Assessment2

April 28, 2025

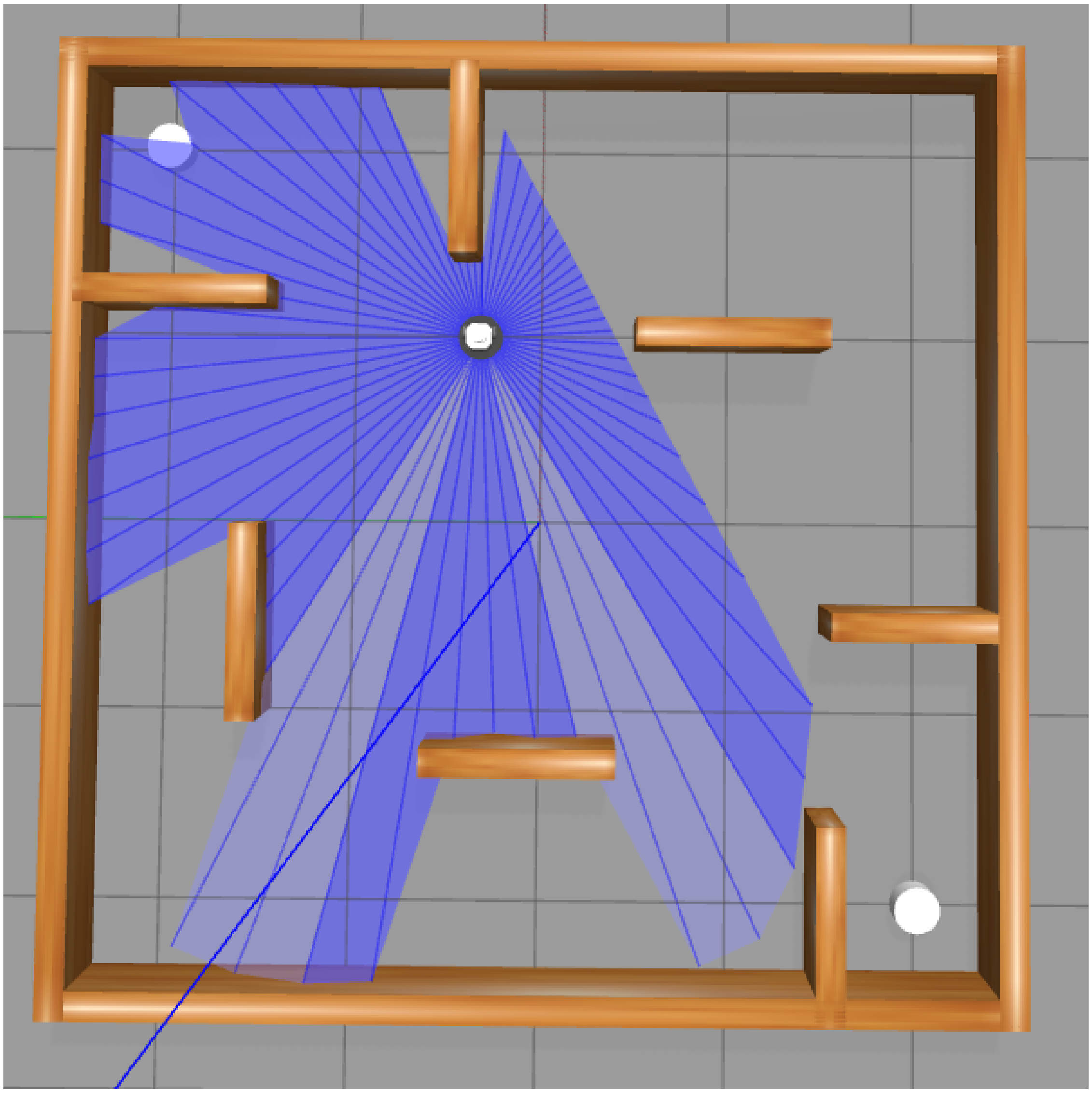
1 Summary and Introduction

This assignment requires the use of Reinforcement Learning to build an agent which controls a TurtleBot3 to reach a goal in the fewest steps possible in a simulated indoor environment. Two robot navigation models must be created a linear model, and a non-linear model.

The main components are the environment robot.py and robot_environment.py, and the the agent eg Sarsa or DQN (see Assessment2.ipynb). The environment provides the reward and state, and the agent learns the policy to pick the best set of actions.

```
RL/
   env/
      grid.py
      gridln.py
      gridnn.py
      mountainln.py
      robot.py           ← Gazebo interface and Environment
      robot_old.py
   rl/
      dp.py              ← Dynamic programming
      rl.py              ← Core RL logic
      rlln.py            ← Linear approximation model
      rlnn.py            ← Non-linear model
      rlselect.py        ← "Runs" code for running experimental trials comparing

   Assessment2.ipynb      ← This code part of submission
   robot_environment.py   ← vRobEnv includes state representation and reward structure

   remote_control.py      ← Allows driving the robot around using wasd and the step function in
   robenv_monitor.py      ← allows you to monitor the "reward" and "s_" of the environment as i
   feature_monitor.py     ← used for testing hand crafted features
```

The map isn't simple, there are maze like features. My intuition was that it should hug walls to the goal, but unfortuneately I didn't get a useful policy.

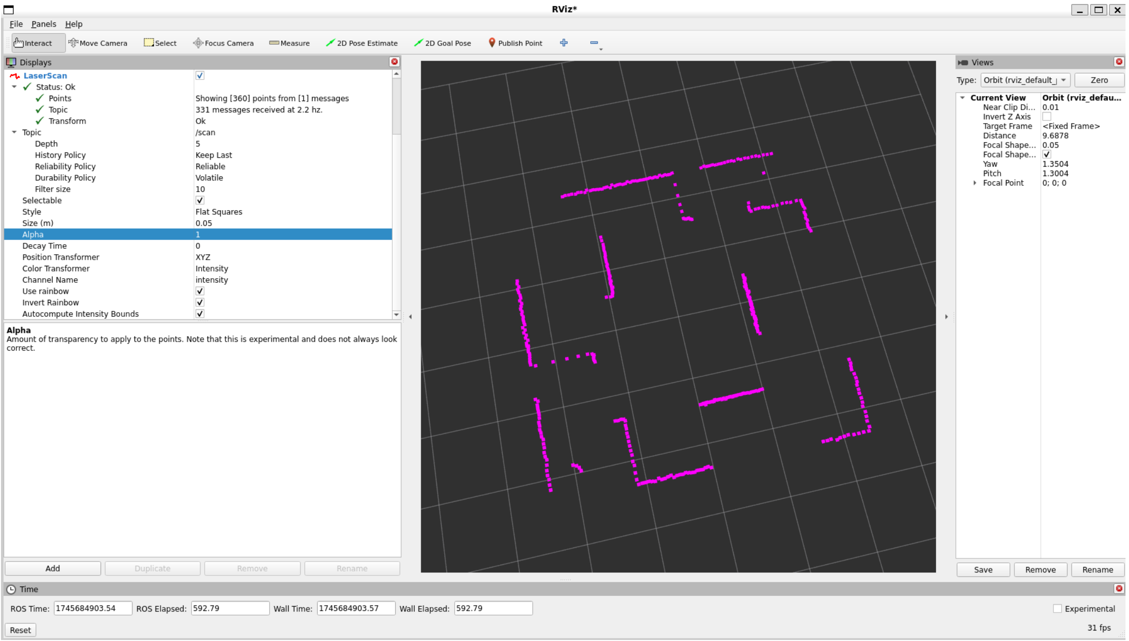


2 ROS Environment

2.1 State Messages

Gazebo runs the simulation of the Robot in it's virtual world. In ROS all communication is message based, so /odom and /scan data are broadbast and "nodes" can subscribe to the messages. This lets you listen to the messages and view them without influencing the system (ie the robot does not know or care who is observing it).

RVIZ and Topic Inspector are two inbuilt tools that let you view published messages



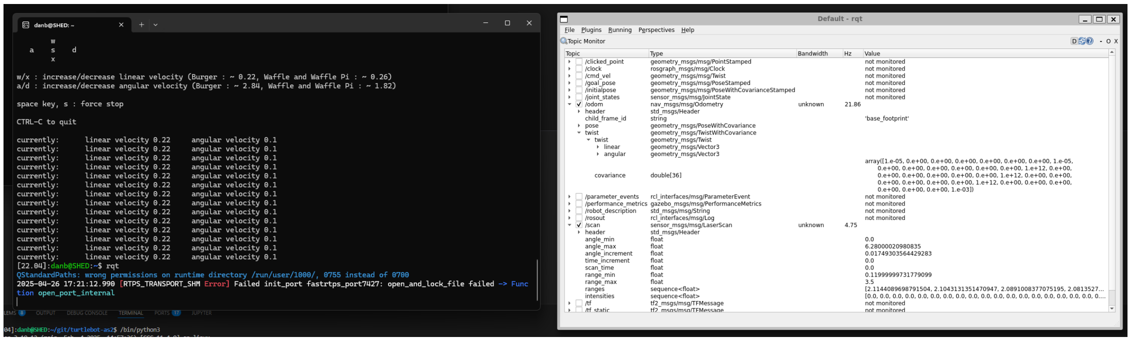
LaserScan	☑
Status: Ok	
Points	Showing [360] points from [1] messages
Topic	331 messages received at 2.2 hz.
Transform	Ok
Topic	/scan
Depth	5
History Policy	Keep Last
Reliability Policy	Reliable
Durability Policy	Volatile
Filter size	10
Selectable	☑
Style	Flat Squares
Size (m)	0.05
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	Intensity
Channel Name	intensity
Use rainbow	☑
Invert Rainbow	☑
Autocompute Intensity Bounds	☑

Alpha
Amount of transparency to apply to the points. Note that this is experimental and does not always look correct.

Type: Orbit (rviz_default_

Current View	**Orbit (rviz_defau...**
Near Clip Di...	0.01
Invert Z Axis	□
Target Frame	<Fixed Frame>
Distance	9.6878
Focal Shape...	0.05
Focal Shape...	☑
Yaw	1.3504
Pitch	1.3004
Focal Point	0; 0; 0

ROS Time: 1745684903.54 ROS Elapsed: 592.79 Wall Time: 1745684903.57 Wall Elapsed: 592.79



```
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)

space key, s : force stop

CTRL-C to quit

currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
currently:      linear velocity 0.22     angular velocity 0.1
[22.04]:danb@SHED:~$ rqt
QStandardPaths: wrong permissions on runtime directory /run/user/1000/, 0755 instead of 0700
2025-04-26 17:21:12.990 [RTPS_TRANSPORT_SHM Error] Failed init_port fastrtps_port7427: open_and_lock_file failed -> Func
tion open_port_internal
```

```
04]:danb@SHED:~/git/turtlebot-as2$ /bin/python3
```

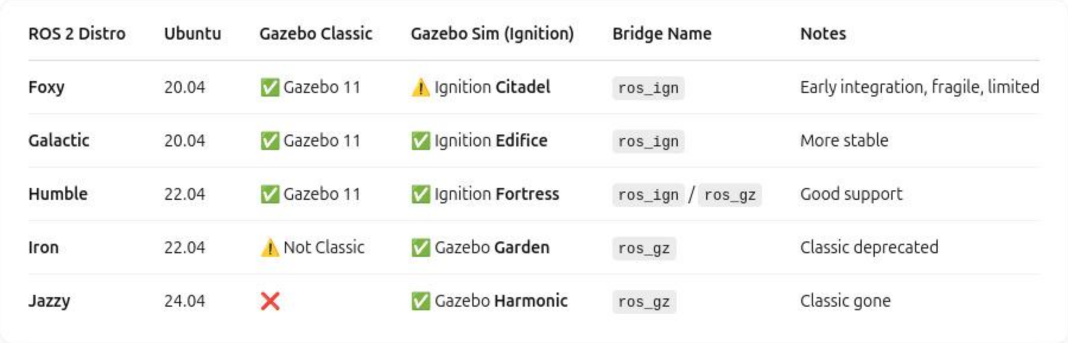
Topic	Type	Bandwidth	Hz	Value
□ /clicked_point	geometry_msgs/msg/PointStamped			not monitored
□ /clock	rosgraph_msgs/msg/Clock			not monitored
□ /cmd_vel	geometry_msgs/msg/Twist			not monitored
□ /goal_pose	geometry_msgs/msg/PoseStamped			not monitored
□ /initialpose	geometry_msgs/msg/PoseWithCovarianceStamped			not monitored
□ /joint_states	sensor_msgs/msg/JointState			not monitored
☑ /odom	nav_msgs/msg/Odometry	unknown	21.86	
header	std_msgs/Header			
child_frame_id	string			'base_footprint'
pose	geometry_msgs/PoseWithCovariance			
twist	geometry_msgs/TwistWithCovariance			
twist	geometry_msgs/Twist			
linear	geometry_msgs/Vector3			
angular	geometry_msgs/Vector3			
covariance	double[36]			array([1.e-05, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e-05, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e+12, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e+12, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e+12, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 0.e+00, 1.e-03])
□ /parameter_events	rcl_interfaces/msg/ParameterEvent			not monitored
□ /performance_metrics	gazebo_msgs/msg/PerformanceMetrics			not monitored
□ /robot_description	std_msgs/msg/String			not monitored
□ /rosout	rcl_interfaces/msg/Log			not monitored
☑ /scan	sensor_msgs/msg/LaserScan	unknown	4.75	
header	std_msgs/Header			
angle_min	float			0.0
angle_max	float			6.28000020980835
angle_increment	float			0.01749303564429283
time_increment	float			0.0
scan_time	float			0.0
range_min	float			0.11999999731779099
range_max	float			3.5
ranges	sequence<float>			[2.1144089698791504, 2.1043131351470947, 2.0891008377075195, 2.0813527...
intensities	sequence<float>			[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0....
□ /tf	tf2_msgs/msg/TFMessage			not monitored
□ /tf_static	tf2_msgs/msg/TFMessage			not monitored

2.2 Command messages

To sent commands to the robot you publish messages to a topic, and the robot listens and processes them. There are standard programs which are part of ROS for sending twist messages to the robot from the keyboard and joystick (ros2 run turtlebot3_teleop teleop_keyboard)

2.3 Computer setup

I originally tried to setup Jazzy on 24.04 but this only works with Gazebo Sim (Ignition) and that needs different model types to the old Gazebo Classic. I then used 22.04 Humble. This worked very well.



ROS 2 Distro	Ubuntu	Gazebo Classic	Gazebo Sim (Ignition)	Bridge Name	Notes
Foxy	20.04	✅ Gazebo 11	⚠ Ignition **Citadel**	`ros_ign`	Early integration, fragile, limited
Galactic	20.04	✅ Gazebo 11	✅ Ignition **Edifice**	`ros_ign`	More stable
Humble	22.04	✅ Gazebo 11	✅ Ignition **Fortress**	`ros_ign` / `ros_gz`	Good support
Iron	22.04	⚠ Not Classic	✅ Gazebo **Garden**	`ros_gz`	Classic deprecated
Jazzy	24.04	❌	✅ Gazebo **Harmonic**	`ros_gz`	Classic gone

2.4 Environment File

`-RobEnv`: A ROS2 node that connects a TurtleBot robot with a Gazebo simulation. It manages robot control, sensor readings, and interactions within the environment. `-vRobEnv(RobEnv)::` Specialisations for vectorised state representation

`-odom`: Processes odometry data, updating the robot's position (`x, y`) and orientation ( ).
`-scan`: Reads laser scan data, replacing infinite values with the maximum sensor range.
`-yaw`: Converts quaternion orientation into a yaw angle in radians.

`- goal`: Computes the angular distance between the robot and a goal.
`-distgoal`: Calculates the Euclidean distance to the closest goal.
`-atgoal`: Checks if the robot has reached a goal.
`-atwall`: Detects potential collisions based on laser scan data.

`-reward`: Assigns a reward based on the robot's state and action. It encourages movement towards goals and penalises collisions or undesired actions. If the robot collides with a wall, it automatically resets the environment.

`-s_`: Converts the robot's real-world position and orientation into a discrete state representation, mapping it to a grid system.

`-spin_n`: Ensures the node updates by calling `ros.spin_once` multiple times.
`-control`: Publishes movement commands.
`-step`: Executes a specified action (`forward, turn left, turn right`), computes the next state, and returns a reward.
`-stop`: Halts all movement.

`-reset`: Restarts the simulation, ensuring a fresh environment for new episodes.

2.5 Note on step wise architecture

The setup is designed in a "step" wise fashion, so that the robot moves forward and then stops, this is so that it matches the techniques we have learnt in the module. I'd be very interested in knowing if we can use RL techniques for continuous control where we can leave the turtlebot moving and not have to stop it after each "step".

3 Imports / Constants

[2]:

```python
%matplotlib inline
import torch
import numpy as np
from env.robot import *
import numpy as np
from math import pi
from time import sleep
#from tqdm import tqdm

from tqdm.notebook import tqdm
import sys
import termios
import tty
import select
import ipywidgets as widgets
from IPython.display import display
from IPython.display import clear_output
import time
import matplotlib.pyplot as plt
import numpy as np

from robot_environment import *
from rl.rlnn import *

#ACTIONS - make the code easier to read
FORWARDS = 1
LEFT = 0
RIGHT = 2

#Really want it deterministic
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

#GPU support?
print(torch.cuda.is_available())
if torch.cuda.is_available():
    print(torch.cuda.get_device_name(0))
```

```
<IPython.core.display.HTML object>
```

```
True
```

```
NVIDIA GeForce RTX 3080 Ti
```

[ ]:

```
#Need to nuke write protection for these files
#sudo chmod 777 /opt/ros/humble/share/turtlebot3_gazebo/worlds/
 ↪turtlebot3_assessment2/burger.model
#sudo chmod 777 /opt/ros/humble/share/turtlebot3_gazebo/models/
 ↪turtlebot3_burger/model.sdf

accelerate_sim(speed=100)
set_nscans_LiDAR(nscans=64)

#Note restart gazebo after changing these
```

3.0.1 Common / Helper Functions

[5]:

```
def print_robot_odom(env: RobEnv):
    #note: env.x and env.y are rounded to 1dp
    print (f"Odom. Pos:[{env.x},{env.y}] Yaw:{env. }")
```

4 Connect to ROS / Configure Environment

1) Launch simulation environment
2) init ros (connect to ROS DDS eventing)
3) create environment

[3]:

```
# Start Gazebo
# Assessment world:
# ros2 launch turtlebot3_gazebo turtlebot3_assessment2.launch.py

# Other worlds
# ros2 launch turtlebot3_gazebo turtlebot3_simple.launch.py
# ros2 launch turtlebot3_gazebo empty_world.launch.py
# ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
# ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
# rviz2

if not ros.ok():
    ros.init()

nscans = get_nscans_LiDAR()
print(f"Num laser scans in sensor:'{nscans}'")
# set_nscans_LiDAR(nscans=64)

# accelerate_sim(speed=10)
# Note running on my machine (AMD Ryzen 9 7950X, 64GB RAM, 3080TI) I got a␣
 ↪stable Real Time Factor of 1.00 in the simulation.
```

```
# I had problems if I set the real time target to more than this, and from␣
 ↪reading up this is a known limitation in the old Gazebo environment
# the new Gazebo Sim properly abstracts time so there is a simulation time␣
 ↪independent from wall clock time. This version doesn't play nicely unless␣
 ↪it's 1:1
# See Simulation Speed in ROS/Gazebo
```

```
Num laser scans in sensor:'64'
```

[7]:

```
test_connection = False
if test_connection:
     speed = pi/3.5
    speed = 10.0
    n = 6

    env = RobEnv(speed=speed,  speed= speed, n=n, verbose=True)
    env.reset()
    print_robot_odom(env)

    for _ in range(10): env.step()

    print_robot_odom(env)

    env.reset()
```

5 Calibrate Speed and speed

The following functions were used to sweep through different settings for speed and speed until the robot number of degrees and actual movememt got unreliable.

