Title:

Using Reinforcement Learning in Robotic Simulations

Members:
Timothy Klein, Liam Jennings, ,Yaşar İdikut

Abstract:

The overall goal of this project is to design and implement a robot that is capable of navigating a simulated environment and pushing a cylinder obstacle to a specific goal. To achieve this, we have decided to use a reinforcement learning approach, specifically Deep Q-learning. Setting up the environment and getting the reinforcement learning model working have been significant milestones in this project. In order to properly train the robot, we have utilized Gazebo and ROS to create a virtual environment for a simulated robot to operate in. We then focused on implementing the reinforcement learning algorithm. This involved setting up the reward function, and configuring the training process. Overall, this project has required a combination of technical skills in robotics, machine learning, and computer simulation.

NOTE: some of the sources in this paper are from earlier years than 2018, however, the information that these sources provide are still very much relevant to today and the information is too valuable not to include them. We still included ten sources that are later than 2018.

Keywords:
Deep Q-learning, reinforcement learning, ROS, Gazebo, Turtlebot3, machine learning, simulated environment, hyper parameters, AI, and vector field.

Introduction:

Combining AI with robotics would improve the versatility of robots, as this would allow them to be implemented in a wider range of situations and environments. Incorporating AI into robotic design can potentially increase the robot's ability to adapt to new tasks and environments more easily, increasing their overall usefulness. Some examples of successfully implementing AI into robotics include researchers at the Georgia Institute of Technology developed a robot that was able to learn how to perform new tasks by observing a human demonstration and using reinforcement learning [1]. The Previous source is from 2013 which is considered outdated, however, the information that it includes is still up to date and relevant for this paper. Additionally, a team at the University of California, Berkeley used deep learning techniques to teach a robot how to manipulate objects it had never seen before [2]. Both examples demonstrate the potential of AI to significantly enhance the versatility of robots. Overall, the goal of our project is to utilize AI to help make robots more versatile, as this would enable them to be used in a wider range of situations and environments.

Implementing reinforcement learning into a robot in order to achieve a specific goal, such as moving a box to a designated location, was a challenging task. There are several key factors that need to be taken into account when designing a reinforcement learning system for a robot, the most important of which is determining the appropriate AI techniques to use. There are several different methods that can be used for reinforcement learning, including Q learning, and SARSA, and deep Q-learning. Each of these methods has its own advantages and disadvantages, and choosing the most suitable one will depend on the specific requirements and constraints of the task. For example, Q-learning is a simple and widely-used reinforcement learning algorithm that is easy to implement, but it can be slow to converge and may not be suitable for tasks that require a high degree of precision. SARSA can learn from its own actions, but it is able to be trapped in suboptimal policies. On the other hand, deep reinforcement learning can handle more complex tasks and can learn faster, but it requires more computational resources and can be more difficult to implement. Based on previous works, other examples of AI-based robotics, and our problem domain, we chose to use the Deep Q-learning algorithm.

Another challenge was setting up the reward system for reinforcement, which determines how the robot will be motivated to achieve the desired goal. The reward system needed to be carefully designed to provide the right balance between encouraging the robot to take actions that push a cylinder obstacle closer to the goal, while also punishing actions that take it further away. Despite these challenges, there are many benefits to using reinforcement learning to enable robots to achieve specific goals. Reinforcement learning allows the robot to learn and adapt to new tasks and environments more easily, increasing its overall versatility. Additionally, it can enable robots to learn from experience, which can be more efficient than manually programming them to perform specific tasks. Overall, implementing reinforcement learning into a robot in order to achieve a specific goal requires careful consideration of the appropriate AI techniques and the design of the reward system. While there are challenges involved, the benefits of using reinforcement learning can be significant, making it a valuable tool for improving the capabilities of robots.

