Automated Hex Nut Fastening using Reinforcement Learning

Jessica Tingley
Department of Mechanical, Industrial, and Systems Engineering
Department of Computer Science and Statistics
University of Rhode Island
Kingston, RI, 02882 USA
jessicatingley@uri.edu

I. Introduction

A. Background

Threaded fastening is a commonly used joining technique due to its low cost, versatility, and reversibility [1] [2]. Specifically, the threaded stud and hex nut fastening method will be discussed. The general stages of the fastening process are initial mating, rundown, and tightening. With various types of cross-threading and jamming scenarios being possible, initial mating is the most difficult stage to successfully complete [3]. During initial mating, the torques and forces must be closely monitored to prevent damage to the threads, fastener, and fastening device.

Automating the fastening process is highly desirable, however the challenges posed by the initial mating stage present significant hurdles to fastening systems aiming to be fully autonomous. Articulated robots equipped with sensors, such as those used by Hsue et al. [4], Rastegarpanah et al. [5] , and Pettinaro et al. [6], are a common approach to the problem.

B. Current Approach

A classical control approach has been implemented on an Epson VT6L articulated robot with a 97% success rate [7]. The overview of this method can be seen in Figure 1 and is briefly described as follows:

- The robot starts from a home position with a hex nut. A photo from the home position of the studs is taken and run through an object detection model to identify studs without hex nuts.
- The user is prompted to select a stud to approach.
- As the arm approaches the selected stud, two phases of centering using visual methods are completed.
- Threading begins in 5° increments. This allows response to a cross-threading detection to be nearly instantaneous.
- If cross-threading is detected, the nut is unthreaded and centering is conducted again.
- Once 240° has been rotated, the nut must be released and the arm must rotate -240° to avoid damaging external wiring.

C. Objective

Despite the high performance of the original method, designing that specific sequence of movements and feedback took months of tedious testing. If the studs were to be moved to a different orientation (mounted vertically or at an angle), the same process would need to be repeated. The goal of this project was to implement a deep Q-learning (DQN) algorithm to replace the initial mating and rundown in the classic method in an attempt to eliminate the need for precise engineering of the process. In the original method, the state of the system directly corresponds to the input from the environment. This provides a natural way for reinforcement learning to be implemented as the model must learn directly from environmental interactions and queues [8]. Using reinforcement learning over a classic controls approach may provide better responses to changes in the environment and allow for a more robust system [9].

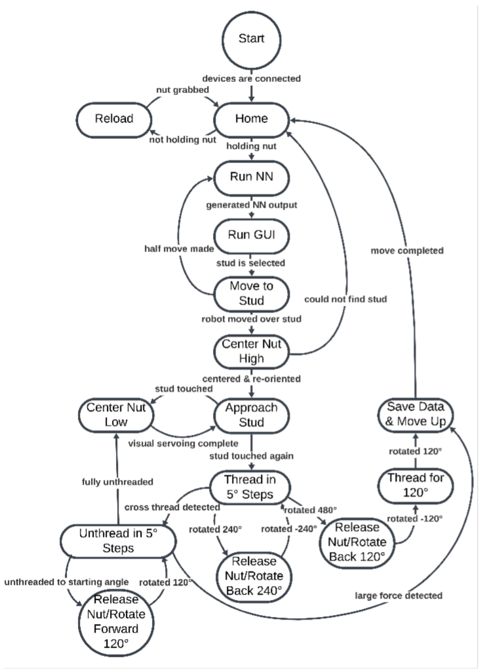

Fig. 1. State diagram of classical method [7].

II. Implementation

The proposed DQN algorithm was implemented using Python with RoboDK used for communication with the robot and PyTorch used for model implementation.

A. Data Collection

Data collection will be conducted using the current robotic setup and the algorithm will be trained on this data in real time. The relevant components of the system are the Epson VT6L articulated robot, a Robotiq Force-Torque sensor, and the Robotiq Hand-E gripper. The force-torque sensor reads the forces and torques in all directions exerted on the end effector of the arm. The state of the system will primarily be deduced from these readings.

