Curious Actor-Critic Reinforcement Learning With the Dynamixel-bot

Nadia M. Ady
Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada
nmady@ualberta.ca

Abstract

Curiosity is a crucial, but not yet well-understood component of intelligence and a better understanding of existing models may lead to a better understanding of curiosity as a whole. In this work, we present a physical robot implementation of the basic curiosity loop introduced by Gordon and Ahissar in 2012. In the same way that Gordon and Ahissar produced a rough simulation of a rat's whiskers, this work presents a physical model using two servos to create the whisking actions.

1 Introduction

The term *computational curiosity* is used to refer to computational methods attempting to create curiosity, a "desire to know", in machines. Curiosity has the potential to benefit many systems. For example, in any setting where we want a system to continue training long-term with a large or changing environment, curiosity is valuable for developing (hopefully useful) knowledge of that environment.

2 Conceptual Goals

This project extends actor-critic reinforcement learning in a frontier direction: implementation of computationally curious control. In particular, I have implemented an adaptation of Gordon and Ahissar's [Gordon and Ahissar, 2012b] basic curiosity loop architecture on the Dynamixel-bot [Ady, 2017c].

Over 2011 and 2012, Gordon and Ahissar presented their simulation of a rat's whiskers, movements which they conjecture are mainly controlled based on curiosity [Gordon and Ahissar, 2011; 2012b]. Their method for control used three parts: a *learner* to develop knowledge of the environment, a *critic* to develop knowledge of the state's value for curiosity, and an *actor* to parameterize a policy based on the critic's knowledge. This structure is suggested in figure 1.

I am conceptually interested in implementing an adaptation of the curiosity loop supported nearly exclusively with RL architectures. While Gordon and Ahissar used RL architectures for their actor and critic components, they implemented their learner component using a neural network with multiple layers. They noted that the running time for their system was long, attributing the slowness to averaging multiple trials of the system. However, with a simpler architecture using a general value functions to fulfill the role of supervised learner, I hope we might be able to improve on the original efficiency.

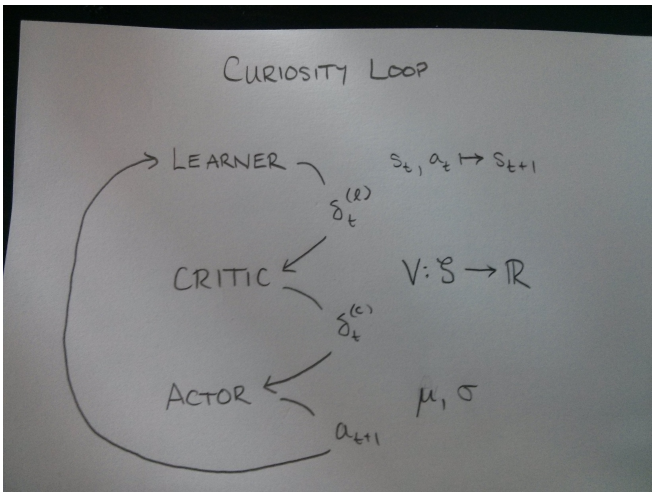

Figure 1: This figure shows the structure of the basic curiosity loop and the relationship between its components.

2.1 The Original Implementation

For more detail on Gordon and Ahissar's original architecture, we can look at their original choices for each component of the loop.

The basic loop includes at least one learner component, but could include multiple learner components making different predictions. The learner component I have chosen to focus on and implement is the one they refer to as the *forward model* (FM). Given a state and an action, the forward model learner component is learning what next state it would end up in if it took the given action from the given state ($s_t, a_t \mapsto s_{t+1}$). To implement this kind of predictive model, they used "a feed-forward neural network with two input neurons, two hidden layers and one output neuron, with linear, hyperbolic tangent and symmetric saturated linear transfer functions, respectively" [Gordon and Ahissar, 2012b, p. 122]).

The important part of the learner's computation that is passed on to the critic component is its prediction error. In

the original implementation, this prediction error is squared (and summed if there are multiple learner components), then used as the reward for the actor-critic component.

