Estimating Robot Configuration from Vision Through Transfer Learning

Pedro L. La Rotta [1] **Shrivatsa A. Mudligiri** [1] **Anand B. Iyer** [1]

Abstract

We present an approach for estimating the configuration of a robotic manipulator using vision. While sensors and encoders are built into the design of most industrial robotic arms, we believe there are settings where the integration of these components into a mechatronic system is impossible and thus alternative methods of state estimation are useful. In this work we describe a transfer learning-based approach to extracting the joint configuration of a robotic arm in 3D space from images. We hope that this work can be a stepping stone to robust implementations of visual servoing for sensorless control.

1. Background & Approach

1.1. Background

Most classical and modern methods of control require knowledge of the state of the system. In robotic manipulation, the minimal representation of the state of the system consists of the values of the n joint angles, where n is the number of joints in the arm, and it often also contains information about the velocity of the joints. This state representation generally is constructed from sensors on the robot, and the accuracy and precision of the state depends directly on the sophistication of the sensors.

Encoders are sensors that report on the state (position/velocity) of a motor, and they are most often designed into robotic manipulators. However, highly articulated robots as well as robots deployed in high-energy radiation environments (such as nuclear plants) are limited to the most unsophisticated of encoders, and many simply do not even employ them which hampers their utility. In this work we explore vision-based state estimation as both a supplement and a complement to sensor-based approaches. Our main objective is to build up to a vision-based end-to-end framework that given an input set of images of the manipulator can output a set of joint values describing the state of the manipulator.

[*]Equal contribution [1]Department of Mechanical Engineering, Columbia University, New York, USA. Correspondence to: Pedro La Rotta <pll2127@columbia.edu>, Shrivatsa Mudligiri <sam2465@columbia.edu>, Anand Iyer <ai2493@columbia.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

1.2. Approach

1.2.1. Environment & data description

To bypass the time-intensive process of capturing images of a robotic arm across thousands of poses, we utilized a simulation environment. PyBullet, a renowned open-source physics engine, was our tool of choice due to its proficiency in robotics, machine learning, and realistic environment simulations. Known for its precise simulation of both rigid and soft body dynamics, PyBullet's compatibility with various programming languages, especially Python, makes it an accessible and versatile tool. This environment was ideal for training and testing our machine learning models, allowing for intricate manipulation of robotic arms and other objects, and the customization of camera positions and angles.

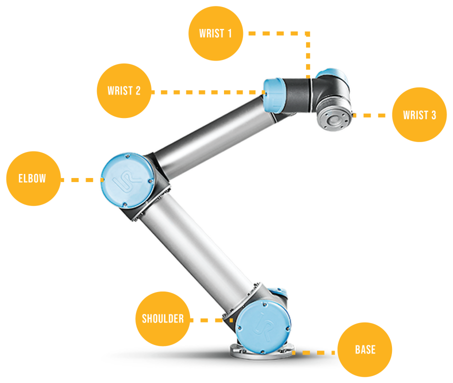

Figure 1. UR5 Robotic Arm

For this project, we used the UR5 robotic arm from Universal Robots as shown in Figure 1. The UR5 is a part of their UR series, designed for safe, collaborative work alongside humans. Its design mimics human arm joints, offering flexibility and precision, critical for our data collection.

1. Planar View Dataset: We initiated our project with a simple setup. Here, the UR5's workspace was limited to a single two-dimensional plane. In this setup, the base joint was fixed, while the other joints (shoulder, elbow, wrist) were allowed to move freely within this plane. A comprehensive dataset was compiled, consisting of 10,000 images of corresponding configuration, with camera shots taken perpendicular to the plane.

