Analysis of a roomba robot simulation, comparing different moving strategies

Vy Ung

Minerva Schools at KGI

I/ Project description:

1/ General description:

In this project, I modeled the movement of a Roomba robot in a rectangular room with some obstacles given different strategies. The goal of this simulation is to evaluate different movement strategies and their effectiveness at covering floor areas.

2/ Rules:

The room that the robot has to clean has rectangular shape with specific sizes in different dimensions. Size along the x-axis is called "width" and size along the y-axis is called "height". This room is represented as a grid of discrete tiles and each tile is labeled as the coordinate of its left lower vertex. Therefore, the object RectangularRoom will have an attribute all_tiles which is a list of tuples, each of which contains the label of each tile. It also contains an attribute that stores all the cleaned tiles, this would be initialized to be empty because no tile is cleaned in the beginning. Some tiles would be randomly assigned to represent obstacles in the room and their labels are also stored in a separate list.

In my simulation, even though the room is discrete, the robot is allowed to move in continuous manner depending on its current direction and velocity. This means that the whole one tile would be visited at one time step if the robot coordinates lie within that tile. For example, if the robot is at position (0.5, 0.7), tile (0,0) would be marked as cleaned on our board if it is an empty tile

Because the rule of the model is at most one tile would be cleaned at one step, the velocity of the robot should be between 0 and 1, to prevent it from going through two tiles at one time step.

The direction of the robot could be represented as angle, in the unit of degree, so it should range from 0 to 360 degrees. To calculate the new position of the robot after one time step, we use the formula:

$$new\ x\ coordinate = current\ x\ coordinate + velocity * cos(angle)$$

$$new\ y\ coordinate = current\ y\ coordinate + velocity * sin(angle)$$

The robot is initialized to start at the middle of the room, with a random direction. I chose the middle because that is where the robot is less likely to be completely blocked (it needs 8 obstacles to block a robot from the middle). The tile where the robot starts would be marked cleaned, if that tile is not an obstacle. The only problem with random initialization of obstacles is that the robot may start right on the tile that is an obstacle, which is a bit unrealistic, but as long as we don't mark that tile as clean, the analysis of the model is still fine.

I implemented three different types of robots which correspond to three moving strategies:

- The standard robot is the robot with a standard movement strategy. At each time step, it attempts to continue moving in its current direction until it hits a wall or an obstacle. When it hits a wall or an obstacle, it changes its direction to a new random angle.
- The wall following robot is the robot with a wall following movement strategy. I created this strategy myself and it is not the same as the typical wall following strategy. Basically, at each time step, it attempts to continue moving in its current direction until it hits a wall or an obstacle. When it hits a wall, it reacts the same as the standard robot - choosing a new random direction. However, when it hits an obstacle, it will change direction to +90 degree (turning right) in regards to the current direction, and in the immediate next time step, it would change direction to -90 degree in regards to that direction (turning left), to get back to

its original direction in the previous step. This is also similar to how us human being reacts when we see an obstacle - we move a bit to either left or right side and then following the edge of the obstacle until the original path is free from obstacle, then we move back to that path . Because this rule involves less randomness, the robot is easy to get stuck; therefore I also calculated the "gap" which means the number of times that the robot stays at the same tile and if the robot stays at one tile after more than 10 time steps, it will change it direction randomly in the next time step.

- The random walk robot is the robot with a random walk movement strategy. It changes its direction randomly after every time step.

3/ Modeling assumptions:

Even though the movement of robot is not discrete, the room is discrete, which is similar to a cellular automaton that consists a regular grid of cells. Each cell has a finite number of states: either cleaned, not cleaned or is obstacle.

One of the assumption in this model is the robots have unlimited battery power. These robots are produced specifically for the purpose of testing to see what strategies work best. The robot will only stop after it reaches the goal (the proportion of room that needs to be clean), or if it is in failure mode (pass 10000 time steps and still gets stuck). In real life, this is not the case. An alternative simulation to avoid this assumption is to have the proportion of room cleaned as the quantity of interest, given a limited amount of power resource.

