Imperial College
London

MSc Human and Biological Robotics
Final Report

Sim-to-Real for Simultaneous Locomotion and Manipulation

Author:
Georges Nomicos

Supervisor:
Prof. Anil Anthony Bharath
Co-Supervisor:
Kai Arulkumaran
Stephen James

January 16, 2020

Word count: 5500

Abstract

The ability to construct robotic controllers for manipulation and navigation would allow us to automate a large range of tasks. Combining both can give access to human-like behaviors, with a navigation to a point and subsequent reaching with the arm. In this work, deep reinforcement learning was explored to enable a wheeled robot to navigate in a virtual environment, and extend an arm toward a target object, a combination of two very different tasks. We outline and successfully train a pipeline for robot learning starting with the sensing and controlling of an agent in a virtual environment. With the final goal of deploying our system in the real-world, the RCAN Sim-to-Real pipeline is extended with a novel dataset for joint locomotion and manipulation, which supports transfer of a learned policy from simulation to the real-world.

Key words: exploration, navigation, manipulation, embedded multi-agent coordination, sim-to-real, partial observability

1 INTRODUCTION

Robotic systems are present in industry and other domains mainly to perform repetitive tasks under human guidance. Machine learning techniques, where algorithms can learn based on real-world data, have the potential to bring generalisation over different environments and robustness to changes in factors such as lighting and weather conditions. Additionally, traditional modular robotic methods which involve constructing and composing separate components such as scene mapping, motion planning and locomotion, suffer from compounding errors and necessitate a considerable amount of prior knowledge [Mishkin et al., 2019]. Reinforcement learning (RL) provides a desirable framework for robotic learning where an autonomous agent receives a (reward) signal at each interaction with the environment, which can then be used to evaluate the pairing between sensory inputs (the state) and actions taken by the robot. Within this framework, end-to-end systems can be learned, so as to map the robot's state to actions, in order to solve the task at hand.

Deep RL (DRL), which combines RL with neural networks as function approximators, has predominantly been applied in simulated environments, initially in games [Mnih et al., 2013, 2015, Narasimhan et al., 2015, Lample and Chaplot, 2017], and later to robotics [Zhu et al., 2017, Rajeswaran et al., 2017, Kalashnikov et al., 2018, James and Johns, 2016, Pinto

et al., 2017, Mnih et al., 2015, 2013, Nair et al., 2017, Gao et al., 2018]. However, damage hazards to hardware and random exploration makes training DRL agents not directly feasible in the real world. Demonstrations of robot manipulation, trained in the real world, required multiple robots and weeks of data collection [Kalashnikov et al., 2018]. Sample efficiency and leveraging privileged information in simulation makes training in simulation desirable. This work investigates a method to train a robot in simulation with the aim of minimising the amount of real data to deploy it in the real world. A second element covered in this research is the training of a mobile manipulator, consisting of navigating to a target and using its arm to reach it.

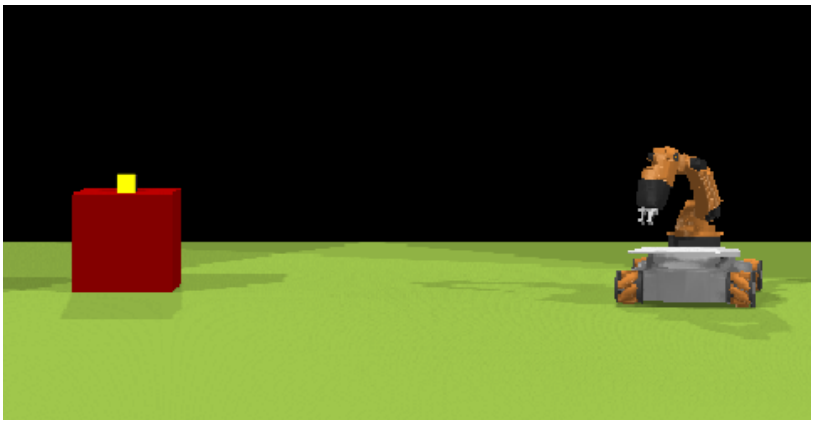


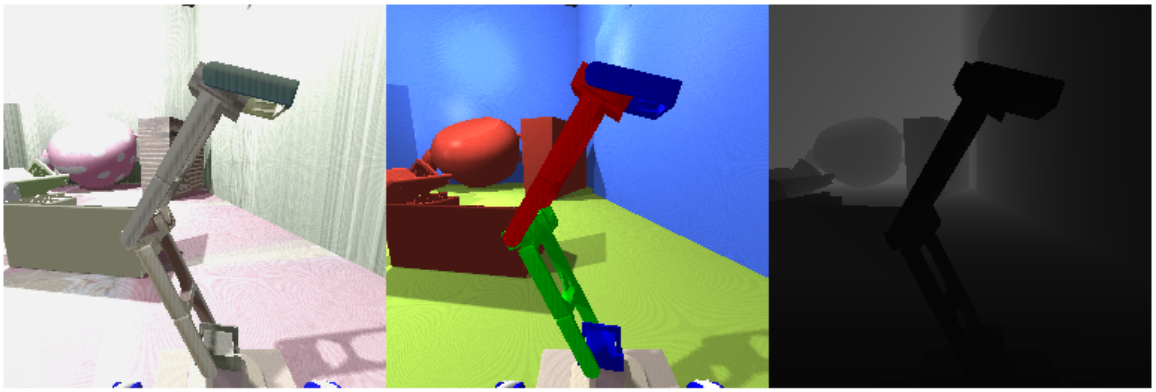
Figure 1: Robot representation in simulation. (Target in yellow)

Mobile manipulator: The latest generation of mobile robots with an aim for assistance and personal use, can also be equipped with a robot arm for manipulation, expanding the work-space. At the stage of software development capabilities, these robots lack flexibility and generalisability in their behavior to be safely and usefully deployed. DRL techniques can potentially be applied to solve this problem. DRL research for robotic sensing and control has focused on either manipulation or navigation on its own. Embedding both a mobile base and a manipulator, within a single unified sensing and control system significantly elevates the complexity of creating time and space efficient controllers. Each of these desired properties are addressed in developing our model pipeline.

The robot employed in this research consists of an omnidirectional platform equipped

with a five degrees of freedom arm (Fig. 1). The considered task for all our experiments is to reach a target, with the base to achieve a positional threshold in two dimensions, with the arm with a positional threshold in three dimensions, and eventually both together.

Sim-to-Real and image-based learning: The gap between simulation and the real world reduces the likelihood of an agent trained in simulation reliably functioning in the real world. The latter is explained by inaccuracies in the mechanical and kinematic models of the robot, the difference between real and simulated sensors, and other physical properties. Similarly to most of the research carried out in DRL, data is collected in a simulated environment: the PyRep robotics tool [James et al., 2019]. The full state of an agent is unavailable in the real-world, directing to informative sensory information such as images. Simulation rendering is usually quite different and simplified—in terms of textures, light attributes and object appearance—from the real-world. The latter indicates two main points requiring change so that a learned simulation model may be transferred to real-world: dynamics and visual appearance. The most reliable domain adaptation [Bousmalis et al., 2018] method applied in DRL for robotics is domain randomisation [Tobin et al., 2017], where several dynamics parameters and visual textures (Fig. 2) (if trained on images) are applied to the scene and robot while directly training the DRL agent on these scenes. However, it complicates training, as the network has to be invariant to both the dynamics of the task and the continuous change in the visual domain. For that reason, we choose to base our Sim-to-Real work on randomised to canonical adaptation networks (RCAN) [James et al., 2018], consisting in learning a mapping from either real-world or randomised images to a canonical (simulation-alike) representation (see Fig. 2); it has previously been applied to robotic manipulation where the majority of the data originated from simulation. We aim to apply this method to our robotic task, thereby expanding its application to navigation and a wider range of environments.