The actor and critic components are implemented using Incremental Natural Actor-Critic [Gordon and Ahissar, 2012b, p. 121], as described by Bhatnagar et al. [Bhatnagar *et al.*, 2009]. For their function approximators, Gordon and Ahissar used radial-basis-functions (RBFs) [Gordon and Ahissar, 2012b, p. 121].

3 Technical Goals

Prior to beginning this project, I already had a suitable physical platform as a starting point: my Dynamixel-bot, as described in [Ady, 2017a]. In terms of software, the useful components of the software developed for Robot Modules 1 and 2 include those which which allow me to observe the load, velocity, and position and to use Kanerva coding for function approximation and a horde architecture [Ady, 2017a; 2017b]. The Horde architecture allows off-policy and on-policy predictions about different values, using either constant or state-conditional continuation probabilities.

The first chunk of this project was composed of completing Robot Module 4. In Robot Module 4, I implemented the necessary learning and control algorithms for continuous-action actor-critic, which formed the basis of two of the three components of the curiosity loop. The main implementation details are available in the report [Ady, 2017c].

This section focuses on the details of how I was able to use existing RL architectures to form a functioning basic curiosity loop.

3.1 Positions and Actions

The action space, $\mathcal{A}$, was set to the set of feasible angular velocities. This was of course limited by the robot, which had speeds representable by numbers between 0 and 2047. Essentially, these were really numbers between 0 and 1023 which represented '0.111 rpm's with an additional sign bit for direction. When I needed to represent the velocity dimension (in sction 3.2) I only considered the range

$$\left[\frac{-1023 \cdot 0.111 \cdot 2\pi}{60}, \frac{1023 \cdot 0.111 \cdot 2\pi}{60}\right]$$

I limited the position space, by placing soft safety limits around the range $[550, 810]$, to only utilize a range making reasonable whisker motions. When the Dynamixel takes an action, if the current position reading θ_t is outside the safe range and setting its velocity to the value of the action would send it further outside the safe range, the action instead sets the velocity to a small value (± 0.0115) in the opposite direction, so the robot cannot get stuck outside its safe range. Otherwise (either its position reading is safe or its action takes it from outside the safe range to inside) the velocity is set to the value of its action. Therefore, the set of positions that could possibly be observed is slightly larger than $[550, 810]$.

3.2 Learner Component

Our adaptation of the learner component is implemented with an on-policy general value function (GVF) using TD(λ).

In both [Ady, 2017b] and [Ady, 2017c], we used a single observation space for each work. In the earlier work, [Ady, 2017b], we used a complex observation space taking into account the position, velocity, and load, while in the latter, [Ady, 2017c], we simplified the observation space to only include position. However, in this latter work, we also hinted that a general value function might be adapted to map not states to values, but state-action pairs to values and used such an adaptation for our discrete-action actor-critic learner.

For the learner component's observation space, we transformed the current observed angular position (subtracting 550 and dividing by 810-550) and paired the transformed value with the action (similar to the the observation space for the actor in discrete-action actor critic RL). The expected upper and lower bounds for that dimension are therefore -0.25 and 1.25 (accounting for the occasions when the robot exceeds its safety limits.

$$\mathcal{O} = [-0.25, 1.25] \times \mathcal{A}$$

We used Kanerva coding as our function approximator, similar to what we used for Robot Modules 2 and 4 [Ady, 2017b; 2017c]. However, in this case, instead of using Hamming distance, we rescaled each dimension to approximately $[0, 1]$, and took the Euclidean distance. In the case of any reading from the obersation space or prototype for the Kanerva coding, (θ, ω), this meant performing the following transformation before computing the Euclidean distance:

$$\frac{(\theta, \omega) - \left(-0.25, \frac{-0.111 \cdot 1023 \cdot 2\pi}{60}\right)}{\left(1.25, \frac{0.111 \cdot 1023 \cdot 2\pi}{60}\right) - \left(-0.25, \frac{-0.111 \cdot 1023 \cdot 2\pi}{60}\right)} \tag{1}$$

We used TD(λ) exactly as described and implemented for Robot Module 2 [Ady, 2017b]. For the cumulant Z, we used the value of the state. We used a continuation probability γ of zero to only consider the next step. In short form, our question is formed by:

$$\pi = \mu, \text{ our behaviour policy,} \tag{2}$$

$$Z(s_t) = \theta_t \tag{3}$$

$$\gamma = 0 \tag{4}$$

3.3 Actor and Critic Components

In my adaptation, I used nearly exactly the implementation of continuous actor-critic as implemented and described for Robot Module 4 [Ady, 2017c]. The only difference is that instead of using the angular position in encoder ticks for the observation space, in this work I used the transformed position as described in section 3.2, so the observation space represented is $\mathcal{O} = [-0.25, 1.25]$. The parameter settings, however, are all identical to those in the previous work.

