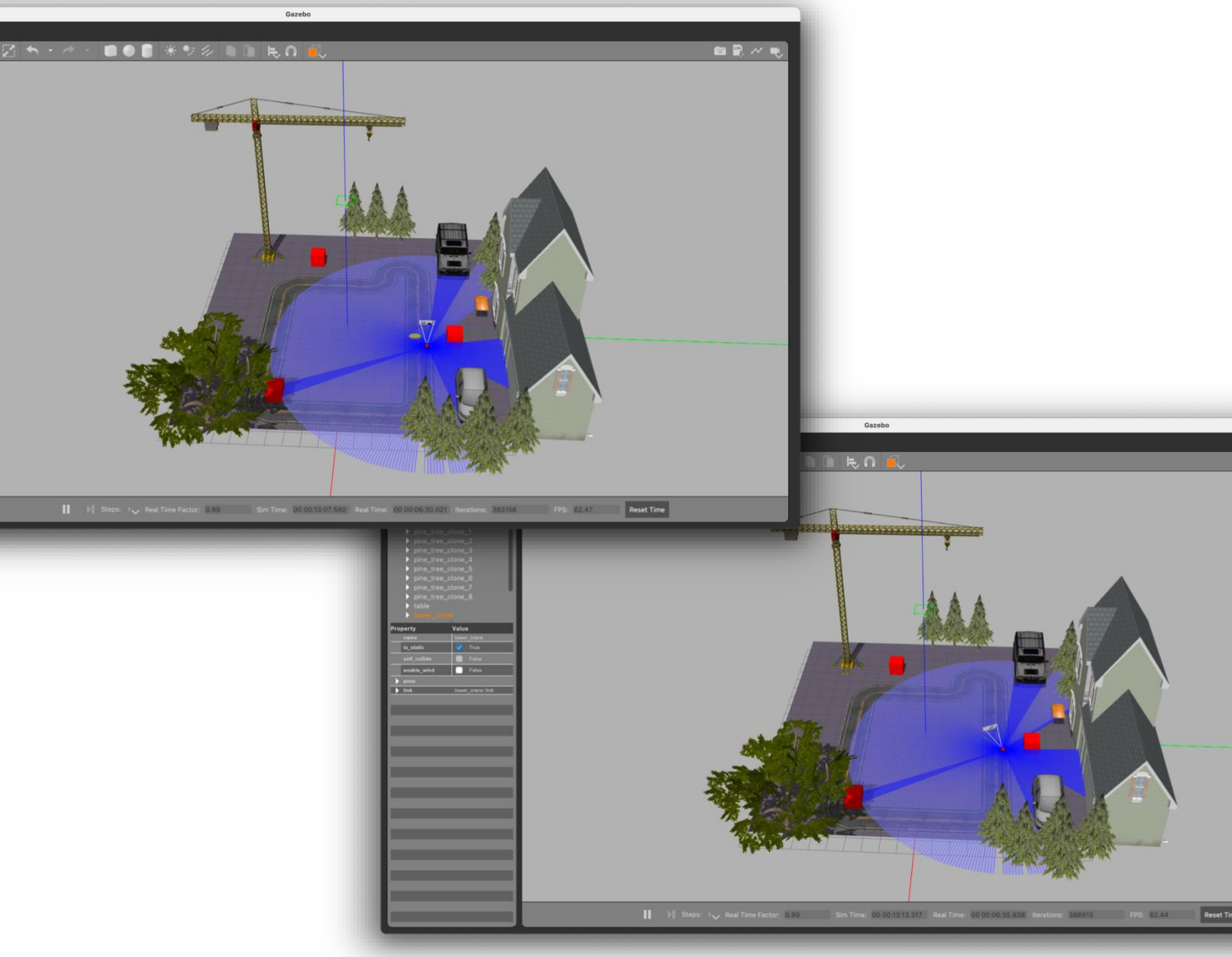
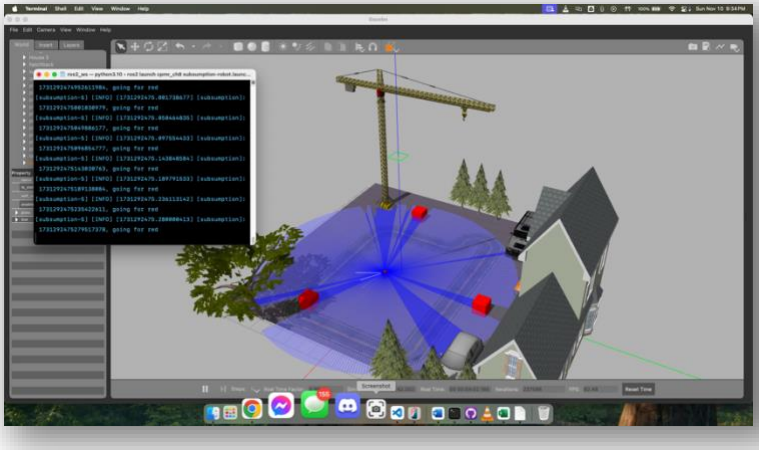
Mahfuz Rahman

