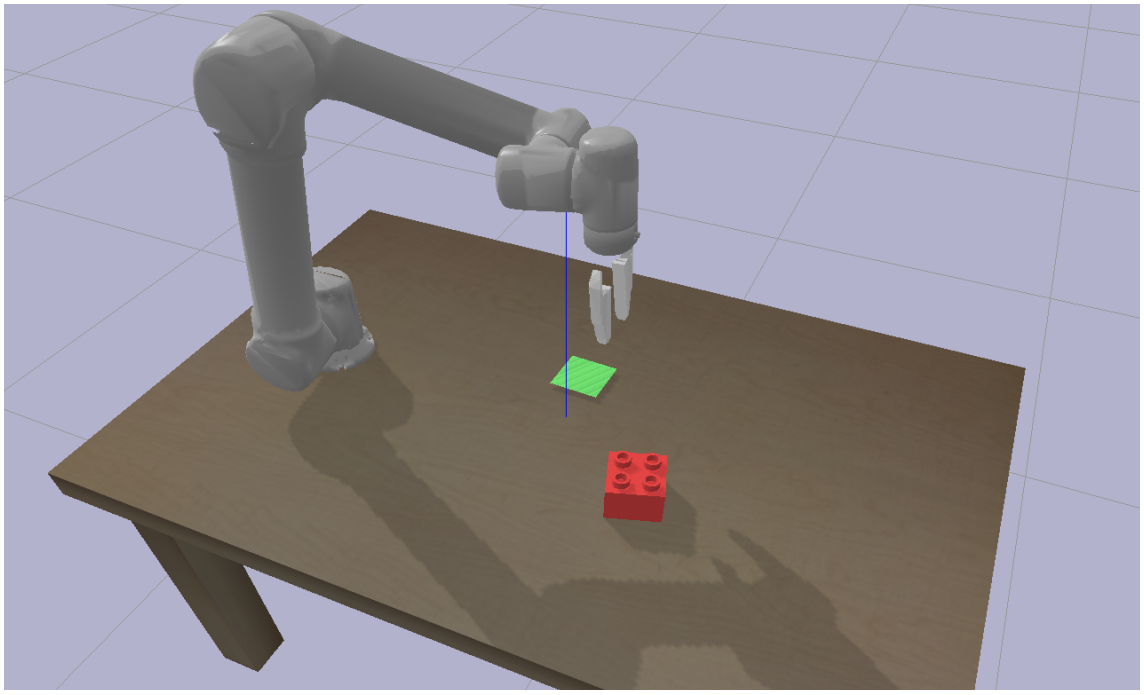
University of Southern Denmark

Project in Advanced Robotics

Visuomotor Behaviour Cloning



Christian Eberhardt	Jakob Grøftehauge Rasmussen	Sina Pour Soltani
chebe17@student.sdu.dk	jakra17@student.sdu.dk	siesm17@student.sdu.dk

Handover Date: 28-05-2021

Supervisor: Rasmus Laurvig Haugaard

Abstract

In this paper, we experiment with Visuomotor Behaviour Cloning within the application of a brick-pushing robot. We design a neural network that uses visual information from a camera setup to arrive at a three-dimensional vector, representing the direction the robot should move to push the brick to the desired position. We design an expert controller to generate training episodes for the neural network, which uses 350 episodes for training. The performance accuracy of the tested models is determined by how successful the robot is in getting the brick to the desired position out of 500 trials. Our baseline model achieved an accuracy of 72.4%. Different strategies for improving the performance accuracy of the robot are investigated, including unfreezing of layers in the ResNet 18 backbone, augmenting the input images, adding noise to the target values, adding an additional camera for a stereo setup, applying dropout and more. The final model used the mentioned improvements and was able to perform with a success rate of 91.8%.

Furthermore, we experimented with reducing the number of training episodes to evaluate the feasibility of this system when trained on few demonstrations. Using the baseline model as a reference, we reduce the number of training episodes linearly. At 35 episodes we observed a performance accuracy of 9.6% - deeming it unfeasible to use in a real-world scenario with only this amount of training.