(a) Randomised (b) Canonical (c) Depth

Figure 2: A sample from our dataset: randomised image in the left, canonical image in the middle and depth image on the right

Challenges in robot learning: A perfect state representation is hard to achieve, especially for a navigation: for example, multiple obstacles cannot be represented in a fixed sized symbolic state space. Letting convolutional layers in a neural network decipher features to solve the task might achieve a more generalisable policy. Reward shaping, for long term and general application to different robotics environment is not ideal, and may lead to suboptimal policies [Ng et al.]. A sparse reward [Smart and Kaelbling, 2002] can therefore be more appropriate, where the agent only receives a large reward signal at the terminal state. The absence of guidance from a reward function can be overcome with improved exploration and use of demonstrations [Schaal, 1997].

Generalisation: Controlling the behaviour of a robot in different environments and for different goals is still a very complex, time-consuming and error-prone process. RL is therefore explored to obtain a control policy, based on robot's sensory inputs, that is generalisable to some extent. Different goals are created in this project by randomising the starting and goal positions. With our Sim-to-Real method, a simplified scene representation (invariant to nuisance factors such as lighting) should allow generalisation to any scene upon consideration that a policy capable of avoiding object, and reaching is learned.

1.1 Aims

Coordinating mobile movement with a robotic arm motion involves complex dynamics and high dimensional state spaces. Our work in this paper focused on developing a pipeline to learn visual navigation and manipulation employing state-of-the-art DRL techniques. The focus is directed towards using a minimum of real data and leveraging unlimited availability of simulation data. Our Sim-to-Real side builds on domain randomisation and image-to-image translation. The following targets are set for this project:

- A full training pipeline and actor architecture enabling arm/base coordination
- No real data used in training and deploying our robot in the real world
- Sim-to-Real method for navigation and manipulation with domain randomisation

2 BACKGROUND

2.1 Markov Decision Processes

The tasks consider an agent able to take actions a from a fixed set at a time-step t, while receiving a reward r_t. Each step is represented as a transition (s_t, a_t, r_t, s_{t+1}) where s_t is the current state and s_{t+1} is the resulting state following action a_t. The full state representation might not be available at all time, when observations are images, and we consider sequences of transitions from which we can learn a policy $\pi(a_t|s_t)$. Our tasks are episodic (finite), meaning a terminal goal or time step limit ends the task. To represent this formalism, the Markov decision process (MDP) is employed and reinforcement learning [Sutton and Barto, 2018] is applied to it.

2.2 Reinforcement Learning

In RL, the agent interacts with the environment to maximise future rewards with an aim of maximising the expected sum of rewards. The objective function J is defined in Eq(1). $\sum_t \mathbb{E}_{(s_t,a_t)\sim\pi}[r(s_t,a_t)]$. The maximum entropy RL objective [Ziebart et al., 2008] allows more stochastic policies (useful for exploration in continuous action spaces) by adding a

state-conditional entropy term to the expected sum of rewards:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\pi}[r(s_t,a_t) + \alpha\mathbb{H}(\pi(\cdot|s_t))], \tag{1}$$

where α, a temperature parameter, trades off between return and entropy maximisation.

A drawback of RL on its own is the representation of all (s_t, a_t) pair combinations, being unfeasible for most complex real-world problems. State and state-action pairs are evaluated by their value function $V(s_t)$ and state-action value $Q(s_t, a_t)$ respectively. The most basic RL algorithm simply finds the optimal state-action values and follows the action having the maximum state-action value. For a state/action space that is modest in size, RL algorithms use a tabular format to keep track of each state or state-action value function. By combining neural networks (deep learning) as function approximators and RL as a framework for optimising these networks, RL may be applied to more complex problems. There are two main classes of RL algorithm: on-policy algorithms which optimise the current policy with the latest learned policy and off-policy algorithms which can also use past-policy generated transitions to improve the current policy, thus, making use of the data more efficiently. This work employs an off-policy RL algorithm.

Experience replay (a buffer storing transitions), introduced in Lin [1992], where data is sampled randomly at each gradient update, makes off-policy methods with function approximators more stable by breaking the correlations between samples and reducing variance during updates. It is used in preference to on-policy method because of its improved sample efficiency, and to avoid the learning of biased behaviors from the current policy. For model-free reinforcement learning—not explicitly learning a model of the environment dynamics—state-action values guide the policy improvement, and errors, during optimisation are propagated based on the temporal difference error; such as in the Q-learning algorithm [Watkins and Dayan, 1992]:

$$\delta_t = Q(s_t, a_t) - (r_t + \gamma \max_{a_{t+1}}(Q(s_{t+1}, a_{t+1}))) \tag{2}$$

Actor-critic methods are a class of RL algorithms where the actor model outputs a policy directly, taking observations as input and producing the actions, while the critic model takes the observation and actions to output the state-action value.

2.3 Imitation Learning

Behavior cloning (see Torabi et al. [2018] for more information) consists of doing supervised learning on demonstrated transitions by feeding the observation, in a neural network for instance, and outputting the relevant action.

$$BCLoss = \sum (a_t - a_t^{dem})^2 \tag{3}$$

It is very simple to train, but does not perform well outside the demonstrated state manifold. This loss function does not take into account information embedded into sequences of observations, and if one error would occur then it would results in an unencountered observations as compared to the demonstrations, leading to compounding errors. The "DAGGER" algorithm [Ross et al., 2011] attempts to correct this drift by requiring that the expert can continue be queried during training.

3 RELATED WORK

Learning robotic navigation and manipulation on its own has actively been researched within RL, although both features within a single model has not been attempted to the best of our knowledge. Robotic research focus is oriented towards sample efficiency, solving complex tasks (multi-stage, requiring several assembly steps), generalisation to new environments and goals, and the ability to train in simulation and transfer to the real-world (Sim-to-Real). In the context of RL, robotic environments do not benefit from a pre-built reward function as opposed to games environments, and engineering a reward function for complex agents and tasks may lead to suboptimal behaviors [Ng et al.]. A common practice within RL for robotics is to use a binary reward function where the agent is only positively rewarded for reaching the final goal. However, exploring randomly from scratch with this type of reward function is not sample efficient as the agent requires to reach the final goal to bootstrap information inducing learning. Starting training with a replay buffer filled with expert demonstrations solves both latter issues by improving exploration and adding states to bootstrap from. Sim-to-Real research aims at training a robot in simulation to then deploy it in a real environment with minimal use of real data. Our works focuses on training navigation and basic manipulation with a unique model in simulation and then transferring it to the real-world.

3.1 Learning from Demonstration

Demonstrations have been employed with Deep Deterministic Policy Gradient (DDPGfD) [Veerk et al., 2017] for robotic insertion tasks, with a prioritized experience replay (PER; see section 4.2) [Schaul et al., 2015] to sample demonstrations and high temporal difference error transitions more frequently. Torabi et al. [2018] also incorporated n-step returns, which prove to better propagate sparse rewards across the trajectory. Similar work was done with deep Q-networks (DQNs) [Hester et al., 2018], using behavioural cloning to imitate the demonstrator actions. Hindsight experience replay and DDPG was combined to learn from demonstrations [Nair et al., 2017], in conjunction with HER resulting in more efficient training, by replacing end transitions as the imagined goal in failed episodes. A behavior cloning loss was added to the policy network along with a Q-filter (see Algorithm 1) to account for suboptimal actions from the demonstrations. Their setup was demonstrated on manipulation robotic tasks: pushing, sliding, pick and place, and, on multi-stage tasks which showed accelerated learning on exploration demanding tasks such as block-stacking. Additionally, dexterous manipulation with a robotic hand has been shown to be trainable with the use of demonstrations [Rajeswaran et al., 2017].