B. Algorithm

The DQN approach was first introduced by Mnih et al. [10] as a variant of Q-learning. The DQN architecture diagram can be seen in Figure 2 and consists of two neural networks, the Q-network and the target network, and an experience replay memory which is used to generate training data. The replay memory interacts with the environment and saves the reward and next state as training data. This set of data is then randomly sampled from, breaking correlations between training steps and helping to stabilize the network training [11]. The introduction of a target network also helps to stabilize training. The network is only updated every 'T' steps, which allows it to pose as a stable target. In this case, the target network was updated every 50 steps, making one episode 50 steps. The randomly sampled batch of training data is input to the Q-network to predict a Q-value and into the target network to generate a target Q-value. The target Q-value is input to the Bellman equation, seen below, to generate an expected future reward.

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

In this equation, $Q(s,a)$ is the Q-value for taking action a in state s, $R(s,a)$ is the immediate reward, γ is the discount factor (0.99 used in this instance), and $\max_{a'} Q(s',a')$ is the maximum Q-value for the next state. The loss between the target Q-value and predicted Q-value is calculated and back-propagated the to the Q-network.

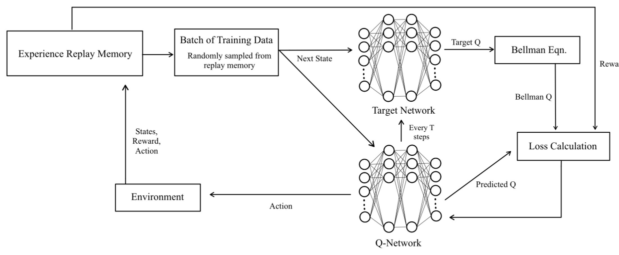

Fig. 2. DQN architecture diagram.

An epsilon-greedy policy was used throughout training. At the beginning of training, exploration was prioritized with an epsilon value of 99%. This value was decayed over 5000 steps to reach a final epsilon value of 5%. The shift from exploration to exploitation allows the system to effectively learn and then gradually begin to apply the experience.

C. Models

The Q and target networks are identical in structure and the architecture can be seen in Figure 3. For both models, the following standard hyperparameters were used:

- Batch Size: 32
- Learning Rate: 0.0001
- Optimizer: Adam
- Loss: MSE
- Activation: ReLU

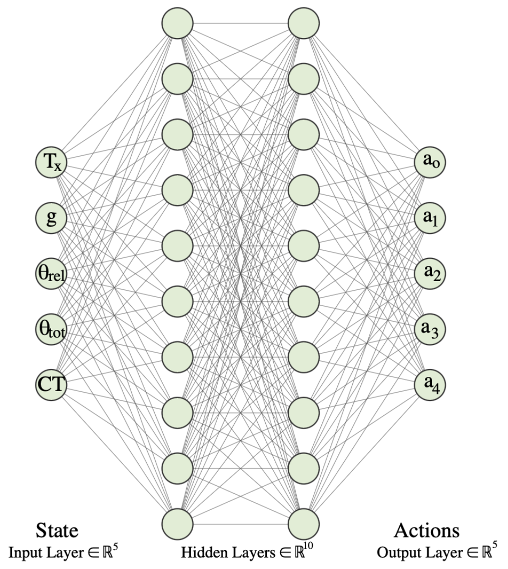
T_x
g
θ_{rel}
θ_{tot}
CT
a_0
a_1
a_2
a_3
a_4
State
Input Layer $\in \mathbb{R}^5$
Hidden Layers $\in \mathbb{R}^{10}$
Actions
Output Layer $\in \mathbb{R}^5$

Fig. 3. Proposed model.

The state input to the network consists of the torque about the z axis (T_z), whether the gripper is open or closed (g), the relative angle traveled (θ_{rel}) to determine whether the nut should be released and the arm angle should be reset, the total angle the nut has traveled (θ_{tot}), and whether a cross-threading event has been detected (CT). These states should provide sufficient information for the agent to respond to various scenarios. The actions a_0 - a_4 are as follows:

- a_0: Rotate +5° – Normal increment for threading.
- a_1: Rotate -5° – If cross-threading is detected, nut should be unthreaded in small increments.
- a_2: Rotate -120° – If 240° has been threaded, the nut must be released and the arm must reset its rotation angle before continuing to thread.
- a_3: Rotate 120° – If -240° has been threaded, the nut must be released and the arm must reset its rotation angle before continuing to thread.
- a_4: Open/close the gripper

D. Metrics of Evaluation

To evaluate the success of the system, multiple metrics were used. The average reward per episode and average Q-value per episode were used as suggested by Mnih et al. [10] and Bellemare et al. [12]. The average reward per episode is a direct indicator of performance and should steadily increase with training time as the agent learns to take better actions. The average Q-value per episode represents the expected future action rewards and reflects the agent's confidence in its policy. A gradual increase should be observed, but increasing Q-value without increasing average reward can suggest overconfidence. It was intended that the success rate, or the percent of threading attempts which are completed in full, be used as well. However, the agent's training did not progress enough for this to be a relevant evaluation metric.