Contents

1 Introduction

This project aims is to investigate the concepts behind Visuomotor and Behaviour Cloning. Visuomotor addresses the problem of relating a visual input and a control signal while Behaviour Cloning is used to developed control strategies based on expert demonstrations. The inspiration for this project is taken from the paper *Self-Supervised Correspondence in Visuomotor Policy Learning* by Florence et al. [1].

The objective of this project is to obtain a control policy that allows a UR5 robot to push a lego brick from an initial position to a desired position on a horizontal plane - solely based on visual information in the form of RGB images.

Various methods for creating control policies relating visual sensory information from an RGB camera and control input to a UR5 robot will be investigated. The control policy will be learned based on expert demonstrations of the task. The challenge exists in obtaining a general policy based on a limited number of demonstrations. An overall conceptual structure of the final system to be implemented can be seen in figure 1.

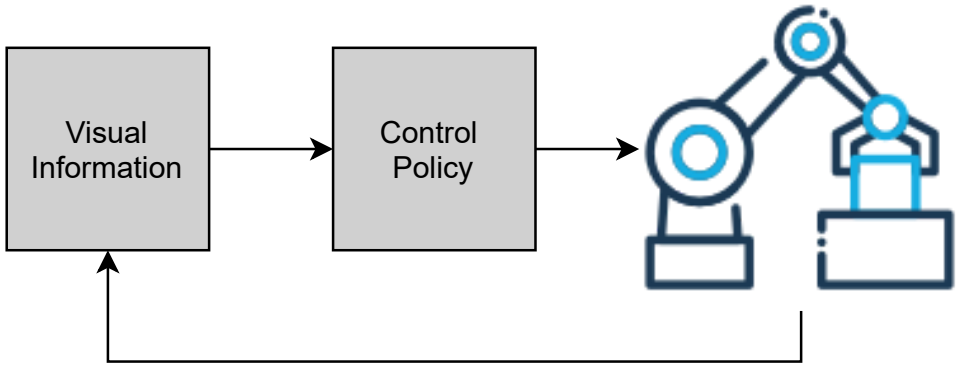


Figure 1: General conceptual structure of the overall system.

This paper is structured as follows: Section 2 outlines how the systems are designed. Section 3 describes the conducted experimentation. Section 4 provides the results of the experiments. In Section 5 the obtained results will be discussed before a conclusion of the findings is made in Section 6.

The work is done exclusively in simulation and the programs are primarily written in Python. The project relies on several Python libraries with worthy mentions of PyBullet [2] for the simulation environment and PyTorch [3] for the ANN. The final implementation of the project can be found at github[1]