3.2 Reinforcement learning for robotics

Model-free DRL has been applied to several robotics tasks: for door opening with a pool of robots collecting experience in real-world [Gu et al., 2017]; visual 2D reaching task with an arm where the real-robot could only succeed with synthetic images as state representation [Zhang et al., 2015]; visual grasping for a large range of objects trained in real-world [Kalashnikov et al., 2018]; navigation with sensory information from a laser transferred well to real-world; cube grasping learned in simulation with deep Q-learning and transfer to real-world with no real-data [James and Johns, 2016]; navigation for collision avoidance using a network to map real images to depth images and a deep Q-network to compute actions [Xie et al., 2017]. Quillen et al. [2018] did a comparative study of various model-free off-policy algorithms for robotic grasping to evaluate generalisation under unseen objects and grasping a targeted object with clutter.

Navigation presents a challenge in learning from images and generalising to different scenes and goals. Zhu et al. [2017] developed an architecture based on the advantage actor-critic DRL algorithm [Mnih et al., 2016] for visual robotic navigation accounting for

different scenes and goals, and trained in a high quality rendering environment to facilitate transfer to the real-world. In our work, real-word images are translated to a general representation (canonical) which facilitates encoding of the scene representation.

3.3 Sim-to-Real

Sim-to-Real is the transfer of a learned model in simulation to real-world with minimal fine-tuning. Robotics research has attempted to train agent in a highly realistic renderer, using sensory information which have similar attributes in the real-world (optical sensor), by performing domain randomisation on the dynamics, by adding noise to the actions to make the model more robust. Domain randomisation appears to achieve the best results for sim-to-real within robotics [Tobin et al., 2017]. Learning from images makes it difficult to include every visual aspect of the real-world in a simulated environment. James et al. [2018] used a generative adversarial network to translate real-world images and domain randomised images to a generic representation (canonical). Instead of training the RL agent on domain randomised input—which the author claims, complicates the learning process—this network translates images to a canonical representation, which the author introduced as Randomised to Canonical network (RCAN). They applied it to visual grasping on a wide range of objects. Our work follows this research but trying to apply it to custom navigation and manipulation task. This technique offers the opportunity to generalise to various scenes by imposing a general color coding and by also predicting depth, giving desired sensory information to learn a navigation task. Within robotic navigation, Sadeghi and Levine [2016] trained a drone in simulation with the task of object avoidance, randomising a lot of visual parameters.

4 METHODS

4.1 Robot and simulator

Our robot consists of a holonomic mobile base with a four degree of freedom arm mounted on it (Fig. 1). The simulation model is made available in VREP. All our simulation experiment are performed in VREP using PyRep [James et al., 2019] to communicate with Python. Mobile robot support was added to PyRep for holonomic and non-holonomic

wheel based robot comprising actuation, collision avoidance and motion planning for linear and non-linear paths. A controller allowing appropriate control of the omnidirectional wheels was as well added. Demonstrations are generated with methods built into PyRep (VREP's motion planning plugin) and do not require any human involvement. For generating demonstration for the mobile manipulation the task is split with two motion planning stages, one for the mobile base and once a threshold distance is reached the mobile base is still, while another motion planner operates the arm to reach the target.

4.2 Reinforcement Learning Agent

An actor-critic architecture is the backbone of our DRL agent, more particularly employing the soft actor-critic (SAC) off-policy algorithm [Haarnoja et al., 2018]. The objective function is augmented with an entropy maximization term with the aim of maximizing both expected return and expected entropy of the policy, thus improving exploration. We chose this algorithm over others for its stability, robustness to hyperparameters and continuous action space, needed for robotics. Two Q-networks are used to compensate for the overestimation of Q-values that occurs when using the same Q-value estimator for both action selection and action evaluation during Q-learning [Fujimoto et al., 2018]. The value network present in the original paper is removed and we instead compute the value function component from the Q-networks. The Q-function approximators are trained by minimising the temporal difference error, while the policy network is updated toward the exponential of the current Q-value.

PER [Schaul et al., 2015] samples some transitions more frequency by altering their priority. We therefore sample demonstrations more frequently by increasing their priority and sample transitions having a higher temporal difference error with a higher probability. For symbolic (low dimensional) observations experiments, the experience replay size is indefinite to keep all transitions, and for visual observation we limit its size to 100000 because of memory availability. We use 1-step returns to not interfere with our reward function. Similarly to Nair et al. [2017], an auxiliary loss for behavior cloning is added to the policy network loss. Furthermore, to account for suboptimal actions from the demonstrator a Q-filter is applied, meaning that when the actor Q-value is higher than the demonstrator's, the behavioural cloning loss is not applied. The behavior cloning loss is calculated as the mean squared error between the demonstrator actions and the agent action at the current stage of

training.

Our reward function is defined in Eq(3) with regards of the distance from the tip of the arm to the target. A positive terminal reward of 1 is accorded when reaching the target and a negative reward penalising high acceleration by taking the difference between the current and previous action. We find that this reward leads to smoother and less abrupt movements, that would otherwise destabilise the base locomotion.

$$r = \begin{cases} -\frac{\Sigma(a-a_{prev})^2}{20}, & \text{if } dist > 0.1. \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

For the mobile manipulation agent, simply training with a simple fully-connected network on symbolic observations did not perform well. Reasons for this originated from destabilisation of the mobile platform with simultaneous motion of both the arm and the base and unnecessary exploration for the arm in specific region of the state space. We therefore introduced a branch in the network to dampen the effect of one another by simply computing a cut-off value with a sigmoid (Fig. 9).

Soft Actor-Critic with demonstrations, algorithm and equations:

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim D}\left[\frac{1}{2}\Big(Q_\theta(s_t,a_t) - (r(s_t,a_t) + \gamma Q_{\bar{\theta}}(s_{t+1},a_{t+1}))\Big)^2\right] \tag{5}$$

$$J_V(\phi) = \mathbb{E}_{s_t\sim D,\varepsilon\sim N}\left[log\pi_\phi(f_\phi(\varepsilon_t;s_t)|s_t) - Q_\theta(s_t,f_\phi(\varepsilon_t,s_t))\right] + \omega_{bc}BCLoss \tag{6}$$

The behavior cloning loss is added to the original actor objective Eq(5). The weight of the behavioural cloning loss is annealed as such: $(N\_DEM/BATCH\_SIZE) \times 0.4$, where N_DEM is the number of demonstrations in the batch, so that as the number of demonstrations in the batch decreases the policy loss privileges current policy improvement based on policy sampled data. When training with visual observations, an auxiliary loss is applied to regress two vectors—the tip to target and base to target—which is added to Eq(5). Also, to reduce the time complexity of the algorithm only the actor is trained on images while the critic is trained on the symbolic observation space, as the critic is not needed during deployment.