By using the continuous-action actor critic implementation from Robot Module 4, I am using an adaptation quite different from Gordon and Ahissar's implementation. Where they used radial basis functions to describe the parameterization of their policy, the Dynamixel-bot simply uses a normal distribution. At the beginning of learning, they say that the policy starts "from a random-like behavior, i.e. their parameters are set such that a random action will be produced" [Gordon and Ahissar, 2012b, p. 123]. To simplify, the Dynamixel-bot's

starting parameters were set so that the normal distribution had mean $\mu = 0$ and standard deviation $\sigma = 1$.

However, the reward observed by the actor-critic is not a goal position. It is instead computed as follows:

$$r_t = \left(\delta_t^{(\ell)}\right)^2$$

3.4 Rat-like Physicality

As a finishing touch on the existing Dynamixel-bot configuration, I added a paper face with ears and whiskers to provide a rat-like semblance to the Dynamixel-bot. This addtion is shown in Figure 5. This addition should not have significant effect on the function the robot, since it was designed to allow free motion of the hinges acting as whiskers.

4 Learning Process

In this section, I would like to discuss my observations as the Dynamixel-bot learns over time and provide commentary on the behaviour.

First of all, the GVF component of the learner appears to work appropriately. In early learning, there is high TD-error, and it takes little time for the learner's predictions to follow the same approximate trajectory (one step early) as its positions. The learning slows rapidly, and there is consistently remaining error.

However, it also becomes clear quite immediately that this setup has its problems. The first clear issue is the development of a clear upward bias in the policy. The parameters for both the critic's estimated value of each state and the learner's predictions result in both being equal to the zero function. Therefore, transformed position readings around 0 result in little error, while transformed position readings around 1 produce relatively more TD-error and quickly increase the estimated value of the higher positions. This results in the policy developing a bias towards positive velocities, rather than negative ones.

The value of higher positions decreases very slowly over time. Given the speed at which the learner's error decreases, this may be reasonable, but I worry that the error may never get small enough to remove this bias, given the noise in the physical system and the nature of approximation. A good solution to this might be utilizing a different reward measure, like decrease in error [Oudeyer *et al.*, 2007].

For the purposes of this initial project, I was unable to run the Dynamixel-bot long enough to decide if the described bias ever disappears or if the state-value estimations every become more reasonable. For around 30,000 timesteps, the standard deviation remained around 1. However, as the number of timesteps approached 40,000, the standard deviation increased to around 2. In terms of physical observations, this lead to faster, wider motions as time went on.

To get a sense of the first 50,000 timesteps, consider figure 3. We can see the bias in the average of the mean, and how the bias decreases over time as the variation increases. The standard deviation generally increases, but its variation also increases dramatically. This figure also allows us to see the spiky profile of the reward, as large errors are encountered once, but cause immediate improvement in the learner component's predictions.

Lastly, there may be a problem with simply using position as the observation for the actor-critic. While in some cases it is useful to generalize about all actions from a particular state, the action has a critical impact on the amount of reward achieved, not just the next state. Though this is not abnormal for a reinforcement learning problem (consider bandits) I wondered what would happen if all three components of the system used both position and velocity as their observation space.

5 Measures of Success

It is challenging to use data to prove success, since 'successful curiosity' is poorly defined and the behaviour may not mimic the empirical results provided by Gordon and Ahissar because my methods are based on theirs, but substantially adapted.

