Assignment – 1

Introduction:

The aim of this project is to develop a simulation of four wheeled robot and avoid objects while the robot is in motion. The simulation of the 3-D object has been created with Webots software and programmed with python programming language. The obstacle avoidance feature works via the real time data of the laser sensor which is placed in the front end of the robot.

Design and Development stage:

Step 1: Double-clicking on the icon launches Webots. By selecting wizards from the menu, a new project directory is created. By changing my project with Four_wheeled_robot, the project directory is named Four_wheeled_robot, and the world file is named Four_wheel_robot.wbt instead of the default empty.wbt. Furthermore, select the rectangle arena check box and hit the finish button to start designing the simulation.

Step 2: When the rectangle arena node option is clicked twice on the scene tree, the nodes and fields in it will be displayed. The size of the floor tile is chosen and modified from 0.5 to 0.25. As soon as the values are entered, the changes are visible in the 3D image of the rectangle arena. The value of the wall height field is altered from 0.1 to 0.05 after selecting it. The rectangle arena's wall height has been reduced.

Step 3: The '+' symbol is clicked to add a new node, and the base node offers several alternatives from which Robot is selected and added to the scene tree. When Robot is clicked on the scene tree, a number of nodes appear, among which Children is picked, the '+' symbol is clicked to add the base node, and form is selected under the base nodes options. The base node is added as PBR Appearance, the base colour is selected, and the values are modified to R=0, G=1, B=0 under shape. The colour of the robot changes in real time as the values are entered. The roughness is set to one, while the metalness values are set to zero. The base node is added as Box after clicking Geometry Null. When the size of the box is clicked under geometry Box, the dimensions of the box are adjusted to x=0, y=0.05, and z=0.2. Then, on the scene tree, Shape is selected, and the DEF name is set to body. Binding object Null is clicked, and Use is selected under Use body (Shape) is added to add a bounding object. The root node Physics is added after clicking Physics Null.

Step 4: The DEF is used to attach the wheels to the robot under the supervision of children. Hinge Joints is clicked and added when the body shape is selected in the basic nodes. Hinge joints are now visible in the scene tree, and when the null button for hinge joints is pressed, the base node is picked as the Hinge joint parameter and inserted. The base node is selected as a rotational motor and inserted, with the name "wheel1" given to the motor. The base node is selected as solid and added in end point Null. Children is clicked under end point null solid, and the base node is selected as shape and inserted. Under form, choose appearance null, then select PBR appearance for the base node and add it. The base node is selected as cylinder and added after clicking the geometry null. The dimensions of the geometry cylinder

