



CMP494-10 - Intelligent Autonomous Robotics

Project 1

Date: 17/03/2024

Jawad Adas b00087996

Koushal Parupudi b00087520

Bingxuan Li b00088619

Muhammad Hamza Umair b00088627

Submitted to: Dr. Michel Pasquier

Task:

1. Goal destination:

(a) Define an APF model that will drive the mobile robot toward the goal, regardless of initial position and orientation, and stop when (approximately) at that location. The goal destination should be located as per the map in Figure 1. Note that it is not a physical object and does not act as an obstacle. You can assume the goal location (X-Y cords on the map) is always known to the robot, and use the positional sensor to retrieve the robot's location. Explain your design clearly in the report, providing the necessary equation/s for the APF vector/s, dropoff, etc.

Python

```
GOAL_POINT = 0.2, 0.2
```

```
Word_Size = 0.74 * 0.74 (based on our measurement of robot size = 0.037)
```

APF:

The APF model for the attractive force toward the goal uses a very simple approach. First, we take two points, the robot's position provided by the GPS denoted as point A, then the GOAL_POINT denoted as point B, and calculate the vector from A to B.

$$\text{Vector from A to B} = C = \langle B_x - A_x, B_y - A_y \rangle$$

Next, we normalize the vector which maintains the direction and sets the magnitude to 1, to achieve a constant vector till the goal. This is done in anticipation of the upcoming tasks of avoiding walls and obstacles. Additionally, this approach avoids extremely low speeds when near the goal and likewise extremely high speeds when far away.

For the dropoff, we set the robot speed to 2 until the robot is within 0.005 units of the goal, then the speed is set to 0. This achieves a step function for the dropoff.

Control:

Translating vector to control is simple when the robot is pointing towards the goal, however, in the case of the robot pointing another direction, we need to turn.

To accomplish this, the angle between the robot's heading vector and the vector to the destination is calculated using the *angle_between_vector* function.

Python

```
left_motor.setVelocity(-angle)
```

```
right_motor.setVelocity(+angle)
```

The function's output range is $-\pi$ to π (radians), where negative values turn right and positive to turn left. At any time the robot should only make a maximum of 180 degrees turn.

(b) Build a Webots world for the map in Figure 3 i.e., an empty arena surrounded by four walls, and implement the simplest controller program as per your design. Experiment in simulation by placing the robot at different initial locations and orientations. Include in your report screenshots (with trace) of your experiments, with appropriate comments, including how you refined your model parameters.



Figure 1: Robot starts at the top left corner and faces west

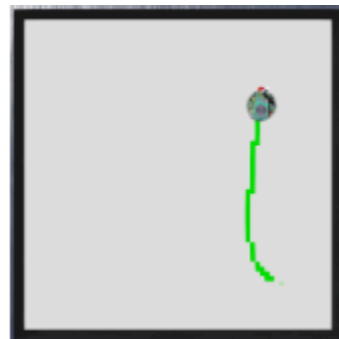


Figure 2: Robot starts at the bottom right corner and faces south



Figure 3, The Robot starts at the bottom left corner and faces south-west

Python

```
#speed = 2, a = angle between the two vectors
#set control based on angle
if(a < 0.0001 and a > -0.0001):
    left_motor.setVelocity(speed)
    right_motor.setVelocity(speed)
```

```

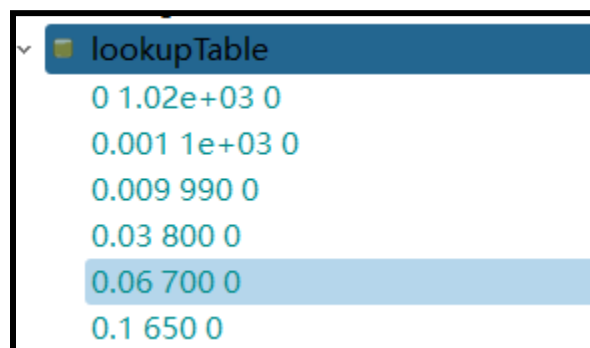
else:
    left_motor.setVelocity(speed-a*0.95)
    right_motor.setVelocity(speed+a*0.95)

```

The angle is subtracted or added from the baseline speed (2), which is necessary to maintain the original robot velocity. This is done to avoid turning in place and a smooth trajectory of the robot to the goal. The 5% ($a*0.95$) reduction is done to dampen the turn.