III. Results

Multiple iterations of rewarding schemes yielded poor performance. This was due to both incorrect rewarding, as the action taken was used to determine correct reward, as well as imperfections in the rewarding scheme that allowed for positive reward for incorrect moves.

A. Iteration One

The first reward scheme awarded a positive reward of one for the agent choosing to move +120° or -120° with the gripper open and zero reward for choosing these moves when the gripper was closed. If the -5° movement action was chosen, a positive one reward was given if the gripper was closed and cross-threading had been detected, otherwise a negative one reward was given. A positive reward of 1 - 2 * T_z was awarded for +5° movement chosen with the gripper closed and a negative one reward if this move was chosen otherwise. Finally, a positive reward of one was awarded if the gripper was opened or closed in specific angle ranges.

In the first two testing iterations, the epsilon value was decayed over only 1000 steps. After the initial exploration period expired, the agent consistently chose to make -5° movements, resulting in a very negative average reward per episode as can be seen in Figure 4. It continued this behavior until discovering an exploit in the reward scheme which allowed it to gain a positive one reward for opening and closing the gripper. It quickly became clear that this reward scheme required reworking.

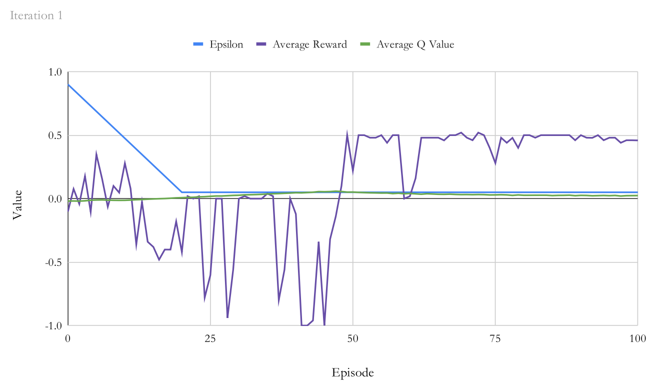

Fig. 4. Iteration one results.

B. Iteration Two

The main change from iteration one to iteration two consisted of a negative reward for the agent choosing a gripper action at an unintended time. However, the agent only chose -5° moves despite the nearly consistent negative reward. This indicated a divergence in Q-learning, wherein the values become increasingly meaningless. This could have been due to many factors, most probably poor random actions during the exploration phase. As can be seen in Figure 5, the agent appeared to experience only a negative reward while the epsilon value was greater than 5%, meaning it had no exposure to actions yielding positive rewards. This could also have been due to unstable learning or due to the underlying issues in the rewarding strategy.

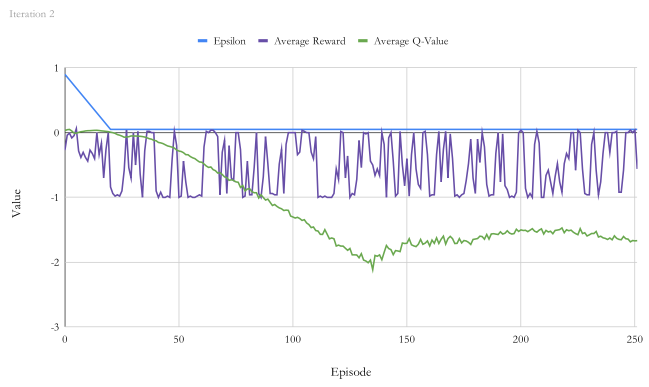

Fig. 5. Iteration two results.

C. Iteration Three

In the third iteration, the rewards were entirely reworked in an attempt to make them more granular. In addition, the epsilon decay period was increased to 5000 steps from this iteration forward. In this scheme, the agent would receive a minimum reward of zero. For each correct move under the correct conditions, 0.2 would be added to the sum reward and one would be subtracted for incorrect actions. A 'goal achievement' reward was added to reward the agent for opening the gripper when 240° was reached. The agent, however, learned to exploit this goal achievement reward when possible, as can be seen from the average reward spikes in Figure 6.

