Sigma-Delta Networks for Robot Arm Control

Wallace Lawson
Anthony Harrison
J. Gregory Trafton
ed.lawson,anthony.harrison,greg.trafton@nrl.navy.mil
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Washington, DC, USA

ABSTRACT

Our autonomous robot, Bight, can be a reliable teammate that is capable of assisting in performing routine maintenance tasks on a Naval vessel. In this paper, we consider the task of maintaining the electrical panel. A vital first step is putting the robot into the correct position to view all of the parts of the electrical panel. The robot can get close, but the arm of the robot will need to move to where it can see everything. Here, we propose to solve this using a sigma delta spiking network that is trained using deep Q learning. Our approach is able to successfully solve this problem at varying distances. While we show how this works on this specific problem, we believe this approach to be general enough to be applied to any similar problem.

CCS CONCEPTS

• **Computer systems organization** → **Robotic autonomy**; • **Computing methodologies** → **Neural networks**; **Reinforcement learning**.

KEYWORDS

Sigma-Delta Networks, Spiking Networks, Deep Q Networks, Robotics

ACM Reference Format:
Wallace Lawson, Anthony Harrison, and J. Gregory Trafton. 2023. Sigma-Delta Networks for Robot Arm Control. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this paper, we examine the problem of how our quadruped Boston Dynamics spot-mini robot (see "Bight" in Figure 1) can assist sailors with one of the tasks involved in maintaining a naval vessel: inspecting, cleaning, and repairing electrical panels.

Maintaining electrical panels involves a series of steps (removing dust, checking for shorts, etc.), but before these can occur, the robot must navigate through the environment, position itself in front of the electrical panel and position the arm so that the sensor is in the correct location for maintenance.

Due to uneven sea-states, and mapping/navigation errors it is likely that the arm will not be in exactly the same 3D location and perhaps not in an actionable location. Further complicating the problem is the fact that Bight, like all mobile robots, has limited on-board computation and power. Power in particular is especially limited on ships.

This task highlights Bight's need to reliably positioning it's arm while consuming as little power and computational resources as possible.

Current state of the art approaches use deep networks (e.g., ResNet-52) to perform tasks such as object detection. In an environment with limited power and connectivity, these traditional methods are less desirable, thus driving the need for a neuromoprhic solution.

Here, we propose ArmNet, which uses deep-Q learning (discussed in section 3.3) to solve the problem of how to properly position the arm. ArmNet learns a policy (π) that predicts the best action to take in each state. The state in this case is the image as seen from the gripper camera of the arm, and the actions that can be taken represent moving the arm in four different orientations (up, down, left, right). Once the arm has reached a new state this process repeats until either the image has been centered or a maximum number of steps have been taken.

One thing to note is that as the arm moves, the image will not change significantly from what the gripper cam saw at the last time step. Traditional approaches recompute all information from scratch, which is computationally and power intensive. Our approach uses sigma delta networks, which focuses on *changes* between subsequent images, reducing power and computation under the assumption that these changes from image to image are small. Because we capture new images fast enough to keep up with movement, the changes between images are small enough to process at each iteration. Sigma-delta networks increase the sparsity of the network by avoiding repeated computations by processing differences in the images (delta) instead of the entire image. To incorporate all information, we sum up the results (sigma) at the end of the sequence. We provide more details on sigma-delta networks and our proposed network in Sections 3.1 and 3.2.

We also present a novel way of training sigma delta networks. Previous work with sigma delta networks learned on image sequences collected in-advance using a supervising learning paradigm (e.g., temporal MNIST [6]). Unfortunately, supervised learning is not always the best approach in robotics because it is impractical

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

to collect large numbers of arm position and movements for every training sequence. We will instead use deep reinforcement learning using deep Q networks to train the sigma delta network to properly position the arm. We provide more details in Section 3.3.

2 RELATED WORK

Deep reinforcement learning has been the subject of an extensive amount of research over the last decade. Some background material can be found in [1]. In this section, we review some of the papers that are most closely related to the proposed work.

Levine *et al.* [4] demonstrated the benefit from simultaneously learning to process the image and control policies simultaneously. Here, the authors use reinforcement learning to complete a number of tasks on a robot including screwing in a top onto a bottle, sorting shapes, and other tasks. Their policy is capable of controlling the joints directly. There have been recent extensions to this work by [8], who use a different network style (vision transformer) and demonstrated some benefits from pre-training. Some have also proposed adding some ground truth sequences [2] to further improve performance.