[1] https://github.com/SinaPourSoltani/Visuomotor-Behaviour-Cloning

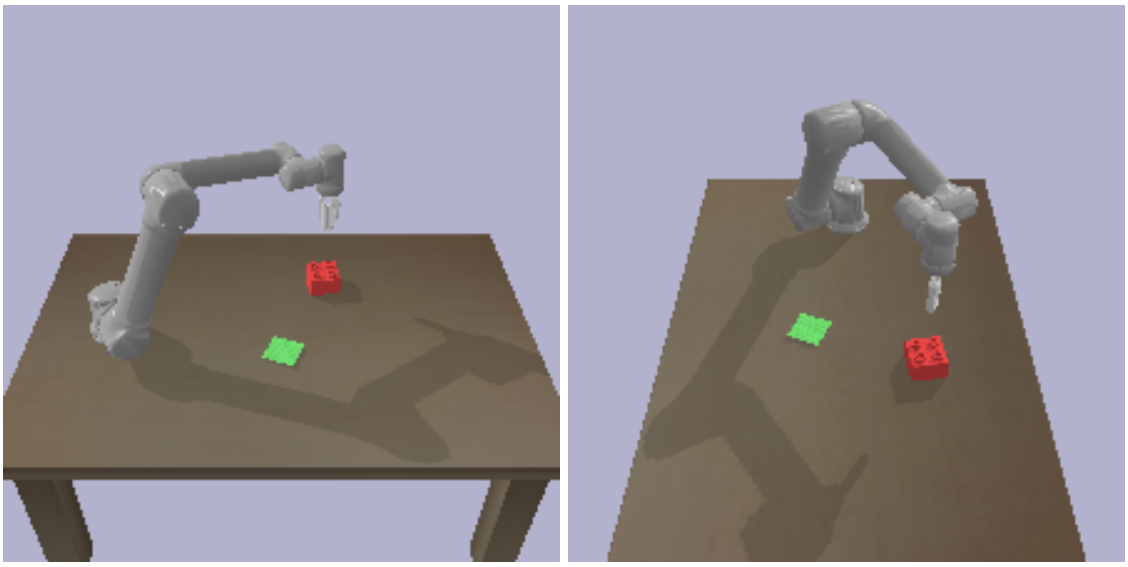
2 Methods

The purpose of this section is to explore different strategies for arriving at a control policy that would complete the given task and further increase the performance. The goal of the task is to push a lego cube from an initial position to a desired position and the system must accomplish this using images as input. For this report, only control policies in cartesian space will be considered.

To learn the control policy a supervised learning scheme is utilised, wherein an ANN is used to predict a target value based on a given input image. The target value comes in the form of a three-dimensional direction vector, indicating the direction the robot should move towards. The image and the corresponding target value is obtained by generating episodes, where the robot pushes the lego brick successfully to the goal, based on an expert policy.

2.1 Simulation

The simulation framework used in the project is PyBullet, which handles the real-time physics and visualisations. The simulation environment is comprised of a four-legged table on one end of which a UR5 robot is attached upright. On the table is also a large red lego brick and a green rectangle indicating the desired position of the lego brick; they are spawned in random configurations in each new episode. The setup is visualised in figure 2. We use two different camera setups for capturing the scene in the simulation. One mono setup with a side view of the table; and one stereo setup which uses the same camera as the mono setup plus an additional camera placed perpendicular to it. Two sample images from the stereo camera setup are visible in figure 2.



(a) Left image. (b) Right image.

Figure 2: Two sample images from the stero camera setup. The red lego cube is the object to be pushed. The green rectangle indicates the desired area of the lego cube to be pushed to.

2.2 Expert Controller

As Behaviour Cloning is a primary objective in this project, the desired behaviour must be designed. The behaviour is expressed by the expert controller, which has full state information at all times in the simulation and therefore knows the pose of the object and the goal. We decided to control the robot in operational space with a three-dimensional vector that denotes the direction in which the TCP of the robot should move. The expert controller is fundamentally a state machine and updates the actions based on the relative position of the TCP, object and goal. The state machine of the expert controller can be seen in figure 3 and visualisation of key areas is visualised in figure 4.

