# Pixel-to-Pose Estimator for Precise Robotic Tool Alignment

Austin Gurley, *Member, IEEE,* Kyle Kubik,
Joel Bjornstad, Seth Cohen, Robert Amaro, and Mark Patterson

***Abstract*— Robotic manipulation in unstructured environments relies heavily on visual information for context. Images from color (RGB) cameras provide rich environmental data at low cost. Recently, deep learning has proven to be a powerful tool for extracting features from images and using these features to estimate the class and location of objects in the image. However, data collection with physical robotic arms is slow and can be dangerous. The application studied in this work is a robotic arm used in a molten-salt nuclear facility to replace failed pipe flanges. In this scenario, access to physical hardware is only available when the plant is not operating so it is not practical to collect real-life data for training. This work trains a deep neural network, using only simulated images, to estimate the pose of targets in the robot's coordinate system. The estimated position is accurate to 52.7 mm in a 0.95 cubic meter workspace. We describe the simulation, the deep pixel-to-pose estimator model and training, environment and domain randomizations used for training, and demonstrate the performance on a precise robotic tool alignment task.**

## I. Introduction

Robots interact with their environment using sensor feedback to adapt to changes and uncertainties in the environment. Feedback control is a well-studied science when the state of the system can be measured or estimated. Cameras can provide rich information about an operating environment, but to use images for feedback control the pixels must be known to encode a set of state variables. Traditionally this encoding was done using fiducial markers placed in the physical environment [1] or by computer vision methods like feature-matching [2]. These methods are useful for highly structure scenarios, but they do not generalize easily to new scenarios without re-engineering for the new task [3]. The objective of this research is to create a model to find the position of objects in an image, a 'pixel-to-pose' estimator network, which requires neither physical fiducial makers nor human-designed feature matching. Deep convolutional networks have proven to be valuable for classification and localization in images. Using deep learning, a pixel-to-pose estimator can be designed for robotic manipulation tasks.

A pixel-to-pose estimator is useful if it is able to estimate the pose of both the robotic arm and a desired target from a single RGB image that contains the robot and target. It should be robust and provide accurate estimates even when there is distracting information in the background of the image. Data collection from a real robot arm is costly, so the model should be trained only in simulation while still providing accurate pose estimates in real images. These goals are achieved by the deep learning model presented in this work. A simulation environment was developed that replicates the experimental scenario but with extensive domain randomization (Figure 1). This simulation is used to generate training data comprising RGB images of the scene and corresponding robot and target poses. This data is used to train a deep convolutional neural network to estimate key features in the images, then to transform those key features into state variables that represent the robot and target pose. By choking the data pipeline between the convolutional and neural network using spatial softmax attention [4], the model achieves 52.7 mm and 0.8 degrees average accuracy with only 125,034 parameters – an order of magnitude below other solutions (c.f. [5] [6]).

Our model provides only pixel-to-pose estimation rather than directly driving the robot from the model output (c.f. [3] [7] [8]). This is done for three reasons. First, many robotic manipulation tasks can be performed in feed-forward once a sufficiently accurate target pose is achieved: bolt-driving, drilling, and many bin-picking tasks fit this criteria. Second, by estimating the pose state directly from a single image, no temporal information is required in training or inference. This allows simple supervised learning training from (image, state) sets. Third, it enables a human to interpret the pose estimate in the image and therefore to easily determine if the output is reasonable, whereas direct drive from the model could lead to motions that are potentially unsafe for the machine.

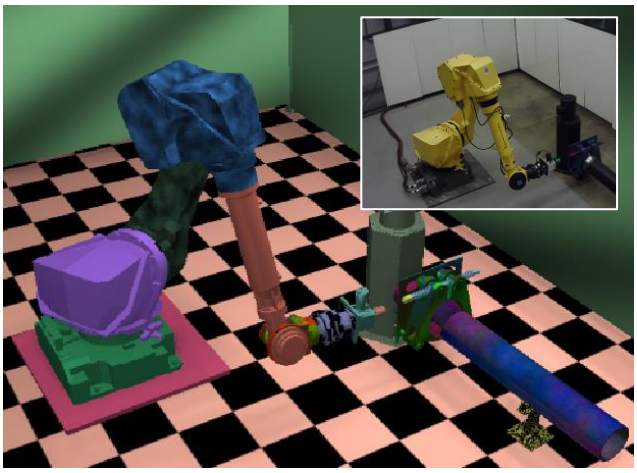

*Figure 1 – The simulation environment demonstrating a typical randomized domain. Inset: Experimental scenario.*