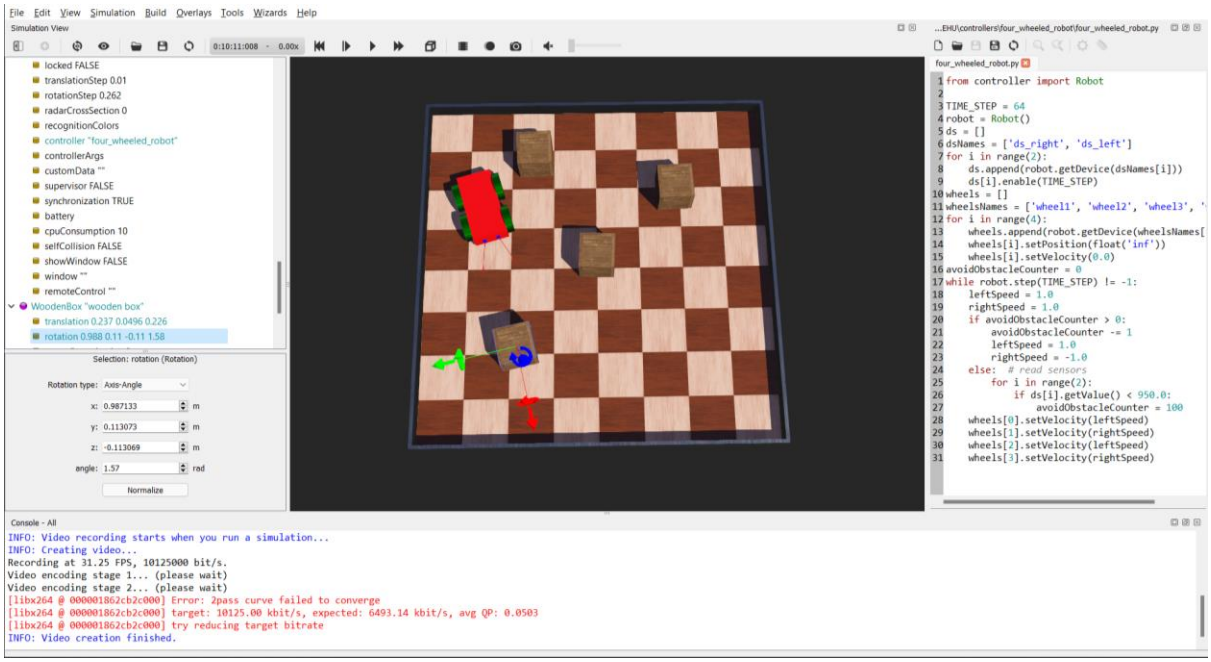
are modified, with the height being 0.02 instead of 0.1 and the radius being 0.04 instead of 0.05. The def is given as Wheel under shape, and it transforms to DEF Wheel shape. The bounding object null is clicked, and the wheel shape is selected and added under usage. In the scene tree, the Physics null is selected, and the base node is selected as Physics and added. Physics wh is the name given to DEF in physics. Now click x=0.06, y=0, z=0.05 under end point solid, and then click x=0, y=0, z=1, angle=1.57 under rotation. As a result, the robot's first wheel has been designed. To alter the colour of the wheel under PBR appearance, pick the base colour and input the numbers R=0, G=0, B=1, and the colour of the wheel is changed to green. The following steps are to be followed in the same way for the next three wheels, by clicking the hinge joint and the "+" symbol, and then repeating the procedure for the next three wheels. The anchor value, which is given under joint parameters, is clicked to make the robot's wheels travel in the right direction, and the values are given as x=0.045, y=0.025. This will assist the robot in moving in the correct direction.

Step 5: In the scene tree, click and add the laser sensor beneath robot hinge joint in the base nodes. The base node box is added after clicking on laser sensor children in the base nodes shape, selecting null in form geometry, then clicking on the base node box. The base node PBR appearance is then selected and inserted after clicking the appearance null. The base colour is adjusted to R=1, G=0, B=0 and the metalness is set to 0. The translation is x=0.02, y=0, z=0.1, and the rotation is x=0, y=1, z=0, and the angle=-1.27 in the PBR appearance. The size of the geometry box is modified to x=0.01, y=0.01, and z=0.01. The shape of the distance sensor is DEF sensor, and the name of the distance sensor is lr left. The bounding object null is picked, and the sensor (shape) is inserted, as well as the physics null and the base node physics. The same technique is followed to integrate another sensor to the robot's right. The first sensor is copied and pasted, and the same steps are followed. The name of the second sensor is changed to lr right, and the translation and rotation are supplied as x=-0.02, y=0, z=0.0, and angle=-1.87. Both sensors are constructed and integrated on the robot once these alterations are made to the second sensor. To see if the sensor is sensing, click the view option on the menu, select see optional rendering, and then select show laser sensor rays. The rays will appear immediately. In the scene tree, select Robot and then click the 'Plus' symbol on the menu. Many types of alternatives will appear in the find search bar type box; choose the wooden box and press add. Because the box appears to be quite large, its dimensions are altered by changing the size from 0.6 to 0. The size of the wooden box will immediately change because of this. The arrows are used to move the wooden box into the desired location.

Video Clip:

Https://drive.google.com/file/d/193epvxps0ecwlrdnvvmnump4_S_dek_m/view?Usp=sharing

Screenshots:



