Deep Reinforcement Learning in FEA Driven Virtual Environments for Control of Flexible Robotics

Sebastian Mettes, Steve Simske

Background

High precision, flexible robotic arms can decrease costs and/or improve performance of robotics.

- Space Robotics
  - Decrease launch cost via weight reduction of flexible materials and reduced launch volume due to flexibility of robot arm links.
  - Decrease manufacturing costs by reducing need for exotic materials (plastic and fiberglass vs. carbon fiber and aluminum).
- Underwater Robotics
  - Near neutral buoyancy of plastics such as ABS and PLA.
  - Corrosion resistance of plastics.

Traditional controllers depend on the near instant and predictable response of arm links to input torque.

- Flexible robotic arms respond neither instantly nor predictably.
- Response varies due to
  - Tip load
  - Flexural rigidity of the arm
  - Torque applied
  - Current system state
- More sophisticated controllers are required to control flexible robotics with high precision

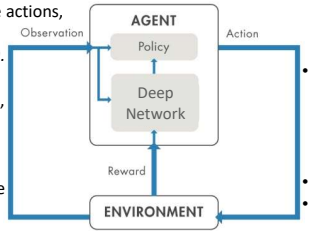
Flexural Rigidity

Flexibility of a robotic link is the product of the material modulus of elasticity (E) and the link's cross section (I), as shown below. In this research, a E value of 2.3E12 results in a highly rigid robotic arm, and 1.65E9 a highly flexible robotic arm.

$$Flexural\ Rigidity = E * I_z$$

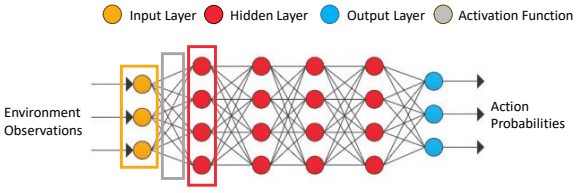
Reinforcement Learning

- Reinforcement learning can be used to construct a model of a dynamic, unknown environment.
- This is accomplished with an *agent* which explores the environment, records the *actions* taken, the *rewards* received for those actions, and the resulting changes to the environment *(observation)*.
- The agent optimizes its action decisions with a deep network, and follows an *action policy,* such as "select the action which will maximize reward" based on the predictions of the deep network.



Deep Learning

- Deep networks are universal function approximators.
- Deep networks consist of multiple *layers* of 3 types:
  - The input layer receives the environment observations.
  - Hidden layers operate on incoming signals (optimized based on a *loss function*). Can vary in *depth* or number of *neurons*. Example below has 4 neurons in each hidden layer.
  - The output layer in this research represents the probability that each corresponding action will lead to a high reward episode.
- Between each layer, an activation function introduces non-linearities which enable the universal function approximation.

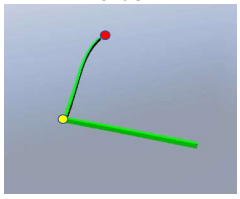


Virtual Environment

- An FEA Environment was built with Python + the Chrono physics engine
- The Environment accepts action decisions (torque applied by motor 1, motor 2) from the agent at every step of the FEA simulation
- Motor 1 (red) fixed at origin, Motor 2 (yellow) at extrema of link 1.

FEA – Rigid Arm

FEA – Flexible Arm



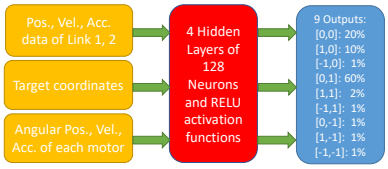
Research Goal: To develop a deep reinforcement learning based controller demonstrating precise control of a virtual 2-D, 2-link, flexible robotic arm.

Specific Aims

- Aim 1: Create FEA driven virtual environment (see Virtual Environment)
- Aim 2: Train a control network for a rigid, 2-link arm with deep reinforcement learning
- Aim 3: Train a control network for a flexible, 2-link arm with transfer learning

Agent – Deep Network & Optimization

- A multi-layer perceptron deep network was selected for this research.



- Each of the 9 outputs represent a probability that a torque combination (ex. [0,1], where 0 Nm is applied to motor 1 and 1 Nm is applied to motor 2) will result in a high reward episode.
- The cross-entropy loss method was implemented for network optimization. In this method, every starting condition is attempted 15 – 100 times (minimum 15, maximum 100 if no positive total reward episode is found), with the highest performing episode of the starting condition retained for optimization. A new agent is optimized after 300 starting conditions are attempted, with the retained episodes considered "high performing".
- Gradient descent is used to optimize the network weights:

$$\omega_i = \omega_i - \gamma \nabla_{\omega_i} H(p,q)$$

$$H(p,q) = -\sum_i p_i \log(q_i)$$

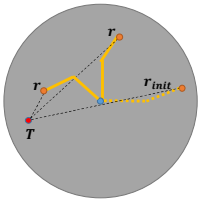
- ω_i = weight of neuron, γ = learning rate
- H = Cross- Entropy prediction, where p is either 0 or 1, where 1 corresponds to actual action taken, and q the predicted probability that the action leads to a "high performing episode".