[8]:

```
def forward_test(env, speed=2.0):
    env.speed = speed
    env.reset()

    steps = 10
    for _ in tqdm(range(steps), desc=f"Moving at speed {speed}", leave=False):
        env.step(FORWARDS)

    yDriftPerStep = env.y / steps
    xPerStep = env.x / steps
    #print(f"Speed {speed}: y drift = {yDriftPerStep:.4f}, x per step:␣
 ↪{xPerStep:.4f}")

    BACKWARDS = -1 #not used in simulation, just used to test repeatability
    for _ in tqdm(range(steps), desc=f"Moving at speed {speed}", leave=False):
        env.step(BACKWARDS)
    xDrift = env.x

    return yDriftPerStep, xPerStep, xDrift

num_trials = 5
speeds = [0.5, 1.0, 2.0, 3.0,5.0, 10.0]
log = {"speed": [], "trial": [], "yDriftPerStep": [], "xPerStep": [], "xDrift": []}

for speed in tqdm(speeds, desc="Speeds"):
    for trial in tqdm(range(num_trials), desc=f"Trials at speed {speed}", leave=False):
        y_drift, x_step, xDrift = forward_test(env, speed=speed)
        log["speed"].append(speed)
        log["trial"].append(trial)
        log["yDriftPerStep"].append(y_drift)
        log["xPerStep"].append(x_step)
        log["xDrift"].append(xDrift)


plt.figure(figsize=(15, 5))

# Plot y drift per step
plt.subplot(1, 3, 1)
for speed in speeds:
    y_drift_vals = [log["yDriftPerStep"][i] for i in range(len(log["speed"])) if log["speed"][i] == speed]
    plt.plot(range(num_trials), y_drift_vals, marker='o', label=f"Speed {speed}")

plt.title('Y Drift per Step across Trials')
plt.xlabel('Trial')
plt.ylabel('Y Drift per Step')
plt.legend()
plt.grid(True)

# Plot x movement per step
plt.subplot(1, 3, 2)
for speed in speeds:
    x_step_vals = [log["xPerStep"][i] for i in range(len(log["speed"])) if log["speed"][i] == speed]
    plt.plot(range(num_trials), x_step_vals, marker='o', label=f"Speed {speed}")
plt.title('X Movement per Step across Trials')
plt.xlabel('Trial')
plt.ylabel('X per Step')
plt.legend()
plt.grid(True)

# Plot total x drift
plt.subplot(1, 3, 3)
for speed in speeds:
    x_drift_vals = [log["xDrift"][i] for i in range(len(log["speed"])) if␣
 ↪log["speed"][i] == speed]
    plt.plot(range(num_trials), x_drift_vals, marker='o', label=f"Speed␣
 ↪{speed}")
plt.title('Total X Drift across Trials')
plt.xlabel('Trial')
plt.ylabel('Total X Drift')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

display(log)
```

```
Speeds:   0%|          | 0/6 [00:00<?, ?it/s]
```

```
Speeds:  17%|         | 1/6 [00:01<00:07,  1.55s/it]
```

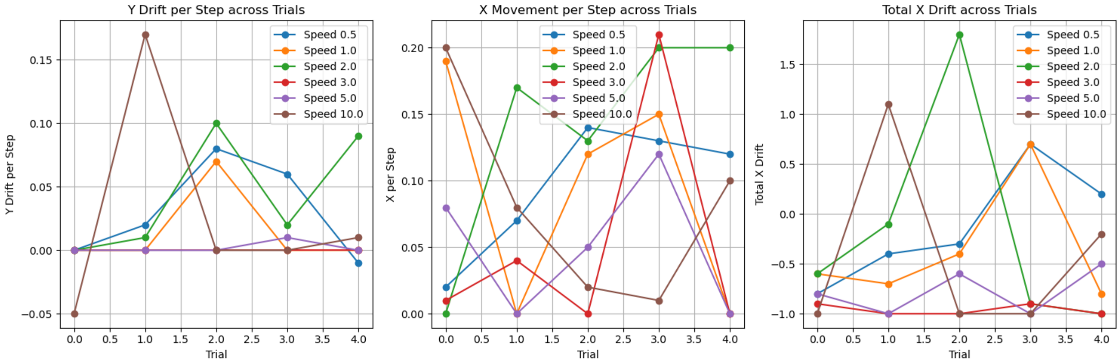
```
Speeds:  33%|        | 2/6 [00:03<00:06,  1.56s/it]
```

```
Speeds:  50%|        | 3/6 [00:04<00:04,  1.53s/it]
```

```
Speeds:  67%|       | 4/6 [00:06<00:03,  1.54s/it]
```

```
Speeds:  83%|      | 5/6 [00:07<00:01,  1.55s/it]
```

```
Speeds: 100%|      | 6/6 [00:09<00:00,  1.54s/it]
```

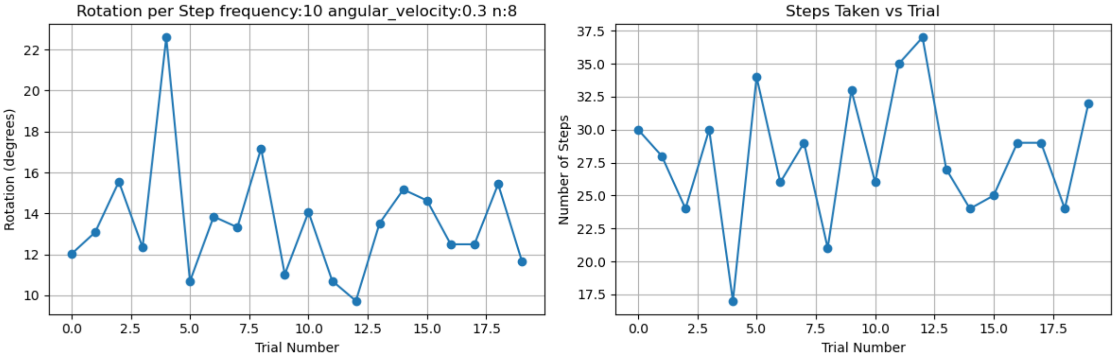


```
{'speed': [0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  1.0,
  1.0,
  1.0,
  1.0,
  1.0,
  2.0,
  2.0,
  2.0,
  2.0,
  2.0,
  3.0,
  3.0,
  3.0,
  3.0,
  3.0,
  5.0,
  5.0,
  5.0,
  5.0,
  5.0,
  10.0,
  10.0,
  10.0,
  10.0,
  10.0],
 'trial': [0,
  1,
  2,
  3,
  4,
  0,
  1,
  2,
  3,
  4,
  0,
  1,
  2,
  3,
  4,
  0,
  1,
  2,
  3,
  4,
  0,
  1,
  2,
  3,
  4,
  0,
  1,
  2,
  3,
  4],
 'yDriftPerStep': [0.0,
  0.02,
  0.08,
  0.06,
  -0.01,
  -0.0,
  0.0,
  0.06999999999999999,
  0.0,
  -0.0,
  0.0,
 0.01,
 0.1,
 0.02,
 0.09,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 -0.0,
 -0.0,
 0.01,
 0.0,
 -0.05,
 0.16999999999999998,
 0.0,
 0.0,
 0.01],
'xPerStep': [0.02,
 0.06999999999999999,
 0.13999999999999999,
 0.13,
 0.12,
 0.19,
 0.0,
 0.12,
 0.15,
 0.0,
 0.0,
 0.16999999999999998,
 0.13,
 0.2,
 0.2,
 0.01,
 0.04,
 -0.0,
 0.21000000000000002,
 0.0,
 0.08,
 -0.0,
 0.05,
 0.12,
 0.0,
 0.2,
 0.08,
 0.02,
 0.01,
 0.1],
'xDrift': [-0.8,
 -0.4,
 -0.3,
 0.7,
 0.2,
 -0.6,
 -0.7,
 -0.4,
 0.7,
 -0.8,
 -0.6,
 -0.1,
 1.8,
 -0.9,
 -1.0,
 -0.9,
 -1.0,
 -1.0,
 -0.9,
 -1.0,
 -0.8,
 -1.0,
 -0.6,
 -1.0,
 -0.5,
 -1.0,
 1.1,
 -1.0,
 -1.0,
 -0.2]}
```

[9]:

```python
def measure_rotation(robot : RobEnv, nTrials=20, frequency=10,␣
 ↪angular_velocity=0.3, nSteps=8):

    robot.freq = frequency
    robot.angular_velocity = angular_velocity
    robot.n = nSteps

    results  = np.zeros((nTrials, 2))
    start_time = time.time()

    for i in range(nTrials):
        robot.reset()

        step_count = 0
        full_rotation_detected = False
        half_rotation_detected = False

        pbar = tqdm(desc="Calibrating Rotation", unit="step")
        while(not full_rotation_detected):

            step_count += 1
            robot.step(LEFT)

            pbar.set_postfix({
                "rotation (°)": f"{np.degrees(robot. ):.2f}",
                "step": step_count,
                "elapsed (s)": f"{time.time() - start_time:.1f}"
            })

            if (robot.  > pi):
                half_rotation_detected = True
            if (robot.  <pi and half_rotation_detected):
                full_rotation_detected = True

        #print(f"step_count:{step_count}: final position: {np.degrees(robot. ):.
 ↪2f} degrees")
        results [i,0] = robot.
        results [i,1] = step_count

    return results

#Let's try to find the best parameters for the rotation
frequencies = [10,20,30]
angular_velocities = [0.3, 0.6, 0.9]
nSteps = [8,16,32]

for frequency in frequencies:
    for angular_velocity in angular_velocities:
        for n in nSteps:
            results = measure_rotation(env, nTrials=20, frequency=frequency,␣
 ↪angular_velocity=angular_velocity, nSteps=n)
            angle_per_turn = (2 * np.pi + results[:, 0]) / results[:, 1]
            results = np.hstack([results, angle_per_turn[:, None]])

            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

            # Plot final errors
            ax1.plot(np.degrees(results[:,2]), 'o-')
            ax1.set_title(f'Rotation per Step frequency:{frequency}␣
 ↪angular_velocity:{angular_velocity} n:{n}')
            ax1.set_xlabel('Trial Number')
            ax1.set_ylabel('Rotation (degrees)')
            ax1.grid(True)

            # Plot step counts
            ax2.plot(results[:,1], 'o-')
            ax2.set_title('Steps Taken vs Trial')
            ax2.set_xlabel('Trial Number')
            ax2.set_ylabel('Number of Steps')
            ax2.grid(True)

            plt.tight_layout()
            plt.
↪savefig(f'plot_measure_rotation_{frequency}_{angular_velocity}_{n}.png')
            plt.show()
```

```
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=0.57, step=30, elapsed
(s)=0.4]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=6.30, step=28, elapsed
(s)=0.9]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=13.18, step=24,
elapsed (s)=1.2]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=10.31, step=30,
elapsed (s)=1.7]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=24.06, step=17,
elapsed (s)=2.0]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=3.44, step=34, elapsed
(s)=2.5]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=0.00, step=26, elapsed
(s)=2.9]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=26.36, step=29,
elapsed (s)=3.4]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=0.57, step=21, elapsed
(s)=3.7]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=4.01, step=33, elapsed
(s)=4.3]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=5.73, step=26, elapsed
(s)=4.6]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=14.90, step=35,
elapsed (s)=5.2]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=0.00, step=37, elapsed
(s)=5.8]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=5.16, step=27, elapsed
(s)=6.2]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=4.01, step=24, elapsed
(s)=6.6]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=5.73, step=25, elapsed
(s)=7.0]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=2.29, step=29, elapsed
(s)=7.4]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=2.29, step=29, elapsed
(s)=7.9]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=10.31, step=24,
elapsed (s)=8.3]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=13.75, step=32,
elapsed (s)=8.8]
```



```
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=30.94, step=12,
elapsed (s)=0.4]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=22.35, step=10,
elapsed (s)=0.7]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=48.13, step=10,
elapsed (s)=1.0]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=9.17, step=13, elapsed
(s)=1.4]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=41.25, step=15,
elapsed (s)=1.8]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=23.49, step=12,
elapsed (s)=2.2]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=6.88, step=11, elapsed
(s)=2.5]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=18.33, step=13,
elapsed (s)=2.9]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=18.91, step=10,
elapsed (s)=3.2]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=36.10, step=12,
elapsed (s)=3.6]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=28.65, step=15,
elapsed (s)=4.1]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=15.47, step=9, elapsed
(s)=4.4]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=31.51, step=10,
elapsed (s)=4.6]
```

```
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=21.20, step=13,
elapsed (s)=5.1]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=14.32, step=18,
elapsed (s)=5.6]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=20.63, step=12,
elapsed (s)=6.0]
Calibrating Rotation: 0step [00:00, ?step/s, rotation (°)=312.83, step=15,
elapsed (s)=6.4]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[9], line 48
     46 for angular_velocity in angular_velocities:
     47     for n in nSteps:
---> 48         results =␣
  ↪measure_rotation(env, nTrials=20, frequency=frequency, angular_velocity=angular_velocity, r
     49         angle_per_turn = (2 * np.pi + results[:, 0]) / results[:, 1]
     50         results = np.hstack([results, angle_per_turn[:, None]])

Cell In[9], line 21, in measure_rotation(robot, nTrials, frequency,␣
  ↪angular_velocity, nSteps)
     18 while(not full_rotation_detected):
     20     step_count += 1
---> 21     robot.step(LEFT)
     23     pbar.set_postfix({
     24         "rotation (°)": f"{np.degrees(robot. ):.2f}",
     25         "step": step_count,
     26         "elapsed (s)": f"{time.time() - start_time:.1f}"
     27     })
     29     if (robot.  > pi):

File ~/git/turtlebot-as2/env/robot.py:223, in RobEnv.step(self, a, speed,  speed
    219 elif a == 2: self.robot.angular.z = - speed  # turn right
    221 # try:
    222 # Now move and stop so that we can have a well defined actions
--> 223 self.spin_n(self.n if a==1 else self.n-1)
    224 self.stop()
    225 # except KeyboardInterrupt:
    226 #     print("Execution interrupted by user. Cleaning up…")

File ~/git/turtlebot-as2/env/robot.py:205, in RobEnv.spin_n(self, n)
    203 for _ in range(n):
    204     self.controller.publish(self.robot)
--> 205     ros.spin_once(self)
    206     if self.sleep: time.sleep(1.0 / 30)
```

```
File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/__init__.py:206, in spin_once(node, executor, timeout_sec)
    204 try:
    205     executor.add_node(node)
--> 206     executor.spin_once(timeout_sec=timeout_sec)
    207 finally:
    208     executor.remove_node(node)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:751, in SingleThreadedExecutor.spin_once(self, timeout_sec)
    750 def spin_once(self, timeout_sec: float = None) -> None:
--> 751     self._spin_once_impl(timeout_sec)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:740, in SingleThreadedExecutor._spin_once_impl(self, timeout_sec)
    735 def _spin_once_impl(
    736     self,
    737     timeout_sec: Optional[Union[float, TimeoutObject]] = None
    738 ) -> None:
    739     try:
--> 740         handler, entity, node = self.wait_for_ready_callbacks(timeout_sec=timeout_sec)
    741     except ShutdownException:
    742         pass

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:723, in Executor.wait_for_ready_callbacks(self, *args, **kwargs)
    720     self._cb_iter = self._wait_for_ready_callbacks(*args, **kwargs)
    722 try:
--> 723     return next(self._cb_iter)
    724 except StopIteration:
    725     # Generator ran out of work
    726     self._cb_iter = None

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:620, in Executor._wait_for_ready_callbacks(self, timeout_sec, nodes, condition)
    617     waitable.add_to_wait_set(wait_set)
    619 # Wait for something to become ready
--> 620 wait_set.wait(timeout_nsec)
    621 if self._is_shutdown:
    622     raise ShutdownException()

KeyboardInterrupt:
```

6 Model 1: Action-value with linear function approximation

6.1 RL method explanation + justification

rlln.py contains a agents which are suitable for linear control

Initially Qlearn(vMDP) was tried but it lead to what seemed like very random behaviour

Sarsa was then picked because this is a "ON POLICY" method, which means it is more conservative and a safer option.

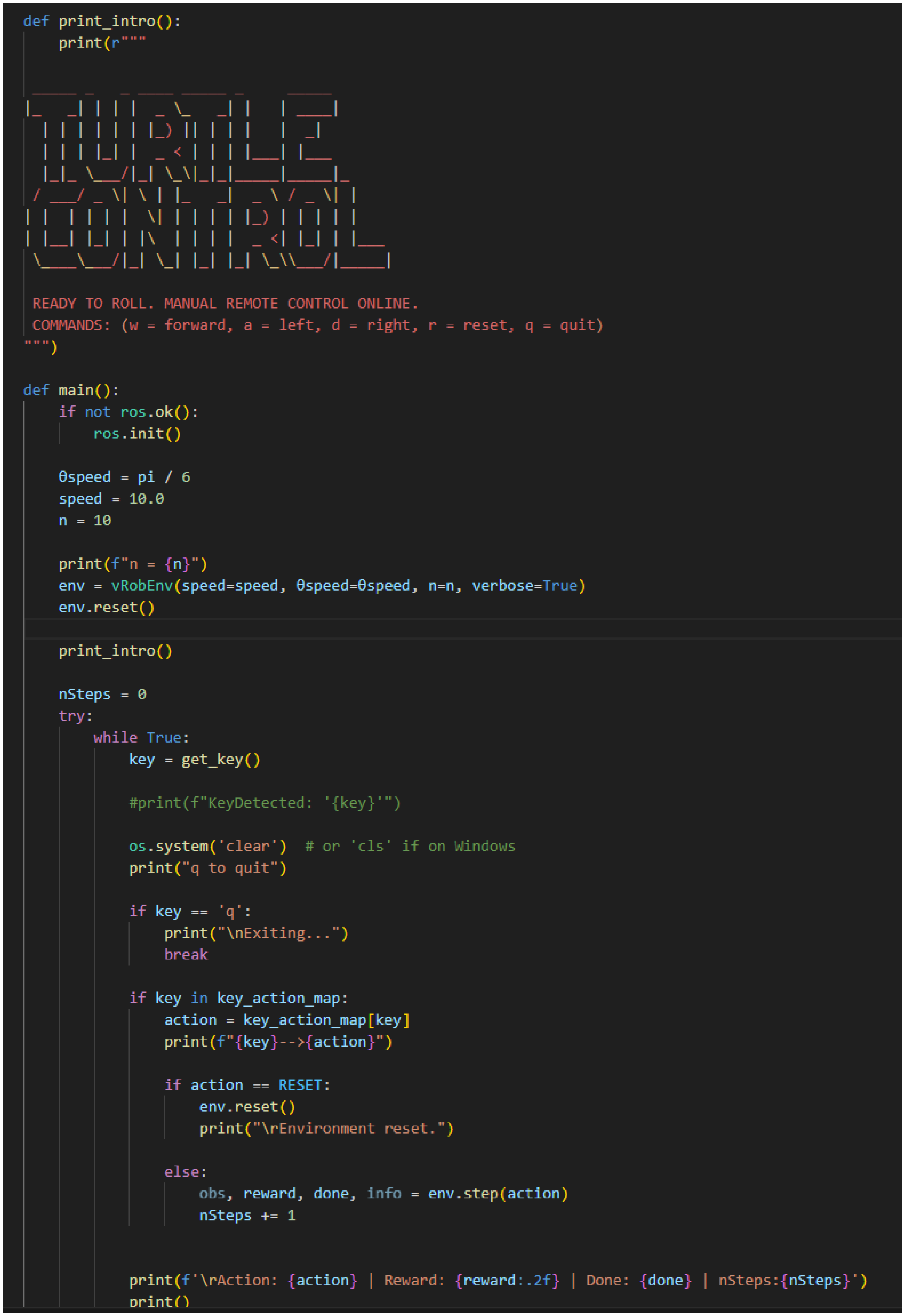
There were so many variables though I struggled to iterate on the method, given more time perhaps SARSA(0) might have been a good choice because it's simpler (I wouldn't have to investigate different values of) and also I would have liked to experiment with an offline learning approach if this was possible.

I didn't discover the "accelerate" sim until quite late, and so this meant each episode was taking 10 minutes or so. After speeding it up (I read how this works, see CETI. (n.d.) *Simulation Speed in ROS/Gazebo*).

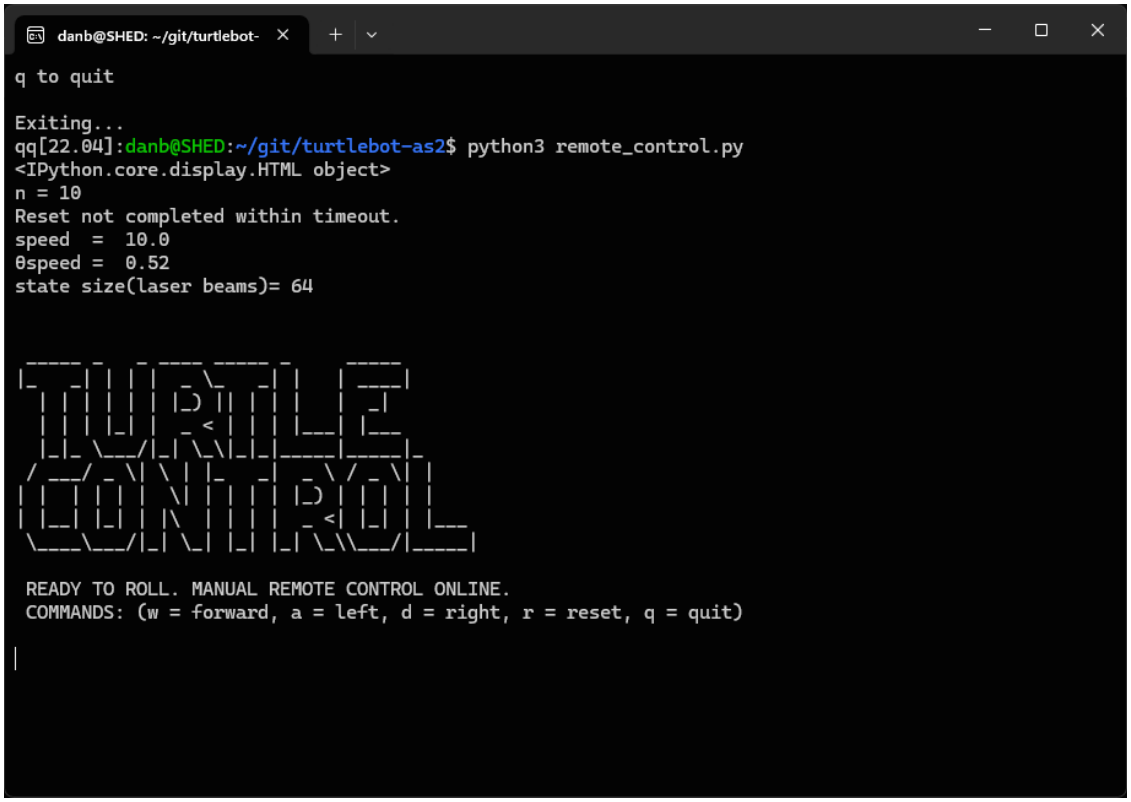
6.2 State representation

The idea of the linear model is that you return a set of binary (one hot) states which represent the prsense of features (eg near wall) and then the agent learns to assign a weight to how important this feature is.