*Supported by the U.S. Dept. of Energy (DE-FOA-0001953-LISE)
Austin Gurley and Kyle Kubik are with Gentle Machine Company, Birmingham, AL 35233 USA (email: austin@gentlemachineco.com)

Joel Bjornstad, Seth Cohen, and Mark Patterson are with Southern Research Institute

Robert Amaro is with Advanced Materials Testing and Technology

This report is arranged as follows: Existing solutions for pixel-to-pose estimators, and examples of training such models only in simulation, are reviewed. The simulation environment that is used to create training data is described along with the domain randomization within the simulation output images. The pixel-to-pose estimator design is explained, especially related to the minimization of network parameters and the design of convolutional layers that provide strong spatial attention. Model training parameters are presented along with a method that was found useful for confirming accurate sim-to-real transfer during training. Finally, the model is used in a real-life robotic maintenance scenario. The accuracy and practical results of that experiment are discussed.

## II. Related Work

The application of deep convolutional networks for extracting image information is not a new concept: they have been successfully applied to classification [9] [10], object localization [11] [12], and 3D depth estimation [13]. By combining a convolutional "feature extractor" with a neural network, deep learning can be trained to play video games at near human level [14] and to pick and place small objects [3]. Many new robotic applications using deep convolutional learning are created every year.

When building deep convolutional networks for robotics, it is important to train models in simulation that can be applied in real life: the "sim-to-real transfer" problem [15]. Models that work very well for purely simulated scenarios (e.g. [14]) will not work in similar real-life scenarios if the model is not carefully designed to transfer. Transfer can be achieved by precise matching of the simulation to reality, by Generative Adversarial Networks (GAN) which transform simulated images to have realistic features [16] [17], and by domain randomization in the simulation environment [5] [18] [19]. Precise matching is usually not practical - the intent of using machine learning is to be robust to variations among any scenario real or simulated. GAN approaches require computing resources that are currently beyond the reach of most practicing engineers. Domain randomization, used in this work, is easy to integrate in most simulation environments and is effective for many types of distractions. It essentially forces the feature extractor to ignore variations in color, lighting, and texture while maintaining a sense of space and dimensions [5].

Machine learning networks for robotics almost always comprise a convolutional feature extractor network which outputs pixel values that are flattened and directly fed into a neural classifier (for object identification) or estimator (for regression). This approach is natural, general, and has proven to be successful in multiple applications [5] [14] [20]. Yet these models requires millions (or tens of millions) of training parameters for common images. By adding a spatial attention layer (spatial-soft-argmax) between the feature extractor and the neural network, these models can be reduced in size by tenfold while still providing sufficient information for the neural network to estimate relevant robotic pose information [4] [7] [21]. The spatial attention layer also provides an excellent restriction in the network. It forces it to encode all state information in the images as a set of coordinate points in that image. Essentially the network is trained to detect a constellation of features which are valuable for the pose estimator and to track the location of those features. This happens in the model optimization process and requires no human design input (besides meta-parameters like convolutional kernel size).

Many applications of deep learning in robotics use reinforcement methods for training [22]. This makes sense for scenarios where the model will be used to output robot actions. However, many robotic tasks can be re-formulated as supervised learning from a single image to an action or a state [7]. There are several benefits to this. A pixel-to-pose network provides position information only, allowing the controls designer to combine the position information in a traditional state estimator that filters it and potentially fuses it with other available measurements [4] [23]. It can also be trained faster than reinforcement learning: since training only requires loading image batches into the optimizer there is no need to record episodes of the model interacting with the environment (c.f. [14] [24]). Finally, many robotic tasks involve accurately moving from a home position to a target position (soldering, bolting, bin-grasping) which can be performed in a single step [25] [26]. While it does not naturally account for potential collision in the motion path (c.f. [27] [28]), it is equally easy to detect obstacles in an image as to detect targets, and design a controller to find a safe path that avoids those obstacles [29].