A limitation of these approaches is that although they have been successful, they are generally only tested with very simple scenes. We extend this in two ways. First, we evaluate on a more complex image that is representative of what a robot will see on a ship. Second, we also explore whether we can use a high resolution image to train a deep-Q network.

Our approach is most similar to the work by James *et al.* [3], who use deep-Q learning to train a robot arm to move in simulation. We build on this by showing how sigma-delta networks can increase power efficiency.

The primary contribution of this paper is to suggest ways that sigma delta networks can increase both the sparsity and the power efficiency of the deep-Q trained network. These properties were initially shown in the work by O'Connor and Welling [6]. In this work they demonstrated how they can use sigma-delta networks to decrease the power usage by an order of magnitude.

3 METHODOLOGY

We provide the details of ArmNet in this section. In Section 3.1 we discuss surrogate gradient descent and methods to train spiking networks using SLAYER. We provide details on sigma delta networks and how that is applied to this problem in Section 3.2. Finally, we provide details on deep-Q learning on how we train in Section 3.3.

3.1 ArmNet and SLAYER

We train ArmNet using Spike Layer Error Reassignment in Time (SLAYER). Rather than relying on ANN-SNN conversion, SLAYER instead introduces a temporal credit assignment technique that enables spiking networks to be trained directly. Very high accuracy can be achieved with ANNs, but they lack the efficiency that can be achieved with an SNN. With this approach, the authors proposed a method that can use a standard error backpropagation algorithm in combination with a temporal error assignment to achieve good performance on SNNs.

The ArmNet network is similar to the network proposed by [3], and is composed 3 convolutional layers (channel, stride): (c=24,s=2; c=36,s=2; c=36, s=1) and three dense layers (1764, 100, 4 neurons). The convolutional layers use a ReLU activation function, the dense layer suse a TanH activation function.

The first layer is preceded by a delta layer, the last neuron has a sigma layer after, we discuss how these layers operate in the next section.

The input to the network is a patch from the high resolution image that is resized to a smaller size (40×40) and then normalized.

We configure the network for our problem in the following ways. First, we use a spiking threshold of 0.5 in the network, with a delta unit surrogate gradient scale parameter of 1.5, which was done to reduce the loss of gradient that we observed in some cases. The spiking threshold also represents a trade-off between sparsity and loss of gradient.

We train ArmNet for 3000 epochs using Adam and a LR of 0.01. We found weight decay to be a contributory factor to loss of gradient so we set weight decay to 0.

3.2 Sigma Delta Networks

During a typical control sequence of moving the robot's arm, we expect that the images will change only very slowly over time as the arm moves from one position to the next. The background of the panel will remain largely unchanged with the primary differences being in the location of the panel components.

One implication of this from a power efficiency perspective is that this results in a lot of (unnecessarily) repeated computations. Rather than re-processing similar / identical images at each time step, sigma delta networks address this problem using a more logical step of only examining differences in each timestep (t), (i.e., Δ_T). At the end of a sequence, to incorporate all of the changes we sum up the output at each timestep (i.e., Σ_T). It was shown in [6] that sigma delta networks can be structured so that they are functionally equivalent to their non sigma delta equivalents. Sigma delta networks have the advantage of increasing power efficiency. For some applications, such as our mobile robot, this efficiency is very important.

To define sigma delta network layers formally, we define temporal difference in 1 and temporal integration in 2. Where $\vec{x}(0) = 0$ and $\vec{y}(0) = 0$

$$\Delta_T(\vec{x}(t))) = \vec{x}(t) - \vec{x}(t-1) \tag{1}$$

$$\Sigma_T(\vec{y}(t))) = \vec{y}(t-1) + \vec{x}(t) \tag{2}$$

A sigma delta network modifies a typical network by adding a delta encoder onto the beginning of the network and a sigma decoder onto the end of the network described in Section 3.1.