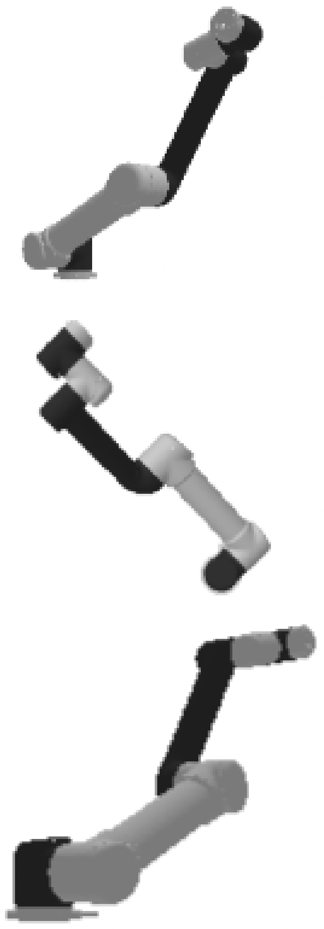

Figure 2. Front, Top and Side view (Top to bottom) of a corresponding robot configuration.

2. 3D space RGB-D Dataset: Advancing from the planar view, we expanded the UR5's operational space to encompass three dimensions. For each of these configurations, determined by randomly assigned joint values, we captured both RGB and Depth images from five distinct perspectives. These included front, top, side, and two additional orthogonal views as shown in figure 2. Our collection comprised 10 images for each of the 10,000 distinct UR5 configurations. In the depth image, each pixel corresponds to distance between the camera and the corresponding point in the scene as shown in figure 3.

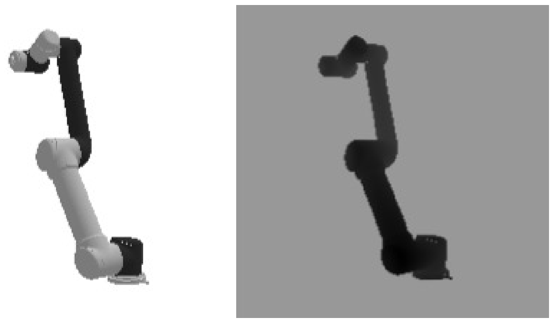

Figure 3. Orthogonal view and its corresponding depth image.

3. Real-world Dataset: Upon achieving promising results with the 3D space depth-enhanced dataset, we introduced more complex elements, such as ground plane and obstacles, into the simulation. This step was designed to mimic real-world scenarios, further testing the robustness of our approach.

1.2.2. Model architecture & data engineering

Our first attempt at building a state estimation model was a shallow Convolutional Neural Networks. This model was comprised of three convolutional layers, followed by max pooling, and three fully connected layers. This architecture was trained on the 'Planar Dataset' with the objective of proving that our concept of estimating joint values from images was feasible. The model demonstrated promising accuracy in this task, providing us with a strong initial validation of its effectiveness. Training and validation loss curves shown in figure 4 and 5.

