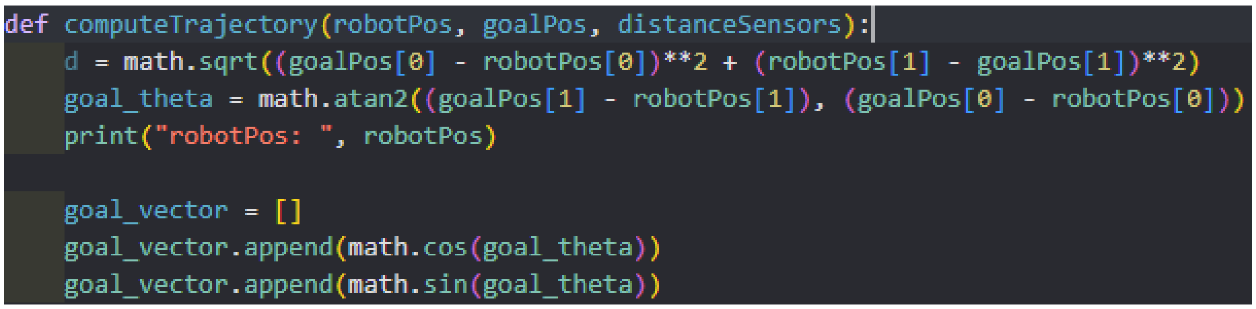
- List the people in your group

: Matt Christensen, Sihnyoung Park

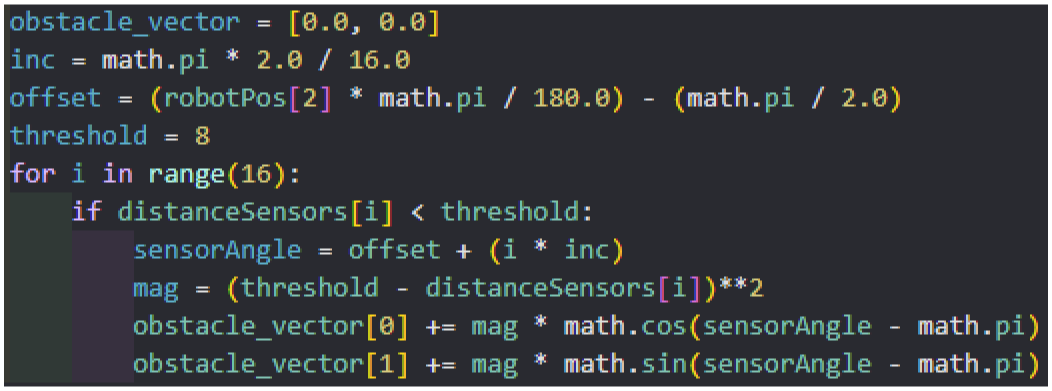
- A 1-paragraph description (or something like that) of what you implemented in your potential fields. Use math if that helps you create a clearer description. The description should have enough detail that another person should be able to duplicate your exact algorithm in code just by reading this brief description. That said, don't over do it — it doesn't have to be polished either.

: First step we was make an attractive potential field was finding the angle between the agent and the goal by using this equation θ = tan−1((yG−y) / (xG−x)) where yG and xG represent y, x coordinates of the goal. Then put the θ value to cos(θ) and sin(θ) to calculate ∆x and ∆y and add ∆x and ∆y value to trajectory vectors so that the agent can move towards the goal.



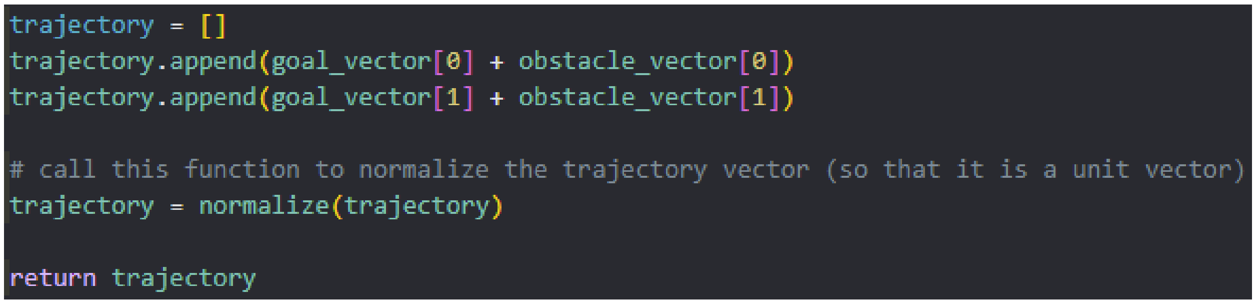
```python
def computeTrajectory(robotPos, goalPos, distanceSensors):
    d = math.sqrt((goalPos[0] - robotPos[0])**2 + (robotPos[1] - goalPos[1])**2)
    goal_theta = math.atan2((goalPos[1] - robotPos[1]), (goalPos[0] - robotPos[0]))
    print("robotPos: ", robotPos)

    goal_vector = []
    goal_vector.append(math.cos(goal_theta))
    goal_vector.append(math.sin(goal_theta))
```