3.3 Deep-Q Learning

The next and probably the most important step is how we train ArmNet. To our knowledge, all of the prior work with sigma delta networks use supervised learning. With supervised learning, a sequence of images associated with an output label is collected and used to train the network [6]. The delta encoder is before the first layer of the spiking network, the sigma decoder sums up all of the

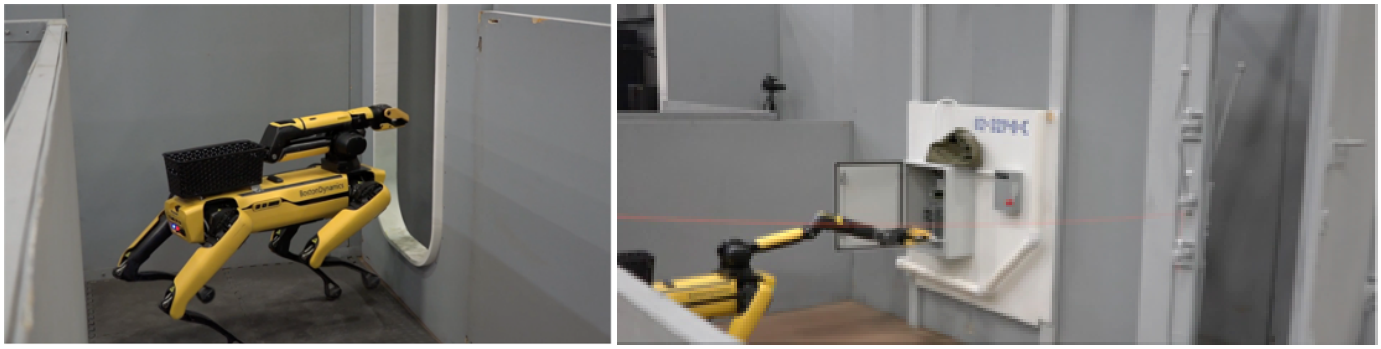

Figure 1: Spot Mini robot "bight" walking through a simulated ship corridor (left) and accessing a mocked-up electrical panel (right).

responses from all of the previous layers, incorporating all available information before making a prediction.

For Bight, supervised learning for this task is challenging [3, 4]. While it may be possible to provide several training sequences, it's difficult to anticipate all possible issues a robot may face (e.g., different viewing angles, different distances). Further it's impractical to collect all such data, making supervised learning approaches to training this type of network unwise.

For these reasons, we propose to use a Deep-Q network (DQN) to train ArmNet. Here we present a high level overview, and direct the interested reader to François-Lavet, *et al.* [1] for a more thorough discussion.

DQNs are based on Q learning, which estimates the expected reward that can be achieved by performing a certain action a in a given state s, that is the network predicts $Q^*(s, a)$, which predicts the Q value for this state. Given Q^*, we can then build a policy π to predict the action that is associated with the highest anticipated reward. We show this in Equation 3.

$$\pi^*(s) = \arg\max_a Q^*(s, a) \tag{3}$$

DQN is based on the premise that it may be difficult to summarize a "state" when it is actually just an image. So, instead, we train a deep network to simultaneously learns to associate the image with the expected utility of each action.

To train our DQN, we use the reward from the given state r combined with the reward from the previous states $Q'(s', a')$ combined with a discount factor γ. This optimal action obeys the Bellman equation [5]. We show this in Equation 4.

$$Q^\pi(s, a) = r + \gamma Q'(s', a') \tag{4}$$

To train a DQN network, we use the loss that's the difference between the expected loss and the actual loss. For efficiency, we do not learn Q^π at each time step, but rather we perform this only periodically (every 10 steps). For stability, we also do not update the policy network directly, but instead update a copy of the network. The copy of the network replaces the policy network periodically (every 50 steps). We also keep a replay memory to store previous episodes, which can help the network remember ways that it failed or succeeded in previous trials.

Figure 2: High resolution image of the electrical panel used during training.

3.4 Training Episodes

DQN learns by playing a "game" of moving the arm. Each episode starts from some point ($p(t)$) in the image (sampled at regular intervals). We crop a small region ($d_w \times d_h$) around $p(t)$ and present this to ArmNet, which makes a prediction about how the arm should move to center this on the middle of the electrical panel, $p(t+1) = p(t) + pred(t)$. The prediction chooses to move the arm up, down, left or right.

Once the center point has been updated, the new image is appended to the end of the sequence and the process repeats.

The episode concludes when a number of time steps has been exceeded ($t > \theta_{time}$), when the network prediction results in a p that is off of the screen, or when the network has found the target ($\hat{p}$), that is, $||\hat{p} - p|| < \theta_{dist}$.

The network is given a reward depending on how it performs. In this "game", a reward of +1 is given when the network moves in the correct direction and a reward of -1 is given when the network moves in the wrong direction. After the episode has completed, a final reward is given that is proportional to the distance to the target $||\hat{p} - p||$. During training, we use the thermostat at the top of the electrical panel as the location that we would like to move the arm towards.

