Reinforcement Learning for Autonomous Driving - Week 5

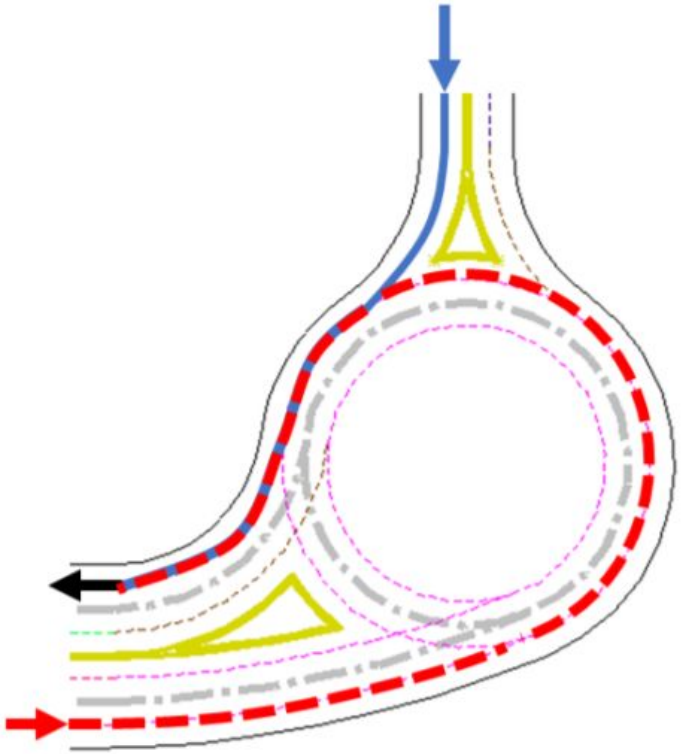
Brandon Dominique, Hoang Huynh, Eric Av, John Nguyen

Overview

- Problem Statement: Learned collision prevention in Gazebo.
  - Use reinforcement learning to train the CAT Vehicle in Gazebo to detect and avoid potential collisions.
  - Utilize a meta-cognitive radio to relay information to nearby vehicles.
  - Use meta-learning to train the vehicle on a variety of situations.

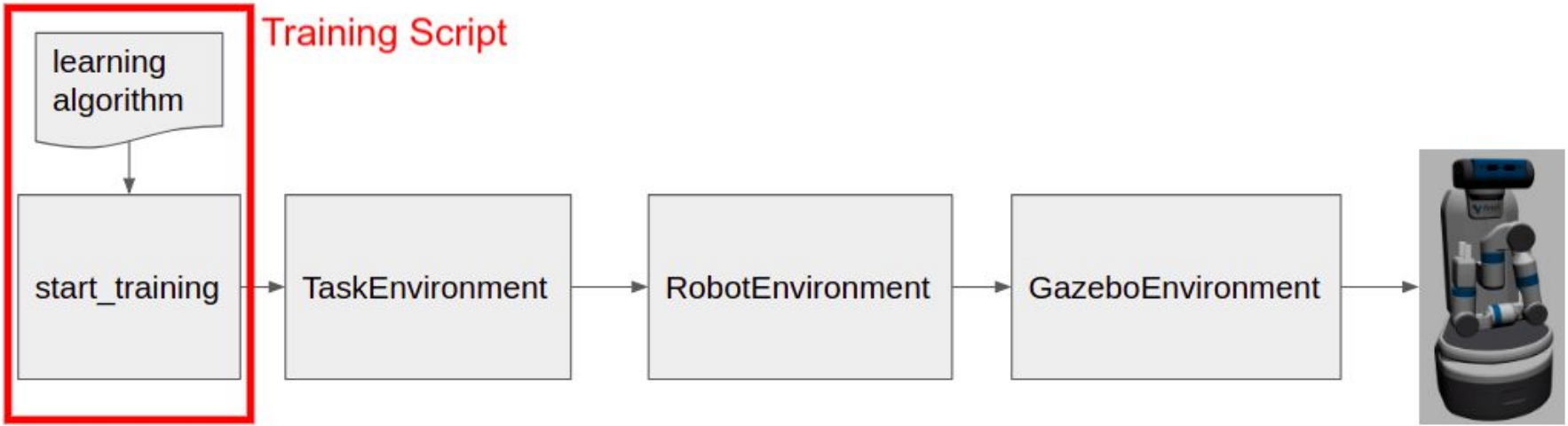
Lit Review: RL and Autonomous Vehicles with Simulators

