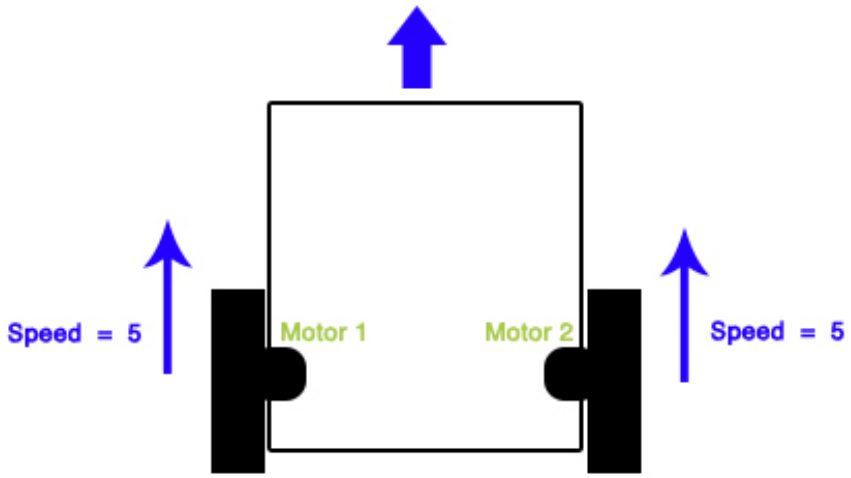
Task Description

The goal of the robot is move through a simple maze and stop when it detects the finish area (a black box). The robot is also required to use the same algorithm to solve different maze configurations. It is a two-wheeled differential drive robot with two sensors (a bumper sensor to detect the wall and a light sensor to detect the floor color). To accomplish this, the robot will need a reference for its movement from one location to another. It will use the existing walls of the maze to guide itself to the finish line. The robot will use the obstacle (the walls) and it sensors to accomplish this task and will complete each maze within 5 minutes. This robot will be designed to follow the wall on the right side.

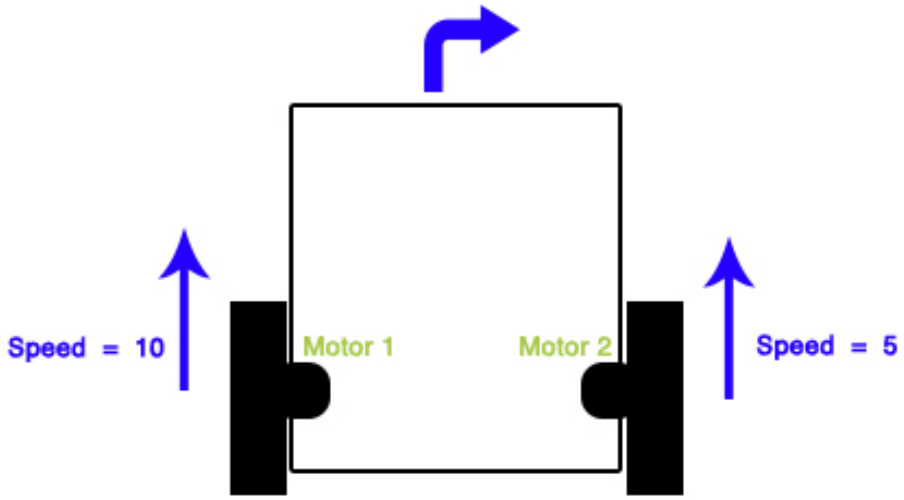
Description of code used and Algorithm

In establishing an algorithm, I first had to understand the steering system of a differential robot. The steering mechanism is as follows:

If both motors (wheels) turn at the same speed, the robot moves in a straight line (see diagram below).



If one wheel rotates faster than the other, the robot follows a curved path inward toward the slower wheel (see diagram below).



As we can see, the steering of the robot has to do with the varying the speeds of the motor. A similar method is used to turn the robot when it is backing up. This fact is major part of the algorithm.