2. Walls and barrels:

(a) Define an APF model for each of the arena walls, so that the mobile robot will not collide into any of them. Explain your design clearly in the report, providing the necessary equation/s for the APF vector/s, dropoff, etc.



lookupTable		
0	1.02e+03	0
0.001	1e+03	0
0.009	990	0
0.03	800	0
0.06	700	0
0.1	650	0

Figure 4: Lookup table for the distance sensor

The APF model for the walls is implemented using the distance sensors on the robot. A vector is generated for each distance sensor, resulting in 8 vectors from 8 distance sensors. The magnitude of each vector is linked to the response from the sensor, and the direction of the vector is calculated relative to the robot's orientation. The corresponding function is *polar_to_cartesian*.

Python

```

#normalize
minValue = sensor.getMinValue()
maxValue = sensor.getMaxValue()
r = (r - minValue) / (maxValue - minValue)
if r == 0: return np.array([0, 0])

```

For the magnitude of the vector, we use the characteristics of the lookup table for the sensor to implement a min-max scaling of the sensor response. This allows us to maintain a balance between the original goal vector and the other repulsive vectors (similar to normalizing a dataset in machine learning).

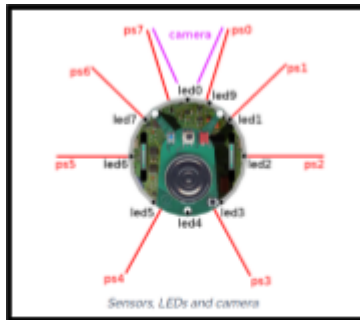


Figure 5: Epuck robot

Device	x (m)	y (m)	z (m)	Orientation (rad)
ps0	0.010	0.033	-0.030	1.27
ps1	0.025	0.033	-0.022	0.77
ps2	0.031	0.033	0.00	0.00
ps3	0.015	0.033	0.030	5.21
ps4	-0.015	0.033	0.030	4.21
ps5	-0.031	0.033	0.00	3.14159
ps6	-0.025	0.033	-0.022	2.37
ps7	-0.010	0.033	-0.030	1.87

Figure 6: Epuck robot distance sensor information table

For the direction, we use the e-puck documentation to get the angle of the sensors relative to the robot. First, we align the 0 rad angle to face the front of the robot, by subtracting 90 deg from ps0 to ps7.

Python

```
# calculate vector relative to the robot
x = r * np.cos(theta + robotAngle)
y = r * np.sin(theta + robotAngle)
```

Next, the robot might be facing in any direction on the board so we need the angle relative to the robot's orientation. To calculate that we add the angle of the robot's orientation to get the final direction of the repulsive vector. This is important because we use cartesian vectors, without this the vector directions are not in the right direction.

(b) Now, define an APF model for barrels i.e., cylindrical obstacles that are placed vertically on the floor, so that the mobile robot will move away from any of them, in addition to the walls. Explain your design clearly in the report, providing the necessary equation/s for the APF vector/s, dropoff, and other parameters as needed.

There are no changes made for the obstacles.

(c) Build a Webots world for the map in Figure 4, where three identical barrels are added to the arena. Each should be sized and positioned exactly as shown on the map above. Implement and test the simplest controller program as per your design. Experiment in simulation by placing

the robot at different initial locations and orientations, including especially the three corners away from the goal. Provide in your report screenshots (with trace) of your various experiments, with appropriate comments, including how you refined your model parameters to make it work.