## III. Methods

The experimental scenario for testing this pixel-to-pose network imagines a molten-salt nuclear facility after a natural disaster: a robotic arm must identify and replace a pipe flange which has been moved to a new, unknown location. The network must estimate both the robot end effector pose and the target flange pose from a single RGB image. When the network can accurately identify the pose of the robot arm and the flange, it will move directly to the target location then loosen the flange using a vendor-supplied soft-insertion routine which uses only end-effector forces for feedback. This scenario was solved in three stages: First, a simulation environment was created with sufficient domain randomization to allow sim-to-real transfer. Second, a deep learning pixel-to-pose estimator was created which maps input image pixels to the robot and target pose. Finally, this network was trained using supervised learning using images from the simulation environment. The trained network can then be used to detect the robot and target pose given only a real life image of the experimental "disaster" scenario.

### *A. Simulation Environment*

A simulation environment was built in Qt (C++) to replicate the experimental setup (Figure 2). This simulation is controlled using a TCP socket and provides control over the virtual robot's joint angles, the camera position, and the position of the target flange. It also provides a domain randomization which modifies the color and texture of each solid body in the scene as well as the location and intensity of the light source (Figure 3).

*Figure 2- The simulation environment approximately matches the colors and arrangement of the experimental scenario.*

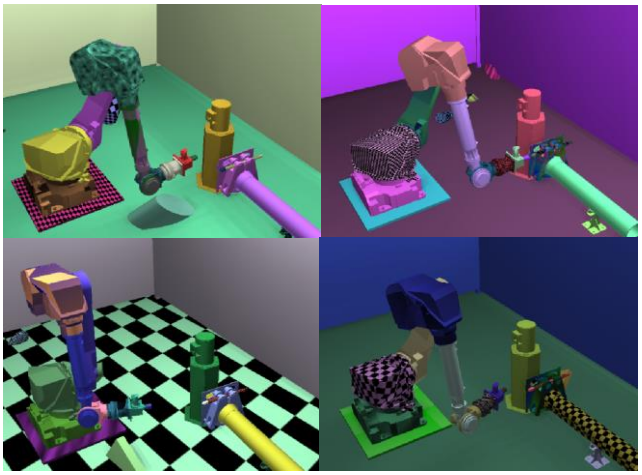

*Figure 3 - Typical examples of simulation images with domain randomization applied*

Qt was chosen as the framework for the simulation environment due to its ability to create multi-platform packaged applications from a single codebase. Through the QtQuick3D module, a 3D environment can be quickly generated and embedded within a traditional graphical user interface so a user can modify the simulation without the need for constant code changes. This environment implements custom 3D models with lighting, shadows, textures, and material properties. Simulated physics are not implemented, but collision checking using oriented bounding boxes allows for detection of any failure states where two objects may interact. In addition, through the use of the Qt QTcpServer object, external programs can be given access to the simulation's features and variables during run-time.

Solid Bodies in the scene are generated either from CAD models or from 3D shape primitives. The robotic arm was generated from models provided by the robot manufacturer, while the target objects were drawn based on measurements of the physical objects. The floor, walls, and distraction objects were generated using 3D shape primitives built into the QT3D framework. The robot and target 3D models are accurate to within 1mm of their real-world counterparts and the overall positions of all objects in the scene are accurate to within 1 cm. The initial camera position is accurate to within 1 cm and 0.5 deg. of the physical camera position. The physical camera has been calibrated using OpenCV to eliminate distortions caused by the lens. These initial accuracies were considered when selecting the range of domain randomization; final random ranges were selected to ensure that the configuration of the physical environment would be guaranteed to fall within the domain, while keeping the amount of training samples needed to be generated as low as possible.

This simulation environment is used to generate 30,000 training samples for the pixel-to-pose network, randomizing the scene and textures in approximately 0.12 seconds per image captured.

*Table 1 - The simulated images were generated over a uniform random range*

| | |
|---|---|
| Robot Pose | Random joint angles<br>End-effector within 0.5 m of target |
| Target Pose | +/- 15cm X, Y<br>+/- 5 deg. Z |
| Camera View | +/- 5 cm X, Y, Z<br>+/- 1 deg. X, Y, Z<br>+/- 0.25 deg. FoV |
| Light Sources | +/- 1m X, Y<br>+/- 50cm Z<br>+/- 30% intensity |
| Object Texture | 13 textures (checkers, gradients, noise)<br>RGB color values from 20% to 100% |
| Object Material | Metalness – 0 to 0.3<br>Rougness – 0 to 10.0<br>Specularity – 0 to 1.0 |