The Robot logic operates as follows:

- If the robot bumps into the wall, then backup (reverse, while slightly turning right)
- Then drive forward (with a slight right angle)

This is represented by the following code:

```
-- When in backward mode, we simply move backward at the desired speed
simSetJointTargetVelocity(leftMotor,-speed/0.4)

simSetJointTargetVelocity(rightMotor,-speed/2)
```

```
-- When in forward mode, we simply move forward at the desired speed
simSetJointTargetVelocity(leftMotor,speed*2.1)
simSetJointTargetVelocity(rightMotor,speed*1.5)
```

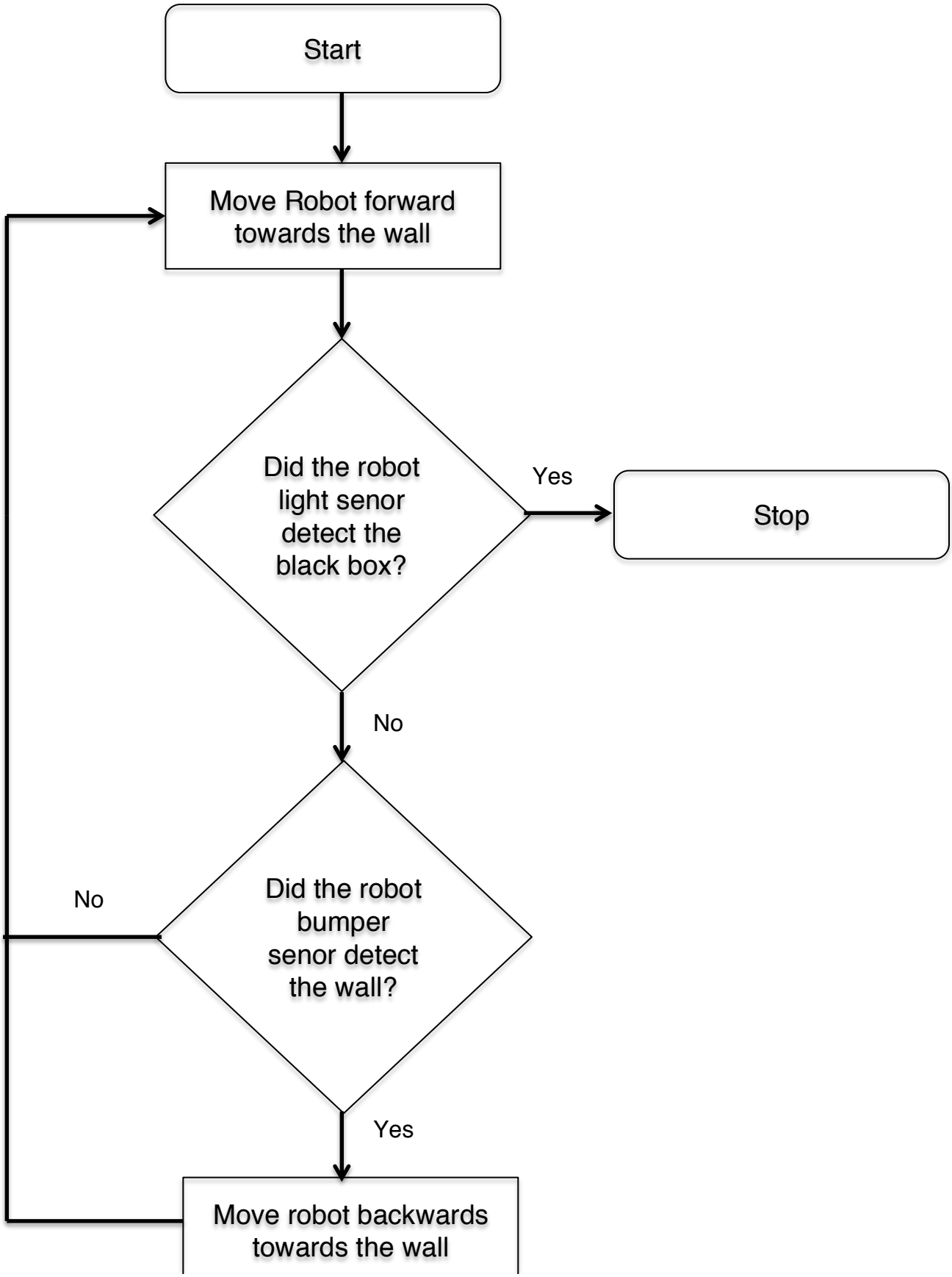
The follow code determines how long the robot will move backward. This is important because if the robot moves backwards too long it may get stuck on an object in the maze or fall off the simulation canvas

```
-- read the Touch sensor
TouchAct=simReadProximitySensor(TouchSensor)
if (TouchAct> 0) then
--  if the touch sensor was activated, provides the robot with 5 seconds to act before rec
    --backUntilTime=simGetSimulationTime()+15
   backUntilTime=simGetSimulationTime()+2.0
```