: Next, we needed to make a repellent force away from obstacles - like a vector field that pushes away from edges. To do this we account for the rotation of the robot and then iterate through all of the distance sensors. If we find the distance to the nearest wall to be within a given threshold we push in the opposite way portotional to how close we are to the wall (with a magnitude that is calculated by the distance).



```
obstacle_vector = [0.0, 0.0]
inc = math.pi * 2.0 / 16.0
offset = (robotPos[2] * math.pi / 180.0) - (math.pi / 2.0)
threshold = 8
for i in range(16):
    if distanceSensors[i] < threshold:
        sensorAngle = offset + (i * inc)
        mag = (threshold - distanceSensors[i])**2
        obstacle_vector[0] += mag * math.cos(sensorAngle - math.pi)
        obstacle_vector[1] += mag * math.sin(sensorAngle - math.pi)
```

: Finally, we add both of these vectors together to find a final transform vector to be used to move our robot.



```
trajectory = []
trajectory.append(goal_vector[0] + obstacle_vector[0])
trajectory.append(goal_vector[1] + obstacle_vector[1])

# call this function to normalize the trajectory vector (so that it is a unit vector)
trajectory = normalize(trajectory)

return trajectory
```

List a thing or two (or three or four) you tried that didn't work (if any -- usually things don't work perfectly the first time). In other words, this step is here to encourage you to experiment just a little bit with making your potential fields better.

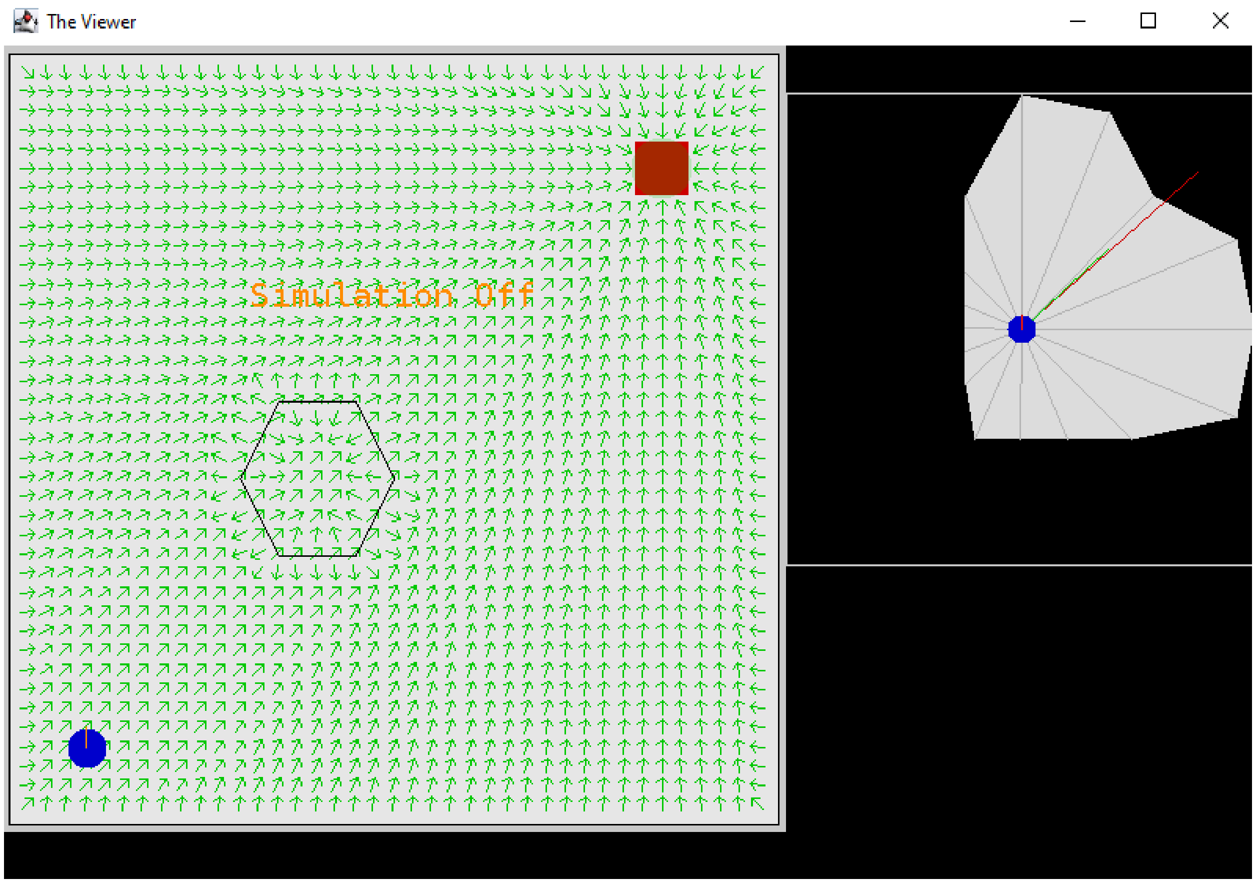
: When we tried to find a rejected potential field, we had to find x,y coordinates of the obstacle. To find the obstacle coordinates, we used `robotPos[2]` as θ value and the equation x = r cosθ and y = r sinθ. However we failed to find the angle between the agent and the obstacle.