To help understand various state representations I built a passive monitor tool which printed the s_ output in a console and a commander tool which let me control the robot using wasd keys.

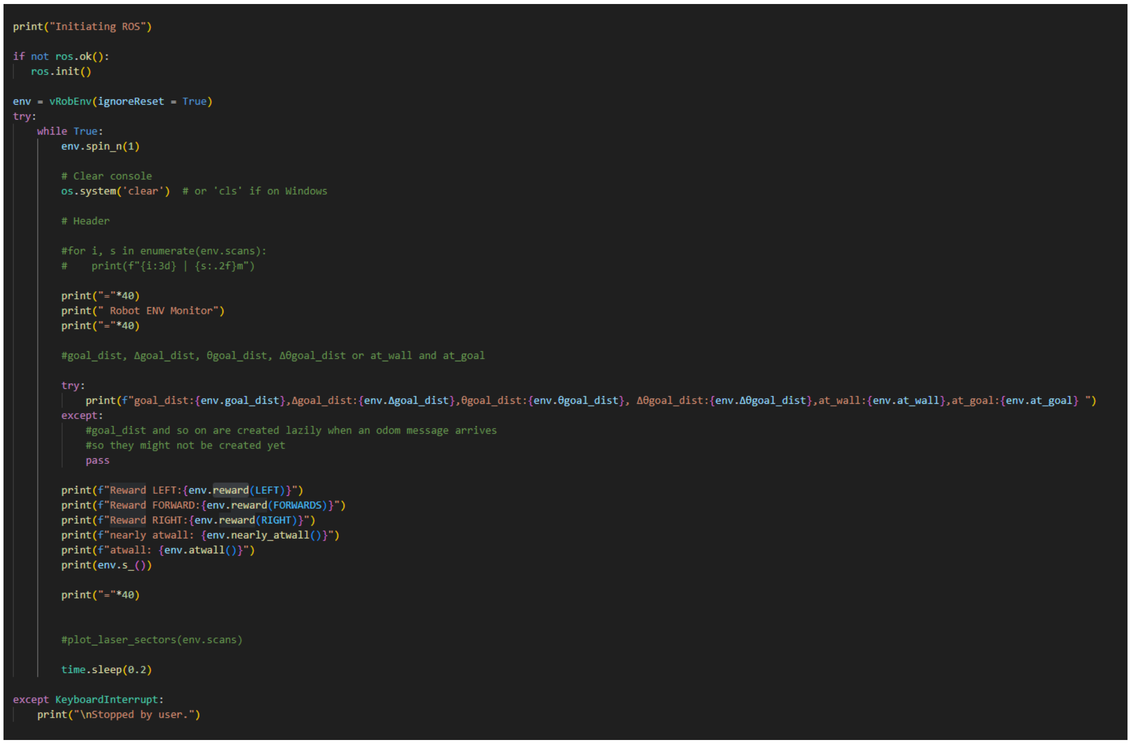


```
def print_intro():
    print(r"""


 _____ _   _ ____ _____ _     _____
|_   _| | | |  _ \_   _| |   | ____|
  | | | | | | |_) || | | |   |  _|
  | | | |_| |  _ < | | | |___| |___
  |_|_ \___/|_| \_\|_|_|_____|_____|_
 / ___/ _ \| \ | |_   _|  _ \ / _ \| |
| |  | | | |  \| | | | | |_) | | | | |
| |__| |_| | |\  | | | |  _ <| |_| | |___
 \____\___/|_| \_| |_| |_| \_\\___/|_____|

 READY TO ROLL. MANUAL REMOTE CONTROL ONLINE.
 COMMANDS: (w = forward, a = left, d = right, r = reset, q = quit)
""")

def main():
    if not ros.ok():
        ros.init()

    θspeed = pi / 6
    speed = 10.0
    n = 10

    print(f"n = {n}")
    env = vRobEnv(speed=speed, θspeed=θspeed, n=n, verbose=True)
    env.reset()

    print_intro()

    nSteps = 0
    try:
        while True:
            key = get_key()

            #print(f"KeyDetected: '{key}'")

            os.system('clear')  # or 'cls' if on Windows
            print("q to quit")

            if key == 'q':
                print("\nExiting...")
                break

            if key in key_action_map:
                action = key_action_map[key]
                print(f"{key}-->{action}")

                if action == RESET:
                    env.reset()
                    print("\rEnvironment reset.")

                else:
                    obs, reward, done, info = env.step(action)
                    nSteps += 1


            print(f'\rAction: {action} | Reward: {reward:.2f} | Done: {done} | nSteps:{nSteps}')
            print()
```

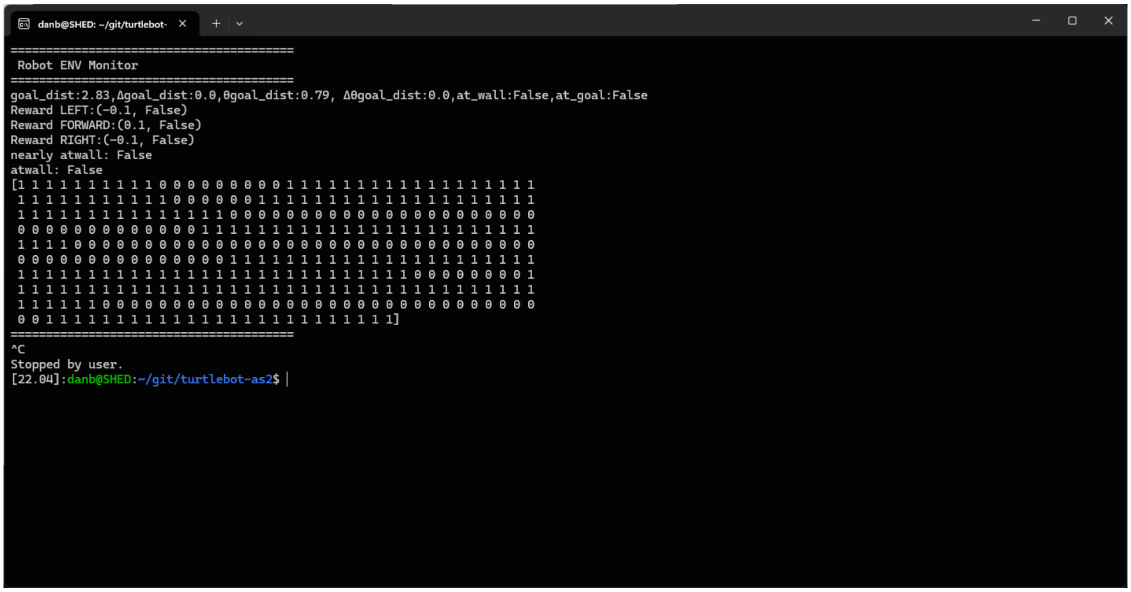


```
q to quit

Exiting...
qq[22.04]:danb@SHED:~/git/turtlebot-as2$ python3 remote_control.py
<IPython.core.display.HTML object>
n = 10
Reset not completed within timeout.
speed  =  10.0
θspeed =  0.52
state size(laser beams)= 64


  _____ _   _ ____ _____ _     _____
 |_   _| | | |  _ \_   _| |   | ____|
   | | | | | | |_) || | | |   |  _|
   | | | |_| |  _ < | | | |___| |___
   |_|_ \___/|_| \_\|_|_|_____|_____|_
  / ___/ _ \| \ | |_   _|  _ \ / _ \| |
 | |  | | | |  \| | | | | |_) | | | | |
 | |__| |_| | |\  | | | |  _ <| |_| | |___
  \____\___/|_| \_| |_| |_| \_\\___/|_____|

 READY TO ROLL. MANUAL REMOTE CONTROL ONLINE.
 COMMANDS: (w = forward, a = left, d = right, r = reset, q = quit)

|
```

and



```python
print("Initiating ROS")

if not ros.ok():
    ros.init()

env = vRobEnv(ignoreReset = True)
try:
    while True:
        env.spin_n(1)

        # Clear console
        os.system('clear')  # or 'cls' if on Windows

        # Header

        #for i, s in enumerate(env.scans):
        #    print(f"{i:3d} | {s:.2f}m")

        print("="*40)
        print(" Robot ENV Monitor")
        print("="*40)

        #goal_dist, Δgoal_dist, θgoal_dist, Δθgoal_dist or at_wall and at_goal

        try:
            print(f"goal_dist:{env.goal_dist},Δgoal_dist:{env.Δgoal_dist},θgoal_dist:{env.θgoal_dist}, Δθgoal_dist:{env.Δθgoal_dist},at_wall:{env.at_wall},at_goal:{env.at_goal} ")
        except:
            #goal_dist and so on are created lazily when an odom message arrives
            #so they might not be created yet
            pass

        print(f"Reward LEFT:{env.reward(LEFT)}")
        print(f"Reward FORWARD:{env.reward(FORWARDS)}")
        print(f"Reward RIGHT:{env.reward(RIGHT)}")
        print(f"nearly atwall: {env.nearly_atwall()}")
        print(f"atwall: {env.atwall()}")
        print(env.s_())

        print("="*40)


        #plot_laser_sectors(env.scans)

        time.sleep(0.2)

except KeyboardInterrupt:
    print("\nStopped by user.")
```



```
======================================
 Robot ENV Monitor
======================================
goal_dist:2.83,Δgoal_dist:0.0,θgoal_dist:0.79, Δθgoal_dist:0.0,at_wall:False,at_goal:False
Reward LEFT:(-0.1, False)
Reward FORWARD:(0.1, False)
Reward RIGHT:(-0.1, False)
nearly atwall: False
atwall: False
[1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
======================================
^C
Stopped by user.
[22.04]:danb@SHED:~/git/turtlebot-as2$ |
```

This was very useful as I could see how much “information” the robot was getting.

6.2.1 Initial state model

Initially I used the example state provided:

```
def s_(self):
        max, min = self.max_range, self.min_range
        # returns a normalise and descritised componenets
        return  1*(((self.scans - min)/(max - min))>=.6)
```

This means normalise to 0 and 1 and if it’s over half way between them then it’s 1 otherwise 0. It’s quite arbitary. I explored the not change much the robot moved around the area, I thought it wouldn’t give you a good signal.

6.2.2 Is near anything

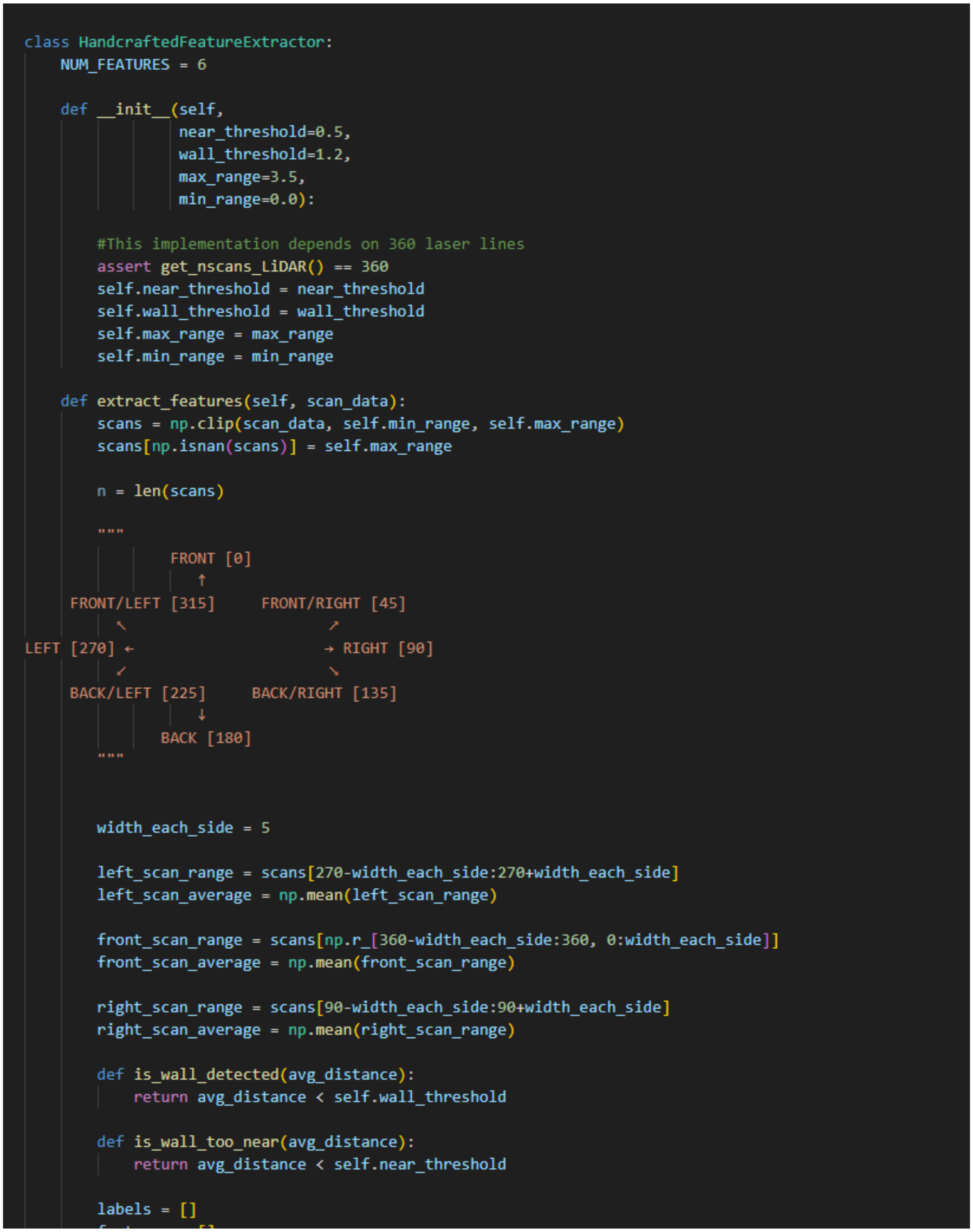
This changed from being normalised to being within a set distance.

```
def s_(self):
        #State is if we're near a wall
        states = (self.scans <= 0.3).astype(int)
        assert states.shape[0] == 64 # self.nF
        return states
```

I tried different thresholds and even also adding more states near | middle | far (so nF = nScans * 3)

6.2.3 Hand crafted features

I thought maybe I could detect edges and walls infront, left right and so on, and even detect if I could “see the round” goal post, but in practice I didn’t get very good results. see robot_environment.py



```
class HandcraftedFeatureExtractor:
    NUM_FEATURES = 6

    def __init__(self,
                 near_threshold=0.5,
                 wall_threshold=1.2,
                 max_range=3.5,
                 min_range=0.0):

        #This implementation depends on 360 laser lines
        assert get_nscans_LiDAR() == 360
        self.near_threshold = near_threshold
        self.wall_threshold = wall_threshold
        self.max_range = max_range
        self.min_range = min_range

    def extract_features(self, scan_data):
        scans = np.clip(scan_data, self.min_range, self.max_range)
        scans[np.isnan(scans)] = self.max_range

        n = len(scans)

        """
                FRONT [0]
                   ↑
     FRONT/LEFT [315]     FRONT/RIGHT [45]
          ↖                       ↗
LEFT [270] ←                     → RIGHT [90]
          ↙                       ↘
     BACK/LEFT [225]     BACK/RIGHT [135]
                   ↓
               BACK [180]
        """


        width_each_side = 5

        left_scan_range = scans[270-width_each_side:270+width_each_side]
        left_scan_average = np.mean(left_scan_range)

        front_scan_range = scans[np.r_[360-width_each_side:360, 0:width_each_side]]
        front_scan_average = np.mean(front_scan_range)

        right_scan_range = scans[90-width_each_side:90+width_each_side]
        right_scan_average = np.mean(right_scan_range)

        def is_wall_detected(avg_distance):
            return avg_distance < self.wall_threshold

        def is_wall_too_near(avg_distance):
            return avg_distance < self.near_threshold

        labels = []
```

6.2.4 Difficult to tell if the state representation was any good

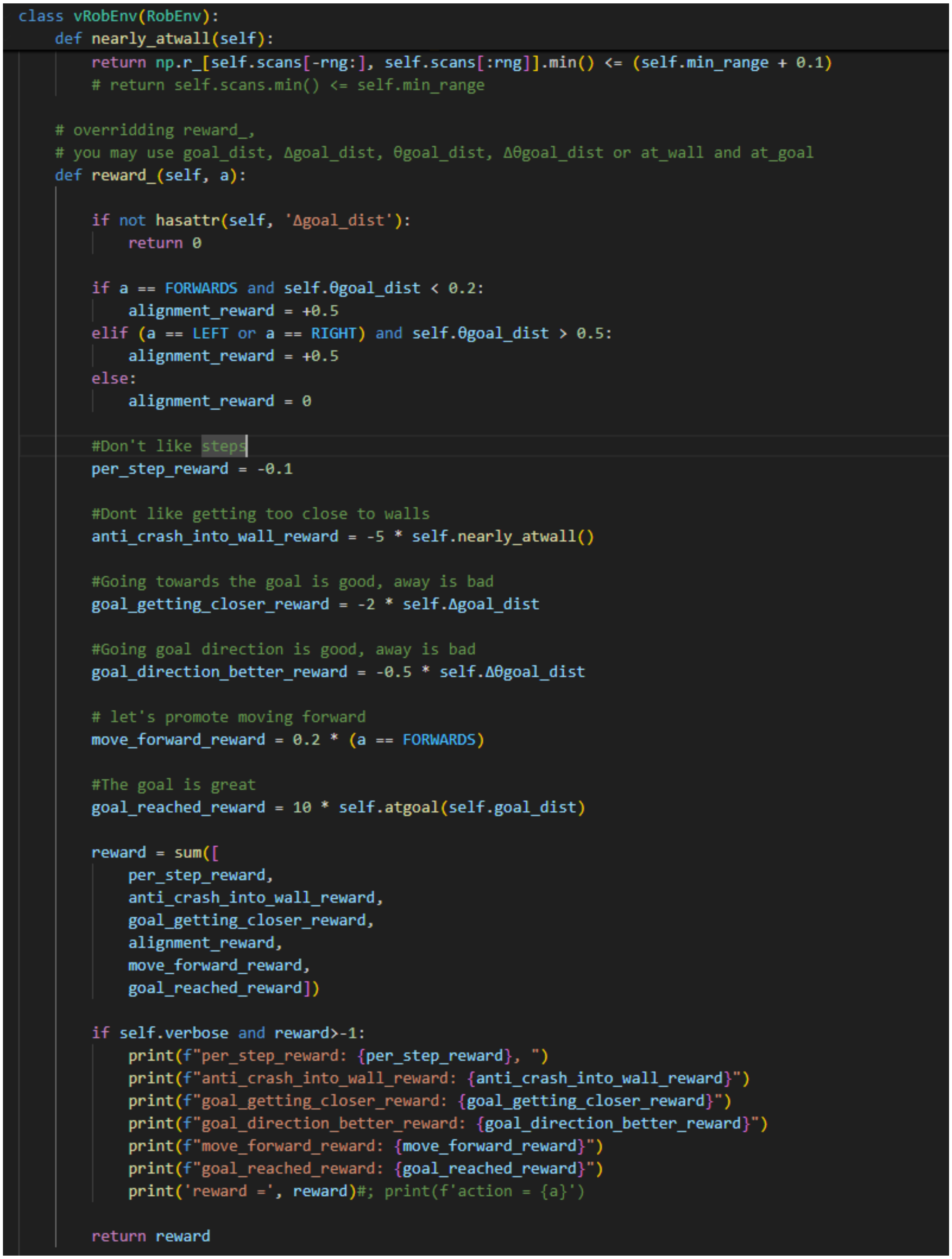
I found it really tricky to know if my bad performance was because of my reward, state or hyperparameters - there's just so much to change and track at once. The agent (robot) could find the goals but didn't seem to converge on an optimum solution. I did some research into state extraction Núñez, P., Vazquez-Martin, R., Bandera, A., and Romero-Gonzalez, C. (2015) 'Feature extraction from laser scan data based on curvature estimation for mobile robotics' and so on, and I think it would be very useful to test state extraction from labeled data - even learn this seperately in a NN.

6.3 Reward function

The idea of the reward function is to provide a signal as to whether the robot is doing well. It shouldn't encode “how to” do the task, just if agent is acheiving the goal.

The odom was used in the reward and was limited to “goal_dist, Δgoal_dist, goal_dist, Δ - goal_dist or at_wall and at_goal” as per the instructions. If goal_dist wasn't “absolute” ie it was signed then I think the task would have been easier because I could reward LEFT and RIGHT appropriately (then the task would be the robot can a compass which points to the goal, but has to navigate thorugh the maze and not get stuck.)