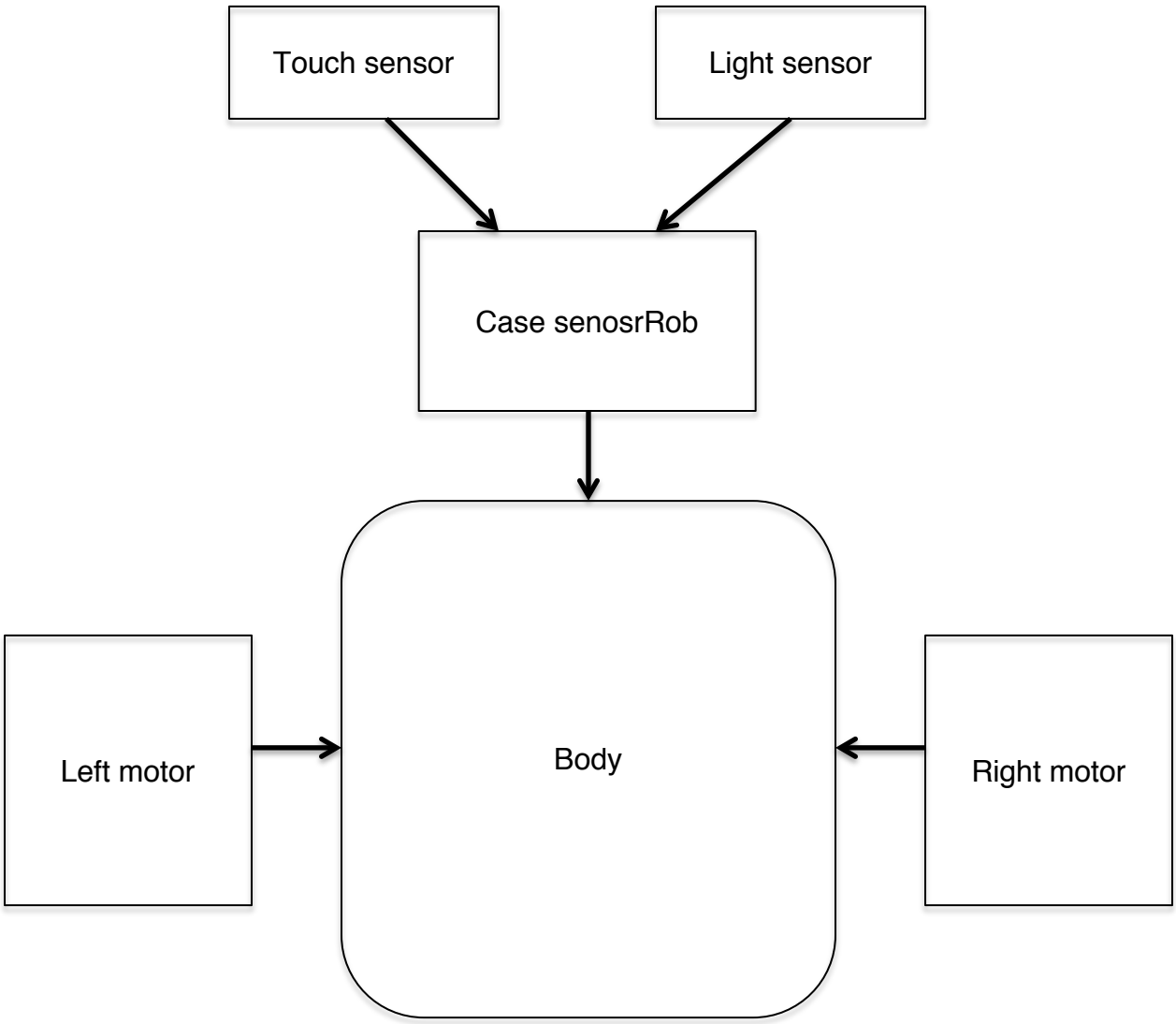
This code checks if the light sensor has located the black box. If it does, the robot will stop. The robot check if the color threshold is lower than 0.3.

```
    if (result>=0) then
-- data[11] is the intensity of the light
-- 0.3 is the threshold to define if there are a black square below of robot
        LightSensor=(data[11]<0.3)
       --LightSensor=(data[11]>0.3)
    end

-- If light sensor detect black square below of robot, then it is assigned zero to speed
if LightSensor then
    --speed=0.1
    speed=0
else
    speed=2.3


end
```