My success in this project is best demonstrated partially by the previous section, which shows both that the robot appears to be learning as expected based on its implementation and a basic understanding of the learning process. In this section, I will discuss future work, which I was only able to determine through developing an understanding of Gordon and Ahissar's idea via its implementation. Both understanding the behaviour of my current system and understanding how to move forward to fulfill my goals for the future of this project.

The performance of the existing implementation could probably be substantially improved by tuning the parameters. A 'faster' critic, which moves the state-value estimations more quickly, might reduce the bias in the policy more quickly. While I could also shift the bias created by the starting position, I believe that the policy would still develop bias towards positions farthest from the learner's initial "guesses". Similarly, optimistic initialization of the critic might change the bias.

However, I think that rather than spending time on tuning parameters, I would prefer to tune the system design. In future work, it would be useful to implement the RBF policy parameterization to compare how that impacts the behaviour and learning of the agent. However, I would first like to replace the standard normal distribution continuous-action actor-critic with Shariff and Dick's truncated normal distribution [Shariff and Dick, 2013]. I believe that a truncated normal distribution would make it easier to aquire the desired almost uniformly random behaviour at the beginning of learning, and make it safer to set the parameters to make the actor's initial variance higher.

For better comparison, it would also be useful to plot a recreation of Figures 1(b) and (d) from Gordon and Ahissar's paper [Gordon and Ahissar, 2012b]. These figures are reprinted in Figure 4. Given that my adaptation considers the problem as a continuing task while they have made their episodic, I may need to adapt their figure 1(b). The closest comparison might be done after 10,000,000 timesteps (since they do 10.000 episodes of 1000 timesteps). Even prior to such a large run, it would be helpful to see how the learner's