Algorithm 1 Soft Actor-Critic with demonstrations

Initialise actor and critic networks with parameters ϕ, θ
Initialise prioritised replay buffer D
Load demonstrations on D
for each iteration **do**
    for each environment step **do**
        $a_t \sim \pi_\phi(a_t|s_t)$
        $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        $D \leftarrow D \cup (s_t, a_t, r(s_t, a_t), s_{t+1})$
    for each gradient update **do**
        Sample a batch B from D
        $\varepsilon_a \sim w_{pr}(Q_\theta(s_t, a_t) - (r(t) + \gamma V(s_{t+1})))$
        for demonstration in B **do**
            Add ε_d to priority
            $a_t \sim \pi_\phi(s_t^{dem})$
            if $Q_\theta(s_t, a_t) < Q_\theta(s_t, a_t^{dem})$ **then**
                $BCLoss \leftarrow \sum(a_t - a_t^{dem})^2$
        Update priorities with ε_d and ε_a
        $\phi \leftarrow \phi + \lambda_V \nabla_\phi J_V(\phi)$
        $\theta \leftarrow \theta + \lambda_Q \nabla_\theta J_Q(\theta)$

4.3 Sim-to-Real

Similarly to James et al. [2018] but with a single convolutional neural network, a U-Net [Ronneberger et al., 2015] style network is trained to translate real-world images to a tuple of depth and canonical images. At training time the input consists of a randomised image where random textures are applied to all renderable objects in the scene, while at test time a real image is fed to the network to output the same representation. This way we bypass the use of real images in the training of our RL agent. The canonical representation is color coded to distinguish scene layout; a single uniform color is applied for each the ground, walls and ceiling, distinct colors for the arm links and a single color for all other objects acting as collision objects in our task.

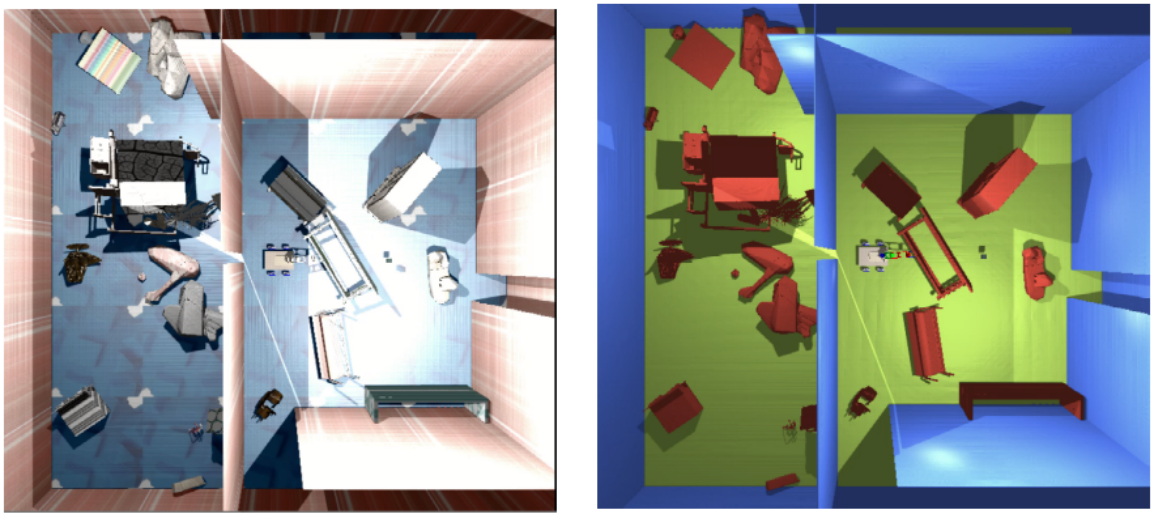
The dataset is generated by generating random scenes, with walls and objects spawning randomly. A fix sized room of 25 m^2 bounds the scene space. Object meshes are collected from the ShapeNet dataset [Chang et al., 2015] and procedurally randomly generated objects. The dataset is customly generated. The steps involve importing a mesh by selecting

a specific number of medium sized and tall objects, creating a collision object with the VHACD algorithm [Mamou et al., 2016], placing the objects at a collision free location, running the simulation for 100 steps to settle all objects in the scene. We then randomly reset the robot position and orientation in the scene while making sure it is not in a collision state and at a correct orientation. We repeat the last step 800 times for each generated scene while applying different textures and reverting back to the canonical representation every time. Scene example of both the randomised and canonical representation are provided in Fig. 3. Over the course of generating the scenes, we vary the number of objects in the scene, walls and lights. One data sample consists of a randomised, canonical and depth image 2. In each scene we collect 800 samples and randomise the scene for generating each sample. The following are randomised: ground, walls, ceiling, each object, each robot link are applied with a texture from a pool of 5000 textures; the height of the ceiling, the position of the lights and their specular and diffuse components, the position of the camera, the brightness and contrast of the image. For the canonical image, all light attributes and camera position are kept fixed in order to learn correct shadow prediction.

The RCAN network is trained with RAdam [Liu et al., 2019] with a learning rate of 0.0003 using the Huber loss. We replace average pooling layers by down-sampling layers from Zhang [2019] to avoid aliasing effect and improve stability.

5 EXPERIMENTS

In this section, our results are presented for the locomotion, manipulation, and mobile manipulation agents. The architecture enabling both manipulation and navigation in a coordinated fashion is evaluated. Training with images is approached while the Sim-to-Real component enabling potential transfer from simulation to the real-world, of a visual trained agent is shown. Evaluation is made during training by running 5 episodes with the current learned policy, every 5000 steps of training, till completion or the step limit. 50 random states are sampled at the start of training and their average Q-values are measured over training. The policy network loss is an indicator of convergence to a suboptimal or optimal policy. All DRL agents are trained on a i7-7700HQ CPU @ 2.80GHz 8 with a GeForce GTX 1060 GPU. Training the locomotion agent took 3 hours, 5 hours for the manipulation task and 15 hours for the final mobile manipulation agent.



(a) Randomised (b) Canonical

Figure 3: Randomly generated scene example. Note the saturation of intensity in one room of the scene, sunlight beams creating apparent lines, clutter, textured surfaces and objects which can confound walls and ground plane (striped). All of these are potentially confusing to the classically engineered vision systems used in robot vision.

5.1 Training Agents in Simulation

5.1.1 Two degrees of freedom (DoF) reacher

Before initiating training on more complex tasks, a simple robotic environment was explored for reaching a target (Fig. 4)- a two DoF robot - to validate our setup and the application of the algorithm. It was successfully trained 5 with symbolic observations, composed of a vector from the tip to the target, joint positions and velocities, and tip position, and with visual observations made of 4 stacked 128x128 grayscale images. The actions are the velocity of each joints bounded with a maximum value. The reward function is the negative of the distance between the tip and the target. Random exploration was performed at the start of training with transitions loaded on the replay buffer. At this stage only the original SAC algorithm and a first-in-first-out (FIFO) replay buffer was employed, no demonstrations were involved due to the simplicity of the task. The reaching error on 10 trials after training was 90% and 70% for symbolic and visual observations respectively.