- Simulated to Scaled City: Zero-Shot Policy Transfer for Traffic Control via Autonomous Vehicles, by Jang et. al.
  - RL on autonomous vehicles using a simulator; did not use ROS/Gazebo.
  - Jang et. al. used deep reinforcement learning to train autonomous vehicles leading cars into a roundabout.
  - Software not applicable to our project since it was not done in ROS.
  - Gave me ideas on what to keep in mind working with ROS.



RL and ROS/Gazebo

- OpenAI ROS: ROS package which integrates ROS/Gazebo and Open AI.
  - Need to define all environments to integrate with CAT Vehicle.
  - Different environments are modular; can change one out for another with minimal changes.
- Other ways to integrate RL and ROS are outdated.



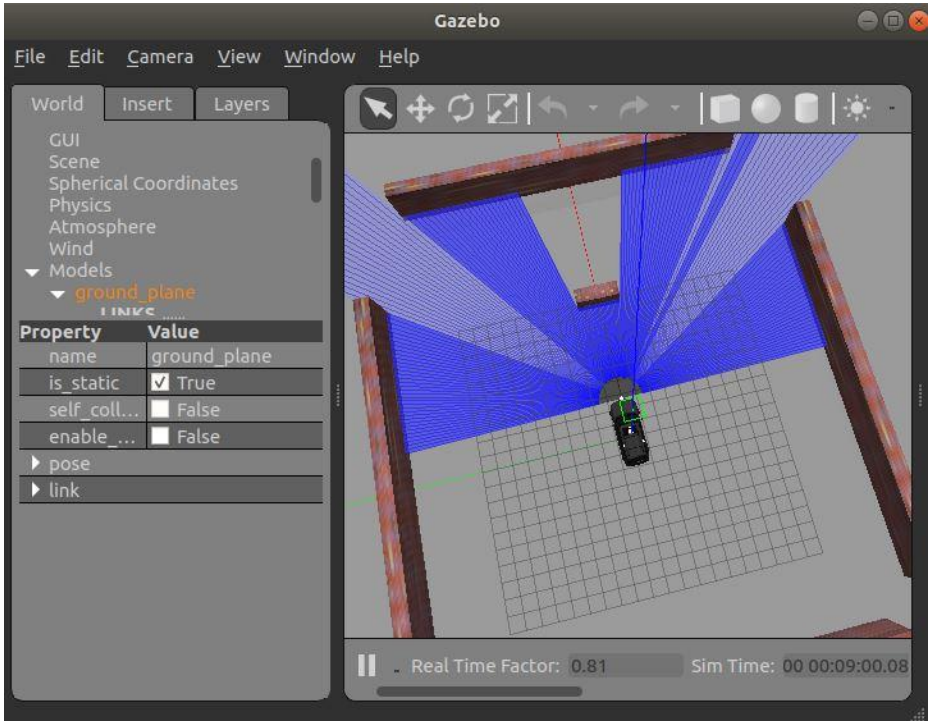
What to do with Open AI ROS:

- Open AI ROS built to work with ROS Development Studio.
  - A little work required to get it to run on our machines.
- Converts a Gazebo environment into an Open AI Gym Environment.
  - Allows us to use the utility of Open AI Gym with ROS.

```
class RobotGazeboEnv(gym.Env):

    def __init__(self, robot_name_space, controllers_list,
reset_controls, start_init_physics_parameters=True,
reset_world_or_sim="SIMULATION"):

        # To reset Simulations
        rospy.logdebug("START init RobotGazeboEnv")
        self.gazebo =
GazeboConnection(start_init_physics_parameters,reset_world_or_sim)
        self.controllers_object =
ControllersConnection(namespace=robot_name_space,
controllers_list=controllers_list)
        self.reset_controls = reset_controls
        self.seed()

        # Set up ROS related variables
        self.episode_num = 0
        self.cumulated_episode_reward = 0
        self.reward_pub = rospy.Publisher('/openai/reward',
RLExperimentInfo, queue_size=1)
        rospy.logdebug("END init RobotGazeboEnv")

    # Env methods
    def seed(self, seed=None):
        self.np_random, seed = seeding.np_random(seed)
        return [seed]