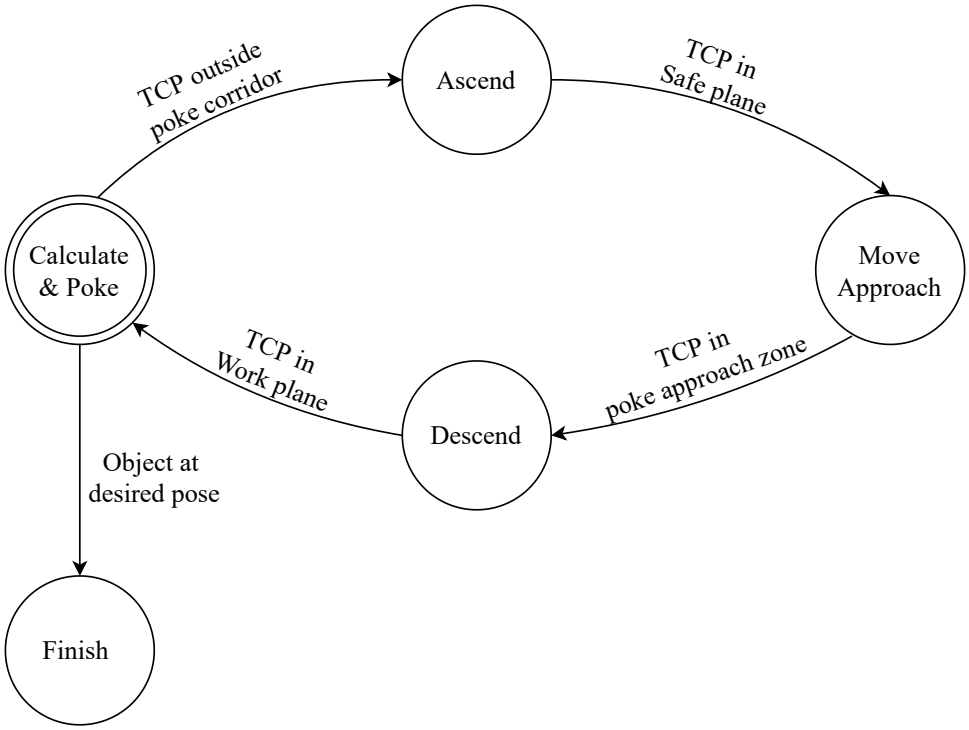


Figure 3: State Machine of the Expert Controller.

The controller starts in the `Calculate & Poke` state. If the TCP of the robot is in the *poke corridor*, the controller uses the state information to calculate a "poke vector"; the direction in which the tool should move to push the object towards the goal. The controller uses the poke vector to move the TCP in the direction of the goal position, which is parallel to the *goal line*. Occasionally, the robot will find itself outside of the *poke corridor*. Either because of the random poses the brick and goal spawn in or as a result of pushing the lego brick and changing the relative angle between the brick and the goal. When this happens the state changes to `Ascend`, where the robot ascends to a predefined *safe plane*; a horizontal plane above the table, where the robot can move around freely without colliding with the environment and in particular with the lego brick in an undesired way. When the TCP is above the *safe plane*, the controller moves the TCP towards

an *approach zone* which lies inside *poke corridor* a specified distance behind the *poke point*. When the TCP reaches inside the *approach zone*, the controller changes state to `Descend`, where it starts moving down to the *work plane* again. Having reached the *work plane* the controller returns to the `Calculate & Poke` state and moves to push the brick towards the desired position. A visualisation of key areas can be seen in figure 4.