Flowchart

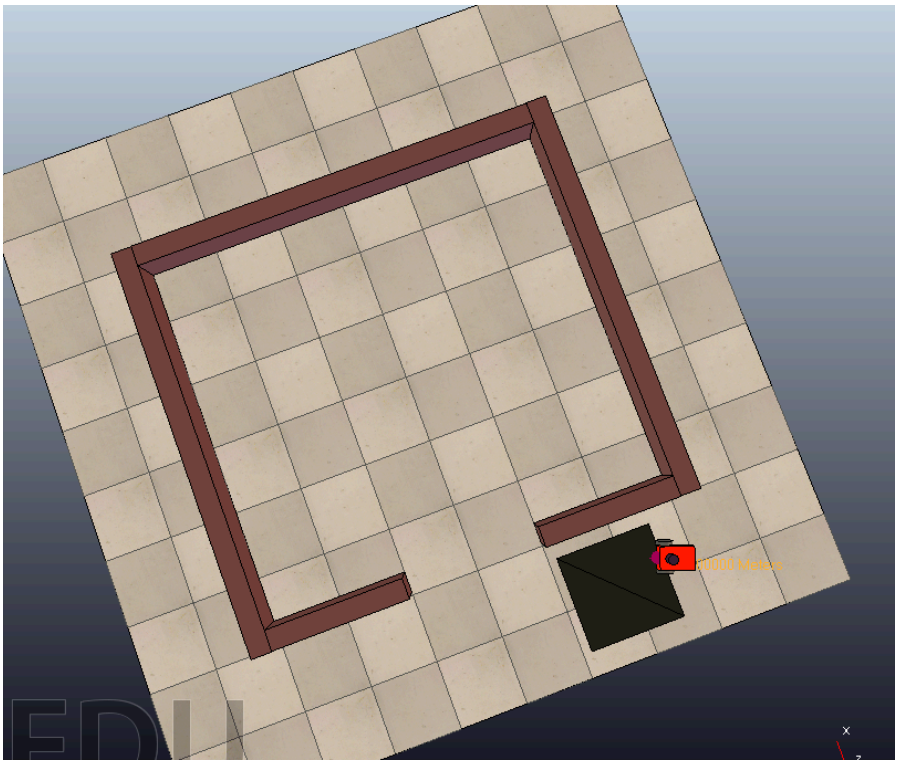

Start
Move Robot forward towards the wall
Did the robot light senor detect the black box?
Yes
Stop
No
Did the robot bumper senor detect the wall?
No
Yes
Move robot backwards towards the wall