```

Basic Example

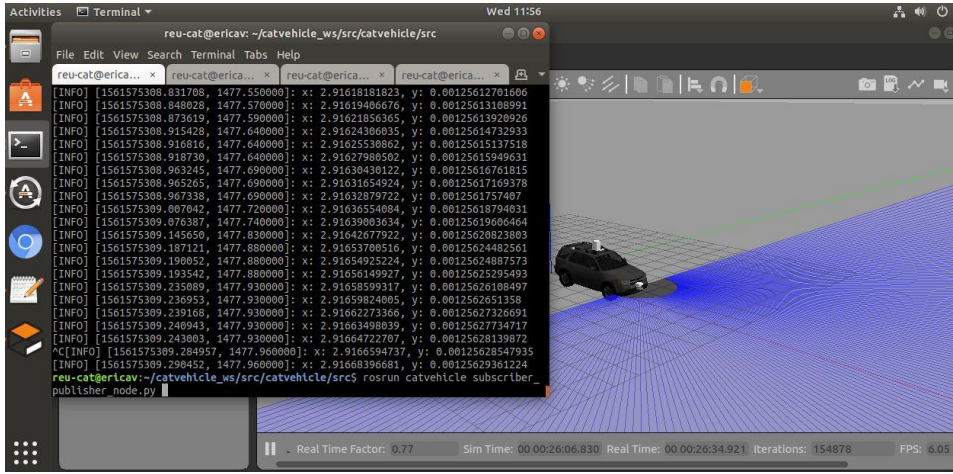
- Car with an obstacle in front, in a closed world.
  - Constant velocity moving forward.
  - Takes sensor data describing distance and angle of object from car.
  - RL algorithm tells car how much to turn.



Gazebo and ROS Progress

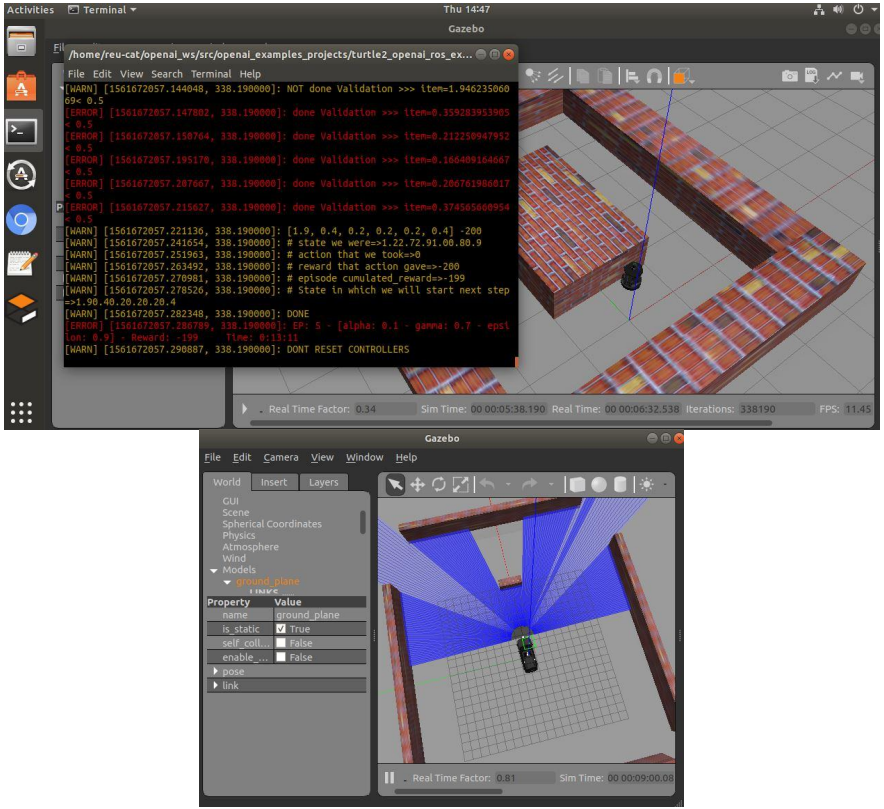
- Learned how to subscribe and publish a topic
- Used that knowledge to subscribe to odometry (position) of the car and publish to a string
- Learned where to find a catvehicle topic's information and package summaries (geometry_msgs, etc.)

```python
#! /usr/bin/env python

import rospy
from std_msgs.msg import String

rospy.init_node('subscriber') #position of car, , distEstimator

publisher = rospy.Publisher('hi', String, queue_size=1)
rate = rospy.Rate(3)

while not rospy.is_shutdown():
    publisher.publish('Hey!')
    rate.sleep()
```