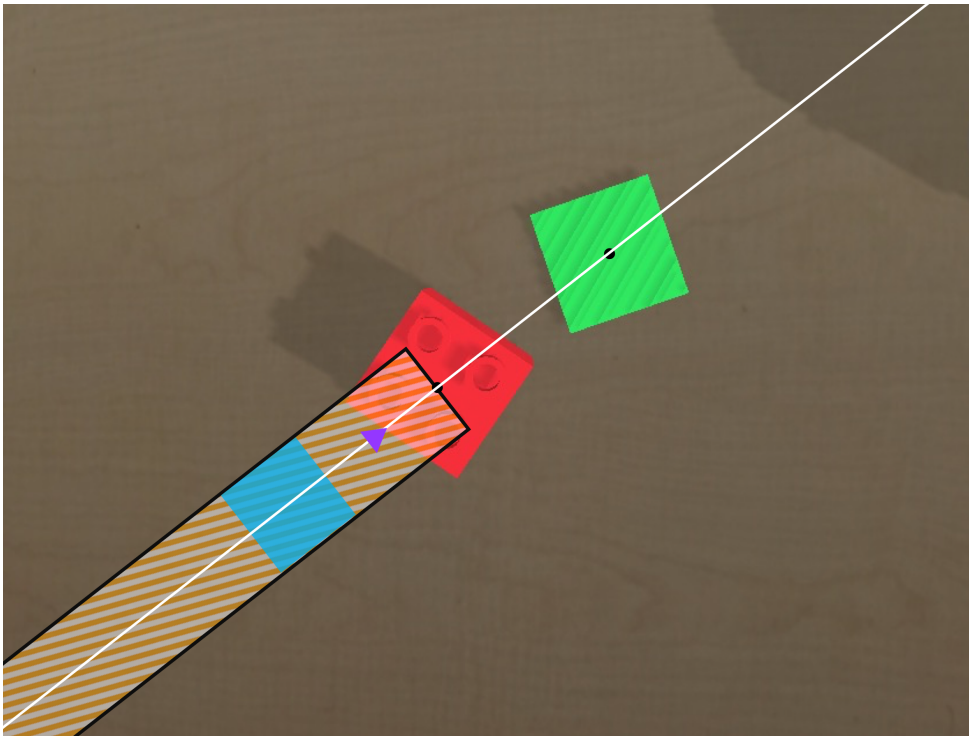


Figure 4: An illustration of the poke corridor (orange stripes), goal line (white line), poke point (purple triangle), and approach zone (blue square).

2.3 Neural Network

To translate the image input into a direction vector, a neural network architecture is employed. It is wanted to create a model, which can regress a directional vector based on an image. The architecture is comprised of a backbone and a single fully connected layer with three output perceptrons. The backbone transforms the image into a feature vector which is forwarded through the fully connected layer and transformed into a three-dimensional vector. This three-dimensional vector represents the directional poke vector. For the backbone, the ResNet18 [4] architecture is utilised. A pre-trained version of ResNet18, trained on ImageNet is used where zero initialisation is used for the fully connected layer. If the model is supplied with multiple images, a feature vector for each image is created by multiple separate ResNet18 models, to designate individual viewpoints in the backbone. The feature vectors are concatenated to a single vector before it is forwarded through the fully connected layer. The network utilises the ADAM optimiser and MSE loss. During training we also experimented with freezing the backbone; this implies that weights

in the backbone are not updated.

The model will be trained using a dataset consisting of 350 episodes unless otherwise stated. Each episode will consist of several data pairs made up of an image and a directional vector, denoting the direction to move the TCP at the given configuration. The direction vector will be used as the target for the model to regress. For validating the performance a validation set of 100 episodes is used.

3 Experiments

To arrive at a high performing control policy, various strategies were explored to test whether they improved or impaired the performance of the brick-pushing task. We started the experiments with a baseline model in a monocular setup trained on 350 episodes and validated on 100 episodes on the architecture described in Section 2. A constant learning rate of $1.0 \cdot 10^{-4}$ was used for the training. What follows are the different approaches that were experimented with, with the purpose of enhancing performance.

Unfreezing layers of the backbone. After training of the fully connected layer for several epochs, we unfroze the last three of the four residual layers of the ResNet18 backbone [5].

Normalising output. As the model is trying to predict a three dimensional unit vector, we tried adding the normalisation step as part of the network to restrict the search space to a unit sphere of unit length rather than allowing any vector.

Noise and augmentation. Adding noise to the target values and to the images should work as a regularising measure, which should reduce overfitting. We added small rotations and translations as well as minimal blur and color jitter to the images. For the targets, we added small rotations to the poke vector. Examples of these augmentations can be seen in figure 5.