Python

```
# calculate vector relative to the robot
x = r * np.cos(theta + robotAngle)
y = r * np.sin(theta + robotAngle)
```

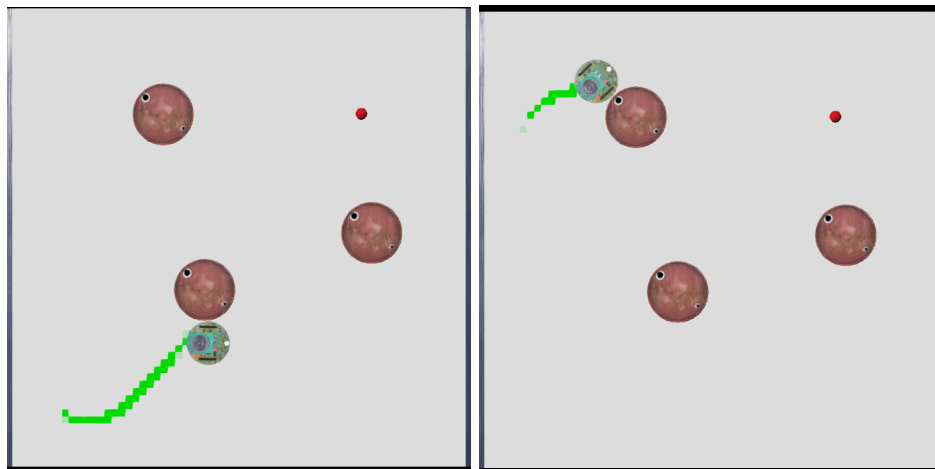


Figure 7: R scaled by 1

The magnitude of the repulsive factor (r) at the original scale. The robot makes contact with the barrel creating some friction, however, it ends up at the goal eventually.

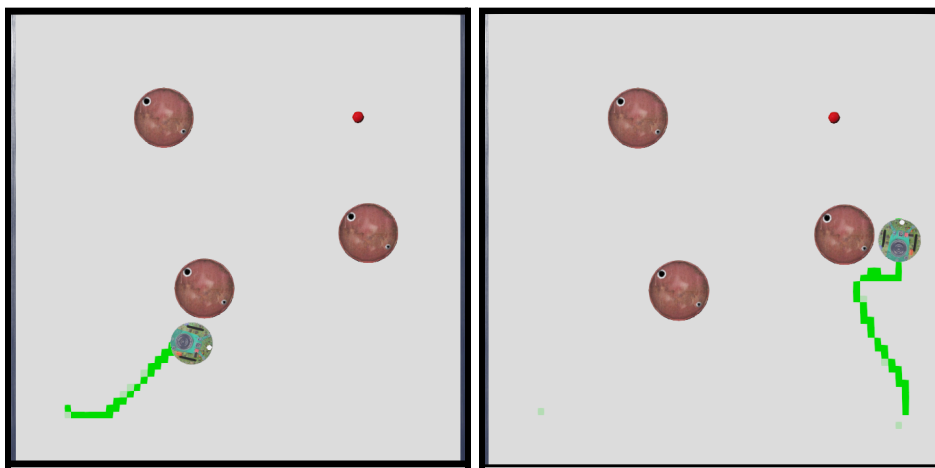


Figure 8: R scaled by 2

The magnitude of the repulsive factor (r) scaled up by 2. The robot comes very close to the barrel but does not make contact with the barrel.

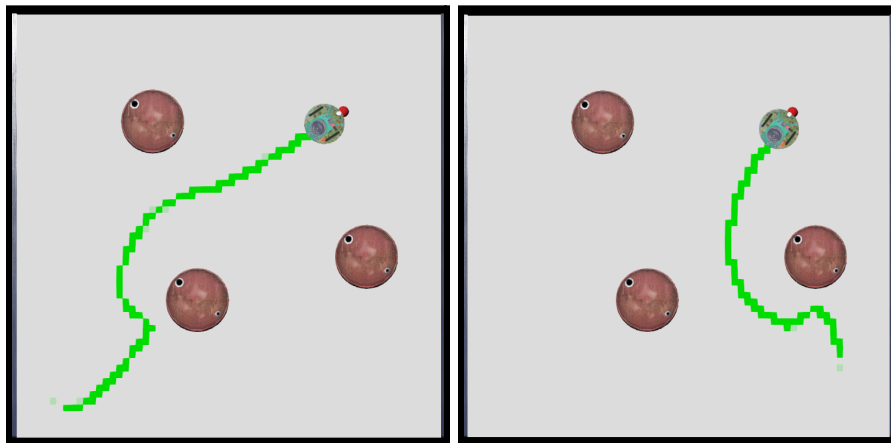


Figure 9: R scaled by 3

The magnitude of the repulsive factor (r) scaled up by 3. The robot safely avoids the obstacle.