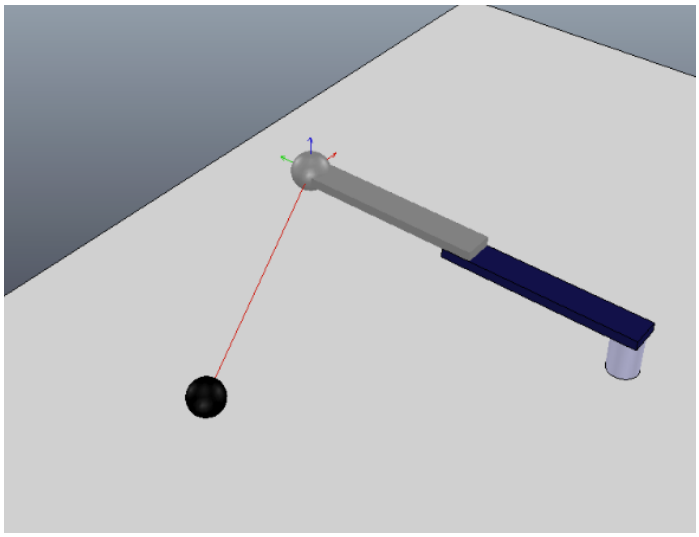


Figure 4: A two DoF robot in a simulation environment

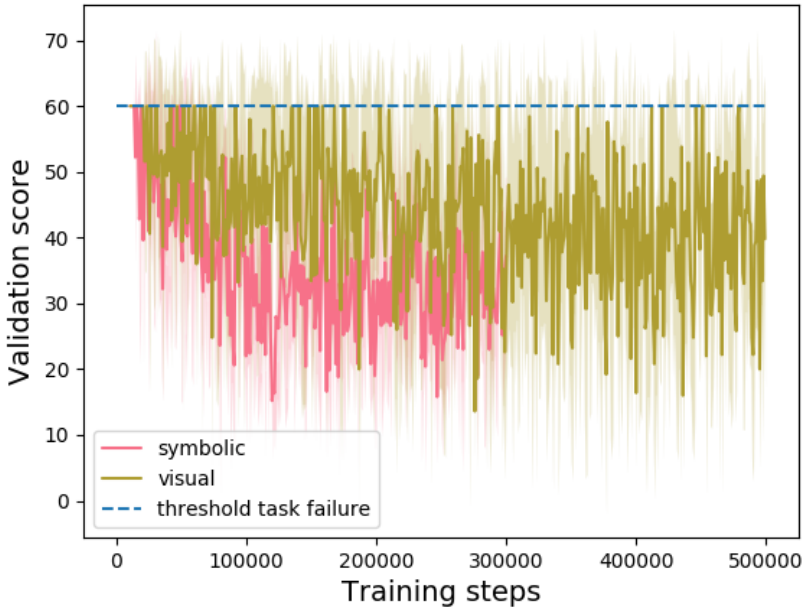


Figure 5: 2 DoF robot, reacher task validation steps over training (evaluated every 1000 steps)

5.1.2 Locomotion and manipulation agent

Training was then performed on the holonomic robot by initially focusing on the mobile base or the robot arm individually. Both were train by employing demonstrations, loaded in the buffer at the start of training, and with the algorithm described earlier in section 4.3. The symbolic state space of the mobile base is composed of the planar position of the base, the translational and rotational velocity, the previous action and the vector components from the center of mass to the target. And for the arm: the joints position and velocity, the tip's Cartesian position, the previous action and the vector to the target from the tip. Two different action spaces were tried: x and y translational velocity and rotational velocity, delta position change for the base. Similarly for the arm: joints velocity and x, y, and z delta position change with a controller taking care of applying the correct joint commands. Both succeeded, and the velocity control was chosen to facilitate control of the mobile manipulator. 30 successful demonstrations were sampled with the intermediate of a motion planning algorithm and store as transitions described in an earlier section. In order, to speed up training, actions are repeated twice at each environment step, by stepping twice in the simulation. The function approximator for the actor and the critic is a three layer fully-connected neural network. The training parameters are: batch size of 128, 200000 gradient update and environment steps for the mobile base and 500000 for the arm, Adam [Kingma and Ba, 2014] as the optimiser with a learning rate of 0.001.

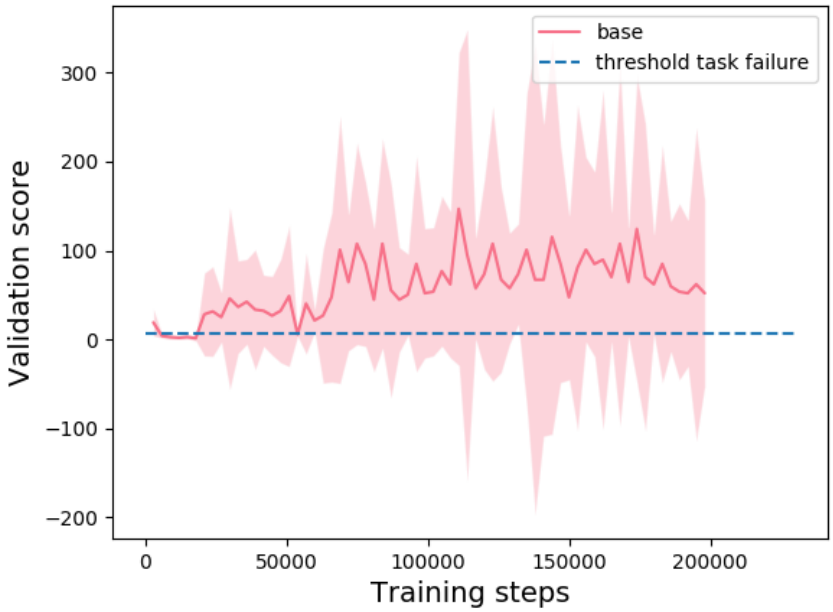


Figure 6: Locomotion reaching task, validation score over training

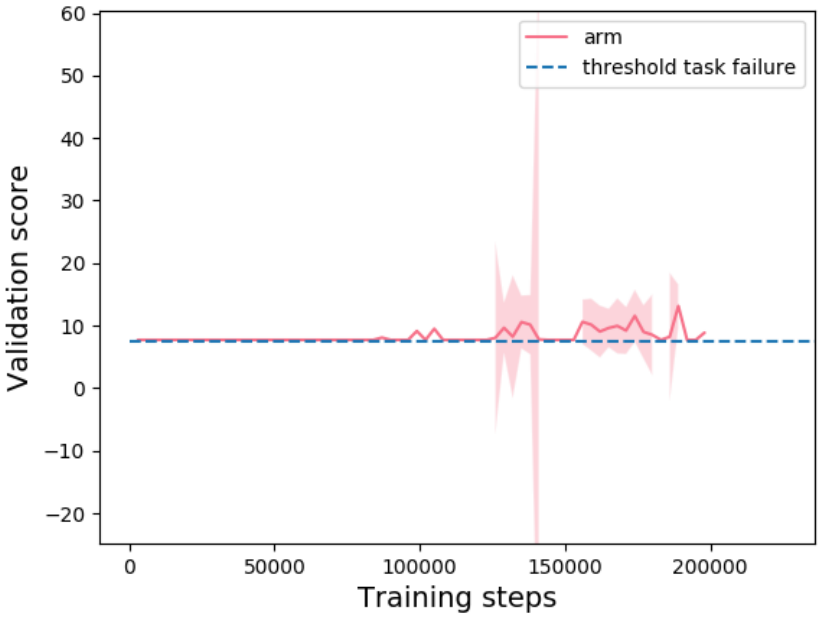


Figure 7: Manipulation reaching task, validation score over training

The algorithm is evaluated on reaching task for both the arm and the base. Both target and agent position are randomised over training for generalisation. After 65000 steps convergence is achieved for the mobile base, with 75000 steps for the arm (Fig. 6 and Fig. 7). The score curve in Fig. 7 for the manipulation task does not show robust learning, the threshold for this experiment was very low thus complicating the task a lot.