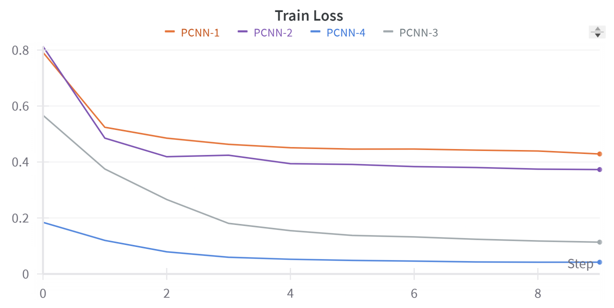

Figure 4. Training Loss curves of shallow CNN's trained on planar dataset

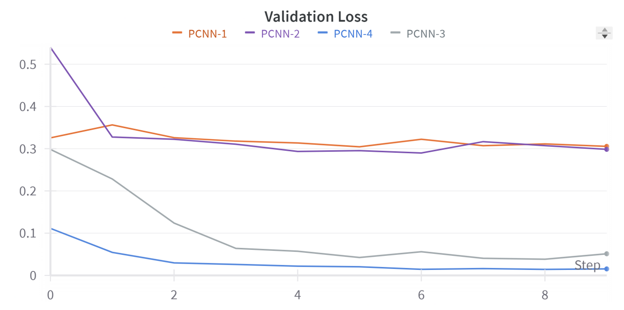

Figure 5. Validation Loss curves of shallow CNN's trained on planar dataset

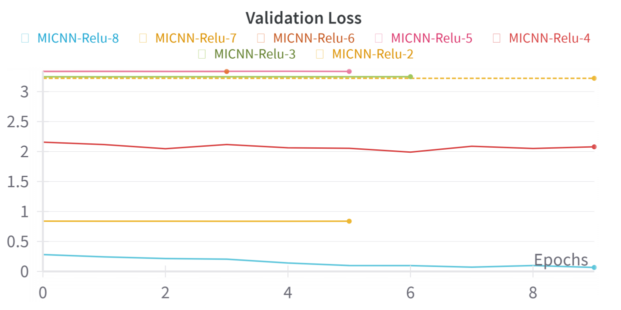

Figure 7. Validation loss curves of shallow CNN's trained on 3D Workspace data

Next we sought to expand the workspace of our model to the full, unrestricted 3-dimensional workspace. Here our approach involved analyzing multiple views of a specific robotic configuration. Each view was independently processed through separate Convolutional Neural Networks (CNNs). The outputs from these distinct CNNs were then concatenated to form a comprehensive feature set. This combined data was subsequently fed through a series of fully connected layers, designed to predict the final output. We conducted extensive experimentation with the network architecture and parameters to optimize performance. Training and validation loss curves shown in figure 6 and 7.

After preliminary experiments with shallow CNNs trained from scratch, we decided that our best chance at building a performant predictor would be through transfer learning. Because feature extraction from images is a well characterized task, several publicly available pre-trained models were at our disposal, and thus most of our architecture engineering was based around the ResNet-18 model pre-trained on ImageNet.

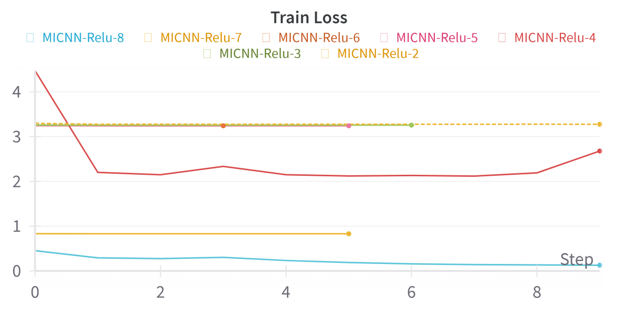

Figure 6. Training loss curves of shallow CNN's trained on 3D Workspace data

We chose a pre-trained ResNet because training a deep CNN from scratch would require prohibitively large amounts of data and compute which we did not have access to. The second reason we chose a pre-trained CNN was that our model simply needs to find the robot-relevant features in the image, and then learn a relationship between these features and the true joint angle values. While ImageNet classification is both a completely different task (classification instead of regression) and a different domain (objects instead of robot arms), we believed that the image embeddings learned by ResNet on ImageNet would likely be relevant for our task or at the very least be easier to fine-tune than a model initialized from scratch

Base architecture: Our networks all modified the input and output heads of the pre-trained ResNet-18 model, but their high-level architecture was the same. The input images were fed into the ResNet to produce a 512-dimensional feature vector which was then passed through 3 fully-connected layers to a final linear 4-output layer of our joint angles (figure 8).

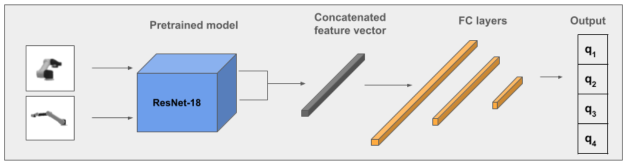

Figure 8. Diagram of base model architecture

Data engineering: One of the main questions that we had in our model building was around the structure of the inputs. Our raw inputs were RGB images of the robot at a single timestep, taken from several different angles. We thus generated 2 different pre-processing workflows to try to answer this question directly: *multi-input* is the approach of treating each RGB as a separate input, running them independently through the ResNet, and then concatenating them together into a unified feature vector. *Multi-channel* is the approach of converting the images from different views to grayscale and then stacking them into a single tensor where each image view is stored in a separate channel.