(d) Identify a starting location and orientation that would cause the robot to be stuck, either by not moving at all or by repeating the same “back and forth” motions... Include in your report a screenshot (with trace) of this scenario. Explain exactly why it is happening, how you could fix it, and what the pros and cons of your modified design are. (If you cannot identify such a situation, make sure the size of the robot, arena, and barrels are exactly as specified.)

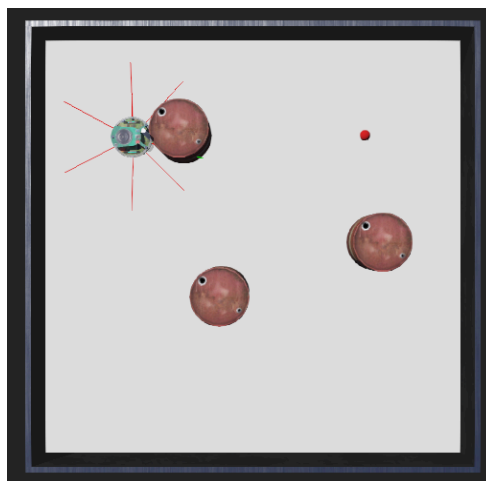


Figure 10: Robot stuck in front of the barrel (initial position)

When $r = 1$ and an obstacle exists between the robot's moving direction and the goal, it would be stuck (not moving at all).

The robot gets stuck when it starts in a location where the sensors detect a barrel on the way to the goal, as seen in Figure 10. This happens when the robot is facing the goal and encounters a

barrel lined up with the goal, the resultant vector from the barrel's repulsive force and the goal's attractive force cancel each other out. This results in the robot not moving at all.

Fix: Increase the repelling force of the robot. The robot will have priority in moving away from the obstacle.

Pros	Cons
Improved navigation, as it will be less likely for the robot to get stuck.	It will ignore the shortest path to the goal.
Smooth movement, optimizing the repelling force for the robot can lead to smoother turns	Higher chances to get stuck in a back and forth loop

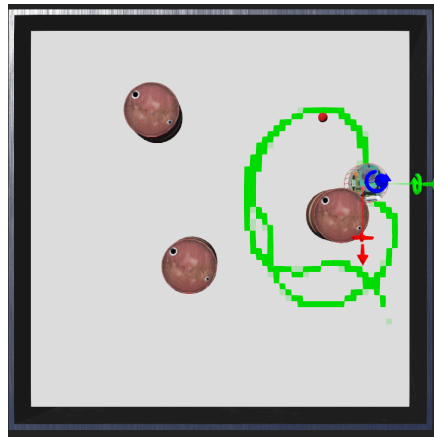


Figure 11: Robot keeps rotating around the goal

Another scenario is when increasing the repulsive force ($r=50$) and the speed ($\text{speed}=4$). The robot can be seen to take wide turns from the obstacles, but it cannot reach the goal due to the increase in speed and repulsive force. As seen in Figure 11, it takes a turn around the goal and ends up stuck on the barrel and cannot turn.

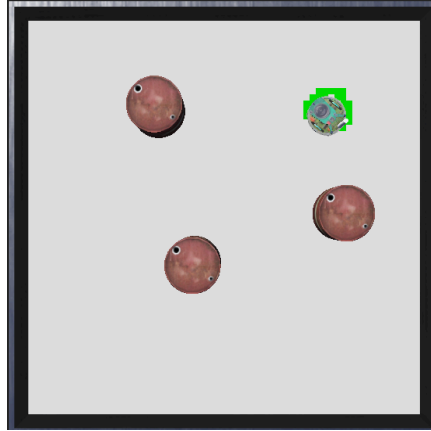


Figure 12: Robot stuck rotating around the goal

As seen in the figure above, the robot will not reach the goal point and instead, “circle” around the goal infinitely. This is evident from the trace of the robot. The reason behind this behavior is the fact that the motor speed of the robot is set too high for the robot to rotate enough to reach the goal point. As such, the robot will rotate **around** the goal point trying to reach the goal but will not be able to do so.