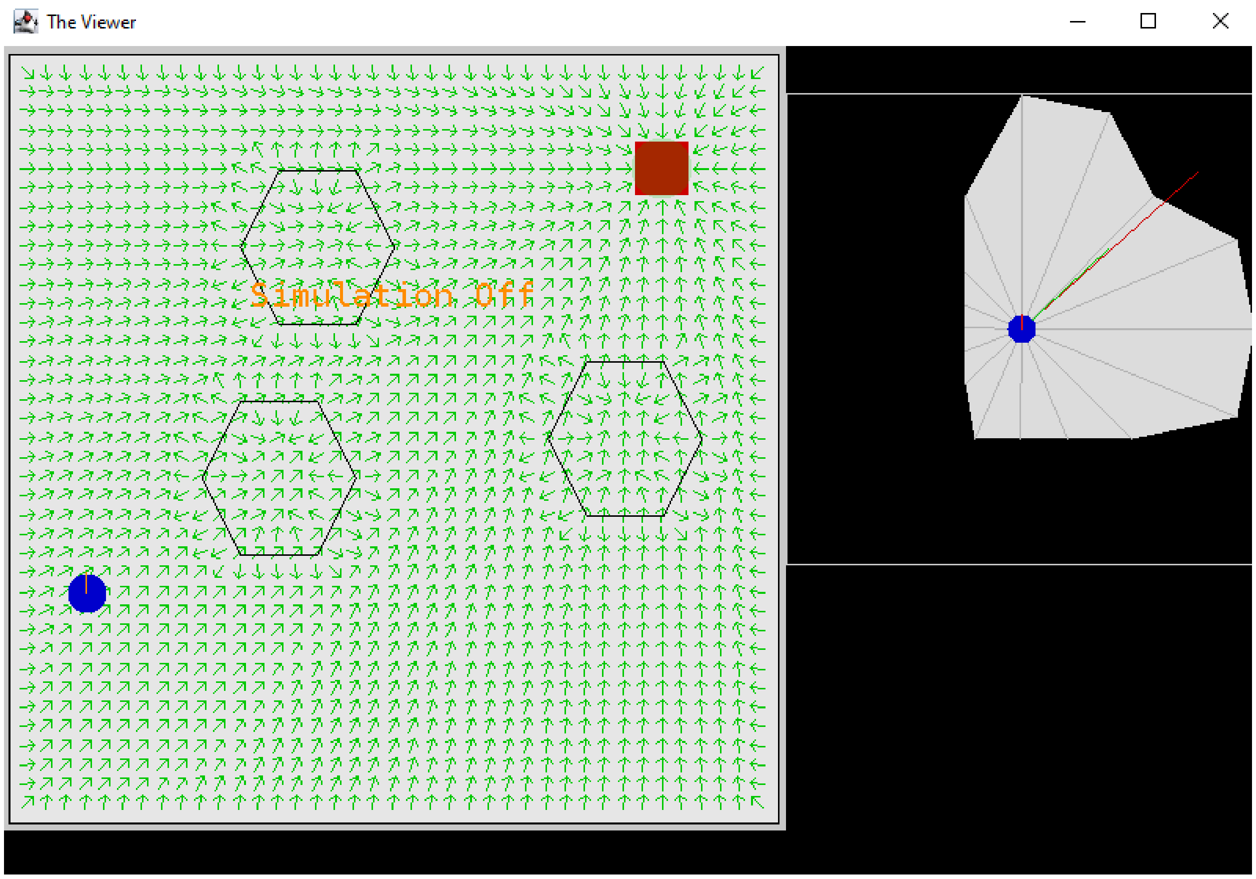
: Additionally, it was quite difficult to try and account for the rotation of the robot for the send field that pushes away from objects. It was hard to normalize the rotation from robot to world space as well as trying to go through all 16 direction sensors and know where they are pointing in order to add them to the final transformation vector.