My thoughts were that having only a reward at the end might be too sparse, and you were allowed to reward getting closer to the goal. I tried various combinations of reward, but again I couldn't get stable results, there are so many variables I could have done with another week or so and some practical guidance.



```
class vRobEnv(RobEnv):
    def nearly_atwall(self):
        return np.r_[self.scans[-rng:], self.scans[:rng]].min() <= (self.min_range + 0.1)
        # return self.scans.min() <= self.min_range

    # overridding reward_,
    # you may use goal_dist, Δgoal_dist, θgoal_dist, Δθgoal_dist or at_wall and at_goal
    def reward_(self, a):

        if not hasattr(self, 'Δgoal_dist'):
            return 0

        if a == FORWARDS and self.θgoal_dist < 0.2:
            alignment_reward = +0.5
        elif (a == LEFT or a == RIGHT) and self.θgoal_dist > 0.5:
            alignment_reward = +0.5
        else:
            alignment_reward = 0

        #Don't like steps
        per_step_reward = -0.1

        #Dont like getting too close to walls
        anti_crash_into_wall_reward = -5 * self.nearly_atwall()

        #Going towards the goal is good, away is bad
        goal_getting_closer_reward = -2 * self.Δgoal_dist

        #Going goal direction is good, away is bad
        goal_direction_better_reward = -0.5 * self.Δθgoal_dist

        # let's promote moving forward
        move_forward_reward = 0.2 * (a == FORWARDS)

        #The goal is great
        goal_reached_reward = 10 * self.atgoal(self.goal_dist)

        reward = sum([
            per_step_reward,
            anti_crash_into_wall_reward,
            goal_getting_closer_reward,
            alignment_reward,
            move_forward_reward,
            goal_reached_reward])

        if self.verbose and reward>-1:
            print(f"per_step_reward: {per_step_reward}, ")
            print(f"anti_crash_into_wall_reward: {anti_crash_into_wall_reward}")
            print(f"goal_getting_closer_reward: {goal_getting_closer_reward}")
            print(f"goal_direction_better_reward: {goal_direction_better_reward}")
            print(f"move_forward_reward: {move_forward_reward}")
            print(f"goal_reached_reward: {goal_reached_reward}")
            print('reward =', reward)#; print(f'action = {a}')

        return reward
```

I added the “promote” moving forwards because it was getting stuck going around in circles. Given more time I’d start very simply and build up the rewards.

6.4 Hyperparameter tuning

With Sarsa the optimum

```
for   in [0.1,0.2,0.5]:
    for   in [0.1,0.3,0.5,0.8,1.1]:
        for   in [0.1,0.5,0.99]:
```

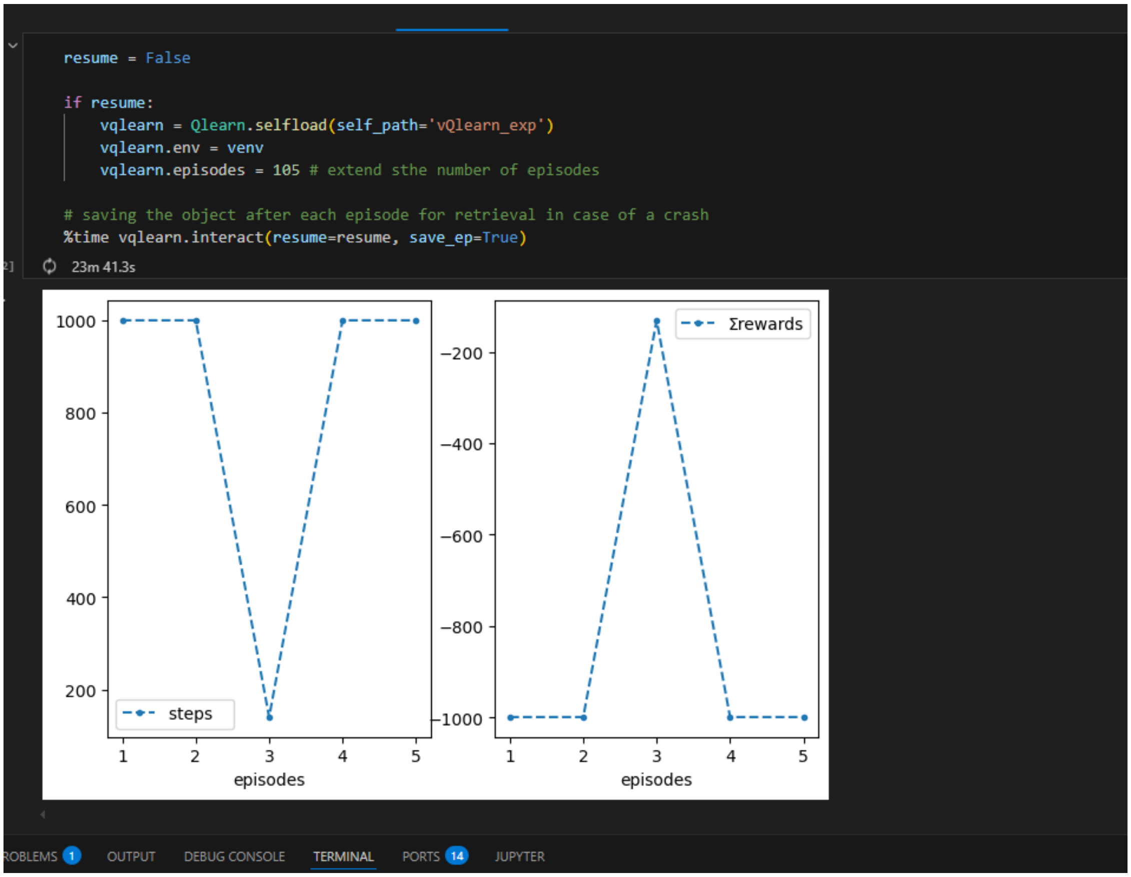
With Sarsa the   changes the algorith from TD(0) to MC on a sliding scale, and different   will be best for these different approaches. I tried a grid search of solutions, however it wasn't until quite late that I learnt how to speed up the simulation.

```
[ ]: for   in [0.1,0.2,0.5]:
         for   in [0.1,0.3,0.5,0.8,1.1]:
             for   in [0.1,0.5,0.99]:

                 max_t = 10
                  min = 0.05
                 d  = (  - min) / max_t

                 hyperparameters = {
                     'max_t':max_t,
                     ' ':  , #Initial value Used in the epsilon Greedy so it will be␣
       ↪random   times per request for a next action
                     ' min':  min, # epsilon decreases (there's a theory that if this␣
       ↪decreases to 0 at infinity you're ll have teh optimum solution)
                     'd ': d , # dtop in epsilon per step
                     ' ': 0.05, # learning rate how much of the new informatin is␣
       ↪used to update the existing value prediction
                     ' ': 0.99, # discount factor, large means that the rewards in␣
       ↪the future contribute to the current action state prediction a lot
                     ' ': 0.6, # trace decay for the eligibility trace 1 is MC, 0 is␣
       ↪just the last step

                     'verbose': False,

                     # Robot Environment params
                     ' speed': pi / 2,
                     'speed': 3.0,
                     'n': 3
                 }
                 print(hyperparameters)

                 env = vRobEnvCornerDetector(
                     speed=hyperparameters['speed'],
                      speed=hyperparameters[' speed'],
                     n=hyperparameters['n'],
                     verbose=hyperparameters['verbose']
                 )
```

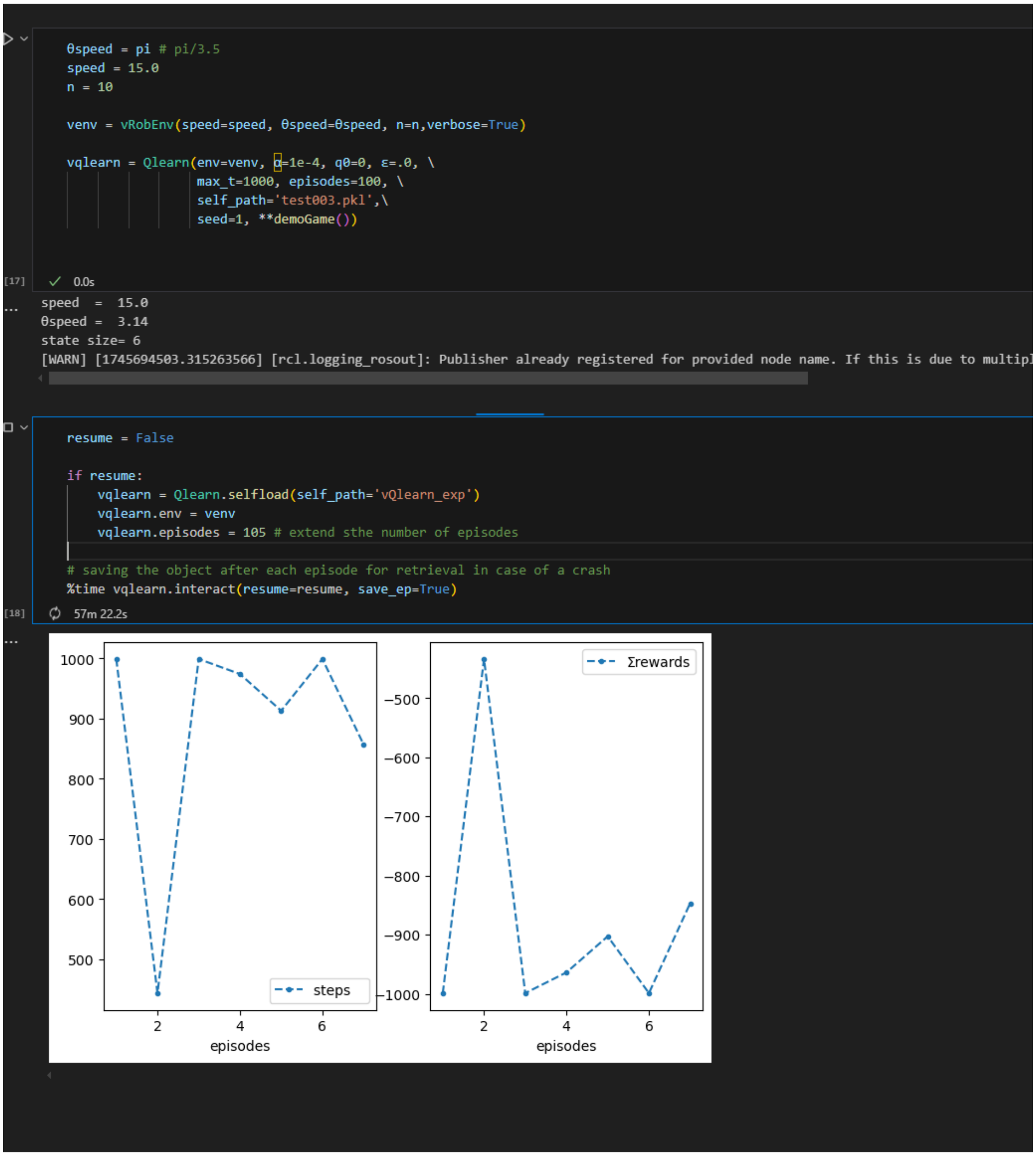
```
vqlearn = Sarsa (
    env=env,
     =hyperparameters[' '],
     =hyperparameters[' '],
     =hyperparameters[' '],
     =hyperparameters[' '],
     min=hyperparameters[' min'],
    d =hyperparameters['d '],
    q0=0,
    Tstar=0,
    max_t=hyperparameters['max_t'],
    episodes=5,
    self_path='SarsaLambda.vRobEnvCornerDetector.test012.pkl',
    seed=1,
    plotT=True,
    plotR=True,
    visual=True,
    animate=False
)

vqlearn.interact(resume=False, save_ep=True, plot_exp = True)
```

6.5 Learning plots + success rates

6.5.1 Q-Learning



```
resume = False

if resume:
    vqlearn = Qlearn.selfload(self_path='vQlearn_exp')
    vqlearn.env = venv
    vqlearn.episodes = 105 # extend sthe number of episodes

# saving the object after each episode for retrieval in case of a crash
%time vqlearn.interact(resume=resume, save_ep=True)
```



```
θspeed = pi # pi/3.5
speed = 15.0
n = 10

venv = vRobEnv(speed=speed, θspeed=θspeed, n=n,verbose=True)

vqlearn = Qlearn(env=venv, α=1e-4, q0=0, ε=.0, \
                 max_t=1000, episodes=100, \
                 self_path='test003.pkl',\
                 seed=1, **demoGame())
```

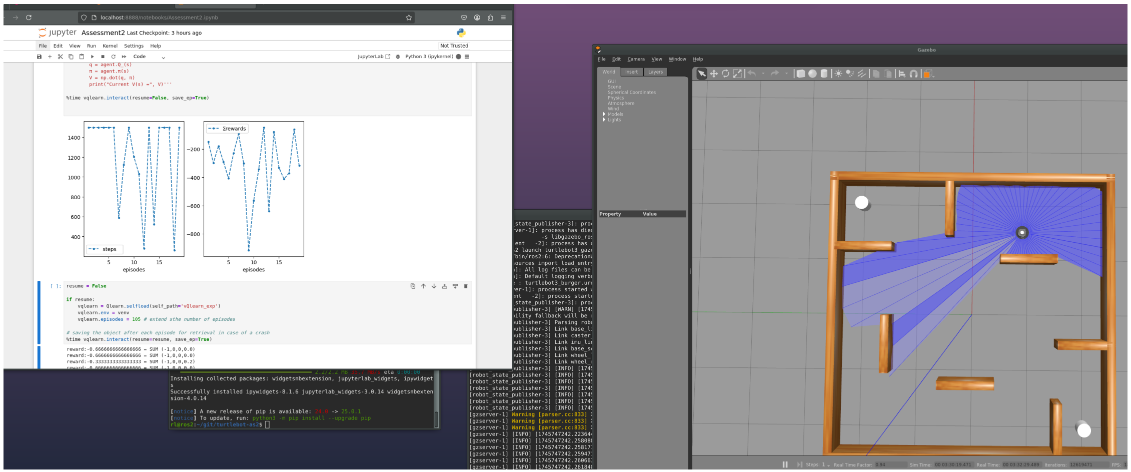
[17] ✓ 0.0s

```
speed  =  15.0
θspeed =  3.14
state size= 6
[WARN] [1745694503.315263566] [rcl.logging_rosout]: Publisher already registered for provided node name. If this is due to multipl
```

```
resume = False

if resume:
    vqlearn = Qlearn.selfload(self_path='vQlearn_exp')
    vqlearn.env = venv
    vqlearn.episodes = 105 # extend sthe number of episodes

# saving the object after each episode for retrieval in case of a crash
%time vqlearn.interact(resume=resume, save_ep=True)
```

[18] 57m 22.2s

6.5.2 SARSA



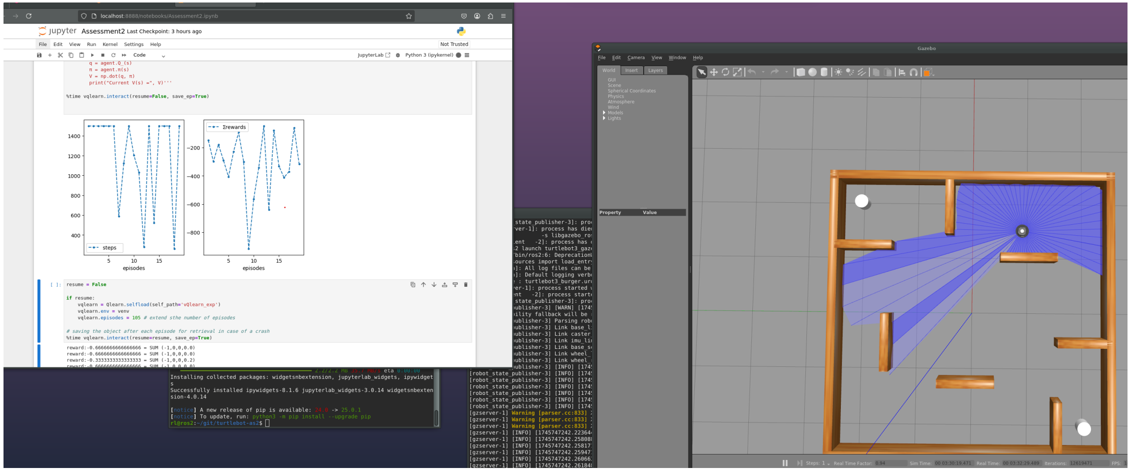
```
        q = agent.Q_(s)
        π = agent.π(s)
        V = np.dot(q, π)
        print("Current V(s) =", V)'''

%time vqlearn.interact(resume=False, save_ep=True)
```

```
[ ]: resume = False

     if resume:
         vqlearn = Qlearn.selfload(self_path='vQlearn_exp')
         vqlearn.env = venv
         vqlearn.episodes = 105 # extend sthe number of episodes

     # saving the object after each episode for retrieval in case of a crash
     %time vqlearn.interact(resume=resume, save_ep=True)
```

```
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
```

```
                                         2.2/2.2 MB 35.7 MB/s eta 0:00:00
Installing collected packages: widgetsnbextension, jupyterlab_widgets, ipywidget
s
Successfully installed ipywidgets-8.1.6 jupyterlab_widgets-3.0.14 widgetsnbexten
sion-4.0.14

[notice] A new release of pip is available: 24.0 -> 25.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
rl@ros2:~/git/turtlebot-as2$ 
```

```
state_publisher-3]: proc
ver-1]: process has died
       -s libgazebo_ros
ent   -2]: process has
2 launch turtlebot3_gaze
bin/ros2:6: Deprecation
ources import load_entry
]: All log files can be
]: Default logging verbo
 : turtlebot3_burger.urd
er-1]: process started w
nt   -2]: process starte
state_publisher-3]: proc
ublisher-3] [WARN] [174
ility fallback will be
ublisher-3] Parsing robo
ublisher-3] Link base_l
ublisher-3] Link caster
ublisher-3] Link imu_li
ublisher-3] Link base_s
ublisher-3] Link wheel_
ublisher-3] Link wheel_
ublisher-3] [INFO] [174
[robot_state_publisher-3] [INFO] [174
[robot_state_publisher-3] [INFO] [174
[robot_state_publisher-3] [INFO] [174
[robot_state_publisher-3] [INFO] [174
[robot_state_publisher-3] [INFO] [174
[robot_state_publisher-3] [INFO] [174
[gzserver-1] Warning [parser.cc:833]
[gzserver-1] Warning [parser.cc:833]
[gzserver-1] Warning [parser.cc:833]
[gzserver-1] [INFO] [1745747242.22364
[gzserver-1] [INFO] [1745747242.25808
[gzserver-1] [INFO] [1745747242.25817
[gzserver-1] [INFO] [1745747242.25947
[gzserver-1] [INFO] [1745747242.26066
[gzserver-1] [INFO] [1745747242.26184
```

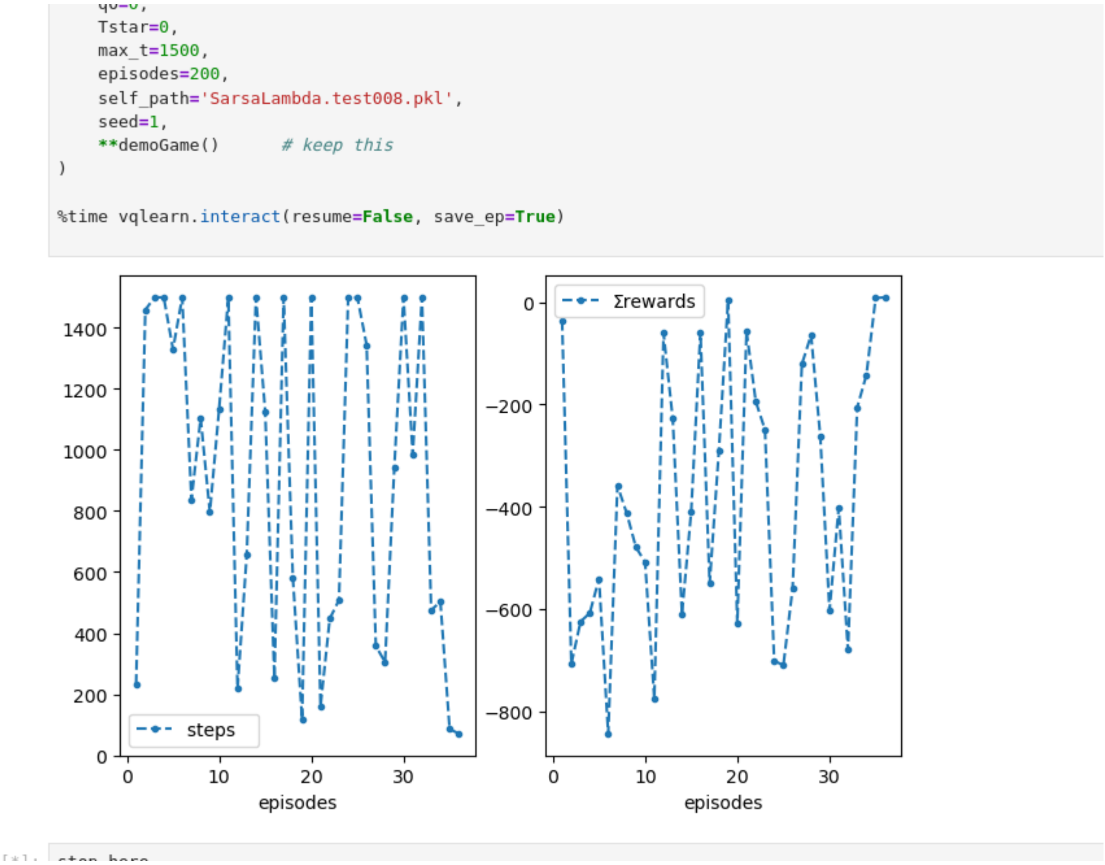


```
        q = agent.Q_(s)
        π = agent.π(s)
        V = np.dot(q, π)
        print("Current V(s) =", V)'''

%time vqlearn.interact(resume=False, save_ep=True)
```

```
[ ]: resume = False

     if resume:
         vqlearn = Qlearn.selfload(self_path='vQlearn_exp')
         vqlearn.env = venv
         vqlearn.episodes = 105 # extend sthe number of episodes

     # saving the object after each episode for retrieval in case of a crash
     %time vqlearn.interact(resume=resume, save_ep=True)
```