5.1.3 Mobile manipulation agent

To combine both the manipulation and the navigation agent as one model, different policy network architectures were attempted. First, a sparse reward must be used in this case to avoid behaviours where the arm would move for no intent. The desired behaviour of the agent (like in the generated demonstrations) is to move the mobile base on its own till the arm is able to reach the target, and then only, the arm is actuated to reach the goal. The sparse reward is 0 for all state-action pairs, and 1 when a threshold of 5 cm is reached by the tip of the arm. Initially, it is hoped that the agent can learn a similar behaviour as the demonstrations by optimising with the behavior cloning loss, while keeping the same network architecture. However, the exploration carried out during training destabilises the base movement due to a high acceleration of the arm. Ideally, we would want the policy network to learn to neutralise either the mobile base or the arm actions. The two attempted architectures are: a fork architecture 8 where the base action are fed back to the penultimate hidden layer and then processed to produce the arm actions, and a gate architecture 9 where a scalar p between 0 and 1 is computed at the penultimate layer and p and $1-p$ are applied to the mobile base and arm actions respectively. Fig. 10 shows a comparison of both the gate and the fork architectures, and their effects on the magnitude of actions for the mobile base and arm over the duration of the task. Both were trained with the same parameters; the fork architecture could not solve the task but approached the target in some instances, while the gate architecture was successful. The gate architecture was incorporated in all subsequent experiments. The gate does not suppress one or the other entirely and seems to have learn a nearby scalar of approximately 0.7 for p which lowers the arm action constantly and does not affect the base extensively.

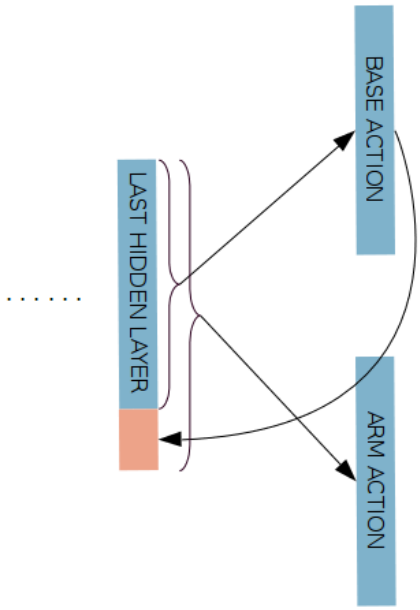


Figure 8: Fork architecture for the policy network of the mobile manipulator

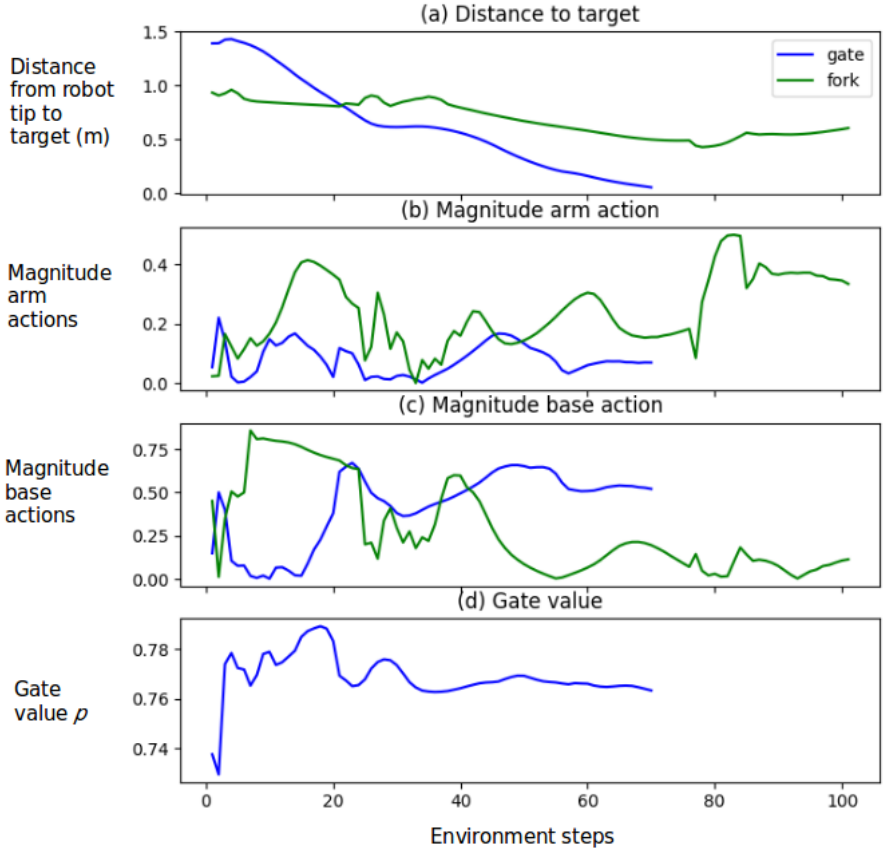


Figure 10: Policy network architecture comparison

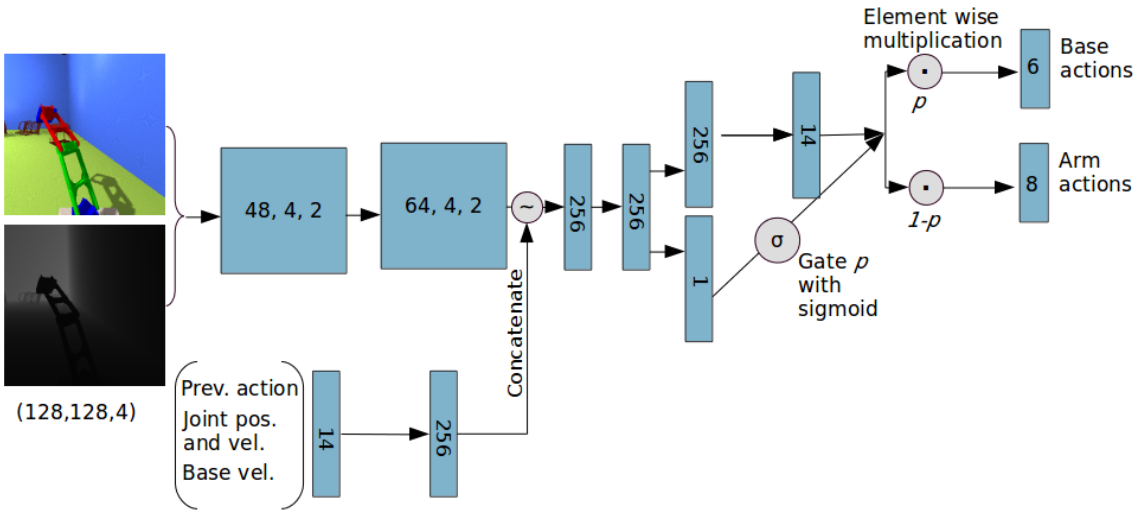


Figure 9: Policy network (Actor) architecture to achieve a coordinated behavior between the mobile base and the manipulator

The validation episodes in Fig. 11 for the mobile manipulator shows learning after around 280000 steps.

5.1.4 Training the robot with visual observation

Deploying the mobile manipulator agent in the real world requires visual sensory inputs. For this, the observation space is built as a combination of images composed of a 128*128 RGB image and a depth image, and low dimensional input composed of joints position and velocity, base velocity, and previous actions. The gate architecture is implemented at the end of the policy network 9. Two convolutional layers at the start of the network decode information about the target, the agent position and obstacles position.
Due to compute availability and to improve training efficiency, while taking of advantage of training in simulation, the critic takes low dimensional observations (the same as in section 5.1.3, when trained on symbolic observations), and the actor is trained and tested on visual input. The high dimensional actor and low dimensional critic setup is similar to Pinto et al. [2017]. Training the critic on low dimensional observation allows us to load a pretrained critic, from the previously trained model on symbolic observations.