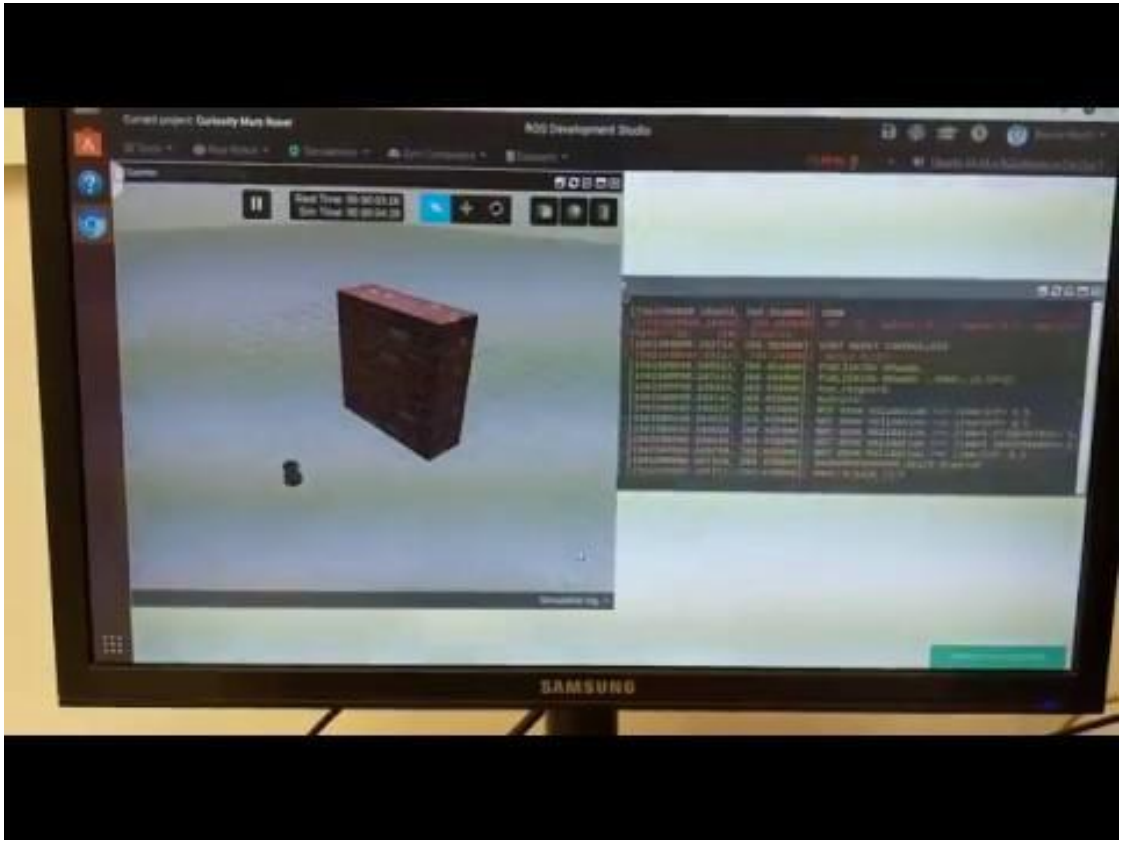


Gazebo and ROS Progress Cont.