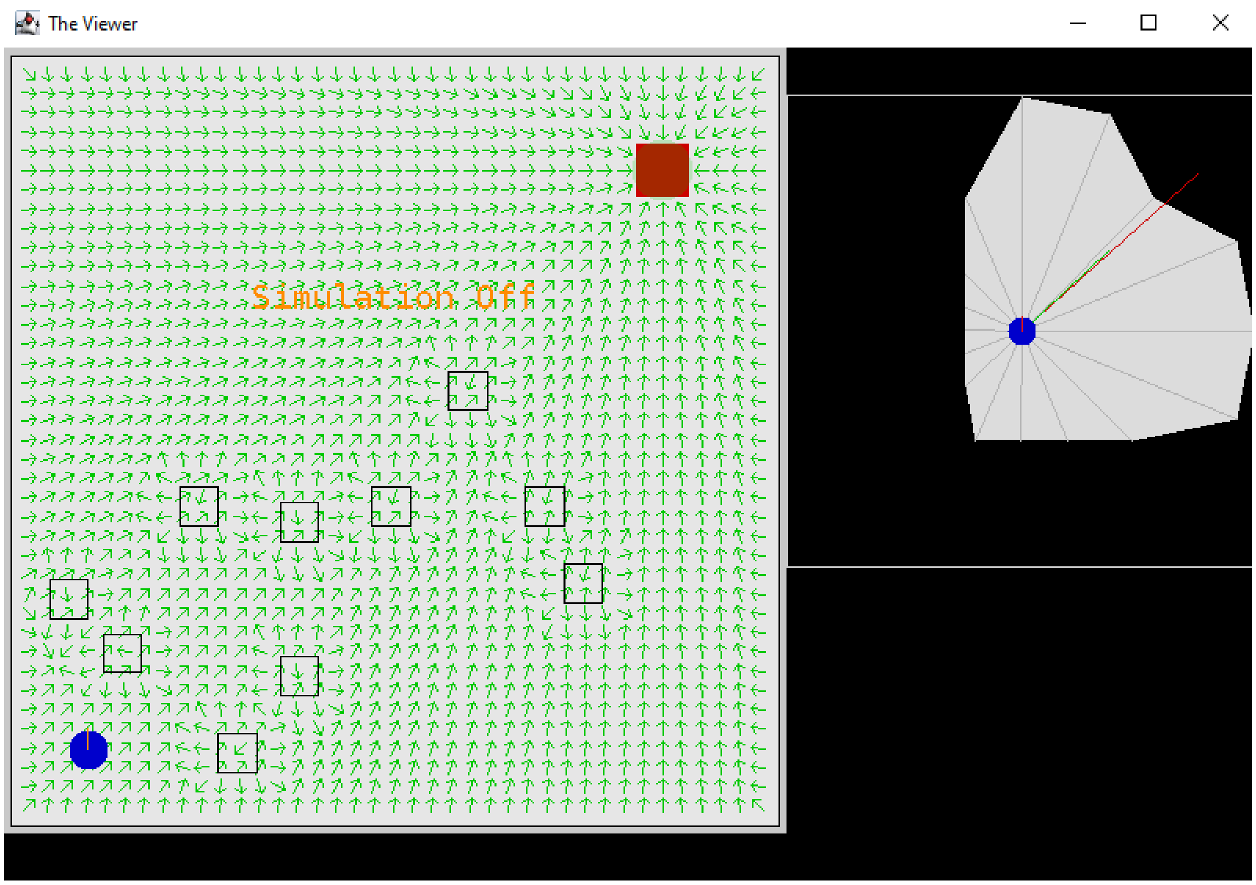
: Finally, we had to tweak the magnitude of the repulsive field (from objects) a few times to make sure we got a solution on all three of the first worlds. Initially our threshold and magnitude were too high, and it would create too large of a field such that we would get stuck in a corner. We didn't realize how much of an affect it would have on the algorithm.