A scene with several distinct objects and with the color coding from the canonical rep-

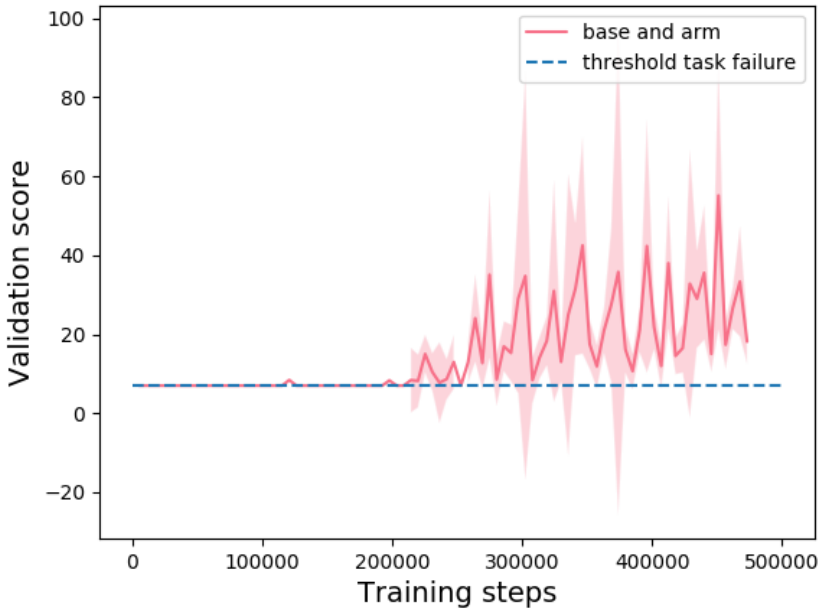


Figure 11: Mobile manipulation reaching task, validation steps over training

resentation is created for the training (Fig. 12.c). The distractor objects are used as clues, object avoidance should also be learned at the same time. Time constraint did not allow us to fully train the visual mobile manipulator agent. Preliminary results are indicated in Fig. 12, where a random state is fed to the current actor network to obtain the first convolutional layer activation. The network seems to grasp the importance of the target and link of the robot arm.

5.2 Training RCAN

In order to train RCAN and to adapt to real-images, a set of 120000 samples are collected over 150 randomly generated scenes. The dataset is generated on a i7-7700HQ CPU @ 2.80GHz 8 with a GeForce GTX 1050 GPU. It took approximately 2 days to generate this amount of data. The U-Net style neural network is trained with a batch of 32 for 60 epochs by parallelising the training over 4 GeForce GTX 1080 Ti GPU. To account for sensory noise from the real camera a Gaussian noise was added to the input training set

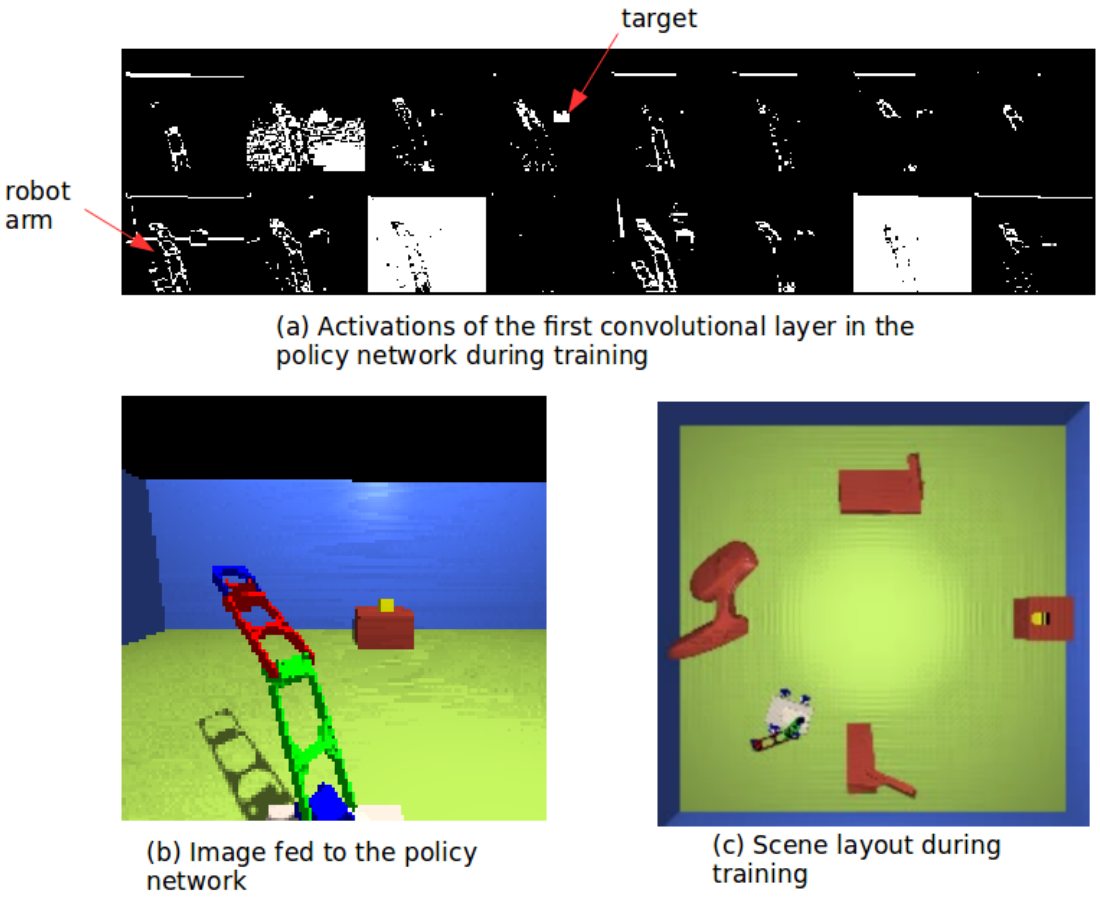


Figure 12: Training the mobile manipulator with visual observations.

with a standard deviation of 0.1. Evaluation is performed over a held out set of randomised simulation images and six real-world images with different attributes (lightning, shadows, long-horizon, lot of objects). Current results are shown in Fig.13 and Fig.14.

6 DISCUSSION

The two DoF reacher environment trained with symbolic and visual observations showed better results with symbolic observations, we might hypothesise than the quality of the images is too low and thus affecting the precision of the reaching movement.