### *B. Pixel-to-Pose Estimator*

The pixel-to-pose estimator network is designed to meet the goals of this scenario by finding a representation of object pose, encoded within a camera image, in a model that can be trained on a common engineering workstation. Practically this restricts the network to 1M parameters, but it was found that a model of only 125,034 parameters provides sufficient accuracy and prevents overfitting. The input to the pixel-to-pose neural network are (a) a single RGB image $I \in R^{180x240x3}$ and (b) the current robot pose $p \in R^6$. The network outputs are (c) the target object pose $x \in R^6$ and (d) the estimated robot pose $\hat{p} \in R^6$. The robot pose output is estimated using only the input image, and is only present as an auxiliary task for faster training ([7] [30]). The network is, conceptually, divided into a vision network comprising convolutional layers, and a pose estimator comprising an arrangement of fully connected layers (Figure 4).

#### *1) Input Image Shape*

The input image size was chosen to provide a balance between network size (number of parameters), spatial resolution of objects in the scene, and the region-of-influence which contributes to each of the key-point outputs. Inputs ranging in size from (60x90) to (320x480) were tested: (180x240) was found to provide accurate spatial resolution while requiring mild computing resource for training.

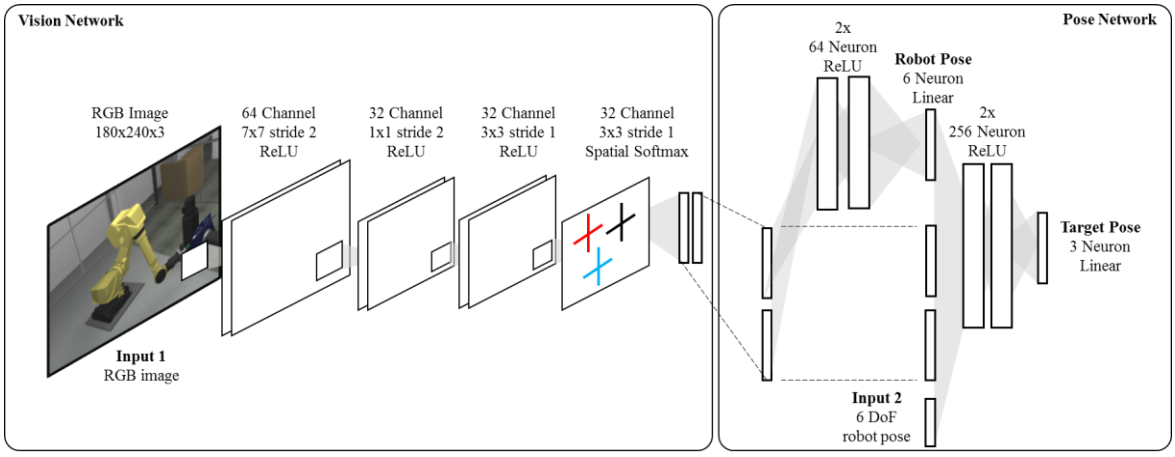


*Figure 4 - The pixel-to-pose estimator combines a vision network and a pose network to estimate the target pose.*

*2) Intermediate Convolutional Layers*

The convolutional layer choices closely follow Zheng et al [7]. The intermediate convolutional layers are common, small convolutions. These layers are initialized randomly with the exception of the first (7x7 kernel) layer which is initialized with the weights of resnet101 [10]. All convolution layers use Rectified Linear Unit (ReLU) activation except the last layer.

*3) Final Convolutional Layer*

The last layer requires special design consideration to achieve accurate tracking. Spatial key-points should provide concentrated activation at small regions of the image, such as corners of the target object or the robot end-effector's tool. Spatial key-points will only be useful if they are unique in the image: due to the nature of the spatial-softmax activation, repeated patterns in the image will be averaged to their (weighted) centroid. To ensure the network will meet these criteria, the final convolutional layer in the network is given L2 regularization so that the total activation of each convolutional channel is small and 'focused' on individual, small regions. An L2 penalty $\lambda_c = 1e^{-5}$ provides good focus without leading to local minima during convergence.

*4) Spatial Softmax*

The vision network output spatial softmax activation is intended to find a minimal set of key-points ($k \in R^{32x2}$), the "constellation", that represent the pose of the robot and the target. Conceptually, a 3D object in an image should be able to be encoded knowing the pixel location of only three features – so tracking the robot pose and the target pose should require only six key-points. However, the model will be more robust if it is trained to determine multiple representations of the scene. It was determined that 32 key-points provides successful tracking for the two outputs in this work. The spatial-softmax temperature was fixed at 0.001 to ensure tracking of concentrated features.