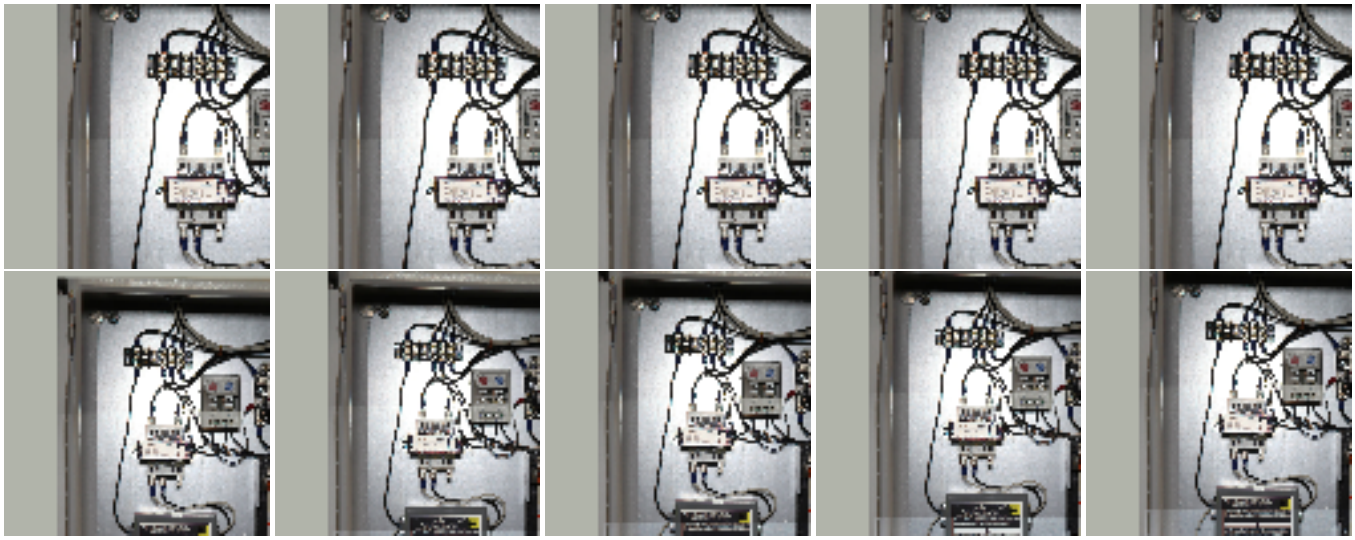

Figure 3: This Figure shows the images being provided to ArmNet for prediction. The top row shows the smaller scenario, whereas the bottom row shows the larger scenario. Both are resized to 40×40 before being presented the network. However, by cropping a smaller region (top) we provide more details on the electrical panel components. By cropping a larger region (bottom) the network can see more of the electrical panel.

When moving the arm, it is also not desirable to move the arm to the same position p repeatedly. This is both wasteful in terms of the amount of time it takes to center the arm as well as the energy used. However, we found that the training episodes could learn more efficiently if they were permitted to make this type of mistake. Over time, the network learns not to repeatedly visit the same points, and this behavior is no longer observed.

Finally, supervised learning approaches have a fixed length sequence that is used by the sigma delta network to make predictions. As we are working in reinforcement learning, it is difficult to predict the sequence length in advance. During training, we fix the length of the sequence to 5 consecutive images. We discuss the implications of variable length sequences in our experimental results.

4 EXPERIMENTAL RESULTS

To evaluate, we used image stitching to create a high resolution image of the electrical panel (see Figure 2). The high resolution image is 6798×5399, which roughly equates to about 100 pixels per cm. We add a padding around the outside of 2000 pixels in each direction, for a total image resolution of 10798×9399 This resolution permits the simulation of what the electrical panel may look like at various distances.

4.1 Results in Simulation

We have designed and run two different experiments in simulation to experimentally validate our approach.

In the first, we use simulated close range images to view an extreme case where the arm may be close to the electrical panel, but we still wish to center in order to properly access off of the components. The image in this case sees a region that is approximately 20cm x 20cm which represents a distance of about 25cm from the sensor to the electrical panel. This represents a challenging problem in that the robot can only see a very small part of the electrical panel, for examples see Figures 3. It is the goal of the robot to move the arm so that it is positioned on top of the thermostat.

To evaluate, first we train ArmNet to predict the movement when viewing the panel from this distance. During training, we sample at a fixed interval in the electrical panel to ensure that we have equal coverage of the entire image.