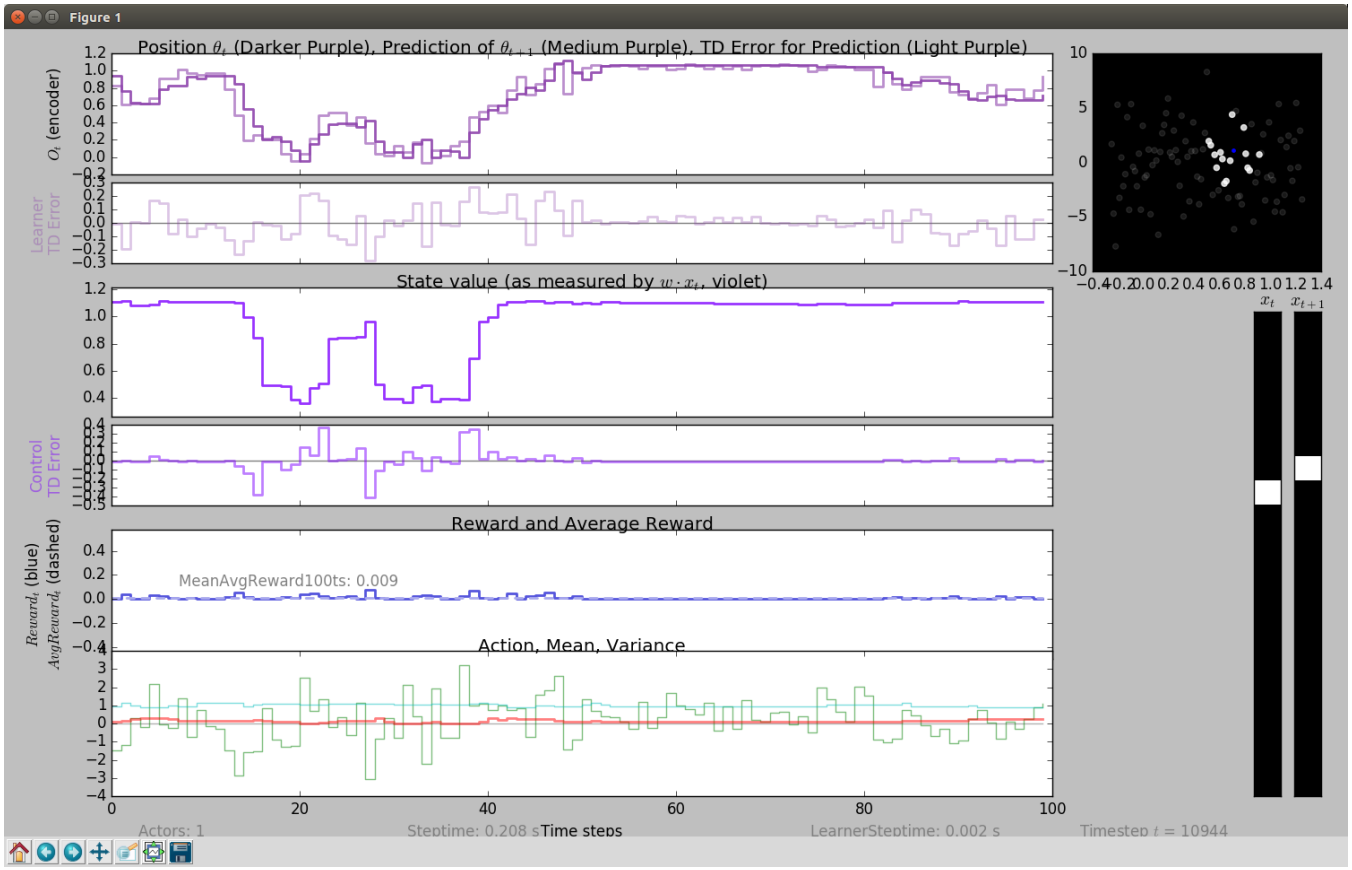

Figure 2: This image depicts the clear bias towards positive velocities. Note that in the bottom plot, the red line (the mean) is typically above zero whenever the position is lower than 1. This bias results in the agent rarely trying negative velocities from positions near 0, and therefore learning those position-action pairs poorly in comparison to other pairs including higher positions. All plots on the left use the same bottom axis showing the most recent 100 timesteps. The vertical axis for the first plot from the top shows tick labels in transformed position ((encoder ticks - 550)/(810-550)
. The bottom plot shows action (a velocity) in green, mean μ for the actor's normally distributed policy in red, and its standard deviation in aqua. The top plot on the right show the Kanerva representation (active features in white, observation in blue) for the learner, with the vertical axis as action and the horizonal axis as transformed position.