*5) Robot Pose Auxiliary Task*

The pose network uses image key-points and a small fully-connected network to estimate the 6 DoF robot end-effector pose as an auxiliary task. While it is not used when the network is on-policy in real life, the model is trained to predict the robot pose as an auxiliary task using only the vision network output. This approach has been shown to provide faster model convergence. In this work it was originally added for that purpose, but was found to also provide a valuable error estimate by comparing the predicted pose to the true pose while on-policy with real images. These two will diverge when the scene is occluded and cannot be used for predictions. For instance, it will report a large deviation when a person walks between the robot and the camera such that the target is not in the scene.

*6) Target Pose Prediction Network*

The desired target output is predicted by a small fully-connected network. The input is created by concatenating the spatial key-points $k$, the known robot pose $p$, and the predicted robot pose $\hat{p}$. The output $x$ is linear since this is essentially a regression problem to predict the target pose. In this work the target pose has three degrees of freedom since the target flange is mounted on the floor, but it is clear that the technique can easily be extended to more targets by increasing the number of spatial key-points and the number of outputs.

## *C. Model Training*

The pixel-to-pose model was implemented using Keras in Python. Training is performed using a data generator to take (image, robot pose, target pose) tuples from a data-frame linked to a local directory of 30,000 randomized simulation images. The data is split so that 20% is held as test data, then all data is normalized to the training set. Images are normalized using the resnet101 input format.

Since pose estimation is essentially a regression problem, L2 loss provides an objective function that directly corresponds to prediction accuracy. The complete objective function

$$\mathcal{L}(\theta) = \lambda_x \mathcal{L}_{l2} + \lambda_{\hat{p}} \mathcal{L}_{l2} + \lambda_c \mathcal{L}_{l2}$$

provides a loss for the output pose $x$, the auxiliary robot pose $\hat{p}$, and the regularization in the last convolutional layer. A meta-parameter search found good results with the weights shown in Table 2.

*Table 2 – Optimization weights*

| | |
|---|---|
| $\lambda_x$ | 0.7 |
| $\lambda_{\hat{p}}$ | 0.3 |
| $\lambda_c$ | 0.0001 |

Optimization was performed using ADAM. A learning rate of 0.0001, lower than the typical starting value, was required to prevent convergence to local minima (convolution kernels with all zeros), consistent with previous research results [5]. A set of 200 real images were used to create a small test dataset for comparison. While this would not be available in a final application, it was available for this experiment to help guide model design and meta-parameter selection. This dataset of real images was input to the model after each epoch, providing a numeric measure of the sim to real transfer. Convergence for the target pose prediction $x$ is shown in Figure 5. In fewer than 100 epochs the network converges. The accuracy for real images is consistently higher than the accuracy for simulation, showing perhaps that the extent of domain randomization in the simulation is more than what is needed for good sim-to-real transfer.

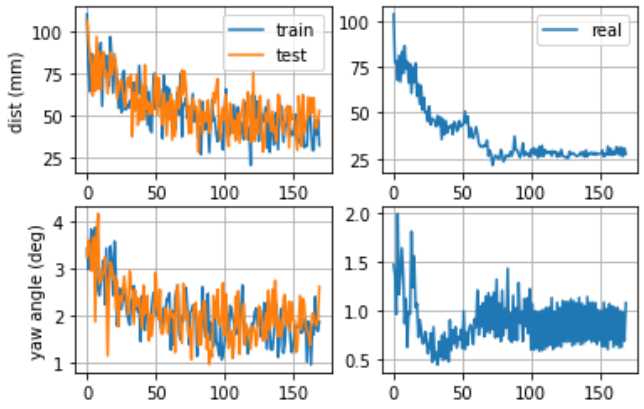


*Figure 5 – Although trained only with simulated data (left), at each epoch the network makes predictions on real images (right).*

## IV. Experiment

To validate the performance of the pixel-to-pose estimator, a scenario was developed to demonstrate a typical robotic arm tool alignment scenario. In this scenario, a molten-salt nuclear reactor is designed with a robotic arm near the working-fluid piping, to assist in case of a system failure. Specifically, in a natural disaster or critical failure of the plant, pipes that contain molten-salt mixture or heat-transfer fluid could be displaced from their mounts and need to be replaced. Each step in the repair requires the precise targeting of an object with the robot end-effector. The targeted object in this experiment is a Grayloc flange which needs to be loosened, lifted, and replaced. If the model can reliably identify the pose of the displaced flange, the robot can move to the flange's locking bolt and remove it (Figure 6).