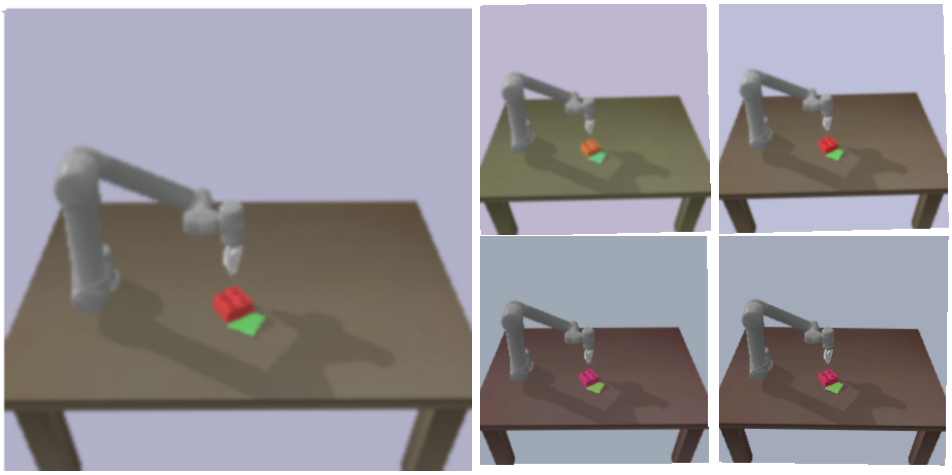


Figure 5: Examples of the data augmentation used at training. The original image is visualised to the left and the 4 small images is the orginal image with the applied augmentation.

Stereo camera setup. Adding several viewpoints would increase the visual information of the robot's relation to the object on the table, and could possibly aid in removing any potential depth ambiguities.

Dropout. Another regularisation approach was used in the form of dropout. Adding dropout between the feature vector and the fully connected layer removes the dependency on particular features and the model should form stronger connections based on a more general view, which should allow for a more robust control policy.

Number of episodes. Finally, we experimented with reducing the number of episodes, that the neural network uses for training. Needing a low number of episodes while arriving at a decent control policy and preserving high performance would be advantageous, should this work be used in a Learning by Demonstration setting, where the demonstrations are manually generated.

Each model, that the system estimated, were then tested and evaluated on a quantitative and qualitative measure. The quantitative performance metric is measured by how many times the robot succesfully pushes the brick to the desired position in 500 different sequences. The configuration of the initial positions of the object and the desired are sampled randomly on the table within reach of the workspace of the robot.

The reason for the qualitative measure is to describe the behaviour of the robot. Initial testing showed that even though the system had not fully estimated a sufficient control policy that allowed it to succesfully complete the task, it had learned *something*. Something, that exhibited a partial learning of the problem (e.g. navigating to the brick, but not pushing it). Something, that would not be implied by the quantitative measure.

4 Results

The results from the experiments mentioned above are shown in the following figures and table. This section features a combined comparison of the performance of all tested models in table 1. Furthermore, a loss graph showing the impact of unfreezing the 3 last residual layers can be seen in figure 6. Figure 7 features a graph showing how the accuracy of the model decreases, as the number of training episodes decreases. The introduced results will be addressed further in section 5.

I	Model	Loss	Performance
0	Expert	—	99.8%
1	Baseline	0.0550	72.4%
2	Unfreeze Layers	0.0472	78.6%
3	**2** + Normalize Output	0.1899	78.0%
4	**2** + Augmentation & Noise	0.0272	84.6%
5	**4** + Stereo	**0.0124**	91.0%
6	**4** + Dropout	0.0232	**91.8%**

Table 1: Performance comparison of the different models when evaluated on 500 different episodes.

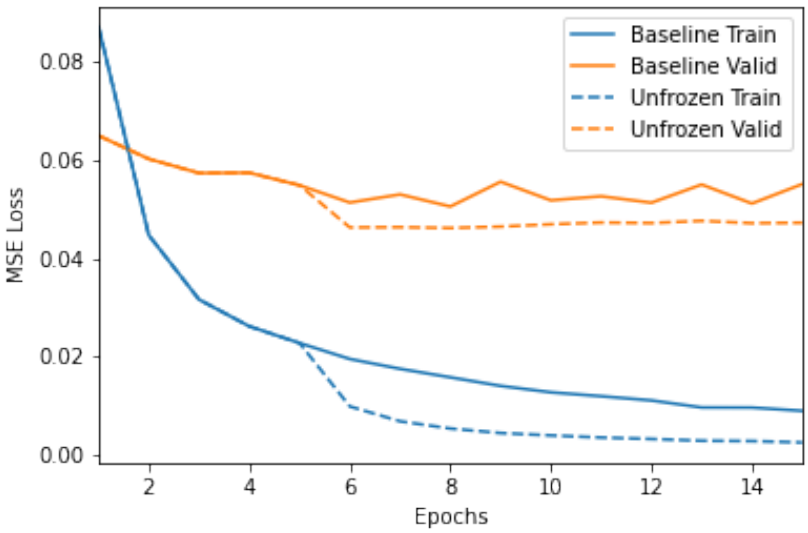


Figure 6: Train and validation loss for the baseline model trained with frozen+unfrozen backbone. Solid line (15+0 epochs). Dashed line (5+10 epochs).