One of the main disadvantages of deep Q-learning is that it can be slow, especially for tasks that are complex or involve a large number of states and actions [4]. Our robot takes a very long time to process since the algorithm relies on trial and error to learn the value of each state and action which makes the editing of parameters very difficult. Another disadvantage is that deep Q-learning can require a large amount of data and computational resources in order to learn effectively. This can make it difficult to apply the algorithm to tasks that require real-time decision-making which is typically required for robotics. The approach that we used was to prioritized experience replay, which helped the algorithm to learn more efficiently by storing and reusing past experiences.

Designing a deep Q-learning system to enable a robot to push a box to a specific goal involved the following steps. The first step was to define the task that the robot needed to perform, as well as the environment in which it would be performing the task. This involved identifying the inputs such as sensor data, location of the box, and location of the goal. We also needed to identify the outputs and constraints of the system such as obstacles of the environment, the robot's motion, and the physical constraints of the robot due to its shape and size. Next, we created the reinforcement learning framework which included the reward system that was used to set the robot on the right path in order to correctly move the cylinder obstacle to the target location. This involved defining the rewards that the robot would receive for taking actions that brought the cylinder obstacle closer to the goal. We also established penalties that the robot would receive for actions that resulted in the cylinder obstacle moving farther away from the goal. Lastly, we trained the deep Q-learning model. This involved feeding the model large amounts of data about the robot taking actions and receiving rewards or penalties. The model would then learn to predict the value of each state and action based on the rewards.

For the rest of the paper, we will discuss related works and how we achieved our goal. For related works, there are many different sources of the use of reinforcement learning to get a robot to achieve some goal but for our paper we will be focused mainly on deep Q-learning. Our proposed application section is essentially what we actually did to achieve the goal of having the robot push a cylinder obstacle to a specific goal in simulation and how we did it which was briefly mentioned above. Experiment results and discussion is just us talking about the results of our testing and the conclusion and future work is discussing what the result of our program is and what other application that it could be used for.

Related Work:

Q learning[8] is a kind of reinforcement learning algorithm that aims to find a policy that maximizes its reward for a given environment. Unlike many other kinds of algorithms, it does not use a model. Instead, it stores the expected reward for every combination of state and action in a Q-Table. This table is updated iteratively as it moves through the environment, using the Bellman equation to update Q values after their action has been taken. The algorithm acts on a Exploration/Exploitation tradeoff - at first, it acts randomly to explore the environment, but later acts more on the information gathered.

Deep Q learning is a type of reinforcement learning algorithm that uses deep neural networks to approximate the Q-function. In normal Q learning, the Q-function is calculated through a Q-Table. The Q-Table stores the expected reward for every combination of state and action. This approach is not scalable- it is applicable for smaller problems, but the size of the Q-Table grows polynomially. There are various optimizations commonly applied to address this, but Deep Q Learning takes a fundamentally different approach by approximating the Q function through a neural network, which takes in a state as input and produces Q values for each action from that state as outputs. However, Q learning can be difficult to train, especially if the Q function is highly nonlinear or the learning problem is complex. This can require large quantities of computational resources and time. Moreover, the model can suffer from instability and divergence during training, especially when using function approximation with neural networks [9].

The SARSA (State-Action-Reward-State-Action) algorithm is another type of reinforcement learning algorithm. In contrast to Q-learning, SARSA is an on-policy algorithm. While still using a Q-Table to store values, it uses the current best policy to train itself. The algorithm uses the quintuple of the current state, action, reward and next state and action to iteratively update the Q-Table, the origin of the acronym. However, not only can SARSA be slower to converge to the optimal policy compared to some off-policy algorithms, but it can also be sensitive to the choice of learning rate: finding the optimal learning rate can be challenging. SARSA can sometimes get stuck in suboptimal policies, especially in environments with multiple local optima. This can occur because the algorithm is biased towards the current policy, which may not always be the optimal policy.