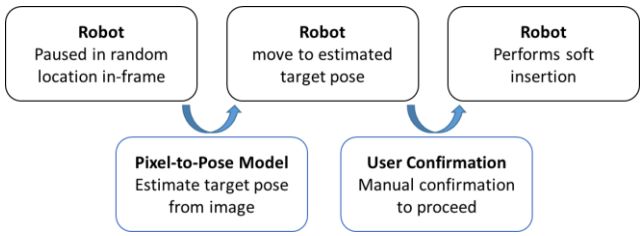


*Figure 6 - Operating process for the robotic maintenance task*

The network designed in this work is well-suited for these scenarios. It allows any object to become a target without requiring fiducial markings and can be trained quickly, after a target is identified. It provides accurate pose estimation even when the camera, target, and surroundings are displaced. It also provides a human-readable constellation of detected key-points to verify a move before it is performed. Finally, it determines the location from a single image, allowing the control interface to regulate the motion of the robotic arm towards the target (Figure 7).

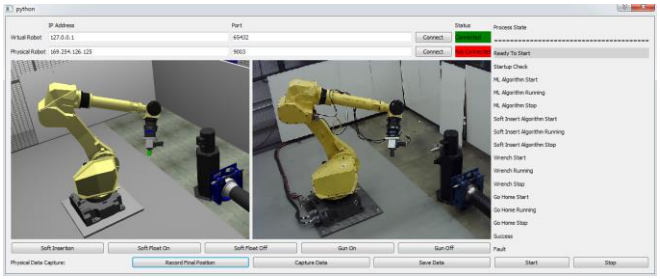

*Figure 7 - A "master program" provides the user interface to observe correspondence between the model and reality.*

## V. Results

Success in the experimental scenario is when the robot arm starts in a random location (in frame), the model determines a flange position, and when the robot moves to that position it is able to unbolt the Grayloc flange. Numerically, that requires the ability to find a target within 30 mm and 1.0 degrees, in a workspace of approximately 0.95 cubic meters. This accuracy is achieved in 240 of 400 tests.

It is perhaps more informative to determine if the solution is stable. As a problem statement: does it provide strong convergence toward the target from any starting position, or will certain starting points cause it to fail? To confirm reliable estimation, the robot arm was moved in a 2D grid near the target with an image and true pose captured at grid points. The network makes a prediction for each of these points with 33.5 mm and 0.3 degree average accuracy. The convergence is demonstrated in Figure 8. This data shows a single minima with a biased minimal solution. This bias appears to be due to limits of the single RGB image: the network's ability to estimate depth is weaker than its estimate of direction.

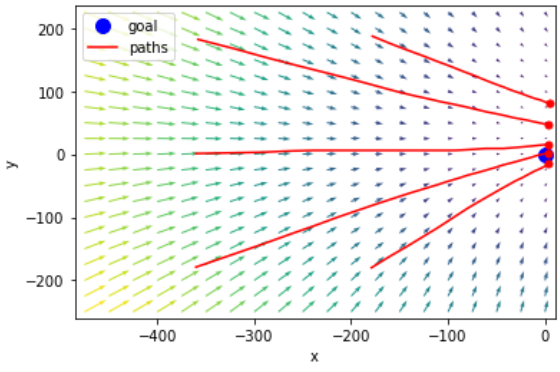


*Figure 8 - The network consistently provides a useful estimate regardless of robot arm starting location.*

## VI. Conclusion

This work describes a model which can reliably predict the pose of objects in a scene relative to a robotic arm. The pixel-to-pose network can be trained using only simulated data then transferred to a real scenario. By encoding the image as a constellation of key-points, the network provides visual validation that it is working correctly.

In future work, this network will be extended to predict the pose of multiple objects in a single scene – this is useful for detection of obstacles in addition to targets. It will also be re-evaluated using stereo cameras to determine if that improves the 3D position estimate. The auxiliary robot pose prediction accuracy was seen to provide a measure of model certainty at run-time – this concept will be explored rigorously in a future experiment. Finally, there are many opportunities to find further efficiency in the model, for instance by optimizing the ratio of key-point to targets, evaluating the input image size, and considering how stereo images, depth images, and grayscale inputs affect the model accuracy.