Code:

```
from controller import Robot                          | #importing libraries

TIME_STEP = 64
robot = Robot()                                       #creating an object called Robot
ds = []                                               #declaring an empty list
dsNames = ['ds_right', 'ds_left']                     #dsNames holds list value of sensor data such as ds_right value and ds_left value
for i in range(2):                                    #looping for 2 times
    ds.append(robot.getDevice(dsNames[i]))            #appending the names to the empty list ds
    ds[i].enable(TIME_STEP)
wheels = []                         #declaring an empty list
wheelsNames = ['wheel1', 'wheel2', 'wheel3', 'wheel4']    #the list wheelNames holds the wheel1, wheel2, wheel3, and wheel4
for i in range(4):                                    #looping for 4 times
    wheels.append(robot.getDevice(wheelsNames[i]))    #appending the wheel names to the empty list wheels
    wheels[i].setPosition(float('inf'))               #setting the positions of wheels
    wheels[i].setVelocity(0.0)                        #setting the velocity of wheels
avoidObstacleCounter = 0
while robot.step(TIME_STEP) != -1:                    #if the time step is not equal to -1 then do below
    leftSpeed = 1.0                                   #speed of left wheel value is 1.0
    rightSpeed = 1.0                                  #speed of right wheel value is 1.0
    if avoidObstacleCounter > 0:                      #if the value of obstacle counter is greater than 0 then do below
        avoidObstacleCounter -= 1                     setting avoid obstacle counter value = avoidobstaclecounter - 1
        leftSpeed = 1.0                               #speed of left wheel value is 1.0
        rightSpeed = -1.0                             #speed of right wheel value is -1.0
    else:  # read sensors                             #if the above condition does not satisfies then do below
        for i in range(2):                            #looping for 2 times
            if ds[i].getValue() < 950.0:              #if value of ds per each loop if less then 950.0 then do below
                avoidObstacleCounter = 100            #setting avoid obstacle counter value = 100
    wheels[0].setVelocity(leftSpeed)                  #speed of wheel 0 = leftspeed
    wheels[1].setVelocity(rightSpeed)                 #speed of wheel 1 = rightspeed
    wheels[2].setVelocity(leftSpeed)                  #speed of wheel 2 = leftspeed
    wheels[3].setVelocity(rightSpeed)                 #speed of wheel 3 = rightspeed
```

Human Robot Collaboration:

Collision avoidance is crucial in mobile robotics for the robots' performance in completing their jobs, especially when they manoeuvre in busy and dynamic situations with people. Traditional collision avoidance algorithms treat the human as a dynamic obstacle, ignoring the fact that the person will also strive to avoid the robot, leading to confusion among people and robots, particularly in dense social settings like laboratories, hospitals, and restaurants. A reactive-supervised collision avoidance system for mobile robots based on human-robot interaction is created to avoid such instances. Both the robot and the human will work together to generate collision avoidance via interaction in this technique. The person will

communicate with the robot to inform it of the avoidance direction, and the robot will search for the best collision-free path in that direction.

Conclusion:

To conclude that, the four-wheel robot has been designed and simulated using Webots software. The result has been added as a video file and shared a link above. Additionally, screenshot of the simulation has been added above.

Bibliography:

- Cyberbotics, Webots User Guide, Tutorial 6: 4-Wheeled Robot, https://cyberbotics.com/doc/guide/tutorial-6-4-wheels-robot

- Soft illusion, Making of Custom Robot | Webots Simulator | [Tutorial 6], (2020), https://www.youtube.com/watch?V=oyvitlwb8lk&ab_channel=Softillusion

Assignment - 2

Introduction:

The environment of the simulation created using Open Ai gym library. From the gym library, by calling the taxi-v3 environment, we can get the expected simulation environment for rendering. Initially, we are creating an array filled with zeros using NumPy library where we will be holding environment observation space and environment action space. After that, we are declaring the reward as 0 then resetting the state. Furthermore, we will be looping and getting the max of state value. Subsequently, we will be gathering values from action variable and storing that information in the respective variables such as, new state, reward, done and info then we will be applying the formula, increasing the reward value, and changing the state for rendering environment. Finally, if the percent of episode if higher then 50 then we will be printing number of episode and reward.