D. Iteration Four

The only change in reward approach between iteration three and four was the removal of the 'goal achievement' reward that was exploited. The results from this approach were fairly normal, as can be seen in Figure 7, though the agent never seemed to learn the optimal moves. This could be due to insufficient training time, but was more likely due to poor rewarding.

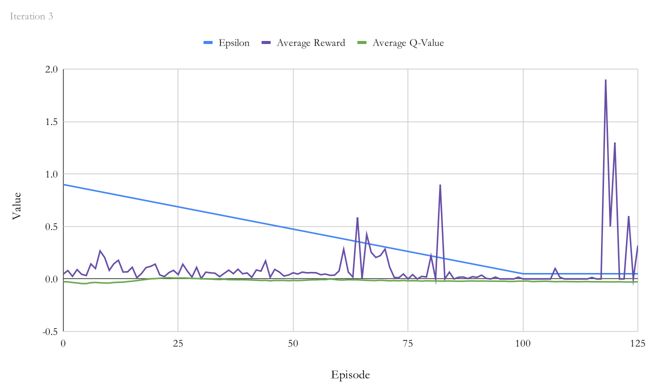

Fig. 6. Iteration three results.

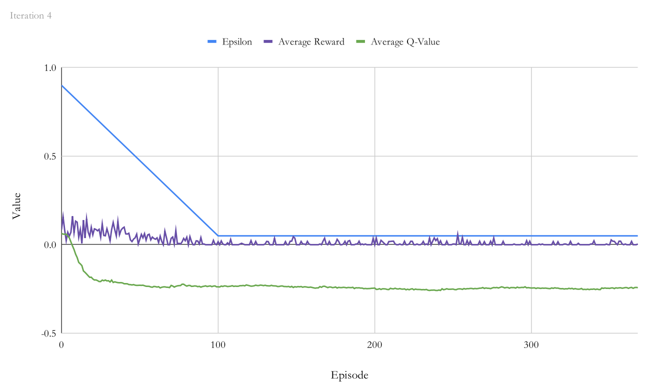

Fig. 7. Iteration four results.

E. *Final Iteration*

The rewards in the final iteration were completely reworked to no longer rely on the action taken. The reward scheme was designed to be fairly granular, with progress rewards given for correct changes in hex nut angle under the correct conditions. Progress rewards were also given for correct changes in relative angle when the arm was not gripping the hex nut. Most importantly, a large award was given for task completion. The result of this completion reward can be seen as large spikes in Figure 8. Interestingly, these completions happened during the mainly exploratory phase of training and did not occur when the agent was making its own decisions. This indicates that more training would be needed for the agent to associate the correct actions with a successful threading.

However, there was promising improvement between the successful threading attempts, as can be seen in Figures 9 and 10. An ideal path to a completed threading attempt would have the nut angle increase nearly linearly with small plateaus where the arm would reverse. From steps 0-13 of training, the hex nut was rotated in a very erratic fashion until finally reaching 600° . From steps 53-55, a much smoother path was taken to reach 600° , indicating improvement in the model's decision making. The reduced time to reach this success with a reduced epsilon is also promising.

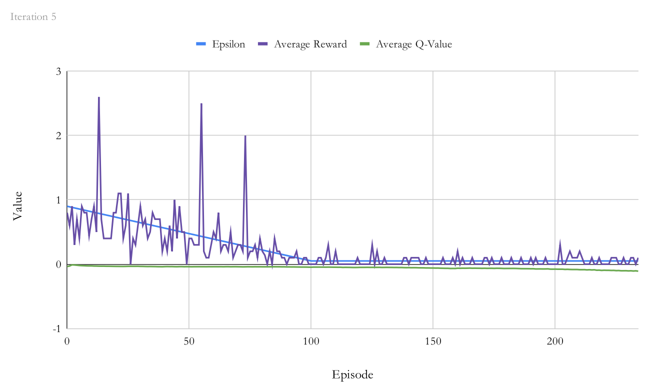

Fig. 8. Final iteration results.

