Pathfinding is a fundamental component of many important applications in the fields of GPS , video games , robotics , logistics, and crowd simulation and can be implemented in static, dynamic, and real-time environments. [[1]](#endnote-1)And obstacle avoidance is an important component for path finding, as avoiding collisions with the environment and not having unrealistic plans to go through an obstacle is an import utility of a navigation program for a mobile robot. Considering its importance, we shouldn’t be surprised to see the wide varieties of approaches people came up with to tackle it, and one of the most popular solution would be reinforcement learning.

Reinforcement learning proposes a way of programming agents by reward and punishment without needing to specify how the task is to be achieved[[2]](#endnote-2). It punishes the agent for actions that goes against the expected behaviour, and rewards the agent if the action is helpful for getting the task achieved. It does come with a formidable computational expense and hasn’t really caught on until the last two to three decades. Two kinds of approaches for reinforcement learning were proposed – model-based and model-free reinforcement learning. Model free means learning a controller without learning a model and model based methods learn a model, then use it to derive a controller. Q-learning and its variations, being one of the simplest method in model-free approaches of reinforcement learning, has appeared in a large amount of research. **(POSSIBLE REFERENCE)**

Q-learning constructs a two-dimensional Q-table for choosing an action under a certain situation. Each value corresponds to the same set of actions and all the actions are assigned a value, default to be 0. The action with the maximum value will be chosen. The value correspond to a state action pair will be decreased if it leads up to failure and increased if it contributes to success. [[3]](#endnote-3)

One of the classical application of Q-learning requires a fully observable environment that can be discretized. It uses the discretized environment as input and come up with a path for the mobile robot to follow using control scheme like inverse kinematic control. It converts planning in continuous environment into grid path planning, which has a great range of applications in areas like robotics and video games[[4]](#endnote-4). There have been many efficient solutions to grid path planning, but all of them has the same limitation – every time the agent encounters a new environment, it has to start the learning process all over again. An application of Q-learning that can be applied to a wide variety of vehicles and only need to train the robot once for it to tackle any possible environment is desired.

In this project a sanity check for such desire is provided. All components and functions are stripped down to its simplest form to give us the simplest implementation of the above idea, and the level of environmental complexity that this agent can handle is tested. The agent is provided with three test environments, with increasing level of complexity.

（ADD SOME OF MY ALTITUDE TOWARDS MY WORK）

The implementation is straightforward - the data from the two distance sensors on the agent is used as ‘state’ required by the Q-learning algorithm, and the algorithm provides a suitable ‘action’, which would be mapped to the torque of the agent’s two motors.

Tests were ran on each of the three environments and three histogram showing number of collisions were produced to reflect the result of the training.

1. (<https://www.hindawi.com/journals/ijcgt/2015/736138/>) [↑](#endnote-ref-1)
2. (<https://arxiv.org/pdf/cs/9605103.pdf>) [↑](#endnote-ref-2)
3. <http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf> [↑](#endnote-ref-3)
4. ([https://www.sciencedirect.com/science/article/pii/S1877050918300553#](https://www.sciencedirect.com/science/article/pii/S1877050918300553)!) [↑](#endnote-ref-4)