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. (<https://www.hindawi.com/journals/ijcgt/2015/736138/>) And obstacle avoidance is an important component for path finding, as the robot should stay intact by avoiding collisions with the environment and shouldn’t come up with unrealistic decision to go through an obstacle to reach the destination. 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

(<https://arxiv.org/pdf/cs/9605103.pdf>). 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. Q-learning and its variations, being one of the simplest method in model-free – learning a controller without learning a model - approaches of reinforcement learning, has appeared in a large amount of research.

What Q-learning does is constructing 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 and action with the maximum value will be chosen. Its value will be decreased if it leads up to failure and increased if it contributes to success. <http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf>

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 ([https://www.sciencedirect.com/science/article/pii/S1877050918300553#](https://www.sciencedirect.com/science/article/pii/S1877050918300553)!). 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. The learning is restricted to learning the given environment instead of learning the actual skill of path finding.

To deal with the limitation of the classical approach, we want to use the precepts of the agent as the input and come up with proper action – in our case, the torque for the two motors – according to the percept and the Q-table acquired from Q-learning process. Thus the Q-learning process only needs to be carried out once for a certain type of vehicle, and after proper learning process we only need

Limited by the timeframe of our project and the package we are using, we are focusing on realising the obstacle avoidance portion of the path finding, which can be extended to a full path finding solution if the performance matches our expectation. And we can only approximate the collision detection and the resetting processes before each iteration.