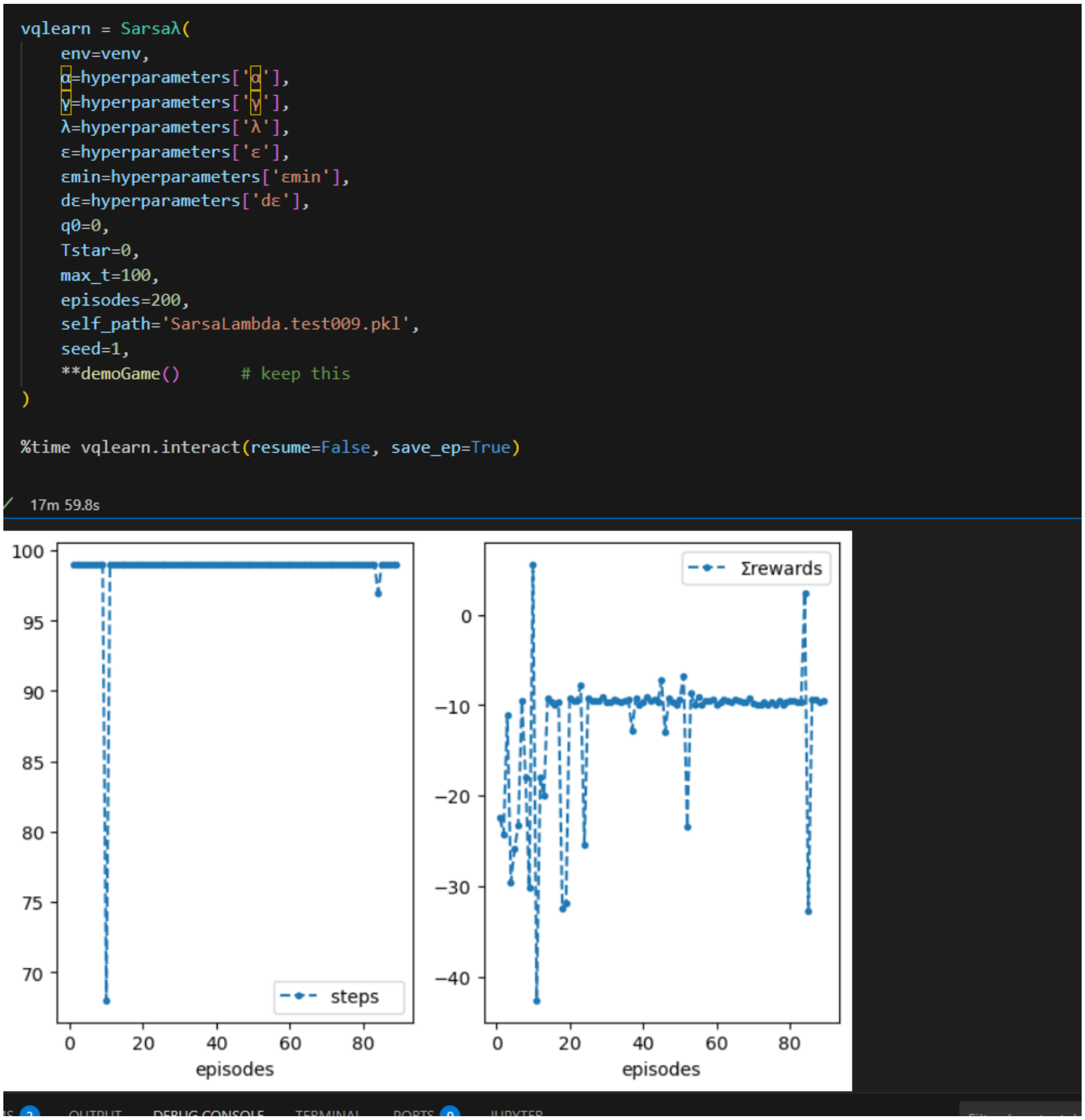
```
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
```

```
                                         2.2/2.2 MB 35.7 MB/s eta 0:00:00
Installing collected packages: widgetsnbextension, jupyterlab_widgets, ipywidget
s
Successfully installed ipywidgets-8.1.6 jupyterlab_widgets-3.0.14 widgetsnbexten
sion-4.0.14

[notice] A new release of pip is available: 24.0 -> 25.0.1
[notice] To update, run: python3 -m pip install --upgrade pip
rl@ros2:~/git/turtlebot-as2$ 
```



```
    q0=0,
    Tstar=0,
    max_t=1500,
    episodes=200,
    self_path='SarsaLambda.test008.pkl',
    seed=1,
    **demoGame()       # keep this
)

%time vqlearn.interact(resume=False, save_ep=True)
```

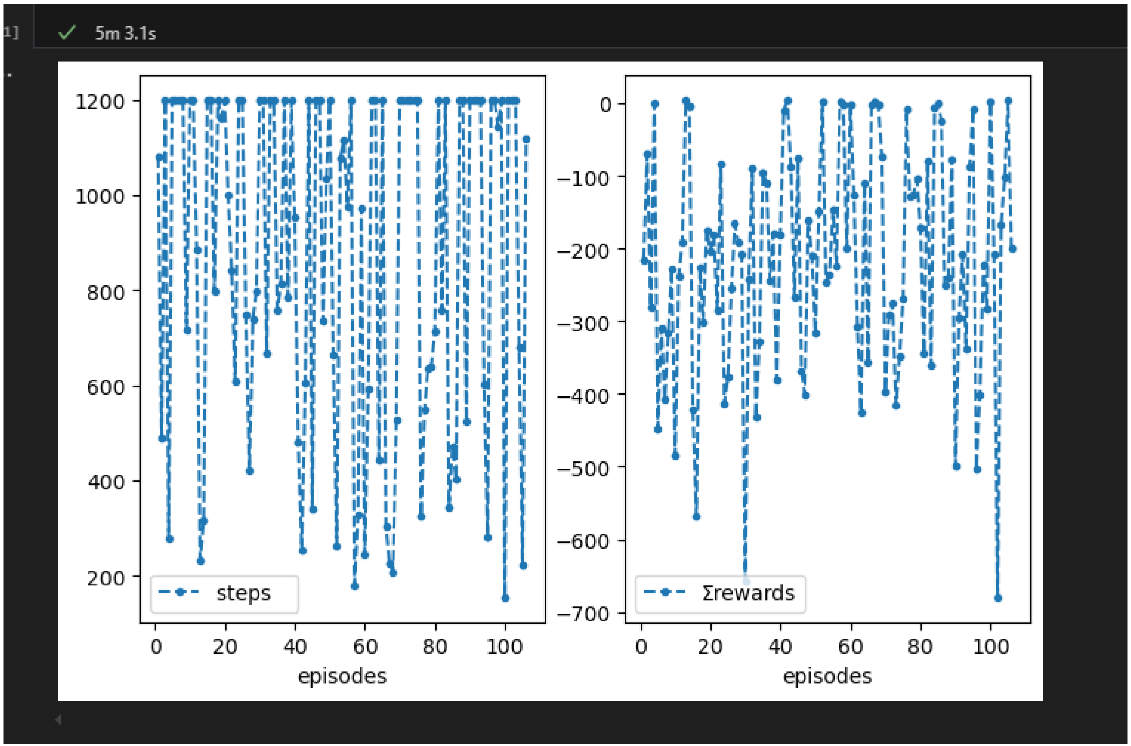
```
[*]: step here
```

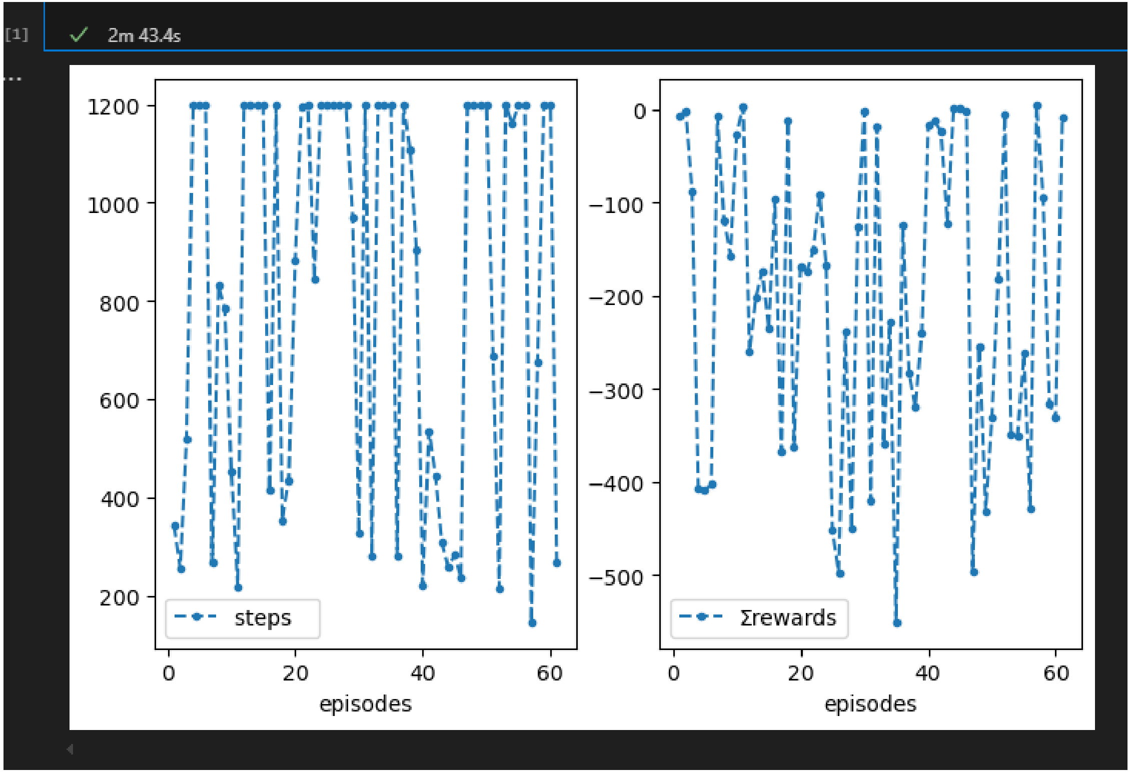


```
vqlearn = Sarsaλ(
    env=venv,
    α=hyperparameters['α'],
    γ=hyperparameters['γ'],
    λ=hyperparameters['λ'],
    ε=hyperparameters['ε'],
    εmin=hyperparameters['εmin'],
    dε=hyperparameters['dε'],
    q0=0,
    Tstar=0,
    max_t=100,
    episodes=200,
    self_path='SarsaLambda.test009.pkl',
    seed=1,
    **demoGame()      # keep this
)

%time vqlearn.interact(resume=False, save_ep=True)
```

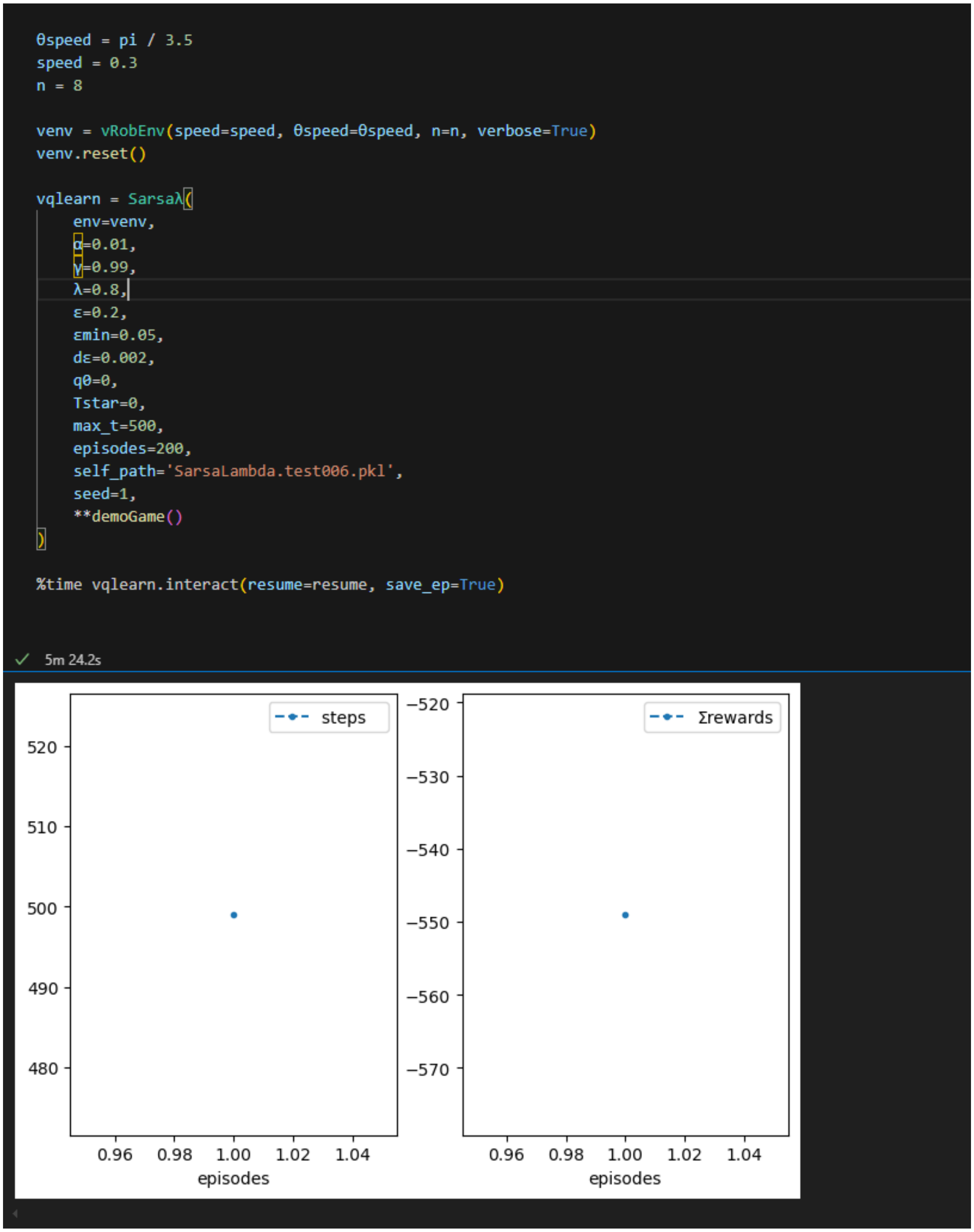
✓ 17m 59.8s

6.5.3 SARSA Changing Epsilon-Greedy



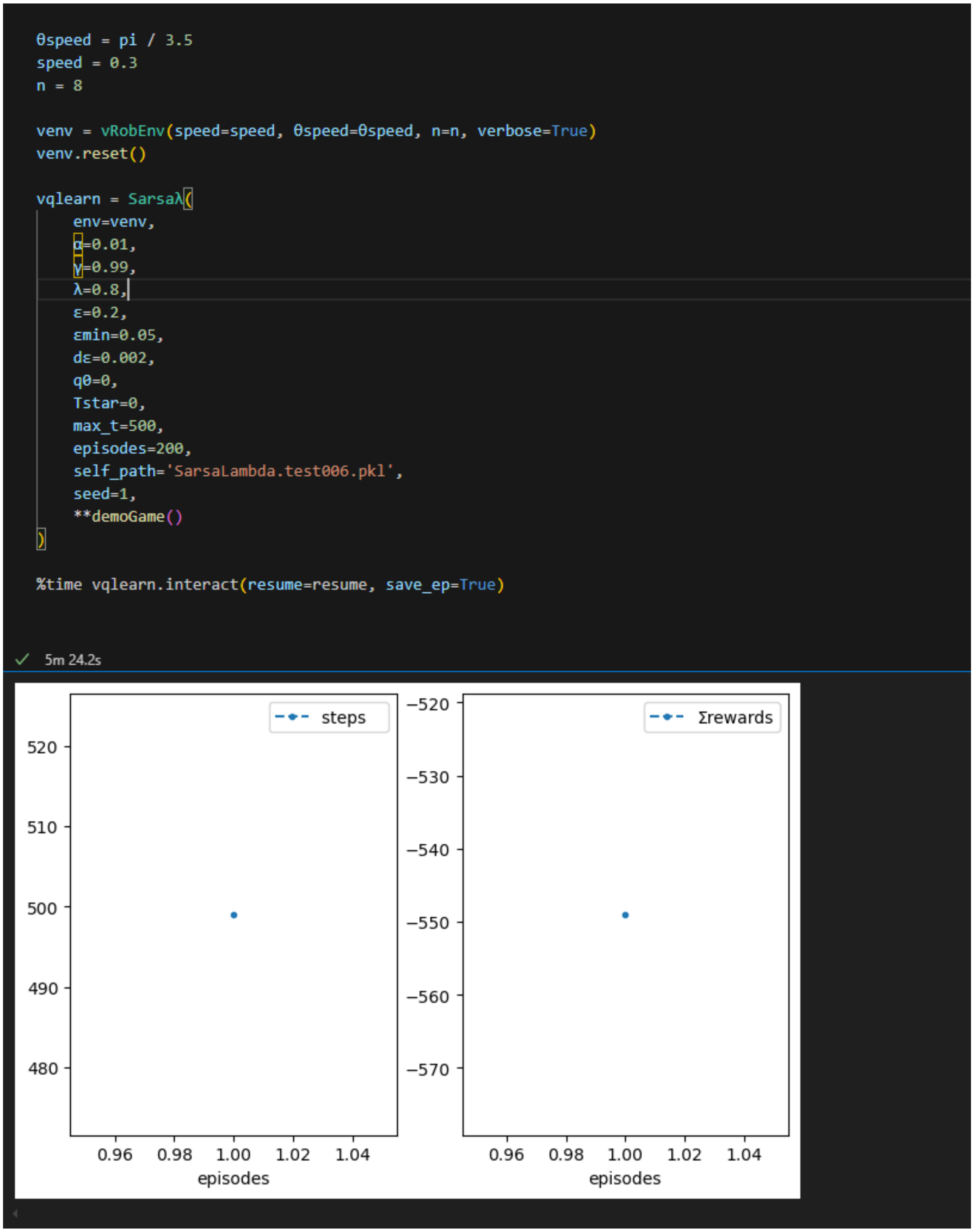


6.5.4 SARSA( )



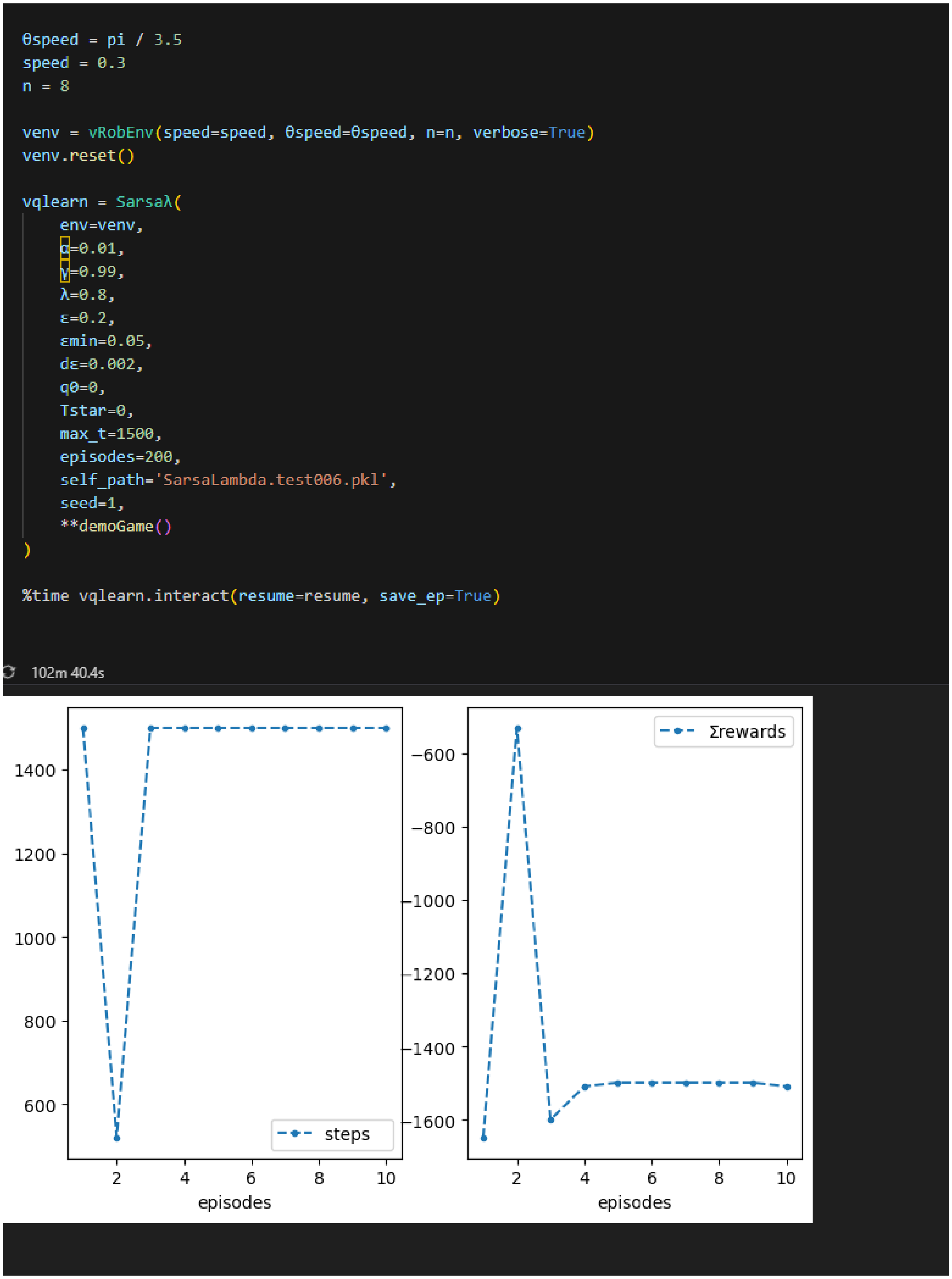
```
θspeed = pi / 3.5
speed = 0.3
n = 8

venv = vRobEnv(speed=speed, θspeed=θspeed, n=n, verbose=True)
venv.reset()

vqlearn = Sarsaλ(
    env=venv,
    α=0.01,
    γ=0.99,
    λ=0.8,
    ε=0.2,
    εmin=0.05,
    dε=0.002,
    q0=0,
    Tstar=0,
    max_t=500,
    episodes=200,
    self_path='SarsaLambda.test006.pkl',
    seed=1,
    **demoGame()
)

%time vqlearn.interact(resume=resume, save_ep=True)
```