Environment - Reward

- Reward (R) was calculated by first calculating the change in distance (D) of the *end-effector* (r) to the target (T) after each action and then dividing that value by the initial distance to target, resulting in a maximum reward of 1.0. Double negative rewards were given for poor results (moving away from the target).

$$D(\boldsymbol{r}, \boldsymbol{r}', \boldsymbol{T}) = |\boldsymbol{T} - \boldsymbol{r}| - |\boldsymbol{T} - \boldsymbol{r}'|$$

$$R(D, \boldsymbol{r}_{init}, \boldsymbol{T}) = \begin{cases} \frac{D}{|\boldsymbol{T} - \boldsymbol{r}_{init}|}, & D > 0 \\ \frac{2D}{|\boldsymbol{T} - \boldsymbol{r}_{init}|}, & D < 0 \end{cases}$$



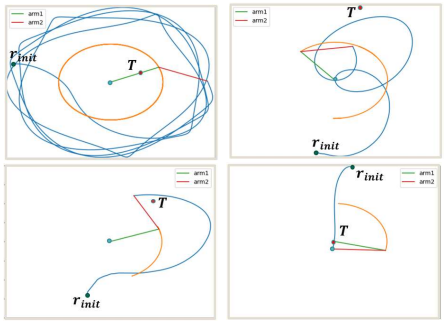
- During training, network performance was improved by introducing a binary "bonus reward" for keeping the end-effector within a small radius of the target.
  - Maximum reward increased to > 3.0
  - Values of 1.0 indicated the controller navigated to the target.
  - Values above 1.0 indicate the controller navigated to the target and remained in its vicinity after arriving.

Results

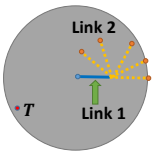
Rigid Arm Controller

Initial Training was conducted without "Bonus Reward" and successfully found solutions for some initial starting conditions. However, by initializing randomly across the entire workspace, not all starting conditions successfully converged on a solution.
The below path plots demonstrate the progression of the paths the agent took to reach the target where the *Agent* number represent the number of optimizations (after each set of 300 episodes) completed.

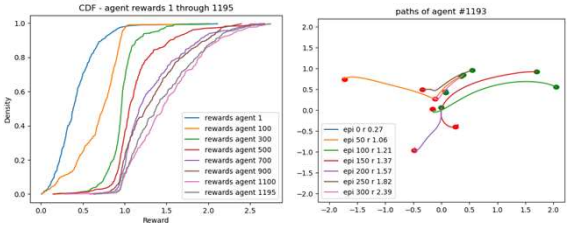
Agent 4 Agent 11



Agent 16 Agent 301

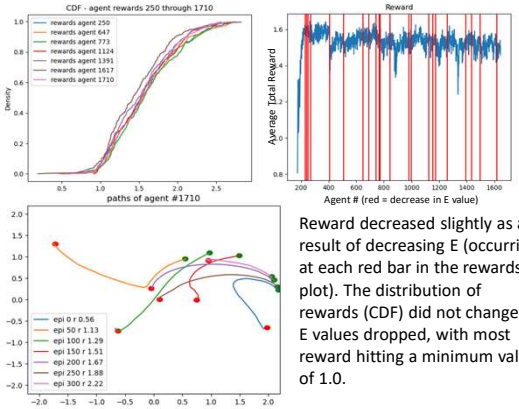


By limiting robot arm starting conditions to the first quadrant, (which represents the entire workspace through geometric transformations), and adding bonus rewards, the network found solutions for the entire workspace.



Flexible Arm Controller

The flexible arm controller began training where the rigid arm controller left off using *transfer learning*. Rather than starting from scratch, the rigid arm controller continued training but with a slow decline in arm modulus of elasticity (E), thus gradually moving from a rigid to flexible arm with an E value of 1.65 E9. This was necessary as training from scratch or jumping immediately to a fully flexible arm converged on a poor performing controller. A flexible arm controller was successfully trained over the course of 1700+ agents.



Reward decreased slightly as a result of decreasing E (occurring at each red bar in the rewards plot). The distribution of rewards (CDF) did not change as E values dropped, with most reward hitting a minimum value of 1.0.

As shown in the final paths plot, nearly all starting conditions successfully reach their targets, and achieved a reward above 1.0, indicated they stopped at or near the target.

Future Research

To further improve these results, investigation into longer simulation time, and a wider range of torque outputs (beyond the -1,0,1 values) may improve the controller's ability to stop exactly at a target and compensate for low frequency vibrations.