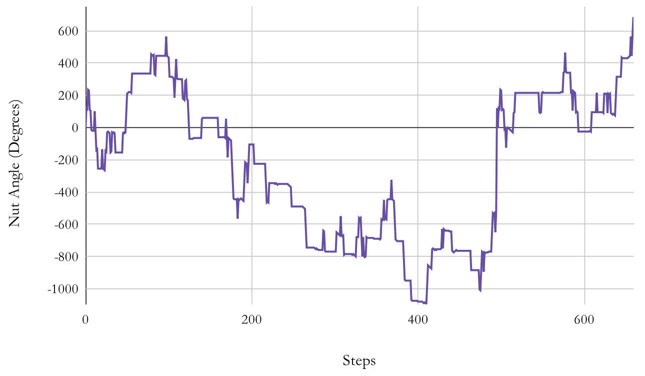

Fig. 9. Steps 0-13 successful threading.

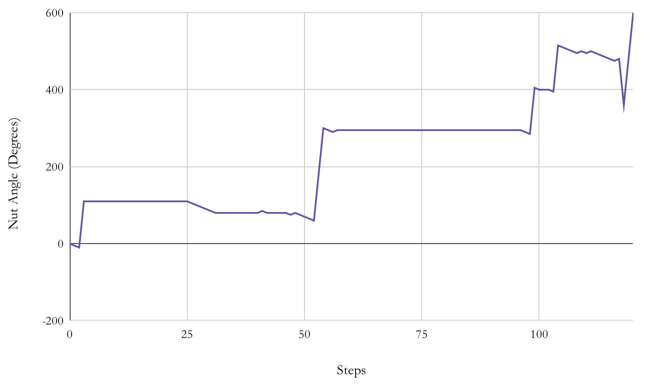

Fig. 10. Steps 53-55 successful threading.

IV. Conclusion

Introducing reinforcement learning to a complex task is a challenging problem. For many systems, it is possible to train a model in simulation before transferring it to a real system, allowing many iterations of training to be completed without significant time investment from the user. For systems such as this where the interactions between components are too complex to simulate, all training must be done on a physical setup. This leads to a considerable time investment for training. Further, it takes a large investment of time to discover flaws in the rewarding scheme or agent structure. On this system, each episode took anywhere from thirty seconds to two minutes to complete, meaning that each failed iteration was trained for at least three hours before the flaws became apparent.

The final iteration of this project yielded promising results. For further improvement, the agent likely needs significantly more training time. However, such a large time investment is unfeasible. Longer episodes or a longer exploration period may also help the model improve more rapidly. Further, a more granular rewarding scheme could help the agent learn which actions to take to most efficiently reach a successful threading. In all, training a reinforcement learning algorithm on a physical system is sometimes necessary and poses many unique challenges to be addressed.

References

[1] N.-A. Noda, X. Chen, Y. Sano, M. A. Wahab, H. Maruyama, R. Fujisawa, and Y. Takase, "Effect of pitch difference between the bolt–nut connections upon the anti-loosening performance and fatigue life," *Materials Design*, vol. 96, pp. 476–489, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0264127516301289

[2] J. Huang, J. Liu, H. Gong, and X. Deng, "A comprehensive review of loosening detection methods for threaded fasteners," *Mechanical Systems and Signal Processing*, vol. 168, p. 108652, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S088832702100978X

[3] Z. Jia, A. Bhatia, R. M. Aronson, D. Bourne, and M. T. Mason, "A survey of automated threaded fastening," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 1, pp. 298–310, 2019.

[4] A. Hsue and C.-F. Tsai, "Torque controlled mini-screwdriver station with a scara robot and a machine-vision guidance," 11 2020, pp. 465–468.

[5] A. Rastegarpanah, R. Ner, R. Stolkin, and N. Marturi, "Nut unfastening by robotic surface exploration," *Robotics*, vol. 10, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2218-6581/10/3/107

[6] G. Pettinaro, "Human modeled robot screw fastening," in *Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No.98CH36190)*, vol. 2, 1998, pp. 1009–1014 vol.2.

[7] K. Wellington, M. K. Jouaneh, J. Tingley, P. Stegagno, and S. Lanzi, "Automated nut fastening," 2024, unpublished.

[8] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013. [Online]. Available: https://doi.org/10.1177/0278364913495721

[9] E. F. Morales, R. Murrieta-Cid, I. Becerra *et al.*, "A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning," *Intelligent Service Robotics*, vol. 14, pp. 773–805, 2021. [Online]. Available: https://doi.org/10.1007/s11370-021-00398-z

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[11] Y. Li, "Deep reinforcement learning: An overview," 2018. [Online]. Available: https://arxiv.org/abs/1701.07274

[12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *CoRR*, vol. abs/1207.4708, 2012. [Online]. Available: http://arxiv.org/abs/1207.4708