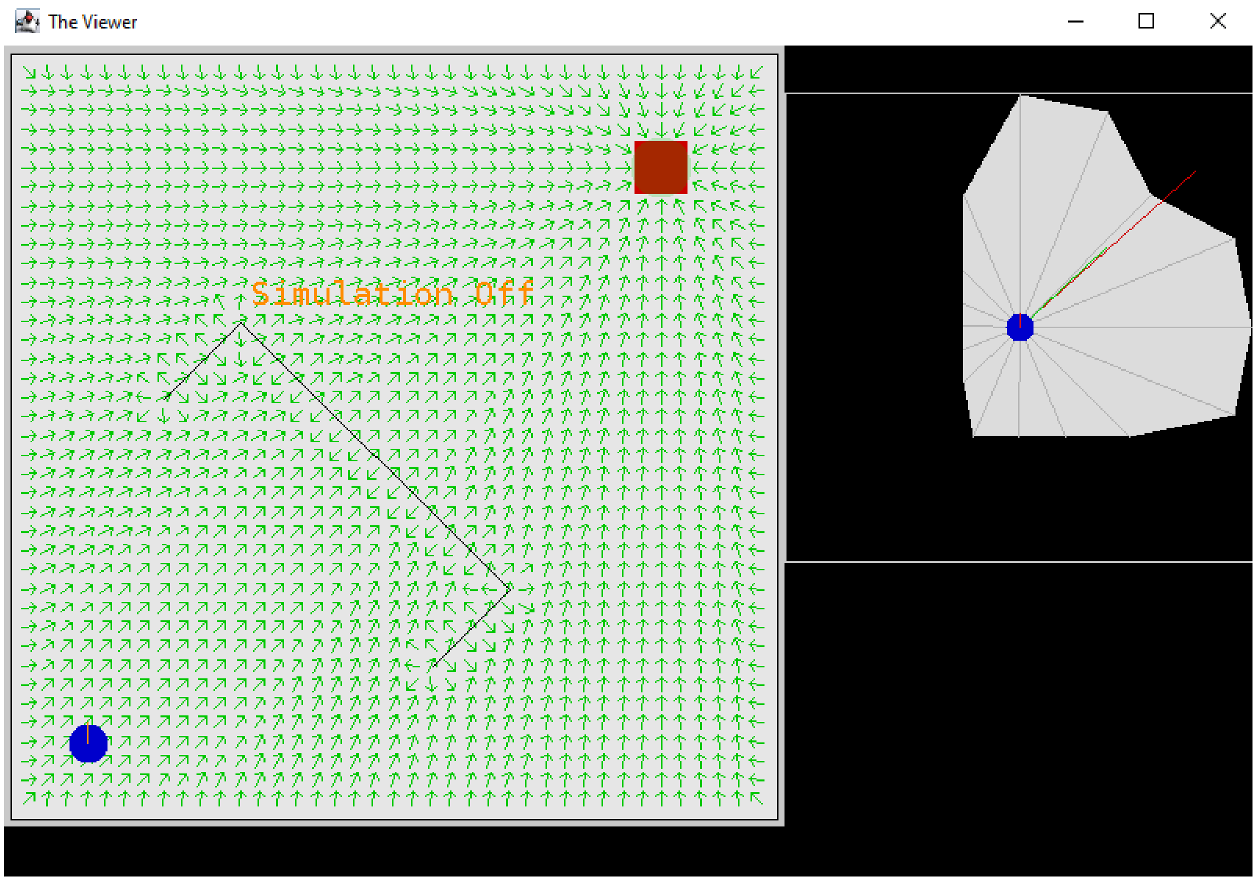
⚉State how long it takes your algorithm to navigate to its charger in each of the 4 worlds. If your robot fails to get to the goal in a world, just note that.

: We were able to get the potential fields to navigate, but the robot would not respond to the updated fields - so we don't know how long it would take. But it should be able to finish the first three worlds without problem.




The Viewer
Simulation Off


The Viewer
Simulation Off


The Viewer
Simulation Off