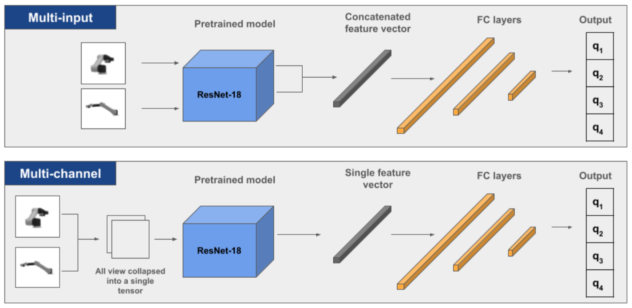

Figure 9. Multi-input and multi-channel approaches to our model inputs

2. Results & Discussion

All submissions must follow the specified format.

2.1. Model Optimization

The first objective of our experiments was to define the optimal architecture and preprocessing workflow for our task. In this experiment we created 12 models, each with a different architecture-preprocessing configuration each probing one of these different areas of the design space:

1. **Number of views:** here we wanted to see how strongly the number of views passed to the model for a single observation affected the training efficiency and the final performance of the model.

2. **Depth:** does switching from RGB to RGB-D data significantly benefit training efficiency and final model performance?

3. **ResNet parameter freezing:** our ResNet was pre-trained on ImageNet, therefore we wondered if the features the ResNet learned to extract are good enough for our task or if we need to learn new ones.

4. **Input stacking**: in this final configuration we sought to compare our two methods of input pre-processing: *multi-channel* vs *multi-input*

Training details: each model was trained for 12 epochs through the AdamW optimizer at a learning rate of 0.0002 and weight decay of 0.0001. Cosine annealing was used to modulate the learning rate throughout training, and inputs were fed into the model in batches of 32. Lastly, all models were trained on a V100 GPU.

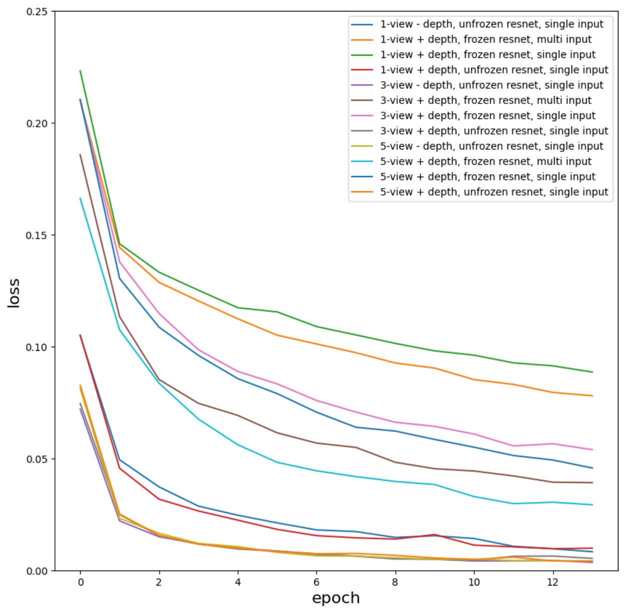

Figure 10. Training loss for the 12 configurations tested.