✓ 5m 24.2s



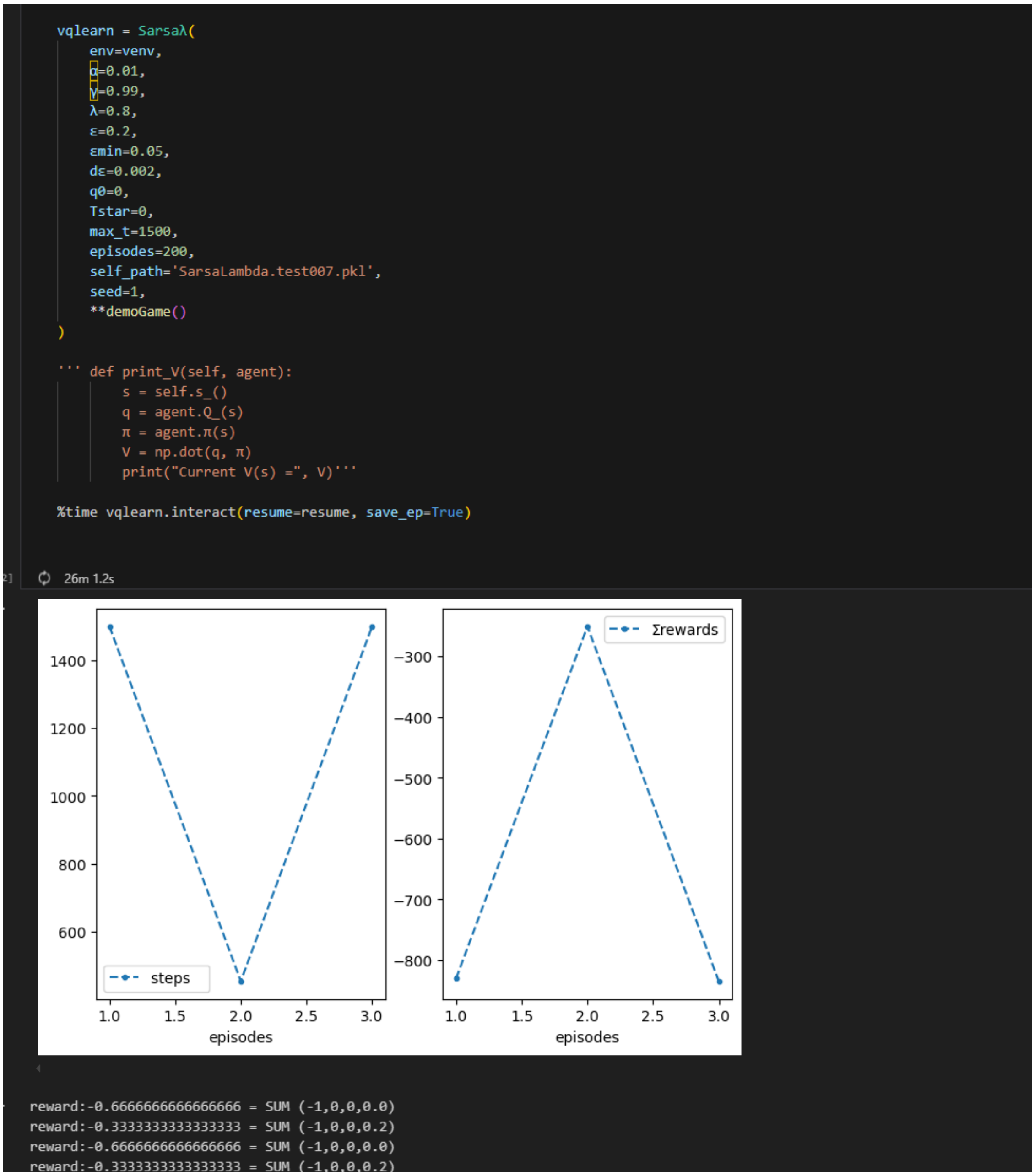
```
θspeed = pi / 3.5
speed = 0.3
n = 8

venv = vRobEnv(speed=speed, θspeed=θspeed, n=n, verbose=True)
venv.reset()

vqlearn = Sarsaλ(
    env=venv,
    α=0.01,
    γ=0.99,
    λ=0.8,
    ε=0.2,
    εmin=0.05,
    dε=0.002,
    q0=0,
    Tstar=0,
    max_t=500,
    episodes=200,
    self_path='SarsaLambda.test006.pkl',
    seed=1,
    **demoGame()
)

%time vqlearn.interact(resume=resume, save_ep=True)
```

✓ 5m 24.2s



```
θspeed = pi / 3.5
speed = 0.3
n = 8

venv = vRobEnv(speed=speed, θspeed=θspeed, n=n, verbose=True)
venv.reset()

vqlearn = Sarsaλ(
    env=venv,
    α=0.01,
    γ=0.99,
    λ=0.8,
    ε=0.2,
    εmin=0.05,
    dε=0.002,
    q0=0,
    Tstar=0,
    max_t=1500,
    episodes=200,
    self_path='SarsaLambda.test006.pkl',
    seed=1,
    **demoGame()
)

%time vqlearn.interact(resume=resume, save_ep=True)
```

102m 40.4s



```
vqlearn = Sarsaλ(
    env=venv,
    α=0.01,
    γ=0.99,
    λ=0.8,
    ε=0.2,
    εmin=0.05,
    dε=0.002,
    q0=0,
    Tstar=0,
    max_t=1500,
    episodes=200,
    self_path='SarsaLambda.test007.pkl',
    seed=1,
    **demoGame()
)

''' def print_V(self, agent):
        s = self.s_()
        q = agent.Q_(s)
        π = agent.π(s)
        V = np.dot(q, π)
        print("Current V(s) =", V)'''

%time vqlearn.interact(resume=resume, save_ep=True)
```

26m 1.2s

```
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
```

[4]:
```
max_t = 2000
 = 0.2
 min = 0.05
d = ( - min) / max_t

hyperparameters = {
        'max_t':max_t,
        ' ': , #Initial value Used in the epsilon Greedy so it will be random ␣
    ↪times per request for a next action
        ' min': min, # epsilon decreases (there's a theory that if this decreases␣
    ↪to 0 at infinity you're ll have teh optimum solution)
    'd ': d , # dtop in epsilon per step
    ' ': 0.05, # learning rate how much of the new informatin is used to update␣
 ↪the existing value prediction
    ' ': 0.99, # discount factor, large means that the rewards in the future␣
 ↪contribute to the current action state prediction a lot

    'verbose': False,

    # Robot Environment params
    ' speed': pi / 2,
    'speed': 3.0,
    'n': 3
}

env = vRobEnv(
    speed=hyperparameters['speed'],
     speed=hyperparameters[' speed'],
    n=hyperparameters['n'],
    verbose=hyperparameters['verbose']
)

vqlearn = Qlearn(
    env=env,
     =hyperparameters[' '],
     =hyperparameters[' '],
     =hyperparameters[' '],
     min=hyperparameters[' min'],
    d =hyperparameters['d '],
    q0=0,
    Tstar=0,
    max_t=1200,
    episodes=200,
    self_path='Qlearn.test54.pkl',
    seed=1,
    **demoGame()
)

print(hyperparameters)
%time vqlearn.interact(resume=False, save_ep=True)
```

```
Reset not completed within timeout.
speed  =  3.0
 speed =  1.57
state size(laser beams)= 64
{'max_t': 2000, ' ': 0.2, ' min': 0.05, 'd ': 7.500000000000001e-05, ' ': 0.05,
' ': 0.99, 'verbose': False, ' speed': 1.5707963267948966, 'speed': 3.0, 'n': 3}
```

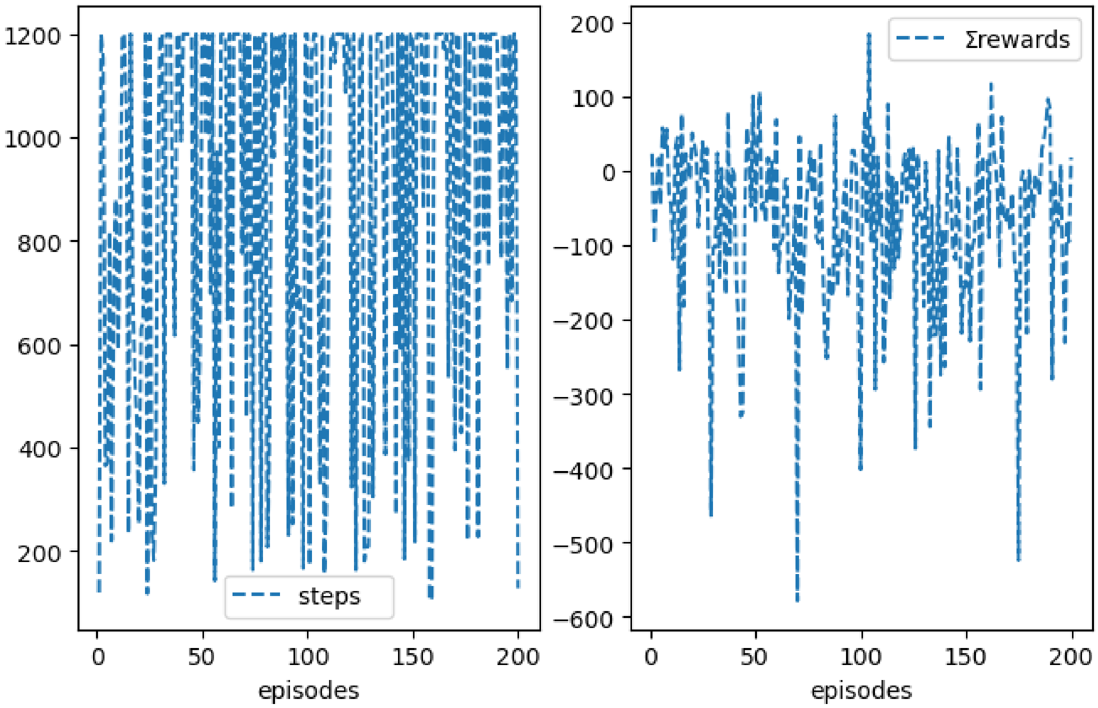
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
File <timed eval>:1

File ~/git/turtlebot-as2/rl/rl.py:194, in MRP.interact(self, train, resume,␣
 ↪save_ep, episodes, grid_img, **kw)
    191 self.t_+= 1
    193 rn,sn, a,an, done = self.step(s,a, self.t)  # takes a step in env and␣
 ↪store tarjectory if needed
--> 194 self.online(s, rn,sn, done, a,an) if train else None # to learn online,
 ↪pass a one step trajectory
    196 self.Σr += rn
    197 self.rn = rn

File ~/git/turtlebot-as2/rl/rl.py:770, in Qlearn.online(self, s, rn, sn, done,␣
 ↪a, _)
    769 def online(self, s, rn,sn, done, a,_):
--> 770     self.Q[s,a] += self. *(rn + (1- done)*self. *self.Q[sn].max() - self
 ↪Q[s,a])

IndexError: index 54 is out of bounds for axis 1 with size 3
```

[ ]:

```
max_t = 2000
  = 0.2
 min = 0.05
d  = (  -  min) / max_t

hyperparameters = {
        'max_t':max_t,
        ' ':  , #Initial value Used in the epsilon Greedy so it will be random ␣
 ↪times per request for a next action
        ' min':  min, # epsilon decreases (there's a theory that if this decreases␣
 ↪to 0 at infinity you're ll have teh optimum solution)
        'd ': d , # dtop in epsilon per step
        ' ': 0.05, # learning rate how much of the new informatin is used to update␣
 ↪the existing value prediction
        ' ': 0.99, # discount factor, large means that the rewards in the future␣
 ↪contribute to the current action state prediction a lot
        ' ': 0.6, # trace decay for the eligibility trace 1 is MC, 0 is just the␣
 ↪last step

        'verbose': False,

        # Robot Environment params
        ' speed': pi / 2,
    'speed': 3.0,
    'n': 3
}

env = vRobEnv(
    speed=hyperparameters['speed'],
     speed=hyperparameters[' speed'],
    n=hyperparameters['n'],
    verbose=hyperparameters['verbose']
)

vqlearn = Sarsa (
    env=env,
     =hyperparameters[' '],
     =hyperparameters[' '],
     =hyperparameters[' '],
     =hyperparameters[' '],
     min=hyperparameters[' min'],
    d =hyperparameters['d '],
    q0=0,
    Tstar=0,
    max_t=1200,
    episodes=200,
    self_path='SarsaLambda.three_levels.test009.pkl',
    seed=1,
    **demoGame()
)

print(hyperparameters)
%time vqlearn.interact(resume=False, save_ep=True)
```

```
CPU times: user 7min 26s, sys: 1min 5s, total: 8min 31s
Wall time: 12min 18s
```

```
[ ]: <rl.rlln.Sarsa at 0x7fa813e048b0>
```



[ ]:

```
max_t = 2000
 = 0.2
 min = 0.05
d = ( - min) / max_t

hyperparameters = {
    'max_t':max_t,
    ' ':  , #Initial value Used in the epsilon Greedy so it will be random ␣
 ↪times per request for a next action
    ' min':  min, # epsilon decreases (there's a theory that if this decreases␣
 ↪to 0 at infinity you're ll have teh optimum solution)
    'd ': d , # dtop in epsilon per step
    ' ': 0.05, # learning rate how much of the new informatin is used to update␣
 ↪the existing value prediction
    ' ': 0.99, # discount factor, large means that the rewards in the future␣
 ↪contribute to the current action state prediction a lot
    ' ': 0.6, # trace decay for the eligibility trace 1 is MC, 0 is just the␣
 ↪last step

    'verbose': False,

    # Robot Environment params
    ' speed': pi / 2,
    'speed': 3.0,
    'n': 3
}

env = vRobEnvCornerDetector(
    speed=hyperparameters['speed'],
     speed=hyperparameters[' speed'],
    n=hyperparameters['n'],
    verbose=hyperparameters['verbose']
)

vqlearn = Sarsa (
    env=env,
     =hyperparameters[' '],
     =hyperparameters[' '],
     =hyperparameters[' '],
     =hyperparameters[' '],
     min=hyperparameters[' min'],
    d =hyperparameters['d '],
    q0=0,
    Tstar=0,
    max_t=1200,
    episodes=200,
    self_path='SarsaLambda.three_levels.test009.pkl',
    seed=1,
    **demoGame()
)

print(hyperparameters)
vqlearn.interact(resume=False, save_ep=True)
```

[ ]:

```
resume = False

if resume:
    vqlearn = Qlearn.selfload(self_path='vQlearn_exp')
    vqlearn.env = venv
    vqlearn.episodes = 105 # extend sthe number of episodes

# saving the object after each episode for retrieval in case of a crash
%time vqlearn.interact(resume=resume, save_ep=True)
```

```
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
```

```
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
```

```
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
```

```
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.6666666666666666 = SUM (-1,0,0,0.0)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
reward:-0.3333333333333333 = SUM (-1,0,0,0.2)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
File <timed eval>:1

File ~/git/turtlebot-as2/rl/rl.py:192, in MRP.interact(self, train, resume, save_ep, episodes, grid_img, **kw)
    189 self.t += 1
    190 self.t_+= 1
--> 192 rn,sn, a,an, done = self.step(s,a, self.t)  # takes a step in env and store tarjectory if needed
    193 self.online(s, rn,sn, done, a,an) if train else None # to learn online, pass a one step trajectory
    195 self.Σr += rn

File ~/git/turtlebot-as2/rl/rl.py:152, in MRP.step_an(self, s, a, t)
    150 def step_an(self, s,a, t):
    151     if self.skipstep: return 0, None, None, None, True
--> 152     sn, rn, done, _ = self.env.step(a)
    153     an = self.policy(sn)
    155     # we added s=s for compatibility with deep learning later

File ~/git/turtlebot-as2/env/robot.py:217, in RobEnv.step(self, a, speed, speed
    214 # try:
    215 # Now move and stop so that we can have a well defined actions
    216 self.spin_n(self.n if a==1 else self.n-1)
--> 217 self.stop()
    218 # except KeyboardInterrupt:
    219 #     print("Execution interrupted by user. Cleaning up…")
    221 reward, done = self.reward(a)

File ~/git/turtlebot-as2/env/robot.py:227, in RobEnv.stop(self)
    225 self.robot.linear.x = .0
    226 self.robot.angular.z = .0
--> 227 self.spin_n(self.n-1)
    228 if self.sleep: time.sleep(1.0 / 30)

File ~/git/turtlebot-as2/env/robot.py:198, in RobEnv.spin_n(self, n)
    196 for _ in range(n):
    197     self.controller.publish(self.robot)
--> 198     ros.spin_once(self)
    199     if self.sleep: time.sleep(1.0 / 30)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/__init__.py:206, in spin_once(node, executor, timeout_sec)
    204 try:
    205     executor.add_node(node)
--> 206     executor.spin_once(timeout_sec=timeout_sec)
    207 finally:
    208     executor.remove_node(node)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:751, in SingleThreadedExecutor.spin_once(self, timeout_sec)
    750 def spin_once(self, timeout_sec: float = None) -> None:
--> 751     self._spin_once_impl(timeout_sec)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:740, in SingleThreadedExecutor._spin_once_impl(self, timeout_sec)
    735 def _spin_once_impl(
    736     self,
    737     timeout_sec: Optional[Union[float, TimeoutObject]] = None
    738 ) -> None:
    739     try:
--> 740         handler, entity, node = self.wait_for_ready_callbacks(timeout_sec=timeout_sec)
    741     except ShutdownException:
    742         pass

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:723, in Executor.wait_for_ready_callbacks(self, *args, **kwargs)
    720     self._cb_iter = self._wait_for_ready_callbacks(*args, **kwargs)
    722 try:
--> 723     return next(self._cb_iter)
    724 except StopIteration:
    725     # Generator ran out of work
    726     self._cb_iter = None

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:620, in Executor._wait_for_ready_callbacks(self, timeout_sec, nodes, condition)
    617     waitable.add_to_wait_set(wait_set)
    619 # Wait for something to become ready
```

```
--> 620 wait_set.wait(timeout_nsec)
    621 if self._is_shutdown:
    622     raise ShutdownException()

KeyboardInterrupt:
```

6.6 Analysis of the results

The robot could reach goals but I couldn't get a policy which improved over time (it would not converge). I don't know if this is because of my reward, hyperparameters, or the state representation. The "solution space" is very big, and I'd like to seek guidance from people with experience as to what type of state representation and rewards should work at all - at least nudge me into the correct area. Speeding up the simulation was very useful.

7 Model 2: Either policy gradient or value-based with non-linear function approximation

7.1 RL method explanation + justification

The non linear agents are definied in rlnn.py DQN was picked, this is an action value based learning algorithm, where the agent learns to estimate the expected return (Q-value) for each possible action, and selects the action with the highest Q-value. It's suited for Non Linear control where the laser scan inputs can be passed into the NN directly, and it can learn to pull "features" out of the high dimensional laser scan data and then associate those features (say near a wall, or something curved and round ahead) to a value.

If I had more time I'd like to extend it to try and be policy based approach (this video by David Silver is really good: RL Course by David Silver - Lecture 7: Policy Gradient Methods) learning a stochastic policy might be good, because there are two goals and you want to choose randomly between them.

7.2 State representation

Previously for the linear model we had to use something which could be combinded "linearly" to create the value function approximation.

For the NN we can use the lasers without having to descritise them. I normalise them to 0 and 1 - this is generally a good idea for inputs into a NN.