## References


[1] M. Košťák and A. Slabý, "Desiging a Simple Fiducial Marker for Localization in Spatial Scenes Using Neural Networks," *MDPI Sensors*, vol. 21, no. 5407, pp. 1-31, 2021.

[2] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features," in *European Conference on Computer Vision (ECCV)*, Graz, 2006.

[3] S. Levine, C. Finn and et al, "End-to-End Training of Deep Visuomotor Policies," *The Journal of Machine Learning Research*, vol. 17, pp. 1-40, 2016.

[4] C. Finn and et al, "Deep spatial autoencoders for visuomotor learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, 2016.

[5] 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," arXiv:1703.06907, 2017.

[6] V. Mnih and et al, "Playing Atari with Deep Reinforcement Learning," arXiv:1312.5602, NIPS Deep Learning Workshop 2013, 2013.

[7] T. Zhang and et al, "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation," in *IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, 2018.

[8] A. Rajeswaran, S. Levine and et al, "Learning Complex Deterous Manipulation with Deep Reinforcement Learning and Demonstrations," arXiv:1709.10087, 2018.

[9] C. Szegedy and et al, "Going deeper with convolutions," arXiv:1409.4842.

[10] K. He, X. Zhang, R. Shaoqing and J. Sun, "Deep Residual Learning for Image Recognition," arXiv:1512.03385, 2015.

[11] D. Mascharka and et al, "Transparency by Design: Closing the Gap Between Performance and Interpretability in Visual Reasoning," arXiv:1803.05268, 2018.

[12] E. Ribeiro, R. Mendes and V. Grassi Jr, "Real-Time Deep Learning Approach to Visual Servo Contorl and Grasp Detection for Autonomous Robotic Manipulation," arXiv:2010.06544, 2021.

[13] A. Kendall and et al, "End-to-End Learning of Geometry and Context for Deep Stereo Regression," arXiv:1703.04309, 2017.

[14] V. Mnih, K. Kavukcuogiu, D. Silver and et al, "Human-level control through deep reinforcement learning," *Nature Research Letters*, vol. 518, pp. 529-533, 2015.

[15] W. Zhao, J. Queralta and T. Westerlund, "Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, Canberra, 2020.

[16] K. Bousmalis, S. Levine and et al, "Using Simulation and Domain Adaption to Improve Efficiency of Deep Robotic Grasping," arXiv:1709.07857, 2017.

[17] K. Rao, S. Levine and et al, "RL-CycleGAN: Reinforcement Learning Aware Simulation-To-Real," arXiv:2006.09001, 2020.

[18] M. Andrychowicz and et al, "Learning Dexterous In-Hand Manipulation," arXiv:1808.00177, 2019.

[19] J. Tobin, P. Abbeel and et al, "Domain Randomization and Generative Models for Robotics Grasping," arXiv:1710-06425, 2018.

[20] R. Rahmatizadeh, S. Levine and et al, "Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstrations," arXiv:1707.02920, 2018.

[21] J. Chang and et al, "Learning Deep Parameterized Skills from Demonstration for Retargetable Visuomotor Control," arXiv:1910.10628, 2019.

[22] D. Kalashnikov, S. Levine and et al, "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," arXiv:1806.10293, 2018.

[23] D. Simon, Optimal State Estimation, Germany: Wiley, 2006.

[24] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proc. 35th Int. Conf. on Machine Learning*, Stockholm, 2018.

[25] G. Schoettler, L. Sergey and et al, "Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Rewards," arXiv:1906.05841, 2019.

[26] E. Johns, "Coarse-to-Fine Imitation Learning: Robot Manipulation from a Single Demonstration," arXiv:2105.06411, 2021.

[27] M. Bajracharya and et al, "A Mobile Manipulation System for One-Shot Teaching of Complex Tasks in Homes," arXiv:1910.00127, 2019.

[28] G. Kahn, P. Abbeel and S. Levine, "BADGR - An Autonomous Self-Supervised Learning-Based Navigation System," arXiv:2002.05700, 2020.

[29] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1994.

[30] M. Jaderberg and et al, "Reinforcement Learning with Unsupervised Auxiliary Tasks," arXiv:1611.05397, 2016.