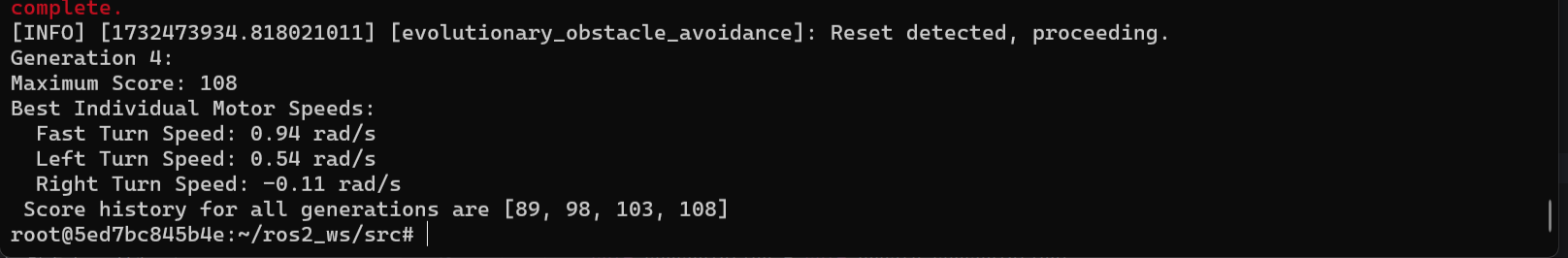
We’ve successfully implemented the ES algorithm on the obstacle avoidance project.

Check out the link below to view the first iteration of the training process:

<https://drive.google.com/file/d/1dJ8ZpfkWNhY868jgy9ndgTaazSGXGA2-/view?usp=sharing>

After four generations of population with population size of six, we get the following results:

At generation 4, the best Motor speeds and scores pair is attached. You can see from the printing of the history of best score of each generation, the scores are



In phase 2, we make some improvements to our codes based on phase 1.

The main process is similar to phase1: initialize -> get sensor -> evaluate -> selection -> mutate & crossover -> evolve generation. However while compiling the code, we found that when the robot hit a wall or an obstacle, it would topple over and never recover. Thus we added a function called reset\_simulation, which used the /reset\_simulation command in the ros2 package to recover the position, speed and angular of the robot. When we detect the robot tip-over has occurred, we will reset the robot and record the score in the previous generation.

def reset\_simulation(self):

self.get\_logger().info("Resetting simulation...")

client = self.create\_client(Empty, '/reset\_simulation')

while not client.wait\_for\_service(timeout\_sec=1.0):

self.get\_logger().info('Waiting for reset\_simulation service...')

request = Empty.Request()

future = client.call\_async(request)

# Wait for the future to complete

timeout\_sec = 0.2 # Max time to wait for the response

start\_time = self.get\_clock().now()

And in the policy part, we added a parameter called fast\_turn, which means when the sensor detects both left and right side obstacles, the robot will turn (default left) at a higher speed. And when the left side or right side has obstacles, the robot will stop moving forward and start turning. We set a threshold and a close threshold. The former one is used to specify the distance from an obstacle when the robot should turn. The latter one is an extreme situation where the robot is too close to an obstacle. In this situation, the individual will receive a high penalty while in other situations the penalty and reward are in a small margin.

# Define a threshold distance to trigger obstacle avoidance

threshold = 0.5 # 0.5 meters

close\_threshold = 0.15 # 0.5 meters

# define our policy here. If left obstacle detected, turn right

# if right detected, turn left

# if both detected, turn left at a faster speed

# if no obstacle detected, moving forward

if right\_dist < close\_threshold or left\_dist < close\_threshold:

self.twist.linear.x = -1.5 # moving back

self.twist.angular.z = 0.0 # Rotate to avoid the obstacle

self.scores += -10

self.get\_logger().info(

f'Hit the obstacle! distance to obstacle: {front\_dist:.2f} meters. Turning at Angular velocity: {self.twist.angular.z} rad/s. current score:{self.scores}')