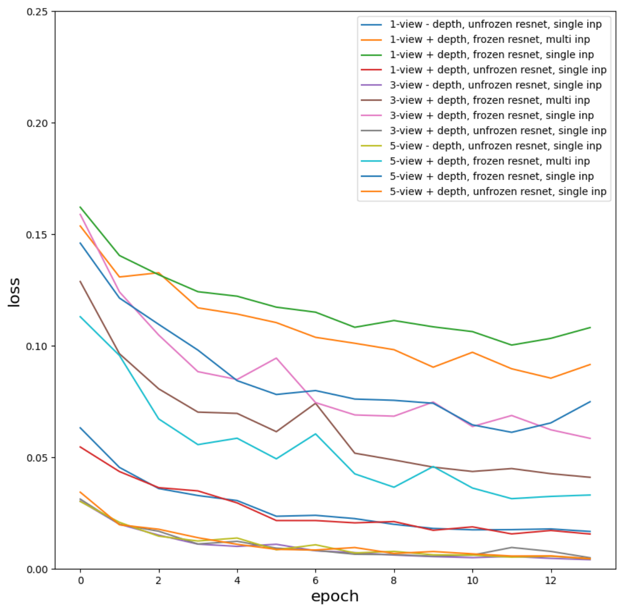

Figure 11. Validation loss

One of our qualitative evaluation metrics was training efficiency because in practice we would want our base model to be easily fine-tuned to new environments and manipulators. As figure 10 shows, the unfrozen ResNets trained more efficiently than the frozen models, and the *single-input* pre-processing workflow outperformed *multi-channel*. As expected, 5-view and 3-view configurations mostly outperformed the single view, but surprisingly the observations also containing depth information did not do significantly better.

Our validation loss plot also shows promising results because the best models have good (¡0.01) training losses (MSE) suggesting strong generalization potential.

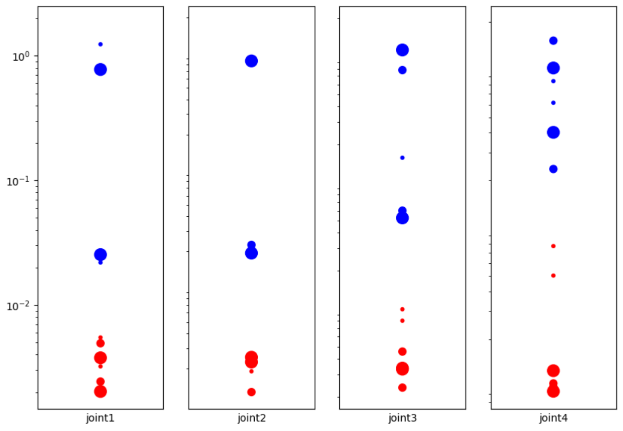

Figure 12. Joint error analysis of the 12 models. Each point represents the mean-squared-error of each model tested on 500 random joint configurations. The size of the point correspond to the number of views contained in each observation (largest points are 5-view and smallest are 1-view) and the color denotes the parameter status of the model's ResNet layers. Red points are models with unfrozen parameters and blue are models with frozen parameters. The y-axis is mean-squared error between the predicted and ground truth joint values.

2.2. Analysis

The final analysis we used to identify the best configuration was a joint-wise error calculation. In this analysis we zoomed in on the errors at each joint for the 12 models. Across all 5 joints, the unfrozen models showed the lowest error, and surprisingly some of the single-view configurations achieved ¡0.01 MSE on 3 out of 4 joints.

2.3. Simulation and Inference

In order to validate our models, the simulation environment used for data collection was used to perform model inference. The cameras used to collect data from the environment were now used as sensors providing input to the trained model. The testing criterion was two-fold, target state estimation and real-time joint inference.

2.3.1. Target State Estimation

For this objective, a sample robot configuration from the test dataset was selected. This sample configuration was then defined as the manipulators target/final state. The vision model was then inferenced using the image input corresponding to this configuration, to retrieve the predicted target joint angles. The arm was then simulated to move to this predicted state, and the difference between the predicted end effector position and the ground truth target was visualized. This can be seen in the figure below.

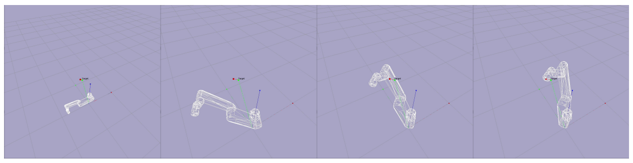

Figure 13. Simulation with predicted vs ground truth target

2.3.2. Real Time Joint Inference

The second objective was to perform real time inference on controlled motion of the manipulator. Multiple target configurations were selected at random, and each was simulated for 50 time steps. At each time step, the vision model was inferenced using the camera sensors in the PyBullet environment, and the predicted joint angles were retrieved. The RMSE between these predicted angles and the ground truth angles of the simulated arm was then calculated and visualized in the figure below. The average RMSE ranged between 0.02 and 0.05 radians, indicating model accuracy consistent with training in real time.