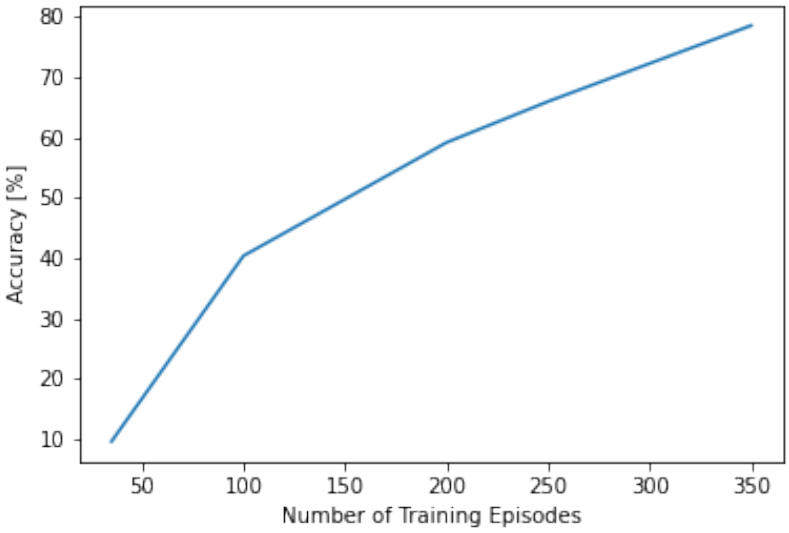


Figure 7: Test accuracy of the baseline model (**1**) as a function of the number of episodes in the trainingset.

5 Analysis & Discussion

Having made the different models, their respective performance accuracies were noted. The Expert Controller (**0**) managed to successfully complete 99.8% of the test episodes. In comparison, the baseline model (**1**) which was trained for 15 epochs successfully completed 72.4% of the test episodes. These results show that there is a lot of room for improvement of the baseline. The first improvement was training with a frozen backbone for 5 epochs followed by training with an unfrozen backbone for 10 epochs. This model (**2**) had an accuracy of 78.6%. This is a substantial improvement, which occurs by optimising the ResNet18 weights to this problem - without disrupting the pre-trained features learned by ResNet, by the large gradients that would occur in the initial epochs. The training and validation loss also decreased sharply when unfreezing was applied as seen in figure 6.

Carrying the improvements forward, we then tried restricting the output to a unit sphere by normalising the output in the model (**3**). This model achieved a performance accuracy of 78.0%. We observed no significant difference between the models with unnormalised and normalised output, implying that the unrestricted model arrives at the correct output of a direction vector without confining the search space. Additionally, normalising the outputs increased the training time of the models and was subsequently discontinued.

Having observed that the movements of the robot occasionally were "jaggy", and that the loss graphs showed that the models were overfitting (figure 6), we tried augmenting the images and adding noise to the target values. This had a great impact on the performance of the model (**4**),

which achieved an accuracy of 84.6%.

We ascribed the remaining unsuccessful test episodes partially to depth ambiguity as well as occasional occlusion of the desired position. Adding an additional viewpoint utilizing a stereo camera setup could add more information to the model and remove the mentioned shortcomings. This is also what we observe as the performance of the stereo setup (**5**) achieves a performance accuracy of 91.0%.

As a final optimisation measure, we tried adding dropout to remove the dependency on specific features and learn to make more general and robust connections. Our results show that adding dropout to the mono setup (**6**) increased the performance accuracy to 91.8%. For a full overview of the performance of the different models see table 1.