(e) Explain the general idea of how we should modify the APFs to make the robot reach the goal destination as fast as possible, while avoiding collision of course. Similarly, explain the general idea of how we should modify the APFs so that the robot trajectories will be as safe as possible. For bonus marks, feel free to actually implement these scenarios, including in your report some screenshots (with trace).

Speed:

The general idea to achieve speed would be to minimize the repulsive force so the robot would take a shorter detour around the obstacle, obviously, it should not be so low that there are collisions.

Another thing to consider is the turning radius, that is the turning ability of the robot. While, higher speed is the goal here, making the robot go faster would constrain the turning ability. Therefore, there needs to be a balance between the turning ability and the speed.

Safety:

The general approach to maximize the safety would be to have a high repulsive force from the obstacles, a force that is definitely higher than the minimum required to avoid the obstacles.

The turning ability matters here too, as the robot should be able to turn precisely enough to allow traversing through narrow spaces. The speed should definitely be low here so the robot not only has a high turning ability but also time to account for the repulsive forces being calculated from the distance sensor. This is correlated to the distance when the sensor starts to get a reading.

Experimentation:

We will adjust the repelling force on the robot from the obstacle, i.e., the r factor/distance sensor reading.

To reach the goal as **fast** as possible, we can reduce the repelling force of the obstacles. In that case, the robot will just look for the shortest path between its position and the goal (as illustrated in Figure 13). However, it might not be possible to recognize the blocked path (as illustrated in Figure 14).

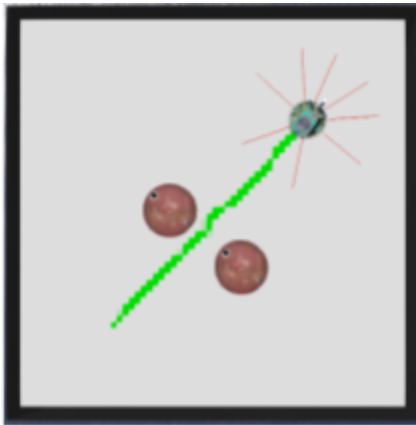


Figure 13: Robot goes through the fast route (experiencing low repulsive force)

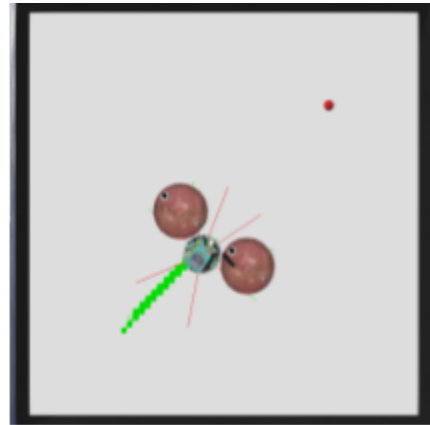


Figure 14: Robot being stuck at the fast route

To reach the goal as **safely** as possible, we can increase the repelling force of the obstacles. So, the robot's avoid-obstacles mechanism will be given higher priority.



Figure 15: Robot goes through the safe route (experiencing high repulsive force).

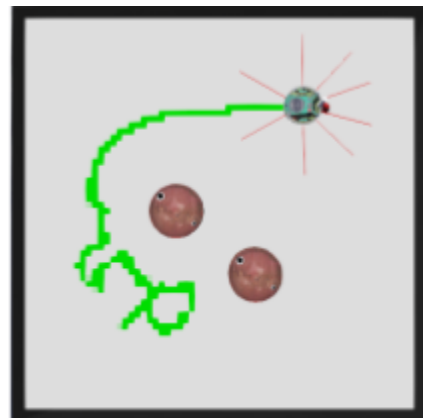


Figure 16: Barrel with increased (passable) distance.

In this scenario, the robot will not be stuck on a path that it can't pass through (as illustrated in Figure 15). Instead, it will choose to take the longer route, rather than risk going through a small path (as illustrated in Figure 16).

3. Crowded warehouse:

(a) Consider the map of Figure 5, which shows a warehouse with five barrels. Explain what new issues will appear that did not exist previously. Modify your APF model of the obstacles to address this issue as much as possible, and provide accordingly the revised equation/s for the APF vector/s, dropoff, etc. Indicate the pros and cons of the new design.