We evaluate using the same scenario, again to ensure that we are capable of evaluating this from all possible angles. This results in our approach stopping with an error of 4.45 cm from the center of the electrical panel. Interestingly, the network performs fairly well, correctly predicting the movements to view the thermostat in 3900 of the 4992 examples, for an accuracy of 78.2%. As can be expected, many of the misclassifications occurred off on the edges where the network could not see any of the actual electrical panel components. This represents a trade-off in this case where ArmNet can see more details on the component, but since it sees less of the panel at once it can perform poorly in some cases.

In the second experiment, the robot can view the electrical panel at a greater distance of 40cm x 40cm, which represents a distance of about 50cm. Again, we train ArmNet for this scenario. We sample at a fixed interval during training and experimentally evaluate this in a similar manner. In this case we typically stop at a distance that is on average 1.6cm from the center of the electrical panel.

We visually show the result for the second experiment in Figure 4, where a blue circle with a white border is drawn around the region where ArmNet predicts the center to be located. The visual results show all of the results, but many of the results are centered on the thermostat. 3654 of the 3672 trials were either directly on the thermostat (i.e., the target) or very near. A total of 99.5% of the sessions were correct, whereas many of the misclassified examples were still relatively close to the thermostat, but terminated before

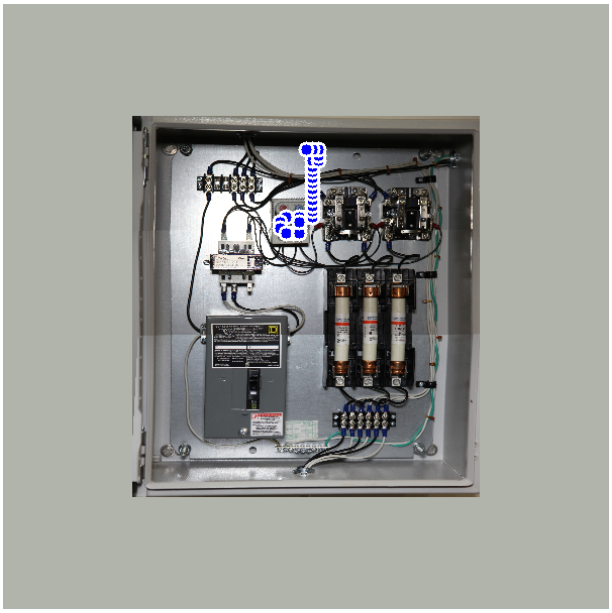

Figure 4: High resolution image of the electrical panel used during training.

it was correctly located. In this case, we can see that although it cannot see much detail on the component, by seeing more of the electrical panel at once it has an easier time navigating.

4.1.1 Variable Sequence Length. The previous two examples use the fixed sequence length of 5, which is used as it was discussed in our methodology section. As an additional evaluate, we explore variable sequence lengths, where we use the sigma delta as it might be used on a robot. That is, we build a sequence one image at a time. Initially, the sequence will only be composed of a single image, and as it grows over time it may grow up to some maximum threshold.

With this setup, we repeated the previous experiment, with the larger network. On average, the error was slightly larger, at 1.75 cm, and 99.2% of the sequences centered right on the thermostat.

Some decrease in accuracy is seen, but this comes with the benefit of fully utilizing the power of sigma delta networks.

4.2 Results on Robot

Finally, we put ArmNet on Bight and test it using the actual configuration. To evaluate, we position the arm at a number of starting positions, approximately corresponding to the edges of the electrical panel with the goal of moving the arm to the thermostat (which was the same goal as we used in simulation). Unlike our simulated result, we could not move this to the far left part of the electrical panel because the door makes it difficult to move the arm.

For safety and efficiency purposes, we set the maximum number of steps to 10. During testing, we found the arm to have a mean distance of 3.4 cm from the thermostat over 8 different starting locations.

It is interesting to note that the distance that the robot used to move from the starting position to the final position was not necessarily a straight line distance. In the future, it's possible that the robot could be encouraged to find a better distance by providing a greater reward to those policies that solved the problem faster.

One of the biggest differences between the robot and the training data is image acquisition on real hardware. Motion blur can occur when images are captured from the gripper camera before the arm has finished moving. In practice, we found this to not be a significant issue, likely because of how much the image was downsized before processing. Motion blur could be lessened by waiting for the arm to finish moving, improving the frame rate of the camera, or even incorporating some motion blur into the training process (i.e., with data augmentation). White balance can also affected the performance due to the reflection from the background of the electrical panel. While this is a harder issue to solve in simulation, it may be possible to change the contrast of the image in training to lessen the impact of white balance.