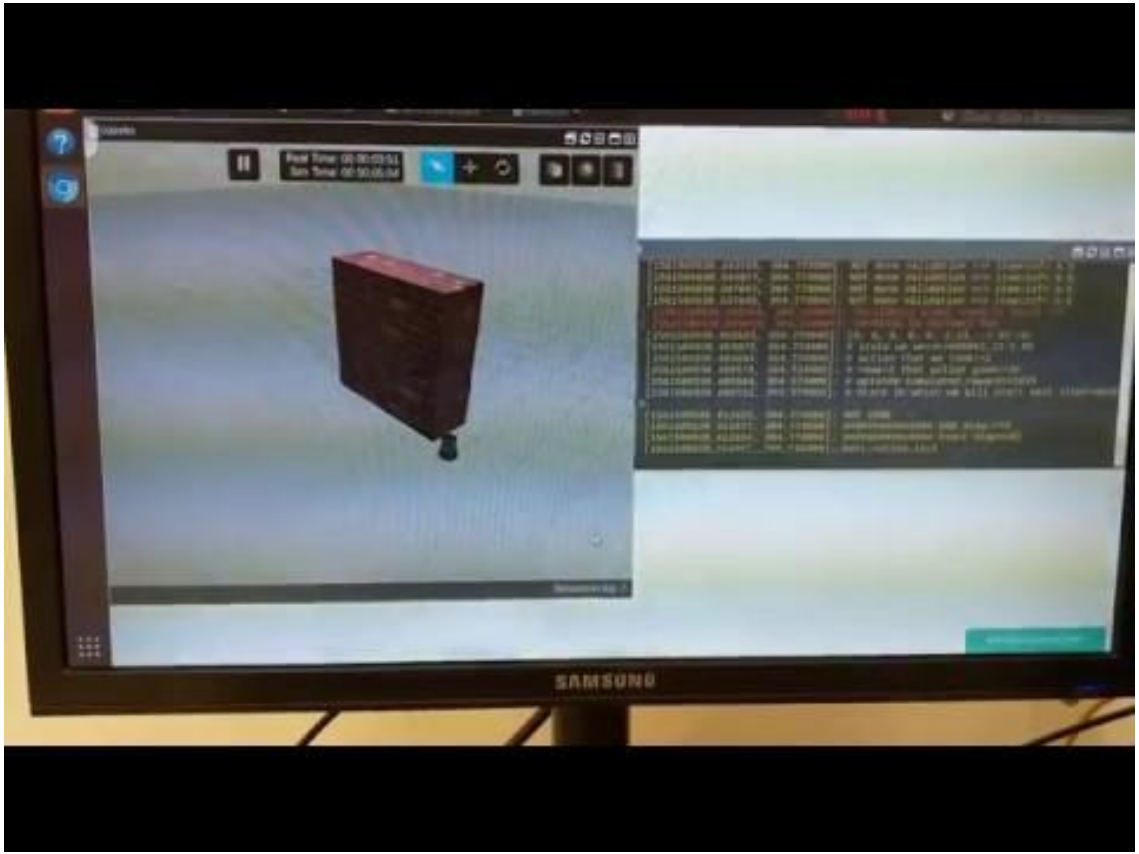
- Installed a working reinforcement simulation of a robot trying to drive around a box maze with OpenAI
- Deconstruct the inner workings of the reinforcement learning algorithm and implementation with ROS and Gazebo for CAT Vehicle usage
- Created world for testing reinforcement learning algorithm