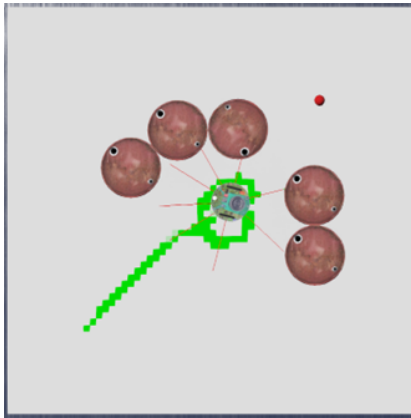


Figure 17: Robot is stuck in a loop

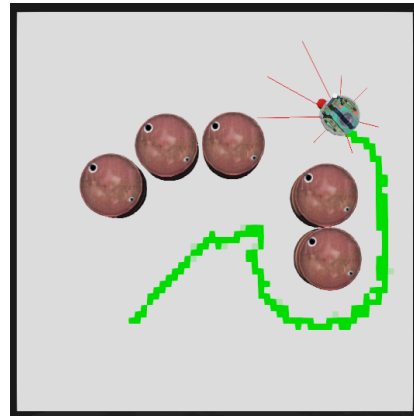


Figure 18: Robot finds a path around the obstacles using the new APF model

We can see in the above figures, a new issue appears where the robot will go in circles because of the surrounding barrels. This happens because the robot is acting extremely safe; even though there is space for the robot to pass, it will never try it because the repulsive forces are too strong. R is scaled by 10 in this case.

We address this issue by changing the lookup table (as illustrated in Figure 19) of the distance sensors, i.e., the maximum range of which the sensor can detect. As illustrated in Figure 16, the Robot has three (ps0, ps6, ps7) long-distance sensors in the front, and the rest of the five distance sensors are short. Besides, we increased the repulsion factor to $r = 14$.

lookupTable
0 1.02e+03 0
0.001 1e+03 0
0.009 990 0
0.12 650 0

Figure 19: Lookup Table of ps0, ps6, and ps7

Pros	Cons
------	------

It doesn't risk going to the fastest path (which might cause it to be stuck), and it figures a path out without going in a circle as illustrated in Figure 15.

It doesn't work in a more crowded warehouse

(b) Build a Webots world for the map in Figure 5, with the five identical barrels positioned exactly as shown. Implement and test a controller program as per your revised design. Experiment in simulation by placing the robot at different initial locations and orientations, including again the three corners away from the goal. Provide in your report screenshots (with trace) of your experiments, including comments about the issues faced and how you addressed them.

Top Left:

When placing the robot in the top left corner we can see that it behaves normally by moving towards the goal and deviating a little bit as it gets closer to the last barrel.

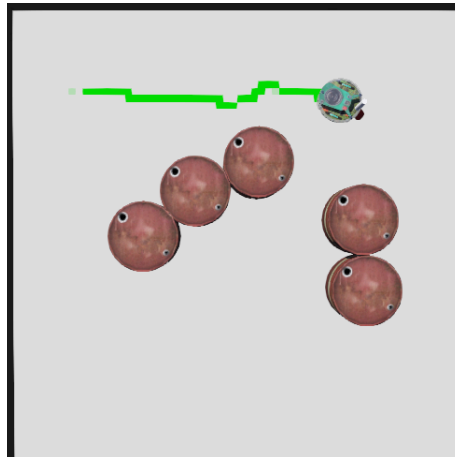


Figure 20: Robot initial position top left

Bottom Right:

When running the simulator with the robot placed in the bottom right corner, we can see that it squeezes itself into the gap between the barrels and the wall to reach the goal.

