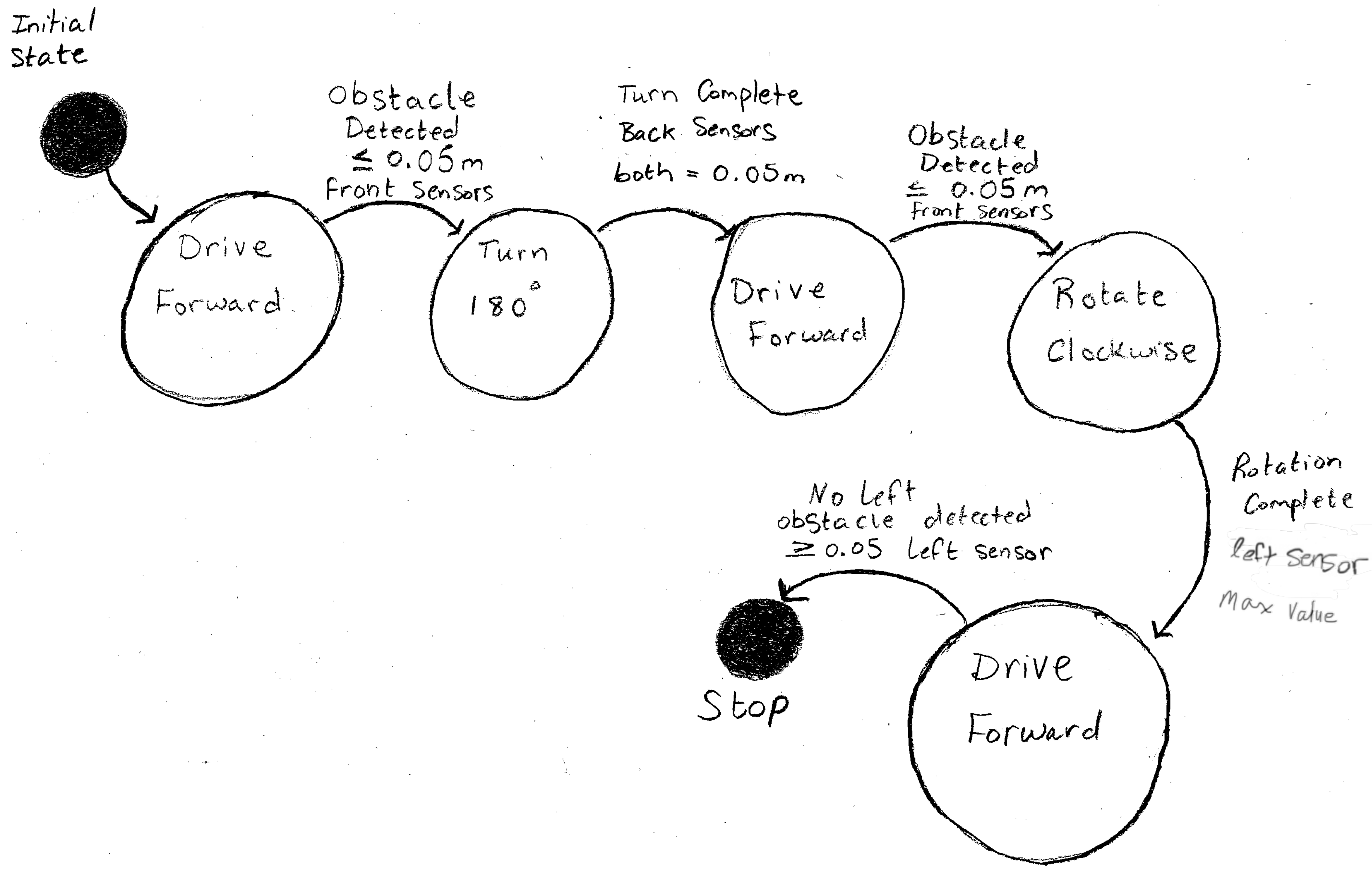
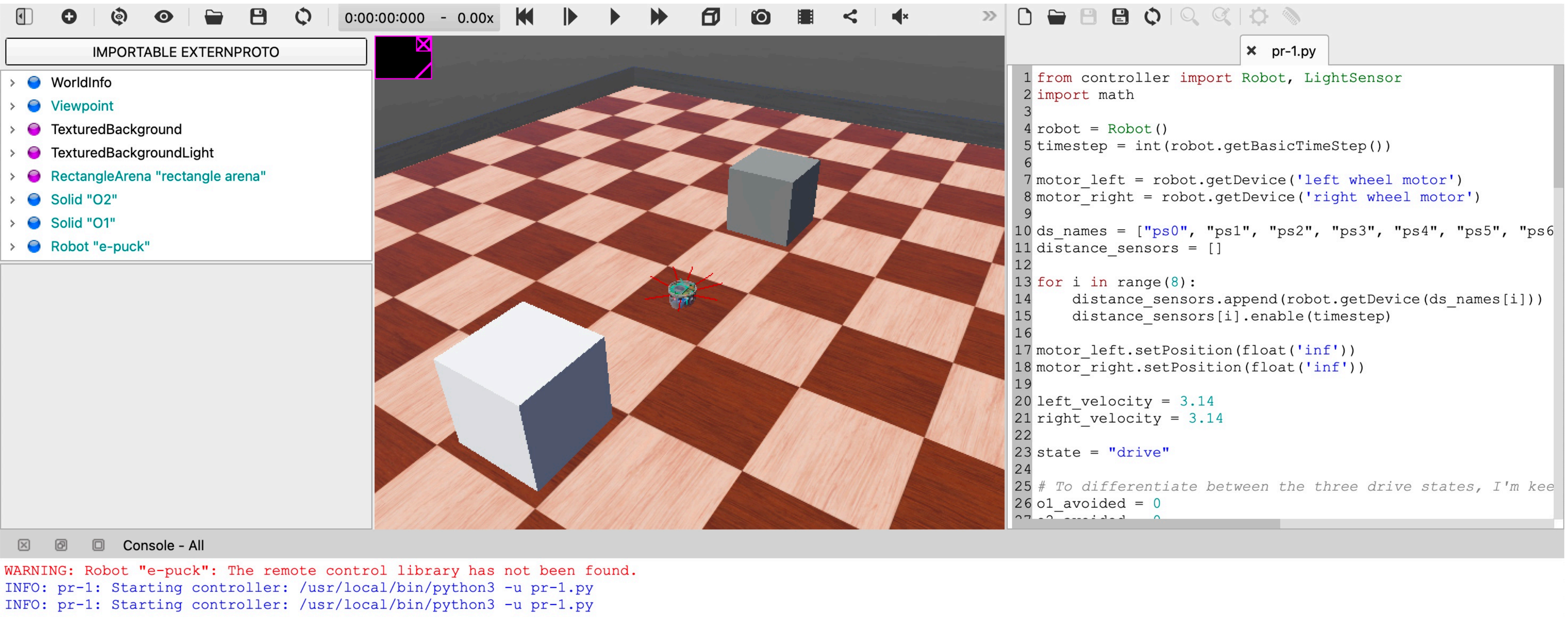