Proposed Application:

We followed source [3] with a slightly modified environment and reward function that favors the robot to go behind a cylinder obstacle and push it to the target location. The robot starts off in the center with the cylindrical object 30cm in front of it. A random goal location is selected at the start. Each time a goal is reached, the robot comes to a stop and a new random location is selected. That random location is highlighted with pink. Robot has a constant forward velocity of 0.15m/s. The AI basically takes in 24 (lidar measurements) + 1 (heading of the robot relative to where it should be facing to in the map) + 1 (distance between the robot and the cylinder) + 1 (distance between the cylinder and the goal position) = 27 variables(state size=27) and outputs one of five actions:

Action	Angular Velocity (rad/s)
0	1.5 (Hard Left)
1	0.75 (Soft Left)
2	0 (Straight)
3	-0.75 (Soft Right)
4	-1.5 (Hard Right)

The lidar sensor used produces a measurement for every one degree, totaling to 360 measurements. However, this was reduced to 24 measurements to reduce complexity and the state size for the purposes of faster training. These lidar measurements are also used to calculate the robot's position and orientation relative to the world frame. AMCL is a system for localization that utilizes lidar. AMCL is used so that a robot is able to accurately determine where it is in a mapped environment [13]. The yaw angle of the robot is extracted from the orientation calculated by the lidar measurements. To calculate the heading portion of the state, a heading vector calculating function is used. This function is "tuned" by hand. Given the position of the robot, cylinder object, and the goal in the map, it generates the desired heading vector. Figures 2 and 3 illustrate this function by creating a vector field map. This function basically favors the robot to go behind the cylinder and push it towards the goal. The yaw angle of the robot is then subtracted from the desired heading angle calculated to come up with the state's heading (25th variable). This heading is essentially the error that needs to be minimized and is done so in the reward function by favoring actions that lead to heading being 0.

As described earlier, the position of the robot is derived from the odometry measurements (24 lidar measurements). The position of the cylinder obstacle is directly polled from the Gazebo simulator. The goal position is determined by the training code and a new random one is generated each time the robot is successful in pushing the cylinder to this position. The 26th state variable is calculated by the hypotenuse of the robot and the cylinder obstacle positions. Lastly, the 27th variable is calculated by the hypotenuse of the cylinder obstacle and the goal positions.