Block Diagram

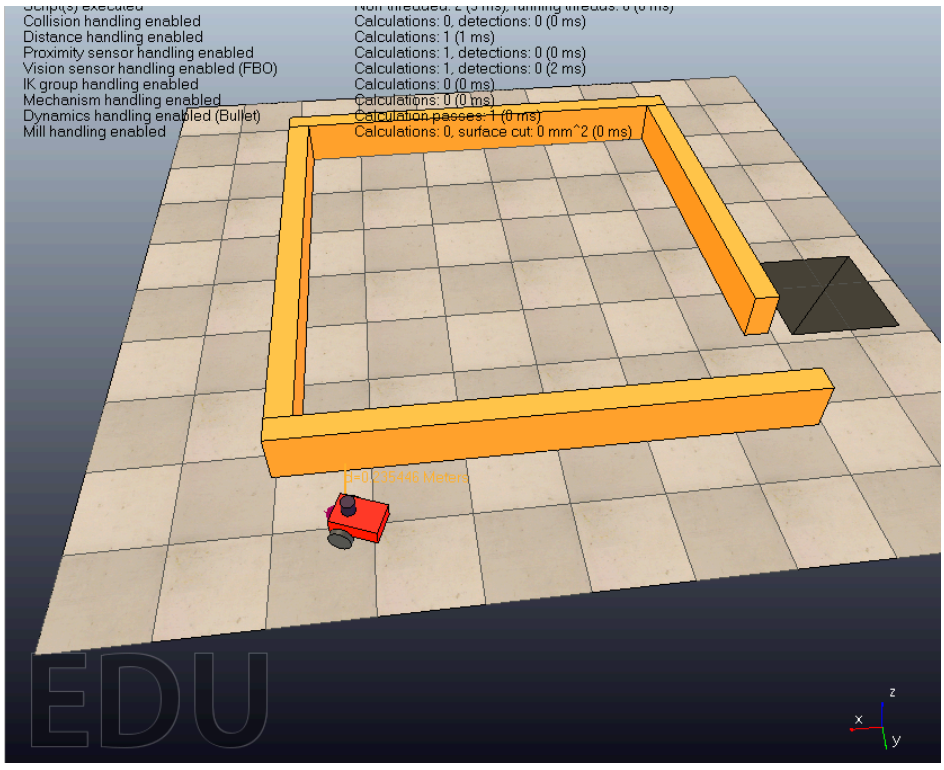


Results (YouTube links and screenshots of simulation)