Figure 5 shows one of the trial runs where the robot moves from a particular starting location to the thermostat.

5 DISCUSSION

In this paper, we showed our approach for using reinforcement learning to train a sigma delta spiking neural network trained in Lava-DL to perform a basic control task on a robot. Our approach is easy to train in that it requires a high resolution image of the target of interest with an annotated keypoint of interest. Our approach is power efficient and takes advantage of recent advances in neuromorphic computing and sigma-delta networks. While we have shown this on a problem of relevance to shipboard maintenance, we believe this can generalize to a wide variety of similar types of situations. Our work is currently in simulation, but we plan to evaluate this with on-chip performance in the future.

Most work with sigma delta networks to date have been with supervised learning, where the entire sequence of images is available at all times (e.g., temporal MNIST from[6]). We believe that for control tasks and deep Q learning this can present a slightly more sensible way to train these networks. Rather than focusing on collecting long sequences of ground truth, instead we can have the network learn on it's own.

As for future work, one extension we would like to consider is a wider evaluation outside of the Lava framework. The basic concepts behind the sigma-delta function are not specific to Lava, and can be applied to other situations. In future work, we would like to consider wider range of chips and possibly exploring other training approaches such as the EONS learning algorithm in the TennLab emulator [7]. We also plan to apply this problem to a wider range of problems

Finally, we also plan to release ArmNet in order to make this available to others using the Spot Mini robot.

ACKNOWLEDGMENTS

Thanks to ONR for sponsoring the work described in this paper

REFERENCES

[1] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. 2018. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning* 11, 3-4 (2018), 219–354.

[2] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Leibo, and Audrunas Gruslys. 2018. Deep Q-learning From

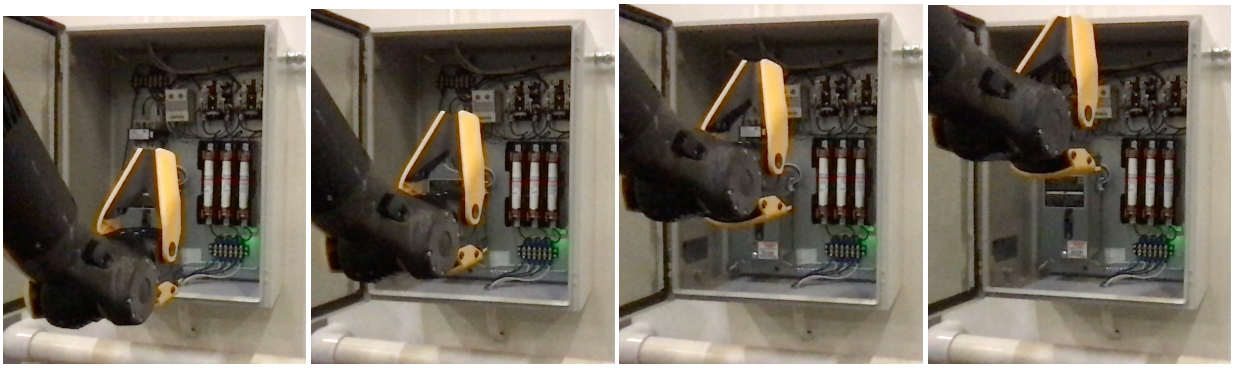

Figure 5: Images of the Bight moving from the starting position (far left) to the final position (far right), with intermediate steps.

Demonstrations. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://doi.org/10.1609/aaai.v32i1.11757

[3] Stephen James and Edward Johns. 2016. 3D Simulation for Robot Arm Control with Deep Q-Learning. *ArXiv* abs/1609.03759 (2016).

[4] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17, 1 (2016), 1334–1373.

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

[6] Peter O'Connor and Max Welling. 2017. Sigma Delta Quantized Networks. In *International Conference on Learning Representations*.

[7] James S Plank, Catherine D Schuman, Grant Bruer, Mark E Dean, and Garrett S Rose. 2018. The TENNLab exploratory neuromorphic computing framework. *IEEE Letters of the Computer Society* 1, 2 (2018), 17–20.

[8] Ilija Radosavovic, Tete Xiao, Stephen James, Pieter Abbeel, Jitendra Malik, and Trevor Darrell. [n. d.]. Real-World Robot Learning with Masked Visual Pre-training. In *6th Annual Conference on Robot Learning*.