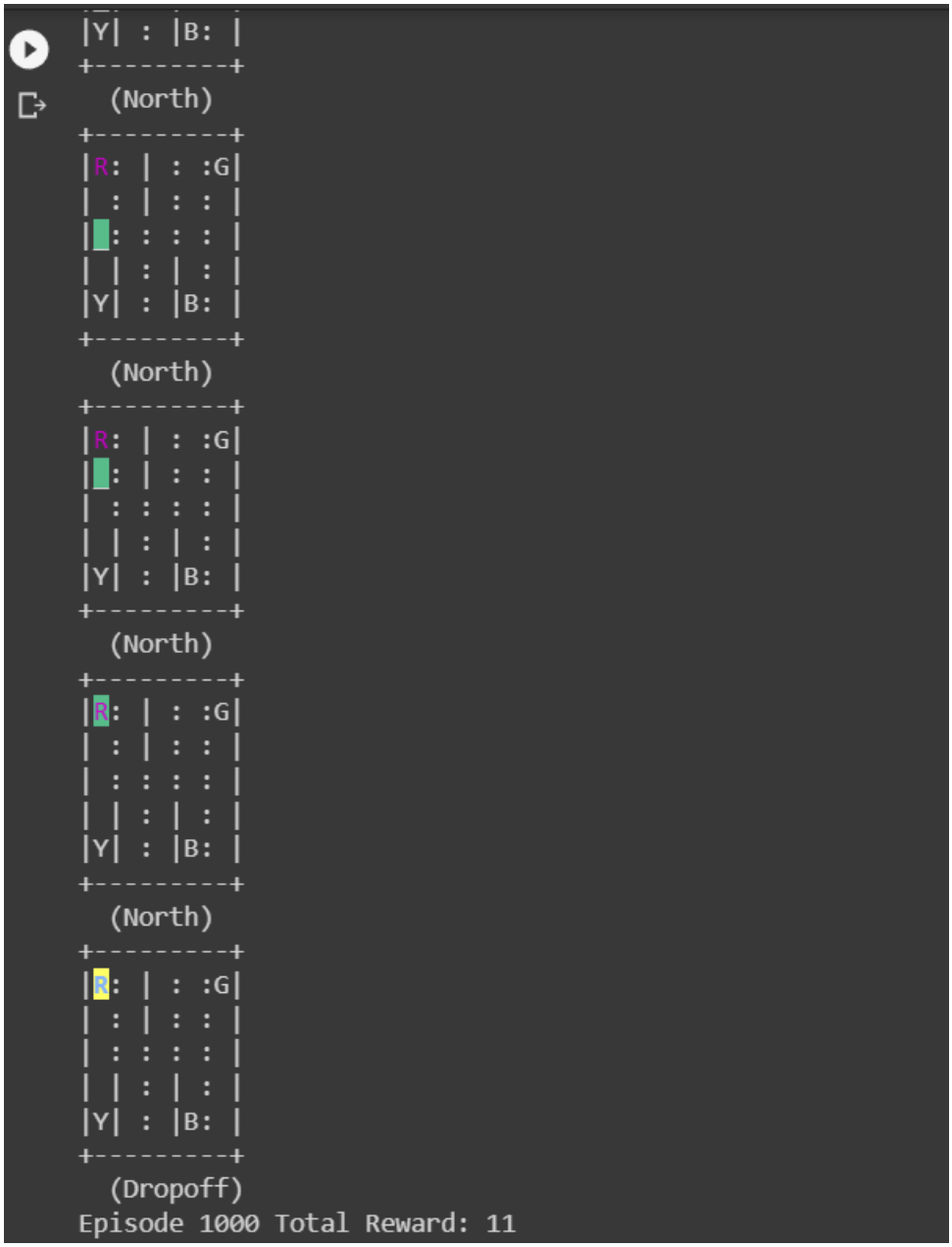
Code:

```
import gym              #importing libraries
import numpy as np

env = gym.make("Taxi-v3").env    #calling the environment
env.reset()            #resetting the environment
env.render()           #rendering the environment

Q = np.zeros([env.observation_space.n, env.action_space.n])       #creating zeros using numpy library.
G = 0
alpha= 0.618

for episode in range(1,1001):        #looping for 1000 episodes
  done = False
  G, reward = 0,0
  state = env.reset()                #resetting the state
  while done != True:
    action = np.argmax (Q[state])    #finding the maximum
    state2, reward, done, info = env.step(action)     #storing the state, reward, action, info from the variable action.
    Q[state, action] += alpha * (reward + np.max (Q[state2])- Q[state, action])      #updating the obtained result.
    G += reward        #increasing the reward, if the condition satisfies.
    state = state2     #changing the state
    env.render()       #rendering the environment
  if episode % 50 == 0:
    print('Episode {} Total Reward: {}'.format(episode, G))
```

Video Clip:

Https://drive.google.com/file/d/1j60xriveqflvrkjieebj9wglw3qwflj9/view?Usp=sharing

Screenshot:



Human-Robot Collaboration:

Reinforcement learning (RL) is one way to robot learning. In real life, the robot receives good or negative rewards from the environment according on how effectively it performs. It is carrying out a task. The purpose of the robot's learning is to improve system responses by increasing the value of a reward function. This is accomplished by obtaining knowledge and experience by interacting directly with the robot's surroundings when faced with ambiguity, it's possible that the robot won't be able to make the proper connections between the observable states and their causes selected activities that result in larger prizes Furthermore, certain issues are frequently overlooked. The values for each state are memory expensive to

store. Another downside of RL-based techniques is that they necessitate extensive contact to environment to evaluate a large number of state-action values before determining an appropriate strategy. To get around this challenge, one solution is to forgo saving every state-action pairings and alternate processing them dynamically as needed. One of RL's key drawbacks is its sluggish convergence to satisfactory solutions.