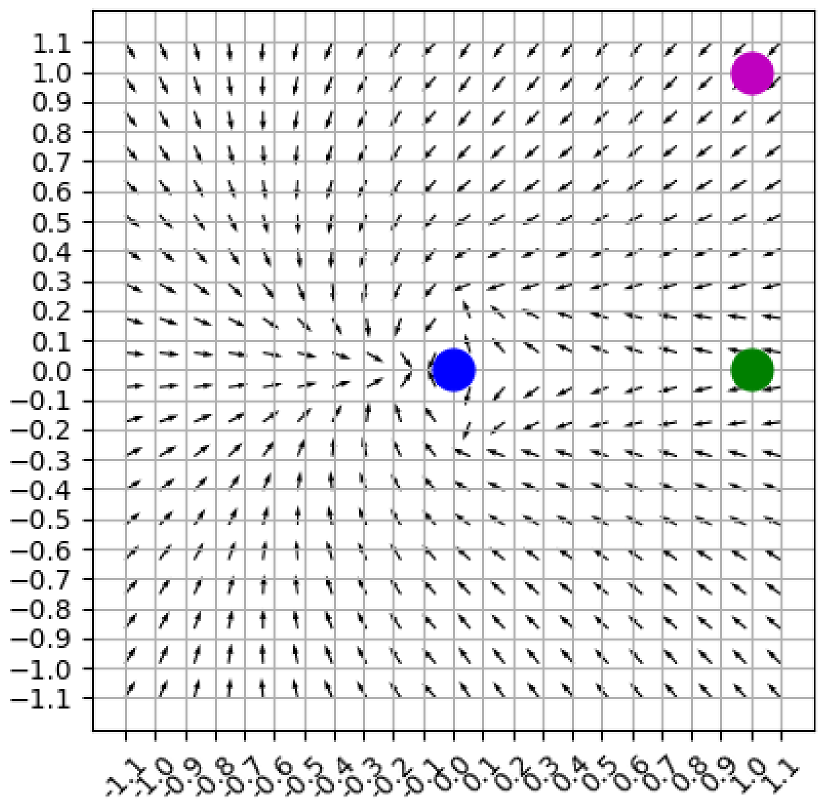

Figure 2: The field vector graph of the desired heading angle based on cylinder obstacle position plotted blue at (0, 0), goal position plotted green at (1, 0), and robot position(all the vectors represent a robot position, but to give an example of where the robot could be, a pink dot at (1, 1) was plotted).

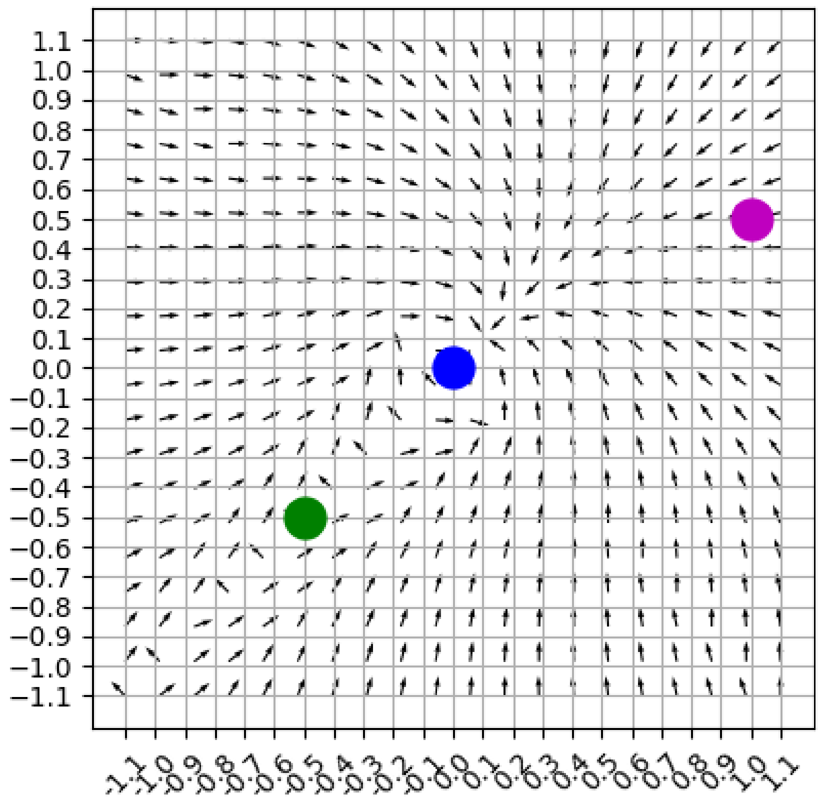

Figure 3: The field vector graph of the desired heading angle based on cylinder obstacle position plotted blue at (0, 0), goal position plotted green at (-0.5, 0.5), and robot position(all the vectors represent a robot position, but to give an example of where the robot could be, a pink dot at (1, 0.5) was plotted).

Our application uses the Deep-Q Learning model to find an optimal policy to maximize reward. Our reward function is shown below:

```
Algorithm for function setReward(state, done, action):
    yaw_reward = []
    for i in range(5):
        angle = -pi / 4 + heading + (pi / 8 * i) + pi / 2
        tr = 1 - 4 * math.fabs(0.5 - math.modf(0.25 + 0.5 * angle % (2 * math.pi) / math.pi)[0])
        yaw_reward.append(tr)

    distance_rate_robot_cylinder = 1 + (self.robot_cylinder_init_dist - robot_cylinder_curr_dist) ** 2
    reward = ((round(yaw_reward[action] * 5, 2)) * distance_rate_robot_cylinder)

    If done:
        Reward = -200 // collision with a wall

    If cylinder_goal_curr_dist < 0.3
```

```
        Reward = 200 // goal reached

    Return reward
```

To simulate the robot, we used Robot Operating System(ROS) and Gazebo. Unfortunately, that means we could not speed it up faster than real time.