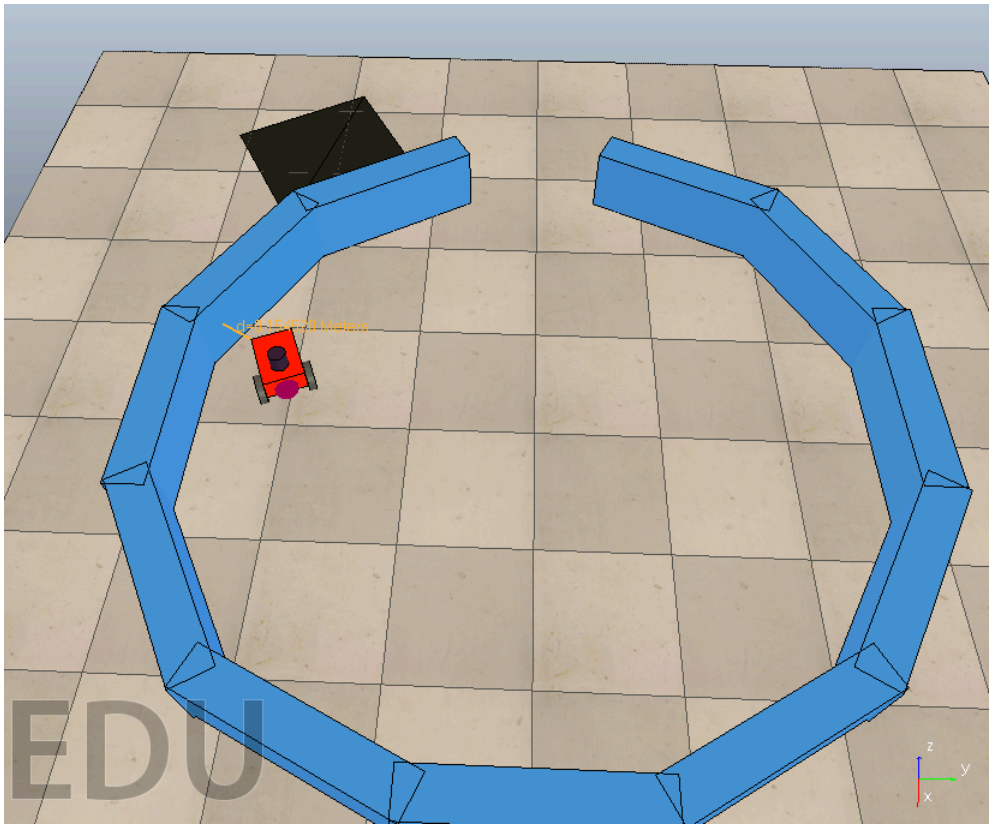
Maze 1: http://youtu.be/F0u08FTDqjM



Maze 2: http://youtu.be/Kzz3qOdEo0Y



Maze 3: http://youtu.be/0e4cQ_B3c8I



Conclusion

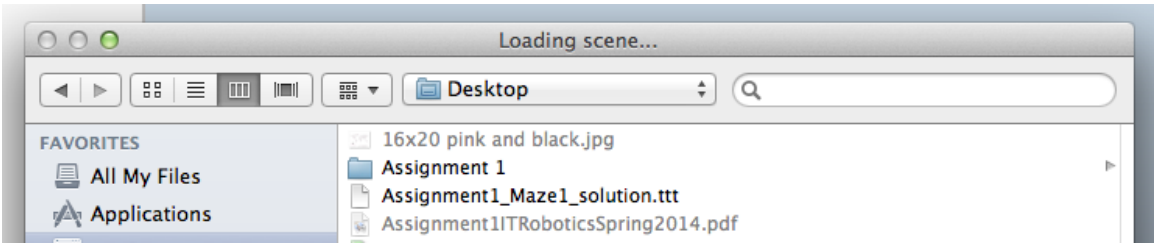
One problem that I faced in programming the robot was trying to avoid the wall while backing up. This was difficult because the robot had to have the ability to navigate multiple maze configurations. I think this problem could be solved if the robot had an addition sensor that could tell it how far it is form the wall.
When the robot is at a “safe” distance from the wall, the sensor will actively check ensure that it does not get too close.
In this scenario,

- if the robot is too close to the wall, steer away from the wall

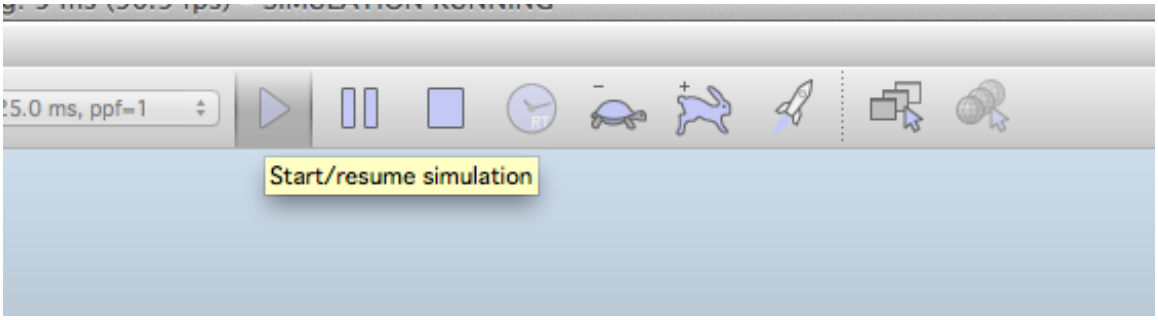
- if the robot is too far from the wall, steer closer to the wall (to the prescribed ‘safe” proximity).

How to run the Program:

Open “Assignment1_Maze1_solution.ttt” in V-REP



Run the program by clicking the start button



I have also included

Assignment1_Maze2_solution.ttt and Assignment1_Maze3_solution.ttt files. They all contain the same code. These can be opened and executed the same way as stated earlier.