```
def s_(self):
        max, min = self.max_range, self.min_range
        normalised =  ((self.scans - min)/(max - min))
        return torch.tensor(normalised, dtype=torch.float32).to(self.device)
```

7.3 Reward function

The reward function design was the same as for the Linear Model.

7.4 Hyperparameter tuning

The main parameters I changes were the h1 and h2 starting small (way to small) and increasing the size, the idea being that the NN would be able to learn the features and map them to a value. However, I think my reward wasn't working well enough, so all these were basically complete failures. Eventually I went up to 256 and also changed the reward to the one provided in the lecture note examples. However the robot would basically spin in a circle.

- `t_Qn`: Frequency (in steps) of updating the target network, used to stabilize learning alongside the main network.
- `save_weights`: Determines how often the model's weights are saved to disk, which is useful for resuming training after interruptions.
- `nbatch`: Size of the *mini-batch* sampled from the experience replay buffer for training.
- `nbuffer`: Minimum number of experiences required in the *replay buffer* before learning commences.
- `h1`: Size of the first hidden layer (set to `0` for no hidden layer).
- `h2`: Size of the second hidden layer (also settable to `0` if not needed).

7.5 Learning plots + success rates

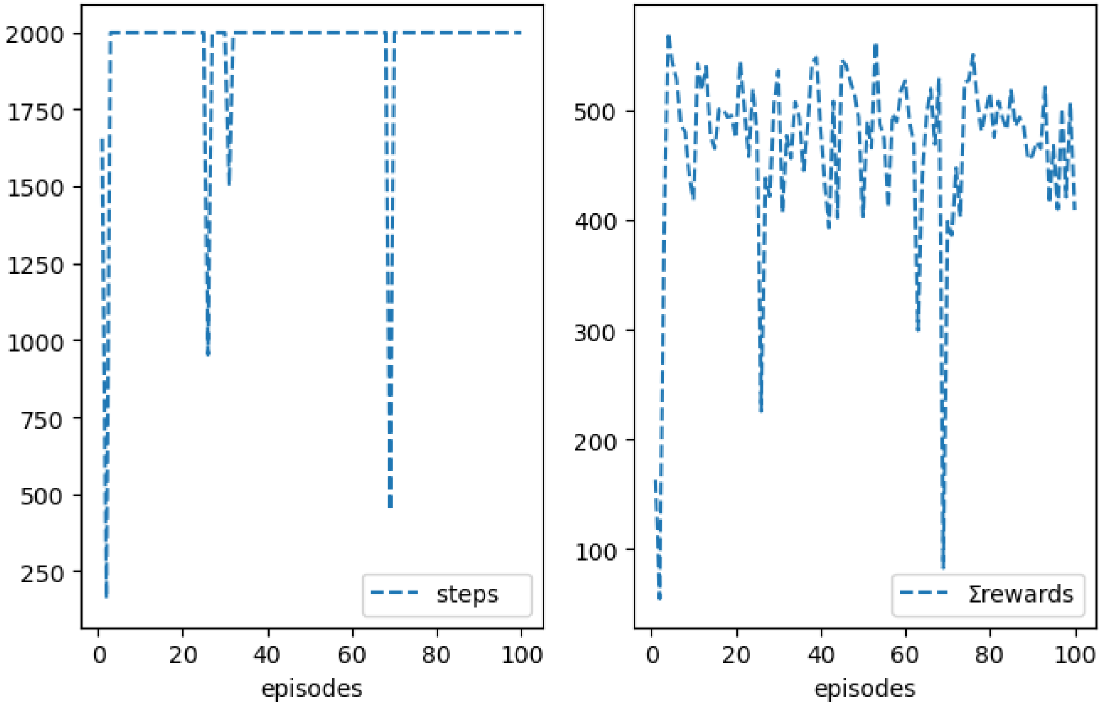
See below, there's very little success. Most maxed out to the max episodes.

[ ]:
```
class nnRobEnv(vRobEnv):
    def __init__(self, **kw):
        self.device = torch.device('mps' if torch.backends.mps.is_available()␣
 ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
        super().__init__(**kw)
        self.nF = len(self.scans)

    def s_(self):
        max, min = self.max_range, self.min_range
        normalised =  ((self.scans - min)/(max - min))
        return torch.tensor(normalised, dtype=torch.float32).to(self.device)

env = nnRobEnv()

class cudaDQN(nnMDP):
    def __init__(self, =1e-4, t_Qn=1000, **kw):
        print('---------------------    cudaDQN is being set up ␣
 ↪----------------------')
        self.device = torch.device('mps' if torch.backends.mps.is_available()␣
 ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
        super().__init__(**kw)
        self. =
        self.store = True
        self.t_Qn = t_Qn

    def greedy(self, s):
        self.isamax = True
        Qs = self.Q_(s)
        Qs_np = Qs.detach().cpu().numpy() #WE NEED TO BRING THE DATA BACK FROM THE GPU for NUMPy
        from numpy.random import choice
        return choice((Qs_np == Qs_np.max()).nonzero()[0])

    def online(self, *args):
        if len(self.buffer) < self.nbatch:
            return

        (s, a, rn, sn, dones), inds = self.batch()

        Qs = self.qN(s)
        Qn = self.qNn(sn).detach()
        Qn[dones] = 0

        target = Qs.clone().detach()
        target[inds, a] = self.  * Qn.max(1).values + rn.to(self.device)
        loss = self.qN.fit(Qs, target)

        if self.t_ % self.t_Qn == 0:
            self.qNn.set_weights('Q', self.t_)
            print(f'loss = {loss}')


nnqlearn = cudaDQN(
    env=env,
    episodes=100,
     =1e-4,
     =0.5,
    d =.99,
     min=0.01,
     =.95,
    h1=3,
    h2=3,
    nF=env.nF,
    nbuffer=5000,
    nbatch=32,
    endbatch=8,
    t_Qn=100,
    self_path='DQN_exp.pkl',
    seed=1,
    **demoGame())

for layer in nnqlearn.qN.layers:
    print(layer.weight)
    # print(layer.bias)
```

```
%time nnqlearn.interact(resume=False, save_ep=True)
```

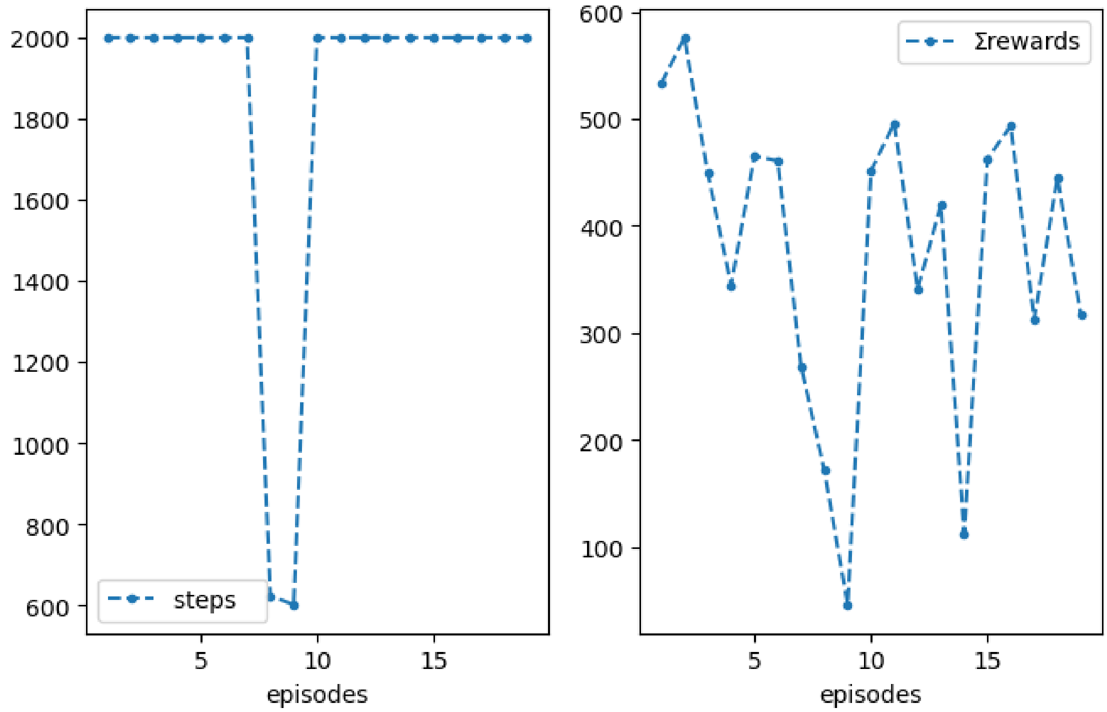
```
CPU times: user 41min 34s, sys: 5min 16s, total: 46min 50s
Wall time: 1h 2min
```

```
[ ]: <__main__.cudaDQN at 0x7fa866e50bb0>
```



```
[5]: class nnRobEnv(vRobEnv):
         def __init__(self, **kw):
             self.device = torch.device('mps' if torch.backends.mps.is_available()␣
      ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
             super().__init__(**kw)
             self.nF = len(self.scans)

         def s_(self):
             max, min = self.max_range, self.min_range
             normalised =  ((self.scans - min)/(max - min))
             return torch.tensor(normalised, dtype=torch.float32).to(self.device)

     env = nnRobEnv()

     class cudaDQN(nnMDP):
```

```
    def __init__(self,  =1e-4, t_Qn=1000, **kw):
        print('----------------------   cudaDQN is being set up ␣
 ↪-----------------------')
        self.device = torch.device('mps' if torch.backends.mps.is_available()␣
 ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
        super().__init__(**kw)
        self.  =
        self.store = True
        self.t_Qn = t_Qn

    def greedy(self, s):
        self.isamax = True
        Qs = self.Q_(s)
        Qs_np = Qs.detach().cpu().numpy() #WE NEED TO BRING THE DATA BACK FROM␣
 ↪THE GPU for NUMPy
        from numpy.random import choice
        return choice((Qs_np == Qs_np.max()).nonzero()[0])

    def online(self, *args):
        if len(self.buffer) < self.nbatch:
            return

        (s, a, rn, sn, dones), inds = self.batch()

        Qs = self.qN(s)
        Qn = self.qNn(sn).detach()
        Qn[dones] = 0

        target = Qs.clone().detach()
        target[inds, a] = self.  * Qn.max(1).values + rn.to(self.device)
        loss = self.qN.fit(Qs, target)

        if self.t_ % self.t_Qn == 0:
            self.qNn.set_weights('Q', self.t_)
            print(f'loss = {loss}')


#increase training time,
nnqlearn = cudaDQN(
    env=env,
    episodes=300,
     =5e-4,
     =0.9,
    d =0.995,
     min=0.05,
     =0.99,
    h1=64,
    h2=64,
    nF=env.nF,
    nbuffer=10000,
    nbatch=32,
    endbatch=8,
    t_Qn=300,
    self_path='DQN_exp.pkl',
    seed=1,
    **demoGame())

for layer in nnqlearn.qN.layers:
    print(layer.weight)
    # print(layer.bias)


%time nnqlearn.interact(resume=False, save_ep=True)
```



```
could not save the file {self.self_path}

/home/danb/git/turtlebot-as2/rl/rlnn.py:132: UserWarning: To copy construct from
a tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  torch.tensor(s,    dtype=torch.float32),
/home/danb/git/turtlebot-as2/rl/rlnn.py:135: UserWarning: To copy construct from
a tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  torch.tensor(sn,   dtype=torch.float32),
```

```
update Q network weights…! at 35400
loss = 0.014120114967226982
update Q network weights…! at 35700
loss = 0.026033079251646996
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
File <timed eval>:1

File ~/git/turtlebot-as2/rl/rl.py:193, in MRP.interact(self, train, resume,␣
 ↪save_ep, episodes, grid_img, **kw)
    190 self.t += 1
    191 self.t_+= 1
--> 193 rn,sn, a,an, done = self.step(s,a, self.t)  # takes a step in env and␣
 ↪store tarjectory if needed
    194 self.online(s, rn,sn, done, a,an) if train else None # to learn online,
 ↪pass a one step trajectory
    196 self.Σr += rn

File ~/git/turtlebot-as2/rl/rl.py:142, in MRP.step_a(self, s, _, t)
    140 if self.skipstep: return 0, None, None, None, True
    141 a = self.policy(s)
--> 142 sn, rn, done, _ = self.env.step(a)
    144 # we added s=s for compatibility with deep learning
    145 self.store_(s=s, a=a, rn=rn, sn=sn, done=done, t=t)

File ~/git/turtlebot-as2/env/robot.py:224, in RobEnv.step(self, a, speed,  speed
    221 # try:
    222 # Now move and stop so that we can have a well defined actions
    223 self.spin_n(self.n if a==1 else self.n-1)
--> 224 self.stop()
    225 # except KeyboardInterrupt:
    226 #     print("Execution interrupted by user. Cleaning up…")
    228 reward, done = self.reward(a)

File ~/git/turtlebot-as2/env/robot.py:234, in RobEnv.stop(self)
    232 self.robot.linear.x = .0
    233 self.robot.angular.z = .0
--> 234 self.spin_n(self.n-1)
    235 if self.sleep: time.sleep(1.0 / 30)

File ~/git/turtlebot-as2/env/robot.py:205, in RobEnv.spin_n(self, n)
```

```
    203 for _ in range(n):
    204     self.controller.publish(self.robot)
--> 205     ros.spin_once(self)
    206     if self.sleep: time.sleep(1.0 / 30)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/__init__.py:206,␣
  ↪in spin_once(node, executor, timeout_sec)
    204 try:
    205     executor.add_node(node)
--> 206     executor.spin_once(timeout_sec=timeout_sec)
    207 finally:
    208     executor.remove_node(node)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:751,␣
  ↪in SingleThreadedExecutor.spin_once(self, timeout_sec)
    750 def spin_once(self, timeout_sec: float = None) -> None:
--> 751     self._spin_once_impl(timeout_sec)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:748,␣
  ↪in SingleThreadedExecutor._spin_once_impl(self, timeout_sec)
    746 handler()
    747 if handler.exception() is not None:
--> 748     raise handler.exception()

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/task.py:254, in␣
  ↪Task.__call__(self)
    251 if inspect.iscoroutine(self._handler):
    252     # Execute a coroutine
    253     try:
--> 254         self._handler.send(None)
    255     except StopIteration as e:
    256         # The coroutine finished; store the result
    257         self.set_result(e.value)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:440,␣
  ↪in Executor._make_handler.<locals>.handler(entity, gc, is_shutdown,␣
  ↪work_tracker)
    438     return
    439 with work_tracker:
--> 440     arg = take_from_wait_list(entity)
    442     # Signal that this has been 'taken' and can be added back to the␣
  ↪wait list
    443     entity._executor_event = False

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/executors.py:365,␣
  ↪in Executor._take_subscription(self, sub)
    363 def _take_subscription(self, sub):
    364     with sub.handle:
```

```
--> 365         msg_info = sub.handle.take_message(sub.msg_type, sub.raw)
    366         if msg_info is not None:
    367             return msg_info[0]

RuntimeError: Unable to convert call argument to Python object (compile in debug␣
  ↪mode for details)
```

[7]:

```
class nnRobEnv(vRobEnv):
    def __init__(self, **kw):
        self.device = torch.device('mps' if torch.backends.mps.is_available()␣
 ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
        super().__init__(**kw)
        self.nF = len(self.scans)

    def reward_(self, a):
        # s_type 0-reached a goal, 1-hits a wall 2-moved forward or 3-turn
        s_type = [self.atgoal(), self.atwall(), a==1, a!=1].index(True)

        # identify the nearest goal and obtain the distance and the angular␣
 ↪distance to it
        dist, goal = self.distgoal()
         goal = self. goal(goal)

        #  reward/penalise robot relative to its orientation towards a goal
         dist = round(abs(abs(self.  - goal) - pi*goal), 2) # subtract pi if␣
 ↪it's goal 1 (behind)

        reward = self.rewards[s_type]
        reward = self.reward(reward, dist,  dist, s_type)

        return reward

    def s_(self):
        max, min = self.max_range, self.min_range
        normalised =  ((self.scans - min)/(max - min))
        return torch.tensor(normalised, dtype=torch.float32).to(self.device)

env = nnRobEnv()

class cudaDQN(nnMDP):
    def __init__(self,  =1e-4, t_Qn=1000, **kw):
        print('----------------------    cudaDQN is being set up  ␣
 ↪-----------------------')
        self.device = torch.device('mps' if torch.backends.mps.is_available()␣
 ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
        super().__init__(**kw)
        self.  = 
        self.store = True
        self.t_Qn = t_Qn

    def greedy(self, s):
        self.isamax = True
        Qs = self.Q_(s)
        Qs_np = Qs.detach().cpu().numpy() #WE NEED TO BRING THE DATA BACK FROM␣
 ↪THE GPU for NUMPy
        from numpy.random import choice
        return choice((Qs_np == Qs_np.max()).nonzero()[0])

    def online(self, *args):
        if len(self.buffer) < self.nbatch:
            return

        (s, a, rn, sn, dones), inds = self.batch()

        Qs = self.qN(s)
        Qn = self.qNn(sn).detach()
        Qn[dones] = 0

        target = Qs.clone().detach()
        target[inds, a] = self.  * Qn.max(1).values + rn.to(self.device)
        loss = self.qN.fit(Qs, target)

        if self.t_ % self.t_Qn == 0:
            self.qNn.set_weights('Q', self.t_)
            print(f'loss = {loss}')


#increase training time,
nnqlearn = cudaDQN(
    env=env,
    episodes=300,
     =5e-4,
     =0.9,
    d =0.995,
     min=0.05,
     =0.99,
    h1=256,
    h2=256,
    nF=env.nF,
    nbuffer=10000,
    nbatch=32,
    endbatch=8,
    t_Qn=300,
    self_path='DQN_exp2.pkl',
    seed=1,
    **demoGame())

for layer in nnqlearn.qN.layers:
    print(layer.weight)
    # print(layer.bias)


%time nnqlearn.interact(resume=False, save_ep=True)
```

```
[WARN] [1745829328.396560857] [rcl.logging_rosout]: Publisher already registered
for provided node name. If this is due to multiple nodes with the same name then
all logs for that logger name will go out over the existing publisher. As soon
as any node with that name is destructed it will unregister the publisher,
preventing any further logs for that name from being published on the rosout
topic.
```

```
speed  =  2.0
 speed =  1.57
state size(laser beams)= 64
---------------------   cudaDQN is being set up   -----------------------
Model on device:cuda

               Model Architecture for Q net

  Id  Layer                           Parameters  Trainable

   0  layers.0.weight                     16,384  Yes
   1  layers.0.bias                          256  Yes
   2  layers.1.weight                     65,536  Yes
   3  layers.1.bias                          256  Yes
   4  layers.2.weight                     16,384  Yes
   5  layers.2.bias                           64  Yes
   6  layers.3.weight                        192  Yes
   7  layers.3.bias                            3  Yes

 Total Parameters:        99,075 | Trainable:         99,075

Model on device:cuda

               Model Architecture for Qn net

  Id  Layer                           Parameters  Trainable

   0  layers.0.weight                     16,384  Yes
   1  layers.0.bias                          256  Yes
   2  layers.1.weight                     65,536  Yes
   3  layers.1.bias                    256  Yes
   4  layers.2.weight               16,384  Yes
   5  layers.2.bias                     64  Yes
   6  layers.3.weight                  192  Yes
   7  layers.3.bias                      3  Yes

 Total Parameters:       99,075 | Trainable:        99,075

Parameter containing:
tensor([[ 0.0764, -0.1153,  0.0458,  …, -0.0125, -0.0370,  0.1180],
        [-0.0098,  0.0229,  0.0124,  …, -0.0301, -0.0701, -0.0039],
        [-0.0227, -0.1144,  0.0558,  …,  0.0095,  0.0368, -0.1171],
        …,
        [ 0.0394, -0.0987,  0.0799,  …, -0.1005, -0.0236, -0.0306],
        [ 0.1058, -0.0113,  0.0374,  …,  0.0546, -0.0075, -0.0898],
        [-0.0825, -0.0742,  0.0318,  …, -0.0499, -0.0408, -0.0932]],
       device='cuda:0', requires_grad=True)
Parameter containing:
tensor([[-0.0282,  0.0331,  0.0120,  …, -0.0623, -0.0014, -0.0086],
        [-0.0344, -0.0619, -0.0021,  …,  0.0449,  0.0494, -0.0168],
        [-0.0206,  0.0425,  0.0459,  …, -0.0438, -0.0411, -0.0339],
        …,
        [ 0.0184, -0.0429,  0.0029,  …,  0.0570, -0.0468, -0.0449],
        [ 0.0526,  0.0466,  0.0437,  …,  0.0213, -0.0251,  0.0406],
        [-0.0553,  0.0456,  0.0526,  …, -0.0064,  0.0170,  0.0009]],
       device='cuda:0', requires_grad=True)
Parameter containing:
tensor([[ 0.0172,  0.0133,  0.0283,  …, -0.0158, -0.0372,  0.0470],
        [ 0.0623, -0.0458, -0.0443,  …, -0.0183, -0.0266,  0.0243],
        [ 0.0369, -0.0468,  0.0389,  …,  0.0425,  0.0130, -0.0200],
        …,
        [-0.0311, -0.0540,  0.0028,  …, -0.0308, -0.0548, -0.0235],
        [-0.0402, -0.0327,  0.0121,  …, -0.0405,  0.0331,  0.0184],
        [ 0.0452, -0.0605,  0.0485,  …,  0.0259, -0.0440,  0.0513]],
       device='cuda:0', requires_grad=True)
Parameter containing:
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],
       device='cuda:0', requires_grad=True)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File <timed eval>:1

File ~/git/turtlebot-as2/rl/rl.py:193, in MRP.interact(self, train, resume,␣
↪save_ep, episodes, grid_img, **kw)
    190 self.t += 1
    191 self.t_+= 1
--> 193 rn,sn, a,an, done = self.step(s,a, self.t)  # takes a step in env and␣
↪store tarjectory if needed
    194 self.online(s, rn,sn, done, a,an) if train else None # to learn online,␣
↪pass a one step trajectory
    196 self.Σr += rn

File ~/git/turtlebot-as2/rl/rl.py:142, in MRP.step_a(self, s, _, t)
    140 if self.skipstep: return 0, None, None, None, True
    141 a = self.policy(s)
--> 142 sn, rn, done, _ = self.env.step(a)
    144 # we added s=s for compatibility with deep learning
    145 self.store_(s=s, a=a, rn=rn, sn=sn, done=done, t=t)