We are able to visualize the total reward collected for each episode.

<u>Experimental Results and Discussion:</u>

The training was run for 300+ episodes(about 14 hours), however, the results after the ~210th episode deteriorated. This corresponds to about the 33rd minute-mark in the training video: https://www.youtube.com/watch?v=sYeYrgIsD40. The video is sped up by 16 times. A notable example of the AI agent is at 32:50-33:05 timestamp of the video. Due to the nature of our project, we can't really provide test or example cases, but the video should demonstrate the learning. Nonetheless, figure 4 shows the total reward and average Min-Q value for each episode. For the rest of our results, please refer to the video. We believe that better results could be achieved with a better reward function.

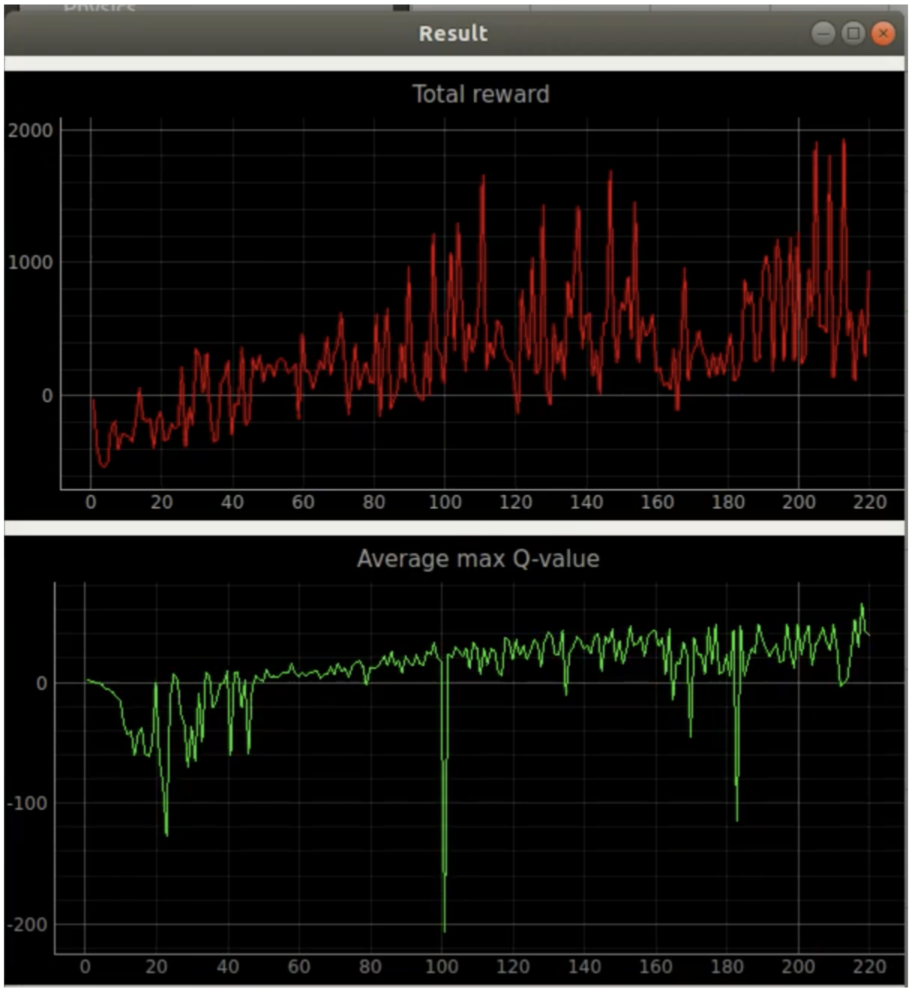

Figure 4: Total reward and average Min-Q value for each episode

Conclusion and Future Work:

For this project, our team was able to successfully apply deep Q-learning to a robot in order for the robot to push a cylinder to a desired goal in simulation. Our system is fully capable and we were able to achieve everything we sought out to do. There are many different real world and practical uses for a robot that is able to manipulate its own environment through the use of deep Q-learning. This project could be used by robots that are designed to explore other planets since it would have to manipulate many different things and due to connection problems and lag, the robot will have to do all the manipulation autonomously for the most precision. This project could also be used for developing autonomous agricultural robots and we could even take this idea a step farther and implement a machine algorithm to determine what each plant would need to maximize crop yield. This project could also lead to more functionable autonomous robots working in health care or even disaster relief efforts. The robot could use the deep Q-learning in order to move patients or debris. The most likely use of this project would be in manufacturing since manufacturing requires the manipulation of many different objects. Manufacturing includes but is not limited to, wielding, carving, injection molding, painting, maching, and essentially anything in manufacturing that requires a high level of precision.

There have actually been many cases of robotics and deep Q-learning being used in the real world. For example, a robot called "Daisy" developed by the University of Cambridge uses deep Q-learning to learn how to perform tasks such as grasping objects and opening doors [5]. Another robot developed by the National Institute of Advanced Industrial Science and Technology in Japan uses deep Q-learning to learn how to perform tasks such as sorting objects by color [6]. Lastly, a robot developed by the University of Maryland uses deep Q-learning to learn how to perform tasks such as navigating through a cluttered environment and grasping objects [7].

References:

1. Georgia Institute of Technology. (2013). Robot learns new tasks by observing human demonstrations. ScienceDirect, 1-11.
2. University of California, Berkeley. (2019). Learning to manipulate. Berkeley Artificial Intelligence Research.
3. Robotis. (n.d.). TurtleBot3 Machine Learning. Retrieved from https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/
4. Wang, D., Hu, Y., & Arulkumaran, K. (2018). Deep reinforcement learning for robotics: A review. Frontiers in Robotics and AI, 5, 1-19. Retrieved from https://doi.org/10.3389/frobt.2018.00024
5. Chen, Y., Leibe, B., & Schiele, B. (2018). Daisy: A deep reinforcement learning approach to visual perception for robotic grasping. IEEE Transactions on Robotics, 34(1), 120-133. https://doi.org/10.1109/TRO.2017.2745184
6. Nagai, Y., Saito, K., & Kajita, S. (2018). Deep reinforcement learning for bin-picking tasks. IEEE Transactions on Industrial Electronics, 65(7), 5930-5938. https://doi.org/10.1109/TIE.2017.2769050
7. Zhou, X., Jia, Z., & Guo, Y. (2018). Deep reinforcement learning for robotic grasping. IEEE Access, 6, 67107-67115. https://doi.org/10.1109/ACCESS.2018.2869041
8. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). Cambridge, MA: MIT Press.
9. Babaeizadeh, M., Czarnecki, W. M., Pachocki, J., Hausknecht, M., & Stone, P. (2018). Reactor: A fast and stable architecture for deep reinforcement learning. arXiv preprint arXiv:1803.09478.
10. Foerster, J., Assael, Y., de Freitas, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In International Conference on Machine Learning (pp. 2926-2935).
11. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., … Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
12. Wang, L., & Yang, J. (2015). Deep reinforcement learning with neural networks: An overview. IEEE Transactions on Neural Networks and Learning Systems, 26(2), 395-417.
13. Burgard, W., Fox, D., Thrun, S., & Welker, K. (1999). The interactive museum tour-guide robot. Autonomous Robots, 7(1), 49-71. doi:10.1023/A:1008839917675