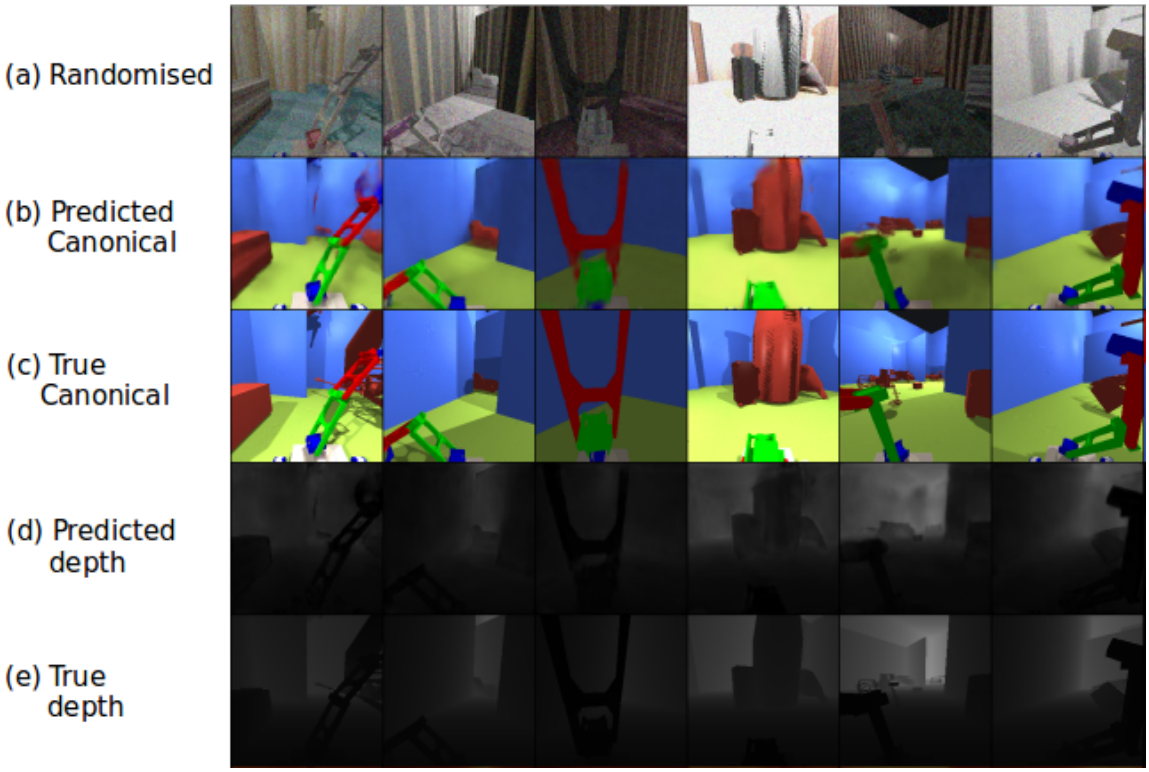


Figure 13: Validation set RCAN after training 50 epochs

The learned behavior for the locomotion and manipulation agent confirmed the validity and reasonable sample efficiency of the built algorithm which learned with approximately 100 episodes for the mobile base and 180 episodes for the arm. Moreover, the mobile manipulator learned the desired policy by navigating close to the target without the arm disturbing its motion. More experiments should be carried out, to evaluate, the range of the learned policy under different training conditions and its generalisation to different starting and goal positions. The learned parameter p for the gate architecture, resulted in a constant value. It could be affected by one of the reward function components penalising acceleration. The inclination of the gate value towards keeping the base actions and damping the arm actions may be explained by the task structure when the mobile base is actuated two-thirds of the time. The visual mobile manipulator setup showed promising initial results, which we observed by analysing the auxiliary loss and the convolutional layer activations. The actor network approximately predicts the goal to tip and goal to base vector components.

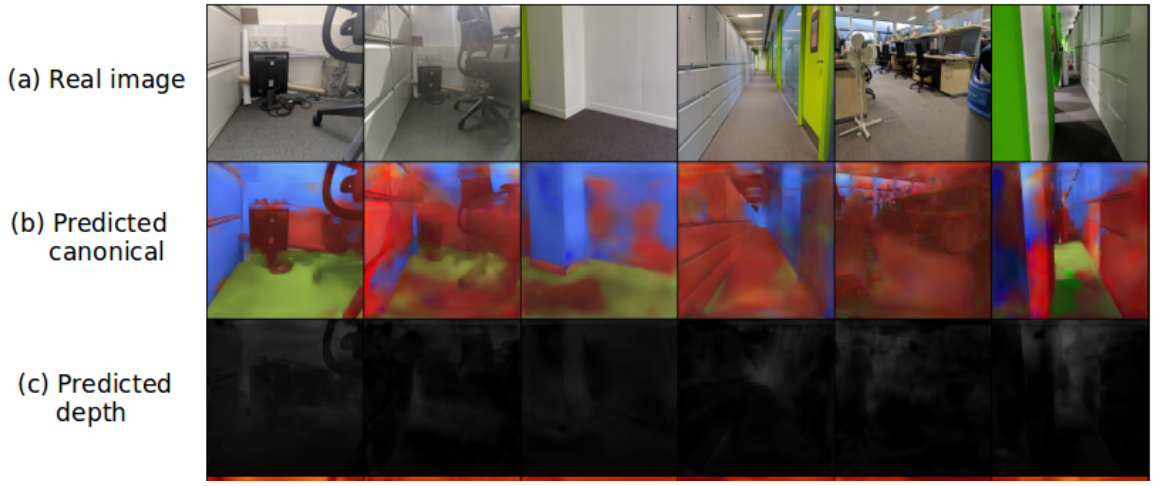


Figure 14: RCAN tested on real images after training 50 epochs

Additionally, the first convolutional layer pays correct attention to the link of the arm and the target. Training for longer periods of time should solve the task.
Concerning the Sim-to-Real component, the RCAN results may be improved by enlarging our dataset, thus generating more images, and with wider randomisation range. All the lighting attributes present in the simulation such as shadows, reflections, intensity contributes to making the network more robust to real scene aspects. In some scene examples, saturation should be changed to improve the distinction between objects. Finally, the Gaussian noise added to the latest training of RCAN, improved the real image predictions, but reduced the validation set prediction quality, indicating that it should only be applied with a certain probability, and thus not to the whole dataset.

7 CONCLUSION

To summarise, a mobile manipulator agent was trained with a novel actor architecture, by incorporating a gate at the penultimate layer of the policy network. The policy exhibited similar behaviour as the demonstration which was desired. The training setup for visual sensor based learning was implemented with promising results showing potential learning of the mobile manipulation task with images. In addition, a novel dataset with complex vision attributes was created with the goal of enabling Sim-to-Real for robotic navigation

and manipulation, and used to train an RCAN model.

ACKNOWLEDGMENT

I would like to thank Kai Arulkumaran and Stephen James for their advice and support along this project. I also thank Prof. Bharath for his comments and supervision. Thank you to the Dyson Robotics Lab for providing computational resources.

References

Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015. URL `http://arxiv.org/abs/1512.03012`.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Yang Gao, Ji Lin, Fisher Yu, Sergey Levine, Trevor Darrell, et al. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*, 2018.

S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, May 2017. doi: 10.1109/ICRA.2017.7989385.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL `http://arxiv.org/abs/1801.01290`.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Stephen James and Edward Johns. 3d simulation for robot arm control with deep q-learning. 09 2016.

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *CoRR*, abs/1812.07252, 2018. URL `http://arxiv.org/abs/1812.07252`.

Stephen James, Marc Freese, and Andrew J. Davison. Pyrep: Bringing V-REP to deep robot learning. *CoRR*, abs/1906.11176, 2019. URL `http://arxiv.org/abs/1906.11176`.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018. URL `http://arxiv.org/abs/1806.10293`.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Khaled Mamou, E Lengyel, and Ed AK Peters. Volumetric hierarchical approximate convex decomposition. *Game Engine Gems 3*, pages 141–158, 2016.

Dmytro Mishkin, Alexey Dosovitskiy, and Vladlen Koltun. Benchmarking classic and learned navigation in complex 3d environments. *CoRR*, abs/1901.10915, 2019. URL `http://arxiv.org/abs/1901.10915`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. *CoRR*, abs/1709.10089, 2017. URL `http://arxiv.org/abs/1709.10089`.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping.

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291, May 2018. doi: 10.1109/ICRA.2018.8461039.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

Fereshteh Sadeghi and Sergey Levine. (cad)$ˆ2$rl: Real single-image flight without a single real image. *CoRR*, abs/1611.04201, 2016. URL `http://arxiv.org/abs/1611.04201`.

Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

William D Smart and L Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 4, pages 3404–3410. IEEE, 2002.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Joshua Tobin, Rachel H Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

Matej Veerk, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothrl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. 07 2017.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *CoRR*, abs/1706.09829, 2017. URL `http://arxiv.org/abs/1706.09829`.

Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, 2015.

Richard Zhang. Making convolutional networks shift-invariant again. *CoRR*, abs/1904.11486, 2019. URL `http://arxiv.org/abs/1904.11486`.

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. 2008.