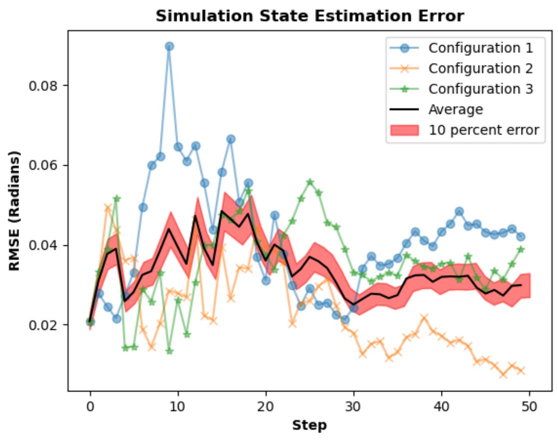

Figure 14. Simulation state estimation error on multiple robot configurations

3. Conclusion

The goal of this project was to estimate the configuration of a robotic manipulator using vision. Specifically, we wanted to extract the joint configuration of a robotic arm in 3D space from images. To do this we collected image data of a UR5 manipulator using a PyBullet simulation environment. This data was then fed into a ResNet18 architecture, with the output being the four joint angles of interest. The performance of the trained vision models was reassuring. With RMSE error on our test dataset well below 0.1 radians, the model was quite accurate. In addition, when inferenced in a real time simulation, this level of accuracy held consistent. These results demonstrate the potential efficacy of a sensorless, purely vision based approach for robot state estimation. This solution could eliminate the need of physical encoder based feedback control in certain environments where sensors are detrimental (like high-energy physics). This would allow for arm independent control, without any additional loads. This in turn would make control of varied robot arm configurations and trajectories much more efficient and cost effective.

4. Future Work

Although our results are promising, there is still need for future work. This project was conducted on a very clean simulation environment, without any background noise to deter model performance. To make this solution more robust and viable for real world application, these models need to be trained in environments with additional noise such as varied flooring, lighting conditions, obstacles, etc. Additionally, to fully verify the viability of this approach, our models need to be validated on a physical robotic arm to determine the effect of domain transfer.

5. Authors contributions statement

P.L. conceived the model architectures and conducted the model optimization experiments, S. M. conceived the simulation environment setup, data collection pipeline and generated all 3 datasets. S.M. conceived the model architectures and conducted model optimization experiments on shallow CNNs for planar and 3D workspace dataset A.I. conceived and conducted the real-time prediction error experiments. P.L. analyzed the results of the model optimization experiments, and A.I. analyzed the results of the real-time prediction experiments, and performed validation tests. All authors edited and reviewed the manuscript.

References

[1] Güler, P., Stork, J. A., &; Stoyanov, T. (2022, July 11). Visual state estimation in unseen environments through domain adaptation and Metric Learning. Frontiers. https://www.frontiersin.org/articles/10.3389/frobt.2022.833173/full

[2] Bateux, Q., Marchand, E., Leitner, J., Chaumette, F., &; Corke, P. (2017, June 7). Visual servoing from Deep Neural Networks. arXiv.org. https://arxiv.org/abs/1705.08940

[3] V. Ortenzi, N. Marturi, R. Stolkin, J. A. Kuo and M. Mistry, ”Vision-guided state estimation and control of robotic manipulators which lack proprioceptive sensors,” 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea (South), 2016, pp. 3567-3574, doi: 10.1109/IROS.2016.7759525.

[4] He, K., Zhang, X., Ren, S., &; Sun, J. (2016). Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://doi.org/10.1109/cvpr.2016.90