elif right\_dist < threshold and left\_dist < threshold:

self.twist.linear.x = 0.0 # Stop moving forward

self.twist.angular.z = self.individual['fast\_turn'] # Rotate to avoid the obstacle

self.scores += -1

self.get\_logger().info(

f'FRONT Obstacle detected! distance to obstacle: {front\_dist:.2f} meters. Turning at Angular velocity: {self.twist.angular.z} rad/s. current score:{self.scores}')

elif right\_dist < threshold:

self.twist.linear.x = 0.0 # Stop moving forward

self.twist.angular.z = self.individual['left\_turn'] # Rotate to avoid the obstacle

self.scores += -1

self.get\_logger().info(

f'RIGHT Obstacle detected! distance to obstacle: {right\_dist:.2f} meters. Turning at Angular velocity: {self.twist.angular.z} rad/s. current score:{self.scores}')

elif left\_dist < threshold:

self.twist.linear.x = 0.0 # Stop moving forward

self.twist.angular.z = self.individual['right\_turn'] # Rotate to avoid the obstacle

self.scores += -1

self.get\_logger().info(

f'LEFT Obstacle detected! distance to obstacle: {right\_dist:.2f} meters. Turning at Angular velocity: {self.twist.angular.z} rad/s. current score:{self.scores}')

else:

self.twist.linear.x = 0.5 # Move forward

self.twist.angular.z = 0.0 # No rotation

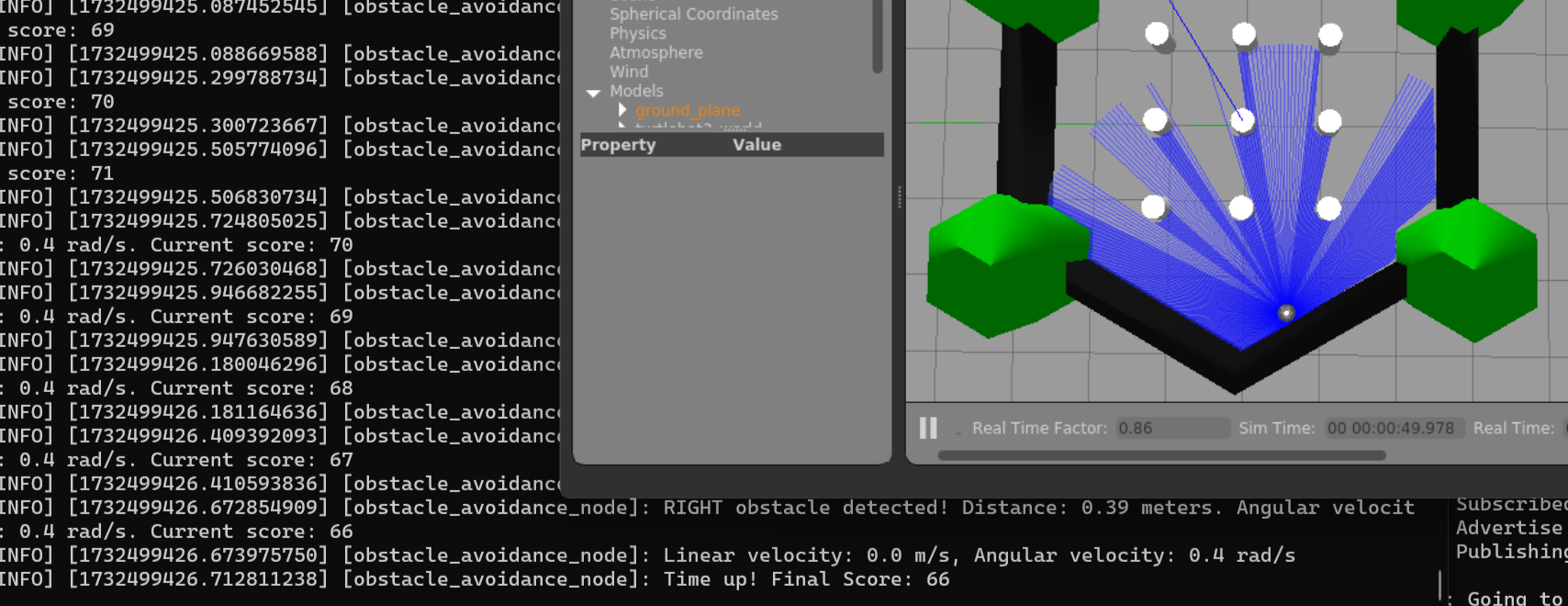
self.scores += 1

self.get\_logger().info(

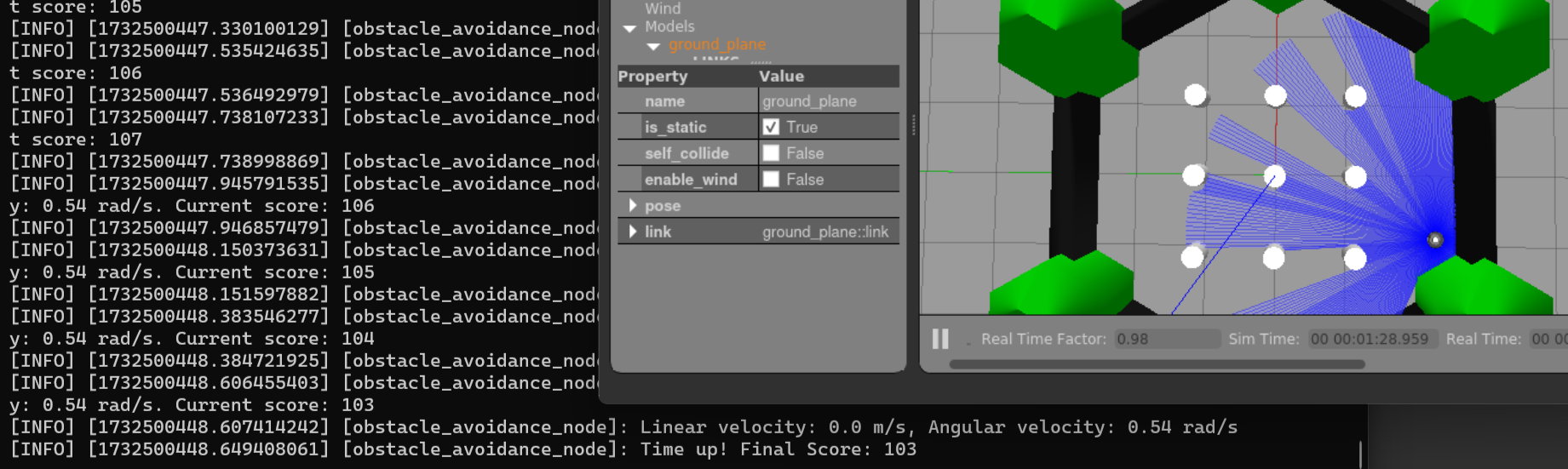
f'Path clear. Moving forward. Front distance: {front\_dist:.2f} meters. current score:{self.scores}')

Results

So, we begin to test out how much motor speed recommended by ES gets better than the initial motor speed we set before. We set the fast turn(turning speed using both sensor detected obstacles), left turn(only when right sensor detected), right turn(only when right sensor detected) to be 0.8, 0.4, -0.4, and got a score of 66.



Then we tried the ES results after four generations, we set the fast turn, left turn, right turn to be 0.94, 0.54, -0.11, and got a score of 103. Thus, this is a huge improvement over the previous set up.



To conclude, we successfully implemented the ES algorithm. In the final presentation, however, we shall use more generation to yield better results.