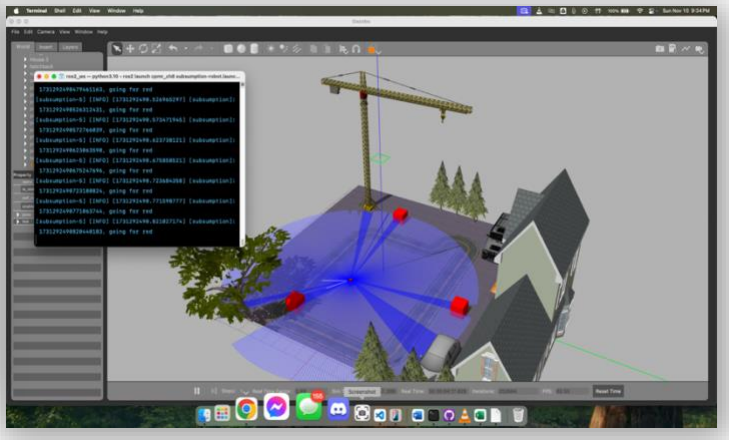
217847518

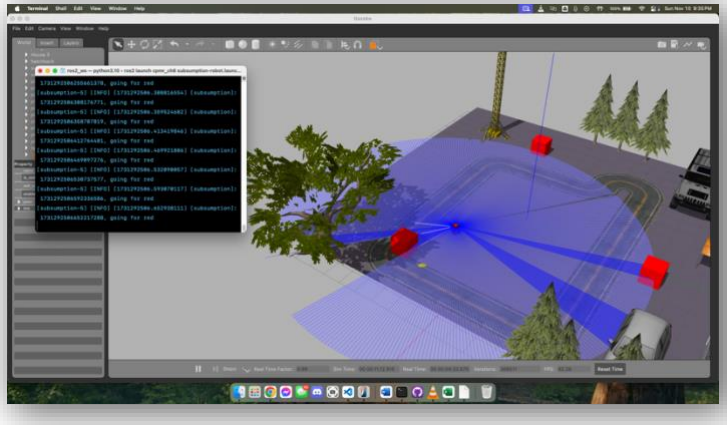
Lab 8 Submission

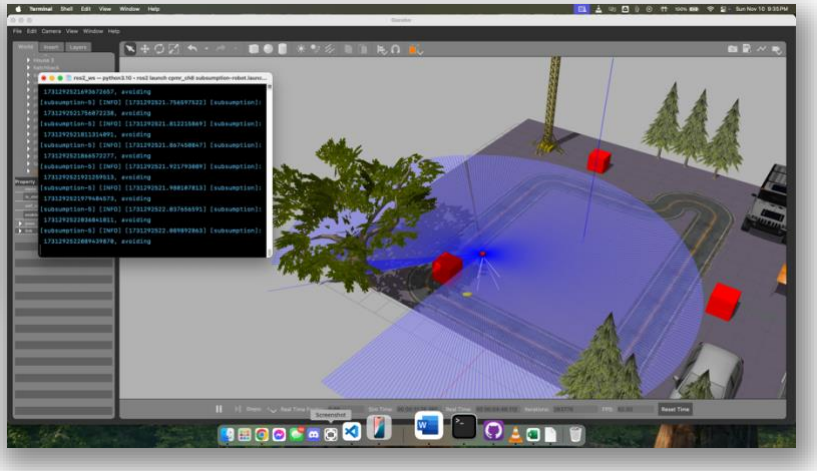
1. Convince yourself that the robot does this. Make a scene, put some non-red objects in the scene, and at least one red one. The snapshots of the robot moving around the world showing that it wanders about, (normally) does not run into objects, and is attracted by red ones. Provide a collection of snapshots of the robot moving in your report. [2 marks]