Another approach based on Confidence-Based Autonomy (CBA) that allows an RL agent to understand a strategy through contact with a human trainer. CBA is made up of two parts that take advantage of the human and computer agents' complementary strengths. Confident Execution is the initial component, and it allows the agent to understand a policy based on representations gained by restricting its autonomy and asking assistance from the trainer. illustrations are chosen which depend on categorization confidence thresholds that are automatically calculated. Corrective Demonstration, the second component, allows the teacher to enhance the learnt policy by cross checking and setting right the agent's errors through supplemental representation. The method was tested in a difficult simulated driving domain, and the results reveal that when strong Execution is used, the agent looks for fewer exhibit to understand the strategy than when representations are chosen by a human trainer. Two methods are used to calculate the confidence threshold: (1) a single fixed threshold and (2) multiple changeable thresholds. A straightforward approach for approximating the high confidence areas of the state-space is to use a single set confidence threshold value. It does, however, lead to the agent requesting far few illustrations of what it already knows and far too few instances of unlearned actions. One can choose between human demonstration and autonomy using a multi-threshold technique in which latest points are classified by first choosing the action class with the maximum surety for the question. The CBA algorithm has been shown to be extremely effective in a range of single-robot and multi-robot task.

Many robotic applications have used the Reinforcement learning based learning algorithm Q-learning, as well as its version Q (), an incremental multi-step Q-learning algorithm that merges one-step Q-learning with eligibility traces (e.g., [1, 4, 5, 40]). Because the learning algorithms such as Q and Q () cannot accept human interference, the agent is situated in an unknown environment and is left to explore it on its own to find the best policy. The main disadvantage of these approaches is that they require a lot of reciprocation between the robot and the environment until a good strategy is found.

This work proposes a collaborative Q () RL method, designated as CQ (), to speed up learning of the Q () algorithm. The collaborative algorithm combines a robot's and a human's knowledge. The work makes two contributions: (1) it provides a threshold dependent way for a robot to seek assistance from a human adviser (HA), and (2) it uses this method to a bag-shaking job where a robot should learn a way to shake a bag to release a knot tying it and therefore the contents inside. For the shaking task, two basic techniques for a person to involve oneself in the learning process are addressed. The primary thing is to award the prize, and then advise on which policy to choose. The other option was chosen as a straight way to assist the robot in learning the appropriate policy. For the HA, "reward intervention" may be used, but for the particular task, a highly accurate way was available, which was mechanised by counting the objects extracted from the bag using a digital scale as a sensor. This study is

also unique in that it addresses human-robot collaboration by permitting a human to actively change Q values using a linguistic-based interface.

Sophisticated algorithms like,

- Deep Q-Learning,
- Deep Deterministic Policy Gradient,
- A3C,
- TRPO,
- PPO can be used to enhance the model with human robot collaboration.

Conclusion:

In a nutshell, the Q-learning agent (taxi) can pick and drop the passenger as short as possible. The environment created and tested via OpenAi gym library. The human robot interaction can achieve better results by following the techniques specified above. The result achieved with the program specified. Output video clip and screenshot has been provided with the appropriate links. The entire setup has been executed with the help of Google Collaboratory.

Bibliography:

- Thomas Simonini, Q-Learning, let's create an autonomous Taxi 🚖 (Part 1/2), (2020), https://thomassimonini.medium.com/q-learning-lets-create-an-autonomous-taxi-part-1-2-3e8f5e764358
- Uri Kartoun · Helman Stern · Yael Edan, A Human-Robot Collaborative Reinforcement Learning Algorithm, J Intell Robot Syst, 2009, https://scholar.harvard.edu/files/kartoun/files/1ca003_83c9850ad8cde33e4cae5bb72077a132.pdf