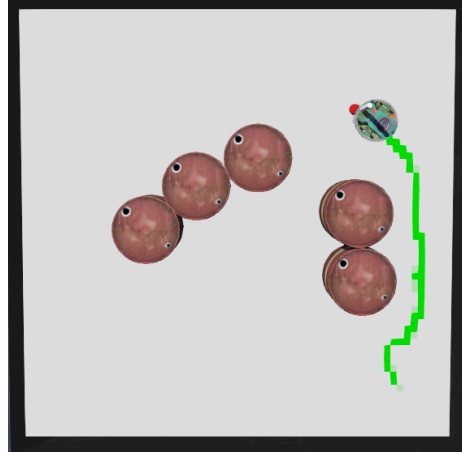


Figure 21: Robot initial position bottom right

Bottom Left:

Once we use the original scaling for R (3), the robot is able to pass through the obstacles as now it is not maintaining high separation from the barrels.



Figure 22: Robot initial position bottom left with $r = r*3$

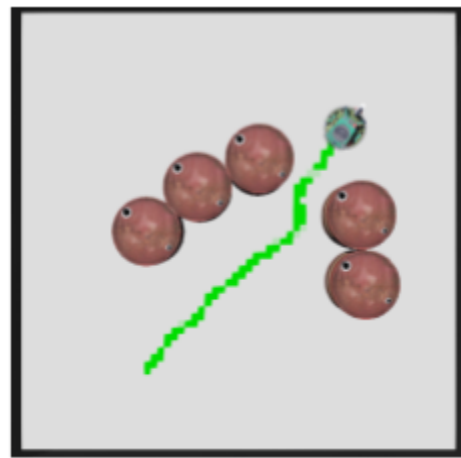


Figure 23: Robot initial position bottom right with $r = r*10$ and speed = 2

However, when adjusting the R value to a factor of 10 and the speed to 2, we can see that the robot loops around and goes around the barrels. As the repulsive force is too high for it to go through the barrels.

(c) Now, build a Webots world for the map in Figure 6, with the additional barrels positioned exactly as shown. Experiment in simulation with your model in part (b), by placing the robot at different initial locations and orientations, including again the three corners away from the goal. Provide in your report screenshots (with trace) of your experiments, then comment on what you see.

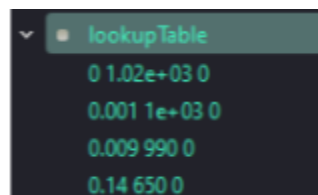
Explain how you could possibly modify your APFs so they still work in this more crowded environment. Indicate the pros and cons of the revised design, considering not only different

starting positions but also different goal locations. For bonus marks again, feel free to actually implement these models, including in your report some screenshots (with trace).

The APF model does not change much for this environment. Most of the model parameters remain constant with very minor changes to the scale of the repulsion forces as well as the speed. Additionally, the distance sensor PS[0] was changed to reduce the range to 0.04 from 0.14.

For the scenarios below, $r = r*11$ and $speed = 1$

Ps[1], PS[7] =



lookupTable	
0	1.02e+03 0
0.001	1e+03 0
0.009	990 0
0.14	650 0

Figure 24: Lookup table for ps1 and ps7

Here the lookup table enables the distance sensors PS[1] and PS[7] to detect obstacles in the max range of 0.14m.

Top Left:

When placing the robot in the top left corner we can see that it behaves normally by moving towards the goal and deviating a little bit as it gets closer to the last barrel.

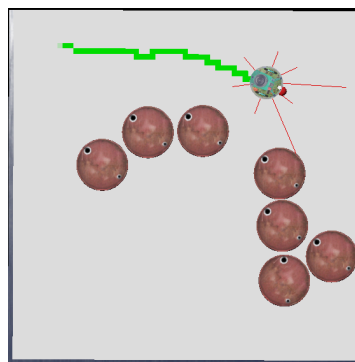


Figure 25: Robot initial position top left

Bottom Left:

When running the simulator with the robot placed in the bottom left corner, we can see that it initially travels towards the gap between the barrels but then traverses around the barrels on the left.

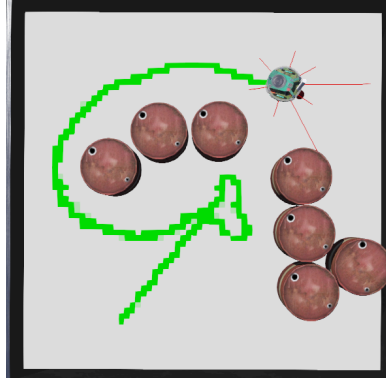


Figure 26: Robot initial position bottom left

Bottom Right:

When running the simulator with the robot placed in the bottom right corner, we can see that it initially travels towards the gap between the barrels but then traverses around the barrels on the left. This is again similar to the behavior when the robot's initial position is in the bottom left of the arena.