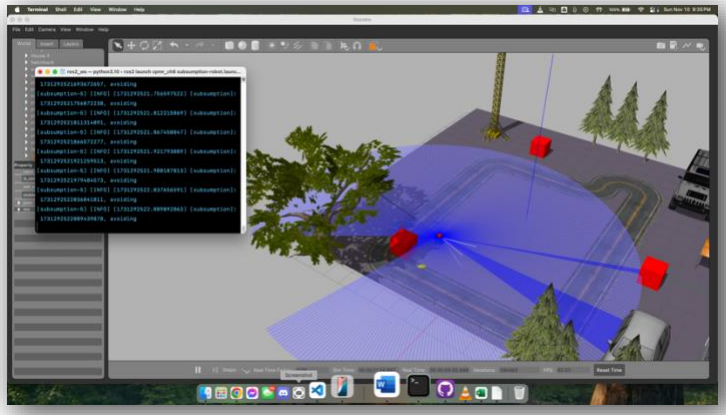


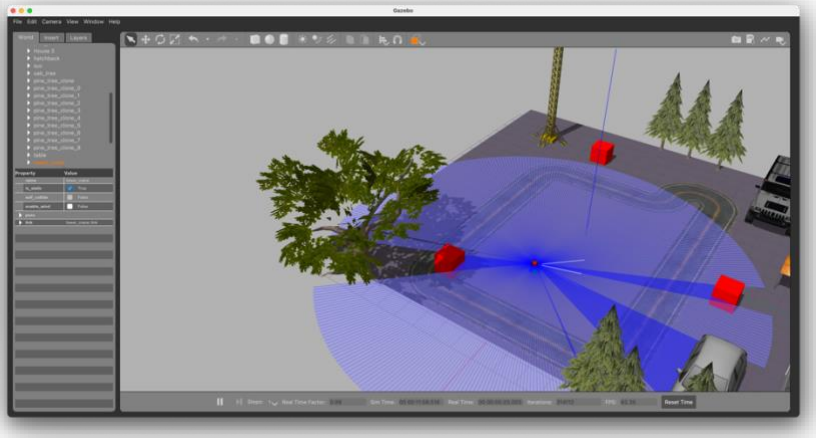










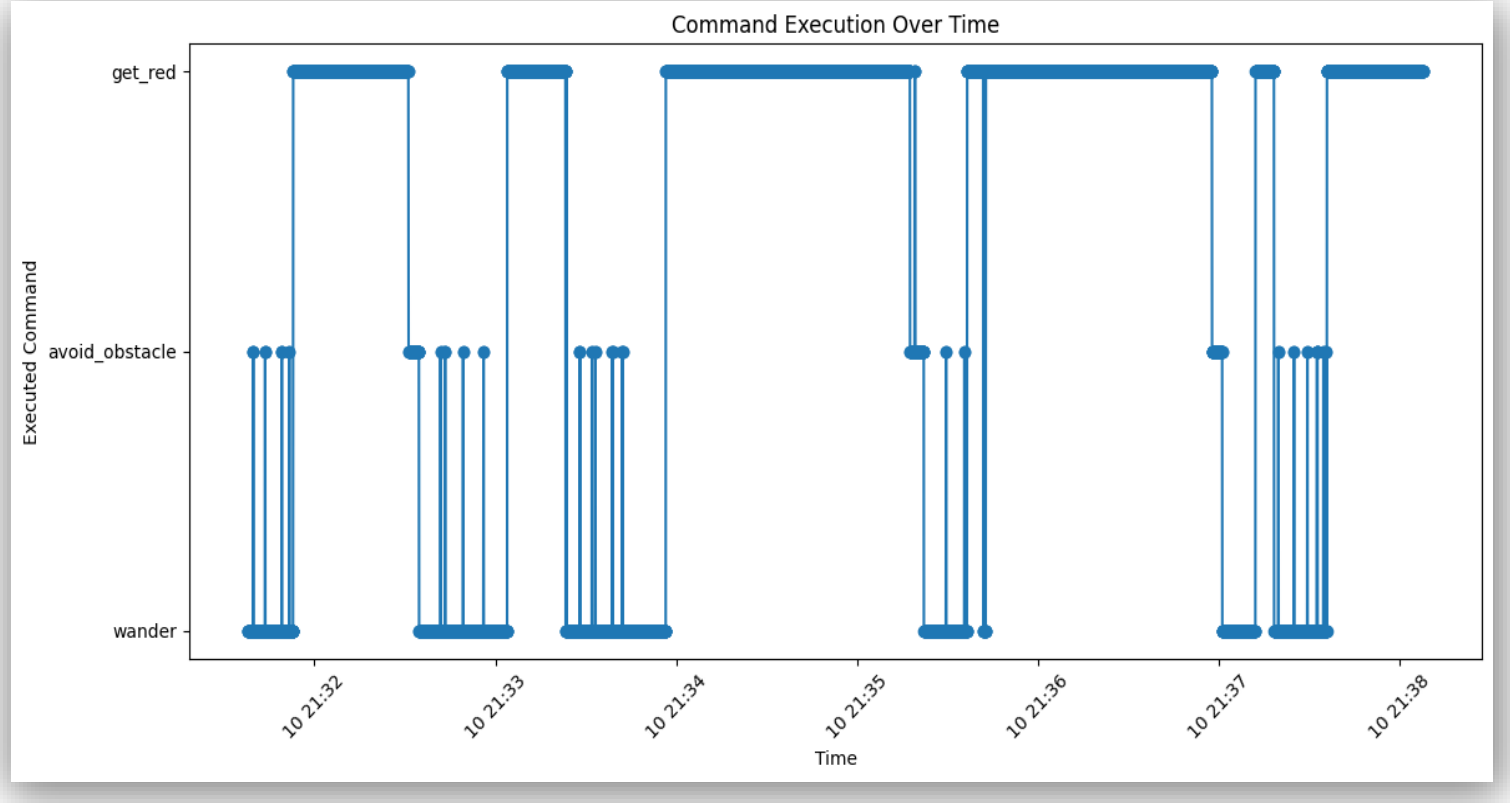


2. Demonstrate that the subsumption architecture interleaves the various control commands over time. To do this, save the time stamps of the output from the code showing the active low-level command as a function of time. Plot this in two ways (i) as a function of time showing which command was actually executed, and (ii) as a histogram of percentage of time each command 'won' in the subsumption architecture. [2 marks]