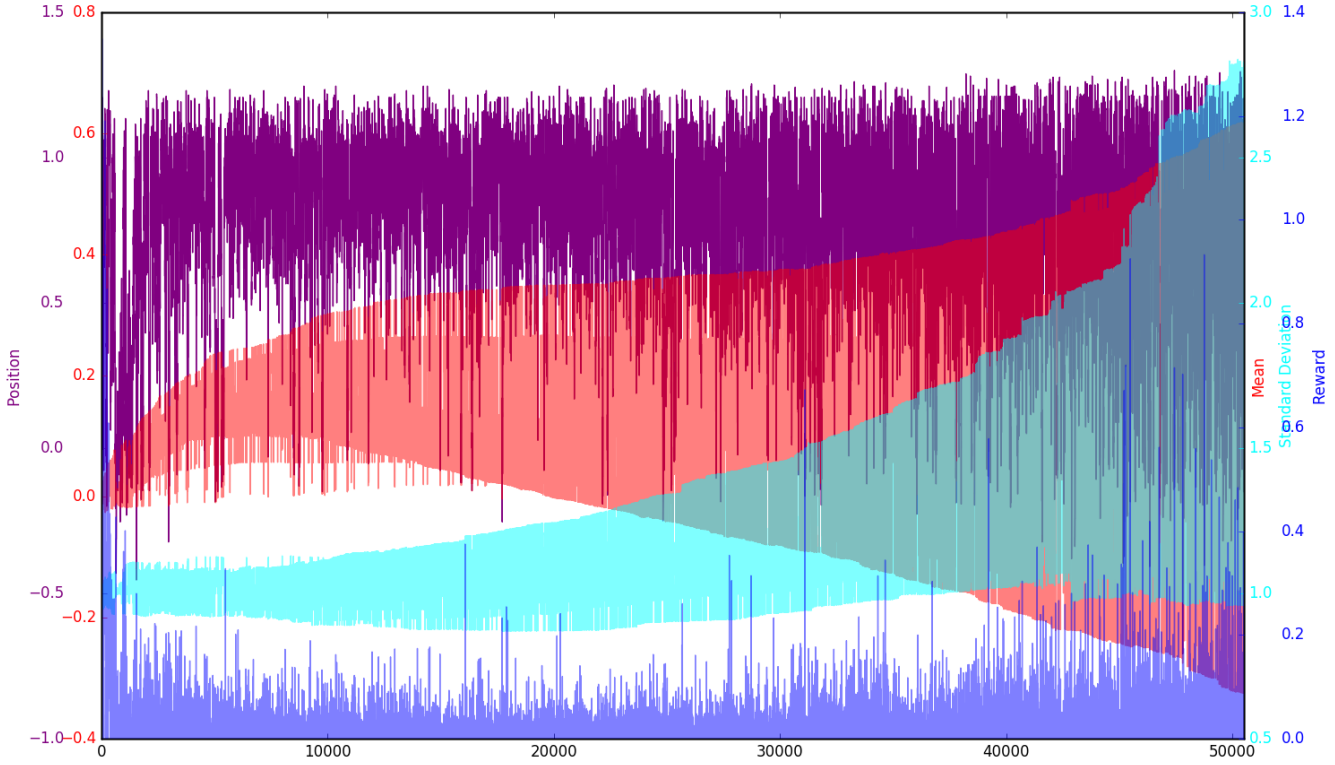

Figure 3: In this plot, we show how position (purple), reward (blue), mean (red), and standard deviation (aqua) vary over the first 50,000 timesteps (note the differing axes).

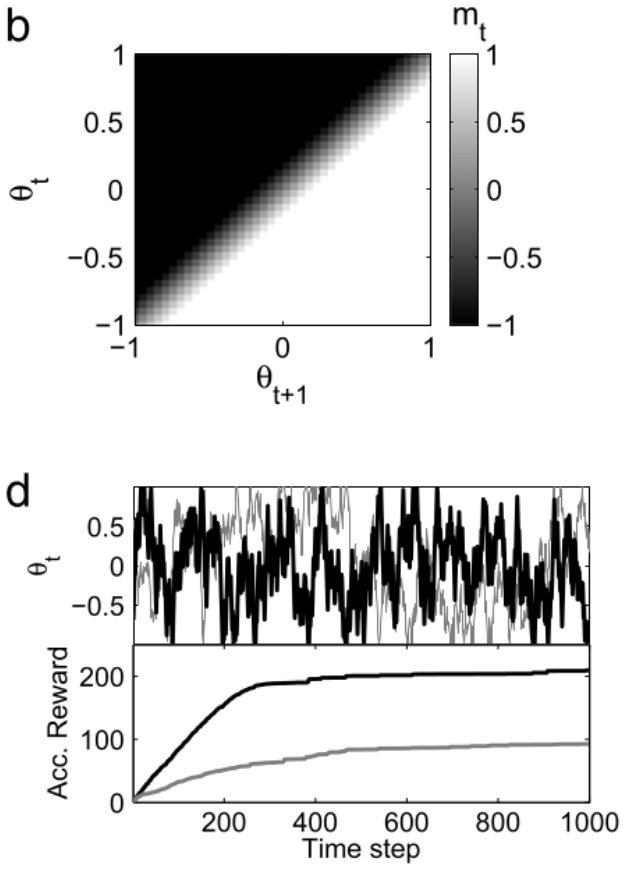

Figure 4: Figures 1(b) and (d), as reprinted from Gordon and Ahiisar 20112 p. 124. For the forward model curiosity loop, (b) shows the forward model mapping, with the position intensity coded. (d) Upper panel: a typical trajectory of the learned (black) and random (gray) actors. Lower panel: accumulated reward of the same trajectory.

predictions for all possible observations improves over time.

After the policy parameterization is improved, the next step should be to implement the second learner, which learns the inverse model $(s_t, s_{t+1} \mapsto a_t)$ to compare how this impacts the process. Gordon and Ahissar consider both individual systems using a single learner (either forward model or inverse model) and a combined system where the sum of the squared error from each is used.