Initial State
Drive Forward
Obstacle Detected ≤ 0.05m Front Sensors
Turn 180°
Turn Complete Back Sensors both = 0.05m
Drive Forward
Obstacle Detected ≤ 0.05m Front Sensors
Rotate Clockwise
Rotation Complete left Sensor Max Value
Drive Forward
No Left obstacle detected ≥ 0.05 Left sensor
Stop



WorldInfo
Viewpoint
TexturedBackground
TexturedBackgroundLight
RectangleArena "rectangle arena"
Solid "O2"
Solid "O1"
Robot "e-puck"

```
from controller import Robot, LightSensor
import math

robot = Robot()
timestep = int(robot.getBasicTimeStep())

motor_left = robot.getDevice('left wheel motor')
motor_right = robot.getDevice('right wheel motor')

ds_names = ["ps0", "ps1", "ps2", "ps3", "ps4", "ps5", "ps6
distance_sensors = []

for i in range(8):
    distance_sensors.append(robot.getDevice(ds_names[i]))
    distance_sensors[i].enable(timestep)

motor_left.setPosition(float('inf'))
motor_right.setPosition(float('inf'))

left_velocity = 3.14
right_velocity = 3.14

state = "drive"

# To differentiate between the three drive states, I'm kee
o1_avoided = 0
o2_avoided = 0
```

```
WARNING: Robot "e-puck": The remote control library has not been found.
INFO: pr-1: Starting controller: /usr/local/bin/python3 -u pr-1.py
INFO: pr-1: Starting controller: /usr/local/bin/python3 -u pr-1.py
```

For this assignment on basic reactive behaviors, I was able to successfully implement all tasks with my E-puck robot's controller based upon the finite state machine on the first page of this document.

States

I used a match-case statement to define the states of the robot: *drive* (forward), *turn* (turn 180 degrees), *rotate_cw* (rotate clockwise), and *stop*. Each of these states sets the wheel velocities according to the motion described.

I introduced two additional variables, *o1_avoided* and *o2_avoided,* to use in conjunction with my state logic. These variables are used to determine which "drive" state I was in, as there are three instances of the drive state; the initial state heading towards the first obstacle, the one after the 180 degree turn heading towards the second obstacle, and the drive state after the right turn after avoiding the second obstacle. After each obstacle was successfully avoided, I set the corresponding variable to 1 (True).

State Transitions

Further down in my logic, I implemented the transition functions with conditions based upon distance sensor readings from the front, back, and left sensors. Once the condition corresponding to each transition was met, such as "no obstacle detected" or "obstacle detected within 0.05m of front sensors," I changed the state value of the robot accordingly. It took some trial and error to determine which intensity level of the sensor represented approximately 0.05m. I also changed the lighting and color of the obstacle to make sure the distance sensor was making as accurate readings as possible. The intensity of around 80 corresponded to the correct distance, so I used this in my conditions. In the "rotate_cw" state, when checking for the maximum sensor reading of the left sensor to determine when it was approximately perpendicular with the obstacle, I used 109 as the threshold; this was also determined through trial and error.

The most complex part of the logic was detecting when the robot had turned exactly 180 degrees after avoiding the first obstacle. I used the two distance sensors on the back to measure when they were approximately equidistant from the obstacle. If the robot changed states and started driving before this condition was met, it would cause the robot to move at an angle and miss the second obstacle. Due to the noise in the sensors, it was impossible to match the back sensors exactly. Therefore, I took the floor of the sensor readings and made sure that their difference was less than some constant value (I chose 3 through trial and error) such that their readings were as close as possible.