(i) as a function of time showing which command was actually executed

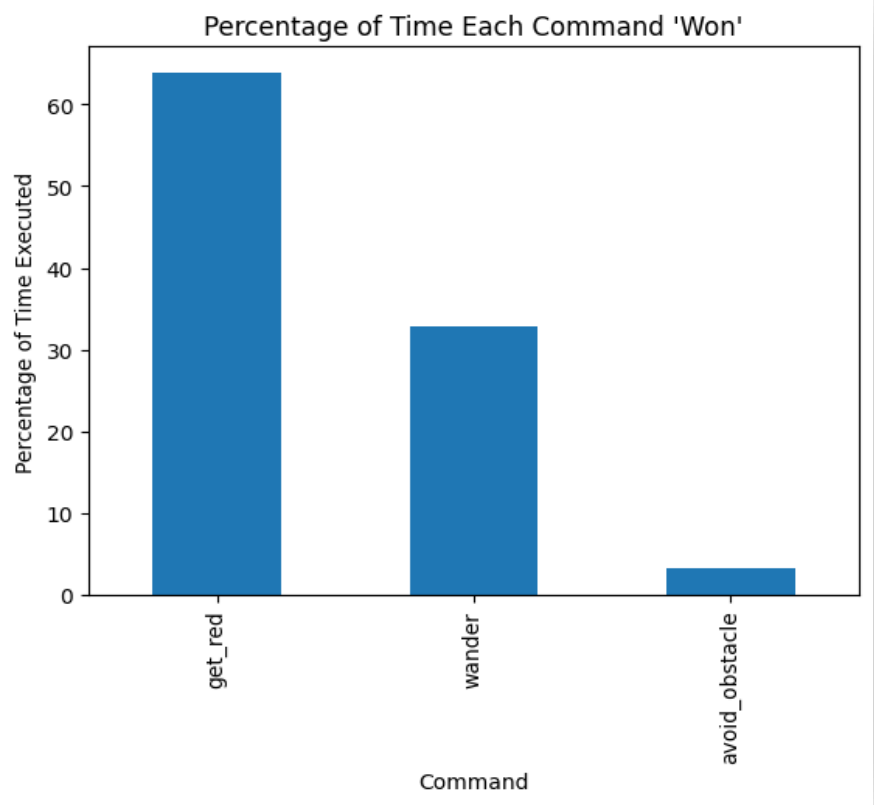
```
import pandas as pd
import matplotlib.pyplot as plt


# Load the log data
df = pd.read_csv("command_log.csv", parse_dates=["timestamp"])
# Plot 1: Command as a function of time
plt.figure(figsize=(12, 6))
plt.plot(df["timestamp"], df["command"], marker="o", linestyle="-")
plt.xlabel("Time")
plt.ylabel("Executed Command")
plt.title("Command Execution Over Time")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



(ii) as a histogram of percentage of time each command 'won' in the subsumption architecture

```
# Plot 2: Histogram of command frequency
command_counts = df["command"].value_counts(normalize=True) * 100  # Percentage
command_counts.plot(kind="bar")
plt.xlabel("Command")
plt.ylabel("Percentage of Time Executed")
plt.title("Percentage of Time Each Command 'Won'")
plt.show()
```


Percentage of Time Each Command 'Won'
Percentage of Time Executed
0
10
20
30
40
50
60
get_red
wander
avoid_obstacle
Command

3. As written the robot is both attracted to and repulsed by red objects. Modify the code so that the robot will stop when it is close to the red object. (At the moment, the robot will be attracted to red objects, but then when it gets close enough to the object it will rotate away from it.) Add a new subsumption layer that stops the robot if there is red in front of the robot and if there is an object in front of it. Use 1.75m as the stopping distance.

   Hand in the code and a video of the robot executing this modified architecture. [6 marks]

```
import math
import random
import numpy as np
import rclpy
import cv2
from rclpy.node import Node
from rclpy.parameter import Parameter
from rcl_interfaces.msg import SetParametersResult
from geometry_msgs.msg import Twist, Pose, Point, Quaternion
from sensor_msgs.msg import LaserScan
from sensor_msgs.msg import Image
from std_srvs.srv import SetBool
from cv_bridge import CvBridge, CvBridgeError

from datetime import datetime


class Subsumption(Node):

    def _wander(self, max_dtheta = math.pi / 20, max_thetav = math.pi / 10, max_dv = .5, max_v = 0.2):
        """wander randomly"""
        twist = Twist()
        v = self._last_twist.linear.x + (2 * random.random() - 1) * max_dv
        t = self._last_twist.angular.z + (2 * random.random() -  1) * max_dtheta

        twist.linear.x = max(min(v, max_v), 0.0)
        twist.angular.z = max(min(t, max_thetav), 0.0)

        return twist

    def _avoid_obstacle(self, minr = 1.5):
        """ if there is an obstacle within minr in front of the robot, stop and rotate"""
        if self._min_r < minr:
            twist = Twist()
            twist.linear.x = 0.0
```