Following from the addition of the inverse model, I would like to continue onto the active-sensing components of Gordon and Ahissar's work. For this component of their work, they added a binary touch signal and predictions of its value. This could likely be mimicked using the load sensor of the Dynamixel-bot.

After including these components suggested in the original work, I would also be interested in expanding their work to include predictions over longer time periods, with non-zero continuation probabilities.

6 Conclusions and Future Work

Studying computational curiosity targets multiple areas of RL. We have high hopes that it will help us make progress in the way we do exploration, but different approaches both pull from and modify aspects of RL from the reward function to representation to the chosen algorithm.

As described in section 5, the next step I would like to take is the implementation of a "truncated normal distribution"-actor. After that, I hope to explore the deeper ideas suggested by Gordon and Ahissar and experiment using predictions that they dis not try.

Thus far, there is only one prior robotic implementation of Gordon and Ahissar's hierarchical curiosity loops [Gordon and Ahissar, 2012a]. That project implemented the curiosity loops on a robot arm, so there has never before been a physical implementation of the whisker simulation from Gordon and Ahissar's original introduction to the concept (see Figure 5). This project has shown that a curiosity loop can be implemented using only RL architectures. In this, this work provides the initial results for an RL-centred extension of actor-critic into the frontier domain of curiosity.

References

[Ady, 2017a] Nadia Ady. Building Nadia's first dynamixel-bot. Technical Report CMPUT 607 Robot Module 1, Department of Computer Science, University of Alberta, Edmonton, Alberta, January 2017.

[Ady, 2017b] Nadia Ady. Forecasting Nadia's dynamixel-bot: First general value functions. Technical Report CMPUT 607 Robot Module 2, Department of Computer Science, University of Alberta, Edmonton, Alberta, February 2017.

[Ady, 2017c] Nadia Ady. Nadia's Dynamixel-bot as actor-critic reinforcement learner. Technical Report CMPUT 607 Robot Module 4, Department of Computer Science, University of Alberta, Edmonton, Alberta, March 2017.

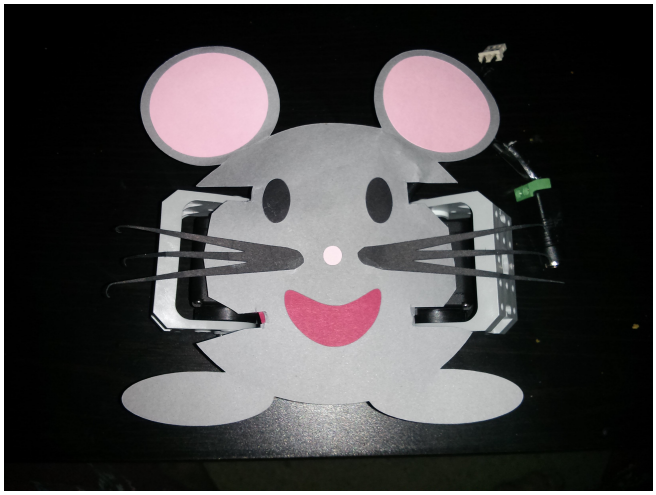

Figure 5: The final physical configuration of the Dynamixelbot.

[Bhatnagar *et al.*, 2009] Shalabh Bhatnagar, Richard Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor-critic algorithms. *Automatica*, 45(11), 2009.

[Gordon and Ahissar, 2011] Goren Gordon and Ehud Ahissar. Reinforcement active learning hierarchical loops. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 3008–3015. IEEE, 2011.

[Gordon and Ahissar, 2012a] Goren Gordon and Ehud Ahissar. A curious emergence of reaching. In *Conference Towards Autonomous Robotic Systems*, pages 1–12. Springer, 2012.

[Gordon and Ahissar, 2012b] Goren Gordon and Ehud Ahissar. Hierarchical curiosity loops and active sensing. *Neural Networks*, 32:119 – 129, 2012. Selected Papers from {IJCNN} 2011.

[Oudeyer *et al.*, 2007] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

[Shariff and Dick, 2013] Roshan Shariff and Travis Dick. Lunar lander: A continuous-action case study for policy-gradient actor-critic algorithms. In *Reinforcement Learning & Decision Making (RLDM) Princeton, NJ, USA*, 2013.