Another interesting consideration is the number of episodes used for training the model. So far all the depicted models have been trained on 350 episodes. In figure 7 we can see the effects of reducing the number of training episodes. As expected the performance of the robot decreases rapidly as we reduce the number of episodes. For this to be suitable in a real-world scenario, a few demonstrations as possible is preferable. Nobody has the time or patience for repeating the task with the robot 350 times for it to be able to complete a task relatively reliably. At the lowest tested value, albeit on the baseline model, the performance accuracy was observed to be 9.6%. Rarely do we find a task that can tolerate such low reliability - hardly useful for anything.

On the other hand, we see that the performance accuracy increases as the number of training episodes increases. If it were possible to efficiently close the reality gap, it would be possible to train the model on a high number of simulation examples and successfully port it to a real-world scenario while maintaining a high performance accuracy.

For this project, we explored solving the problem of the robot occluding the objects in the scene by utilising a stereo setup. We acknowledge that multiple techniques for solving this problem exist. Other solutions to the occlusion problem could be to embed knowledge about the previous states of the environment into the model, for example by using an LSTM or other recurrent network structures. This would allow for the robot to learn where the occluded objects are placed.

The stereo setup with augmentation did perform really well in our experiments. Looking at table 1 it is clear that model 5 performs best considering only the loss, which is almost half that of the next best model (**6**) which uses a monocular camera setup. Comparing the two models based on the test performance, however, model 6 has sligthly better performance. During testing of the models we noticed that when the two models fail to complete the task, they do not fail in the same way. When model 6 failed, it was often noticeable that there was a depth ambiguity that caused the model to believe the object was in another position. It would therefore perform the pushing action offset from the real position of the brick and thereby fail to complete the task. Typically when this problem was affecting the model it would do so from the beginning of the episode and the brick would be pushed very little or not at all towards the goal. The problem observed with

model 5 was different. Here the robot almost always managed to get the brick at the goal position or close to the goal position. In episodes where it failed, the TCP of the robot was observed to start oscillating between two positions as soon as it got the brick really close to the goal. This could indicate that the model with the stereo camera setup, could perform much better if the model parameters could be tuned in such a way that it would not stall just before reaching the goal.

Originally, the aim of the project was for the object to reach a target pose. Initially, we then limited the scope of the problem to only include a goal position, believing this to be simpler at first with the possiblity of expanding it at a later stage to include a goal orientation as well. Due to time constraints we never developed an expert controller with this capability and have therefore not trained and tested a model that could also achieve a target orientation. However this would be a more interesting problem to solve and could be considered for future work.

6 Conclusion

In this paper, we examined different strategies for implementing and optimising a visuomotor behaviour cloning system. We used a neural network comprised of a backbone that uses the ResNet18 architecture with a fully connected layer on top. The neural network took an image as an input and would output a three-dimensional vector representing the direction in which the TCP of the robot should move in the workspace to push a brick to a desired position.

We experimented with unfreezing some layers in the backbone, after an initial training exclusively on the fully connected layer. Other strategies for improving the performance of the robot were also tested; including adding an additional camera, using regularising elements such as augmentation, noise on the target value and dropout.

Our experimentation showed that the best performing model would push the brick to the desired position successfully 91.8% of the time. This model was trained on 350 episodes - the highest number of episodes experimented with within this project - and used augmentation on the images, noise on the targets, and dropout as a further means of regularising the model.

References

[1] P. Florence, L. Manuelli, and R. Tedrake, "Self-supervised correspondence in visuomotor policy learning," 2019.

[2] (2021, may) Pybullet. [Online]. Available: https://pybullet.org/wordpress/

[3] (2021, may) Pytorch. [Online]. Available: https://pytorch.org/

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. [Online]. Available: http://dx.doi.org/10.1109/cvpr.2016.90

[5] P. Marcelino. (2018, mar) Transfer learning from pre-trained models. [Online]. Available: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751