```
            twist.angular.z = math.pi / 10
            return twist
        return None

    def _get_red(self, count = 2000):
        """Detect red color and update a flag if red is detected"""
        self._red_detected = self._redcolcount > count  # Update flag
        if self._red_detected:
            twist = Twist()
            twist.linear.x = 0.1
            twist.angular.z = 0.0
            return twist
        return None

    def _stop_near_obstacle(self, stop_distance = 1.75):
        if self._min_r < stop_distance:
            twist = Twist()
            twist.linear.x = 0.0
            twist.angular.z = 0.0
            self._run = False
            return twist
        return None

    def __init__(self):
        super().__init__('subsumption')
        self.get_logger().info(f'{self.get_name()} created')

        self.create_subscription(Image, '/mycamera/image_raw', self._image_callback, 1)
        self._bridge = CvBridge()
        self.create_subscription(LaserScan, "/scan", self._laser_callback, 1)
        self._publisher = self.create_publisher(Twist, "/cmd_vel", 1)
        self.create_timer(0.01, self._timer_callback)
        self.create_service(SetBool, '/startup', self._startup_callback)

        self._run = False
        self._last_twist = Twist()
        self._min_r = 10000
        self._redcolcount = 0
        self._red_detected = False  # Flag for red detection

        # Adding a list to store command timestamps and names
```

```
    self.command_log = []

def _startup_callback(self, request, resp):
    self.get_logger().info(f'Got a request {request}')

    # Determine the current time for logging
    now_log = datetime.now()
    self.command_log.append((now_log, "_startup_callback"))


    if request.data:
        self.get_logger().info(f'subsumption starting')
        self._run = True
        resp.success = True
        resp.message = "Architecture running"
    else:
        self.get_logger().info(f'subsumption suspended')
        self._run = False
        resp.success = True
        resp.message = "Architecture suspended"
    return resp

def _timer_callback(self):
    if not self._run:
        return

    # Determine the current time for logging
    now_log = datetime.now()

    # Check each behavior in order of priority
    stop_near_obstacle = self._stop_near_obstacle()
    avoid = self._avoid_obstacle()
    red = self._get_red()
    wander = self._wander()

    # Highest priority: Stop if near red and an obstacle
    if stop_near_obstacle is not None:
        self.get_logger().info(f'Stopping near obstacle: {self._min_r}')
        self._publisher.publish(stop_near_obstacle)
        self.command_log.append((now_log, "stop_near_red"))
        return
```

```
        # Second priority: Avoid obstacles
        if avoid is not None:
            self.get_logger().info(f'Avoiding obstacle.')
            self._publisher.publish(avoid)
            self.command_log.append((now_log, "avoid_obstacle"))
            return

        # Third priority: Move towards red if detected
        if red is not None:
            self.get_logger().info(f'Heading towards red object.')
            self._publisher.publish(red)
            self.command_log.append((now_log, "get_red"))
            return

        # Lowest priority: Wander
        self.get_logger().info(f'Wandering.')
        self._publisher.publish(wander)
        self.command_log.append((now_log, "wander"))

    def _laser_callback(self, msg, mind=1.5):
        min_range = mind * 10
        for i, r in enumerate(msg.ranges):
            angle = msg.angle_min + i * msg.angle_increment
            if (abs(angle) < math.pi/4) and (r < min_range):
                min_range = r
        self._min_r = min_range

    def _image_callback(self, msg, width=25):
        """Detect if there's a large amount of red in the image center."""
        image = self._bridge.imgmsg_to_cv2(msg, "bgr8")
        redcolcount = 0
        for i in range(image.shape[0]):
            for j in range(int(image.shape[1]/2-width), int(image.shape[1]/2+width)):
                if (image[i][j][2] > 100) and (image[i][j][0] < 50):
                    redcolcount += 1
        self._redcolcount = redcolcount

    """Saving the logs to a CSV file."""
    def save_command_log(self, filename):
        import csv
        with open(filename, 'w', newline='') as file:
```

```
            writer = csv.writer(file)
            writer.writerow(["timestamp", "command"])
            for entry in self.command_log:
                writer.writerow([entry[0], entry[1]])

import os

def main(args=None):
    rclpy.init(args=args)
    node = Subsumption()
    try:
        rclpy.spin(node)  # This will keep the node running until interrupted
    except KeyboardInterrupt:
        pass
    finally:
        # Save the command log before shutting down
        #filepath = os.path.expanduser("~/Desktop/robotics/Lab8/command_log.csv")
        #node.save_command_log(filepath)
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```