Video Demonstration

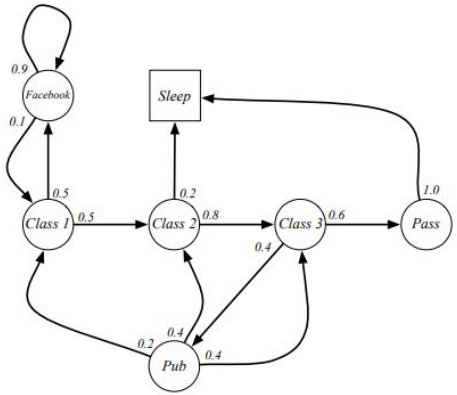


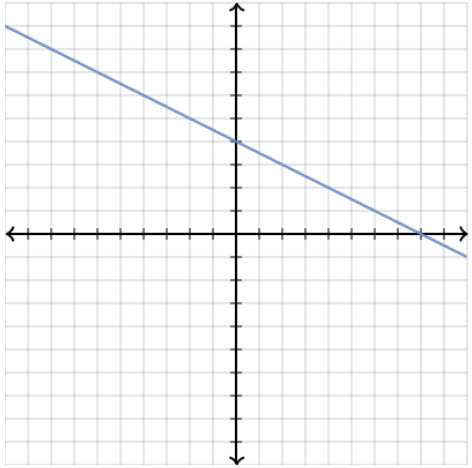
Video Demonstration



OpenAI Gym

- Pendulum Problem
- Continuous-Space
  - How can we make it finite?
    - NNs and Discretization
- https://github.com/openai/gym/wiki/Pendulum-v0





```
env = gym.make("Pendulum-v0")

def QLearning(env, learning, discount, epsilon, min_eps, episodes):
    list_of_rewards = []
    ave_reward_list = []

    obs_size = (env.observation_space.high - env.observation_space.low)
    obs_size = np.round(obs_size, 0).astype(int)+1

    action_size = (env.action_space.high - env.action_space.low)
    action_size = np.round(action_size, 0).astype(int)+1

    q = np.zeros(((obs_size[0] * obs_size[1] * obs_size[2]), action_size[0]))
    Q = np.random.uniform(low = -16, high = 0, size =(obs_size[0], obs_size[1], obs_size[2], action_size[0]))

    reduction = (epsilon - min_eps)/episodes

    # idk which q-table I want to use yet, we'll see

    for i in range(episodes): #define episodes later


        done = False
        state = env.reset()
        total_reward = 0
        action_array = []

        state_adj = (state - env.observation_space.low)
        state_adj = np.round(state_adj, 0).astype(int)


        while done!= True:
```

```
while done!= True:

    if i >= (episodes - 10):
        env.render()


    if np.random.random() < 1 - epsilon:
        action = np.argmax(Q[state_adj[0], state_adj[1], state_adj[2]])
    else:
        action = np.random.randint(0,5)

    if action == 0:
        action2 = -2.
    elif action == 1:
        action2 = -1.
    elif action == 2:
        action2 = 0.
    elif action == 3:
        action2 = 1.
    elif action == 4:
        action2 = 2.
    action_array.append(action2)

    state2, reward, done, info = env.step(action_array)



    state2_adj = (state2 - env.observation_space.low)
    state2_adj = np.round(state2_adj, 0).astype(int)

    if done: #this is the reward for if the Mountain Car went up successfully, probably need to change this later
        Q[state_adj[0], state_adj[1], state_adj[2], action] = reward

        # Adjust Q value for current state
    else:
        delta = learning*(reward + discount*np.max(Q[state2_adj[0], state2_adj[1], state2_adj[2]]) - Q[state_adj[0], state_adj[1], state_adj[2], action])

        Q[state_adj[0], state_adj[1], state_adj[2], action] += delta

    total_reward += reward
    state_adj = state2_adj
```

```
        if epsilon > min_eps:
            epsilon -= reduction

        list_of_rewards.append(total_reward)

        if (i+1) % 100 == 0:
            ave_reward = np.mean(list_of_rewards)
            ave_reward_list.append(ave_reward)
            reward_list = []

        if (i+1) % 100 == 0:
            print('Episode {} Average Reward: {}'.format(i+1, ave_reward))

    env.close()
    return ave_reward_list
```

```
>>> exec(open(r"C:\Users\njdom\Documents\openai\gym\Tutorial\Pendulum.py").read())
>>> rewards = QLearning(env, 0.2, 0.5, 0.8, 0, 500)
Episode 100 Average Reward: -1353.5795785797914
Episode 200 Average Reward: -1345.5482896884541
Episode 300 Average Reward: -1341.7599514040883
Episode 400 Average Reward: -1337.2446116417832
Episode 500 Average Reward: -1346.7466291453977
>>>
```

Observations

- Discretization isn’t always ideal
- Picking the right parameters is key
- Is there an easy way to create an environment in OpenAI Gym?
- What methods have been done for DCS in the past?

Future Plans

- Be able to subscribe and publish velocity, position, and lidar data topics as needed for testing simulations
- Continue deconstructing OpenAI example for our simple object detection simulation
- Define the CAT Vehicle Robot Environment and Object Detection Task Environment in OpenAI ROS.
- Research/Implement the methods of Q-Learning Used on Cognitive Radio Problems, as well as Environments used to train these models