File ~/git/turtlebot-as2/env/robot.py:228, in RobEnv.step(self, a, speed, speed
    224 self.stop()
    225 # except KeyboardInterrupt:
    226 #     print("Execution interrupted by user. Cleaning up…")
--> 228 reward, done = self.reward(a)
    229 return self.s_(), reward, done, {}

File ~/git/turtlebot-as2/env/robot.py:338, in RobEnv.reward(self, a)
    335 def reward(self, a):
    336     # keep the order as is to benefit from distances calculation
    337     done = self.goal_seeking()
--> 338     reward = self.reward_(a)
    339     return reward, done

Cell In[7], line 9, in nnRobEnv.reward_(self, a)
      7 def reward_(self, a):
      8     # s_type 0-reached a goal, 1-hits a wall 2-moved forward or 3-turn
----> 9     s_type = [self.atgoal(), self.atwall(), a==1, a!=1].index(True)
     11     # identify the nearest goal and obtain the distance and the angular␣
↪distance to it
     12     dist, goal = self.distgoal()
```

```
TypeError: RobEnv.atgoal() missing 1 required positional argument: 'goal_dist'
```

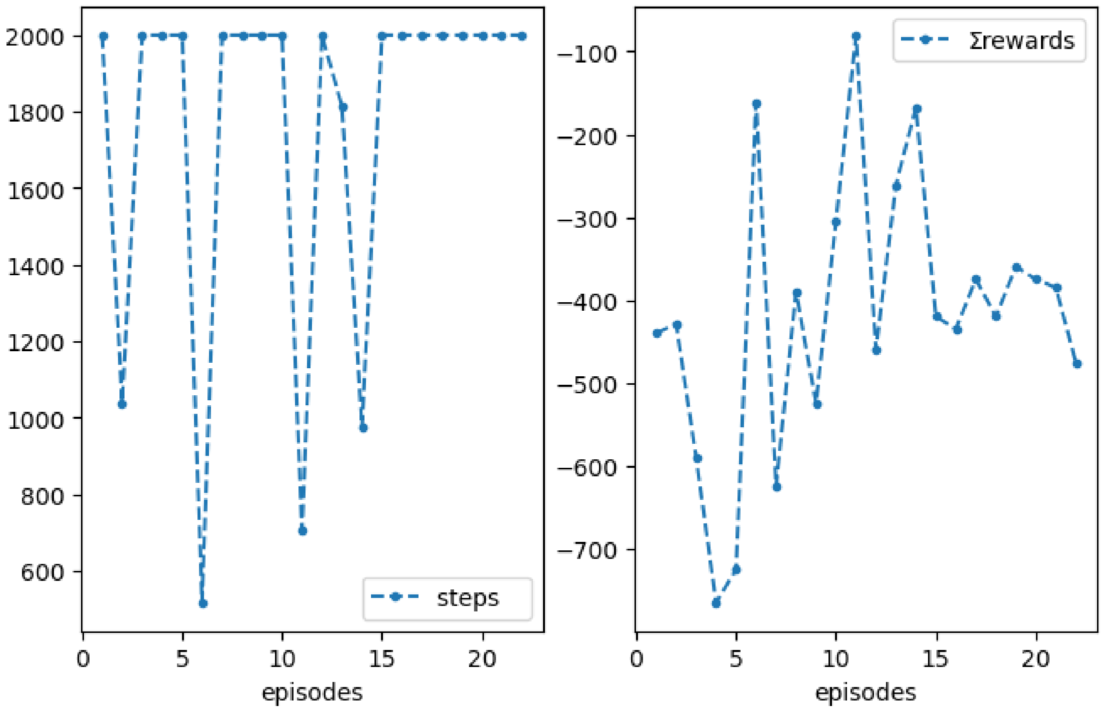
[8]:

```
#accelerate_sim(speed=100)
set_nscans_LiDAR(nscans=360)
```

[9]:

```
class nnRobEnv(vRobEnv):
    def __init__(self, **kw):
        self.device = torch.device('mps' if torch.backends.mps.is_available()␣
 ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
        super().__init__(**kw)
        self.nF = len(self.scans)

    def reward_(self, a):

        if not hasattr(self, 'Δgoal_dist'):
            return 0

        #Don't like steps
        per_step_reward = -0.1

        #Dont like getting too close to walls
        anti_crash_into_wall_reward = -5 * self.nearly_atwall()

        #The goal is great
        goal_reached_reward = 10 * self.atgoal(self.goal_dist)

        reward = sum([
            per_step_reward,
            anti_crash_into_wall_reward,
            goal_reached_reward])

        if self.verbose and reward>-1:
            print(f"per_step_reward: {per_step_reward}, ")
            print(f"anti_crash_into_wall_reward: {anti_crash_into_wall_reward}")
            print(f"goal_reached_reward: {goal_reached_reward}")
            print('reward =', reward)#; print(f'action = {a}')

        return reward


    def s_(self):
        max, min = self.max_range, self.min_range
        normalised =  ((self.scans - min)/(max - min))
        return torch.tensor(normalised, dtype=torch.float32).to(self.device)

env = nnRobEnv()
```

```
class cudaDQN(nnMDP):
    def __init__(self,  =1e-4, t_Qn=1000, **kw):
        print('---------------------    cudaDQN is being set up ␣
 ↪-----------------------')
        self.device = torch.device('mps' if torch.backends.mps.is_available()␣
 ↪else 'cuda' if torch.cuda.is_available() else 'cpu')
        super().__init__(**kw)
        self.  =
        self.store = True
        self.t_Qn = t_Qn

    def greedy(self, s):
        self.isamax = True
        Qs = self.Q_(s)
        Qs_np = Qs.detach().cpu().numpy() #WE NEED TO BRING THE DATA BACK FROM␣
 ↪THE GPU for NUMPy
        from numpy.random import choice
        return choice((Qs_np == Qs_np.max()).nonzero()[0])

    def online(self, *args):
        if len(self.buffer) < self.nbatch:
            return

        (s, a, rn, sn, dones), inds = self.batch()

        Qs = self.qN(s)
        Qn = self.qNn(sn).detach()
        Qn[dones] = 0

        target = Qs.clone().detach()
        target[inds, a] = self.  * Qn.max(1).values + rn.to(self.device)
        loss = self.qN.fit(Qs, target)

        if self.t_ % self.t_Qn == 0:
            self.qNn.set_weights('Q', self.t_)
            print(f'loss = {loss}')


#increase training time,
nnqlearn = cudaDQN(
    env=env,
    episodes=300,
     =5e-4,
     =0.9,
    d =0.995,
     min=0.05,
     =0.99,
    h1=64,
    h2=64,
    nF=env.nF,
    nbuffer=10000,
    nbatch=32,
    endbatch=8,
    t_Qn=300,
    self_path='DQN_exp.pkl',
    seed=1,
    **demoGame())

for layer in nnqlearn.qN.layers:
    print(layer.weight)
    # print(layer.bias)


%time nnqlearn.interact(resume=False, save_ep=True)
```



```
could not save the file {self.self_path}
```

```
/home/danb/git/turtlebot-as2/rl/rlnn.py:132: UserWarning: To copy construct from
a tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
```

```
  torch.tensor(s,    dtype=torch.float32),
/home/danb/git/turtlebot-as2/rl/rlnn.py:135: UserWarning: To copy construct from
a tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  torch.tensor(sn,   dtype=torch.float32),
```

```
update Q network weights…! at 39300
loss = 0.0035610822960734367
update Q network weights…! at 39600
loss = 0.004673745948821306
update Q network weights…! at 39900
loss = 0.014230512082576752
update Q network weights…! at 40200
loss = 0.2563660740852356
update Q network weights…! at 40500
loss = 0.06051730364561081
update Q network weights…! at 40800
loss = 0.26349371671676636
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
File <timed eval>:1

File ~/git/turtlebot-as2/rl/rl.py:193, in MRP.interact(self, train, resume,␣
 ↪save_ep, episodes, grid_img, **kw)
    190 self.t += 1
    191 self.t_+= 1
--> 193 rn,sn, a,an, done = self.step(s,a, self.t)  # takes a step in env and␣
 ↪store tarjectory if needed
    194 self.online(s, rn,sn, done, a,an) if train else None # to learn online,
 ↪pass a one step trajectory
    196 self.Σr += rn

File ~/git/turtlebot-as2/rl/rl.py:142, in MRP.step_a(self, s, _, t)
    140 if self.skipstep: return 0, None, None, None, True
    141 a = self.policy(s)
--> 142 sn, rn, done, _ = self.env.step(a)
    144 # we added s=s for compatibility with deep learning
    145 self.store_(s=s, a=a, rn=rn, sn=sn, done=done, t=t)

File ~/git/turtlebot-as2/env/robot.py:223, in RobEnv.step(self, a, speed, speed
    219 elif a == 2: self.robot.angular.z = - speed  # turn right
    221 # try:
    222 # Now move and stop so that we can have a well defined actions
--> 223 self.spin_n(self.n if a==1 else self.n-1)
    224 self.stop()
    225 # except KeyboardInterrupt:
```

```
    226 #     print("Execution interrupted by user. Cleaning up…")

File ~/git/turtlebot-as2/env/robot.py:204, in RobEnv.spin_n(self, n)
    202 def spin_n(self, n):
    203     for _ in range(n):
--> 204         self.controller.publish(self.robot)
    205         ros.spin_once(self)
    206         if self.sleep: time.sleep(1.0 / 30)

File /opt/ros/humble/local/lib/python3.10/dist-packages/rclpy/publisher.py:70, in Publisher.publish(self, msg)
     68 with self.handle:
     69     if isinstance(msg, self.msg_type):
---> 70         self.__publisher.publish(msg)
     71     elif isinstance(msg, bytes):
     72         self.__publisher.publish_raw(msg)

KeyboardInterrupt:
```

7.6 Analysis of the results

I got this working on my local GPU which was good, I learn't a lot about the .to device and tensors.

However, with regard to the actual learn't policy, not very good at all. I really do think it's my reward function, I need to investigate more.

8 Conclusion and Reflection on what went wrong and what worked well

I've learnt a lot about the messy world of robotics, and it is very frustrating. I think if this was a paired team project I might have performed better. My challenges were

- It took a while to understand ROS, its components and how they interacted.
- I wanted to run this on my own machine, this took a long time to work out, but I'm pleased I did.
- Understanding the RL models (linear vs non-linear) code - I really had to explore and understand the structure, once I spent time on the code base then it clicked, but it took a while.
- I'm still very unsure about how to design the rewards properly I highly suspect this contributed to the bad policy improvement.
- I'm not sure about the state representation, I think this is a wide area, there's so many different ways to approach this I wasn't sure.
- Hyperparameter tuning took so long because the solution space is massive.
- I left it too late and ran out of time, I could have done with another couple of days (my fault)

Positives

I've learnt a lot about the application of the theory. I'd keen to try and spend more time on this type of project.

9 References

- RL Course by David Silver - Lecture 7: Policy Gradient Methods - reinforcement learning. YouTube video, Available at: https://www.youtube.com/watch?v=KHZVXao4qXs [Accessed 28 April 2025].
- Núñez, P., Vazquez-Martin, R., Bandera, A., and Romero-Gonzalez, C. (2015) 'Feature extraction from laser scan data based on curvature estimation for mobile robotics', *Robotics and Autonomous Systems*, 70, pp. 103–114. Available at: https://robolab.unex.es/wp-content/papercite-data/pdf/feature-extraction-from-laser.pdf (Accessed: 26 April 2025).
- Ramos, J., Rocha, R., and Dias, J. (2022) 'Efficient approach for extracting high-level B-spline features from laser scan data', *Sensors*, 22(24), 9737. Available at: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9737135/` (Accessed: 26 April 2025).
- Shen, S., Michael, N., and Kumar, V. (2012) 'Method for corner feature extraction from laser scan data', *ResearchGate*. Available at: `https://www.researchgate.net/publication/288577925_Method_for_corner_feature_extraction_from_laser_scan_data` (Accessed: 26 April 2025).
- Stack Overflow (2019) 'How can I detect the corner from 2D point cloud or LiDAR scanned data?', *Stack Overflow*. Available at: `https://stackoverflow.com/questions/59049990/how-can-i-detect-the-corner-from-2d-point-cloud-or-lidar-scanned-data` (Accessed: 26 April 2025).
- CETI. (n.d.) *Simulation Speed in ROS/Gazebo*. Available at: `https://ceti.pages.st.inf.tu-dresden.de/robotics/howtos/SimulationSpeed.html` (Accessed: 26 April 2025).
- Furrer, F., Wermelinger, M., Naegeli, T., et al. (2021) 'Dynamics and Control of Quadrotor UAVs: A Survey', *IEEE Transactions on Robotics*, 37(5), pp. 1381–1400. Available at: `https://ieeexplore.ieee.org/document/9453594` (Accessed: 26 April 2025).
- Perez-Perez, J., Jimenez, F. and Mata, M. (2023) 'An Overview of Reinforcement Learning in Autonomous Driving: Fundamentals, Challenges, and Applications', *Applied Sciences*, 13(12), p. 7202. Available at: `https://www.mdpi.com/2076-3417/13/12/7202` (Accessed: 26 April 2025).

10 Appendicies

10.1 Cool links / interesting reading:

- https://github.com/hello-robot/stretch_ros/blob/master/stretch_funmap/README.md
- https://arxiv.org/pdf/2502.20607

10.2 Miscelaneous Notes

10.2.1 Setting up ROS

- https://emanual.robotis.com/docs/en/platform/turtlebot3/sbc_setup/

- https://ros2-industrial-workshop.readthedocs.io/en/latest/_source/navigation/ROS2-Turtlebot.html
- https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/
- https://emanual.robotis.com/docs/en/platform/turtlebot3/bringup/#bringup

10.2.2 Multicast traffic (for DDS) through Windows FW to WSL2:

- https://eprosima-dds-router.readthedocs.io/en/latest/rst/examples/repeater_example.html#execute-example
- New-NetFirewallRule -Name 'WSL' -DisplayName 'WSL' -InterfaceAlias 'vEthernet (WSL (Hyper-V firewall))' -Direction Inbound -Action Allow
- New-NetIPAddress -InterfaceAlias 'vEthernet (WSL (Hyper-V firewall))' -IPAddress '192.168.1.217' -PrefixLength 24
- https://github.com/DanielBryars/multicast-test.git

10.2.3 VM

- https://labs.azure.com/virtualmachines?feature_vnext=true

10.3 Framework modifications

10.3.1 Ignore Reset parameter

Modified RobEnv so that I can run it in "passive" mode for monitoring what the state of the robot and rewards are doing in realtime

```
self.ignoreReset

self.reset_world = self.create_client(Empty, '/reset_world')
        if (ignoreReset):
            print("ignoreReset is True Skipping world reset")
        else:
            while not self.reset_world.wait_for_service(timeout_sec=4.0):
                print('world client service...')

def reset(self):
        '''Override the original so we can skip the reset (used for monitoring applications)''
        if (self.ignoreReset):
            print("Reset called BUT self.ignoreReset is True, so ignoring")
            return self.s_()
        else:
            return super().reset()
```

10.3.2 modify to save a picture and a json file of the parameters work in progress.

```
def plot_ep(self, animate=None, plot_exp=False, label='', savefig=False):
        if len(self.eplist)< self.episodes: self.eplist.append(self.ep+1)

        if animate is None: animate = self.animate
        if not animate: return
        frmt='.--'if not plot_exp or self.ep==0 else '--'

        if self.visual:
            if self.ep==self.episodes-1: self.render(animate=False) # shows the policy
            else:                        self.env.render(animate=False)
        if self.plotV:  self.plot_V(ep=self.ep+1)

        i=2
        for plot, ydata, label_ in zip([self.plotT, self.plotR, self.plotE],
                                       [self.Ts,    self.Rs,    self.Es   ],
                                       ['steps   ', 'Σrewards', 'Error   ']):
            if not plot: continue
            plt.subplot(1,3,min(i,3)).plot(self.eplist[:self.ep+1], ydata[:self.ep+1], frmt, la
            plt.xlabel('episodes')
            plt.legend()
            i+=1

        # if there is any visualisation required then we need to care for special cases
        if self.plotV or self.plotE or self.plotT or self.plotR:
            figsizes = list(zip(plt.gcf().get_size_inches(), self.env.figsize0))
            figsize  = [max(figsizes[0]), min(figsizes[1]) if self.plotV or self.plotE else fig
            plt.gcf().set_size_inches(figsize[0], figsize[1])
            clear_output(wait=True)
            if not plot_exp:
                plt.show()
            else:
                if savefig:
                    descriptive_name = f'{self.desctime}.{self.__class__.__name__}.episode.{sel
                    safe_params = self.make_json_safe(self.params)
                    with open(f'{descriptive_name}.json', 'w') as f:
                        json.dump(safe_params, f, indent=2)
                    plt.savefig(f'{descriptive_name}.png')
```

There were a couple of bugs which I fixed

10.3.3 Cuda/Device movements in the nn stuff

10.3.4 Race for state not seen

```
'''
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
File <timed eval>:1

File ~/git/turtlebot-as2/rl/rl.py:192, in MRP.interact(self, train, resume, save_ep, episodes,
    189 self.t += 1
    190 self.t_+= 1
--> 192 rn,sn, a,an, done = self.step(s,a, self.t)  # takes a step in env and store tarjectory
    193 self.online(s, rn,sn, done, a,an) if train else None # to learn online, pass a one step
    195 self.Σr += rn

File ~/git/turtlebot-as2/rl/rl.py:153, in MRP.step_an(self, s, a, t)
    151 if self.skipstep: return 0, None, None, None, True
    152 sn, rn, done, _ = self.env.step(a)
--> 153 an = self.policy(sn)
    155 # we added s=s for compatibility with deep learning later
    156 self.store_(s=s, a=a, rn=rn, sn=sn, an=an, done=done, t=t)

File ~/git/turtlebot-as2/rl/rl.py:487, in MDP.<locals>.MDP. greedy(self, s)
    484 if self.d  < 1: self.  = max(self. min, self. *self.d )              # exponential decay
    485 if self. T > 0: self.  = max(self. min, self. 0 - self.t_ / self. T) # linear      decay
--> 487 return self.greedy(s) if rand() > self.  else randint(0, self.env.nA)

File ~/git/turtlebot-as2/rl/rl.py:477, in MDP.<locals>.MDP.greedy(self, s)
    474 # print(s)
    475 # print(Qs)
    476 if Qs.shape[0]==1: raise ValueError('something might be wrong number of actions ==1')
--> 477 return choices(np.where(Qs==Qs.max())[0])[0]

File /usr/lib/python3.10/random.py:519, in Random.choices(self, population, weights, cum_weigh
    517     floor = _floor
    518     n += 0.0    # convert to float for a small speed improvement
--> 519     return [population[floor(random() * n)] for i in _repeat(None, k)]
    520 try:
    521     cum_weights = list(_accumulate(weights))

File /usr/lib/python3.10/random.py:519, in <listcomp>(.0)
    517     floor = _floor
    518     n += 0.0    # convert to float for a small speed improvement
--> 519     return [population[floor(random() * n)] for i in _repeat(None, k)]
    520 try:
    521     cum_weights = list(_accumulate(weights))

IndexError: index 0 is out of bounds for axis 0 with size 0
```

FIXED BY EDITING rl.py

```
#------------------------------------ add some more policies types -------------------------
        # useful for inheritance, gives us a vector of actions values
        def Q_(self, s=None, a=None):

            #Originally return self.Q[s] if s is not None else self.Q

            if s is None:
                return self.Q

            #just initialise to 0 for now, not sure how to handle this.
            if s not in self.Q:
                self.Q[s] = np.zeros(self.env.nA)

            return self.Q[s]

'''
```

10.3.5 S_ instead of s_ in rl.py

```
--------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
File <timed eval>:1

File ~/git/turtlebot-as2/rl/rl.py:182, in MRP.interact(self, train, resume, save_ep, episodes,
    179 done = False
    180 #print(self.ep)
    181 # initial step
--> 182 s,a = self.step_0()
    183 self.step0()                                    # user defined init of each episode
    184 # an episode is a set of steps, interact and learn from experience, online or offline.

File ~/git/turtlebot-as2/rl/rl.py:134, in MRP.step_0(self)
    132 def step_0(self):
    133     s = self.env.reset()                                 # set env/agent to the start p
--> 134     a = self.policy(s)
    135     return s,a

File ~/git/turtlebot-as2/rl/rl.py:499, in MDP.<locals>.MDP. greedy(self, s)
    496 if self.d  < 1: self.  = max(self. min, self. *self.d )            # exponential decay
    497 if self. T > 0: self.  = max(self. min, self. 0 - self.t_ / self. T) # linear      decay
--> 499 return self.greedy(s) if rand() > self.  else randint(0, self.env.nA)

File ~/git/turtlebot-as2/rl/rl.py:485, in MDP.<locals>.MDP.greedy(self, s)
    483 self.isamax = True
    484 # instead of returning np.argmax(Q[s]) get all max actions and return one of the max a
--> 485 Qs = self.Q_(s)
    486 # print(s)
    487 # print(Qs)
    488 if Qs.shape[0]==1: raise ValueError('something might be wrong number of actions ==1')

File ~/git/turtlebot-as2/rl/rlln.py:149, in vMDP.Q_(self, s, a)
    145 def Q_(self, s=None, a=None):
    146     #print(f"{s.shape}, {a}")
    148     W = self.W if a is None else self.W[a]
--> 149     return W.dot(s) if s is not None else np.matmul(W, self.env.S_()).T

AttributeError: 'vRobEnv' object has no attribute 'S_'
```