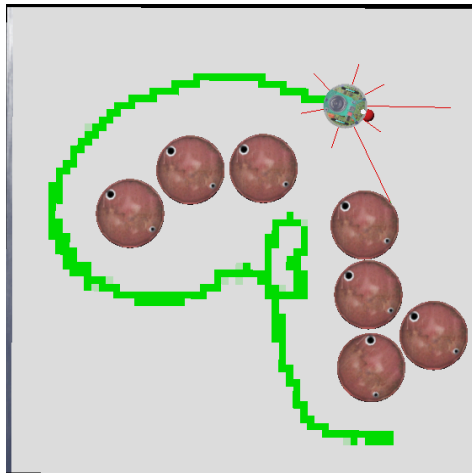


Figure 27: Robot initial position bottom right

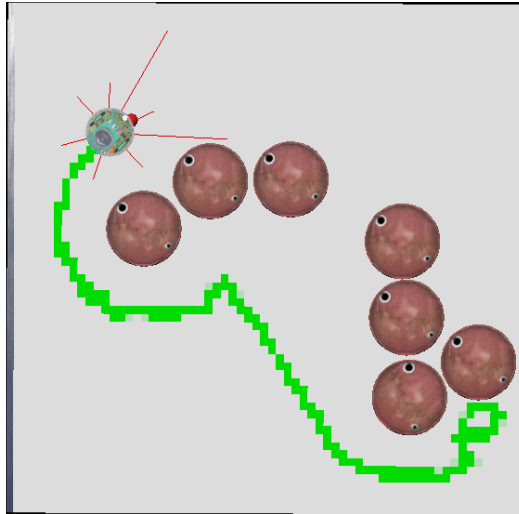


Figure 28: Robot initial position bottom right with goal at $[-0.2, 0.2]$

Bonus: Different barrel positions (more crowded):

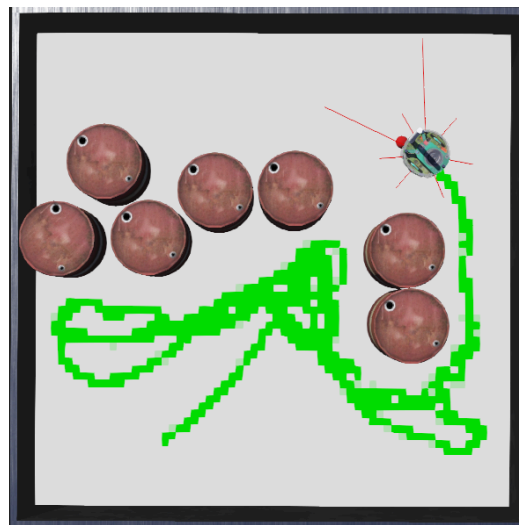


Figure 29: Robot initial position bottom left with crowded left

$r = r*14$

The robot initially tries to find a path on the left. Since it is crowded, it tries to go in the opposing direction and eventually finds a path on the right side of the arena.

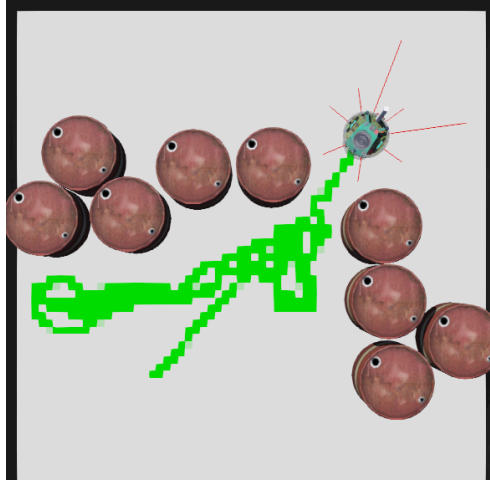


Figure 30: Robot initial position bottom left with crowded left and right

Here, the robot is met with no paths on the sides of the arena and is therefore forced to go between the barrels.