4/ Some failure modes:

Because the obstacles are initialized randomly on the grid, we cannot avoid situation where one tile is blocked by obstacles on other tiles. As you see in the figure on the left, the middle tile is blocked

horizontally and vertically by 4 obstacles; however, because the robot can move diagonally, it would not be a problem, even though for some strategies with less randomness, the robot may have higher chance of getting stuck in the middle tile after it managed to get there. However, the case on the figure on the right is quite different. The two tiles on the upper right hand side are completely blocked from the rest of the floor by 4 obstacles and they would never get cleaned. In scenarios where we have quite many tiles that are blocked, it may take outstandingly for the robot to fulfill the goal of cleaning a portion of the area, or in the worst case it will never fulfill the goal. Moreover, as mentioned, even when tiles are not completely blocked, there is a potential that the robot gets stuck at one corner given the strategy it is following (most of the time with wall following strategy) and it will also never reach the goal.

This is the reason why I implemented the simulation in a way that if the robot gets stuck or cannot reach the goal after 10000 time steps, we break the loop and don't include the result of that trial, to avoid skewness in the final distribution.

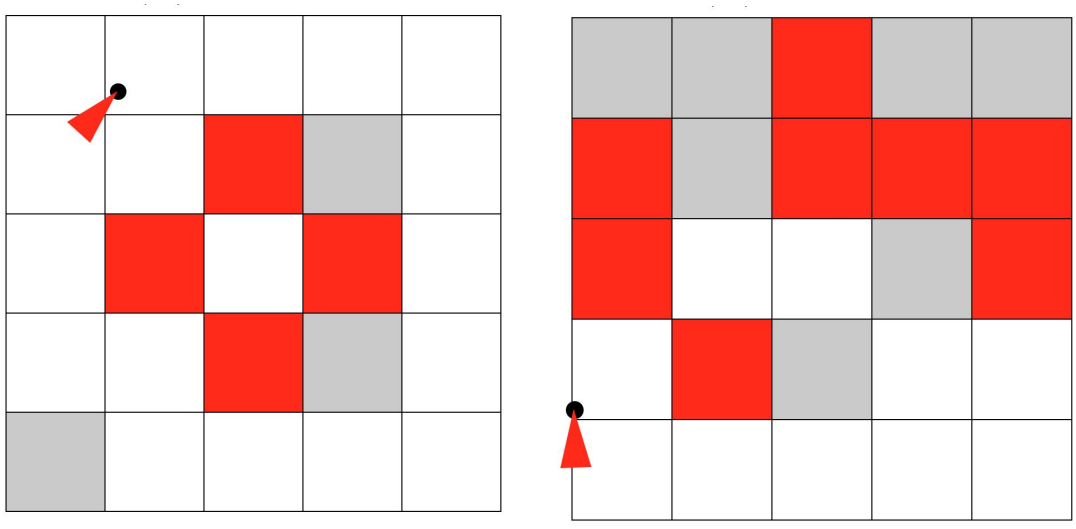

***Figure 1**: Comparison of scenarios where tile is completely blocked (right) and not completely blocked (left)*

5/ Measurements:

The quantity of interest here is the total number of time steps it takes for the robot to reach its goal. This goal is measured as the proportion of the room are that is clean. I was also thinking of other quantities such as resources (battery power), or number of tiles revisited. However, they are all directly proportional to the number of time steps. The longer it takes, the more battery power is used and it also means that more tiles are revisited multiple times.

6/ Parameters:

There are certain parameters of the simulation that are important: the number of obstacle present in the room, the aspect ratio of the room (width/height), and the minimum coverage required for the robot to reach its goal. I would analyze how they affect the behavior of the simulation and compare the performances of different robots in regards to those parameters in the next section - "Result Analysis"

II/ Result Analysis:

1/ Comparing strategies:

Figure 2 shows us how much time it takes for the robots to clean minimum coverage of the floor. Minimum coverage is the minimum proportion of the floor that the robot has to clean to reach goal. In this simulation, only minimum coverage is changed, other parameters are chosen to be {velocity: 1, width: 5, height: 5, number of obstacle: 5} and kept the same. We can see that the average time needed increases very slowly when the minimum coverage increases from 0.2 to 0.8, then rockets (by around 10 times) when the minimum coverage increases from 0.8 to 1. This result leads to a

good insight that if we want to trade off between time and coverage, it is reasonable to set the goal to be around 0.8 so that we can save time by 10 times, which also corresponds to resources (battery power).

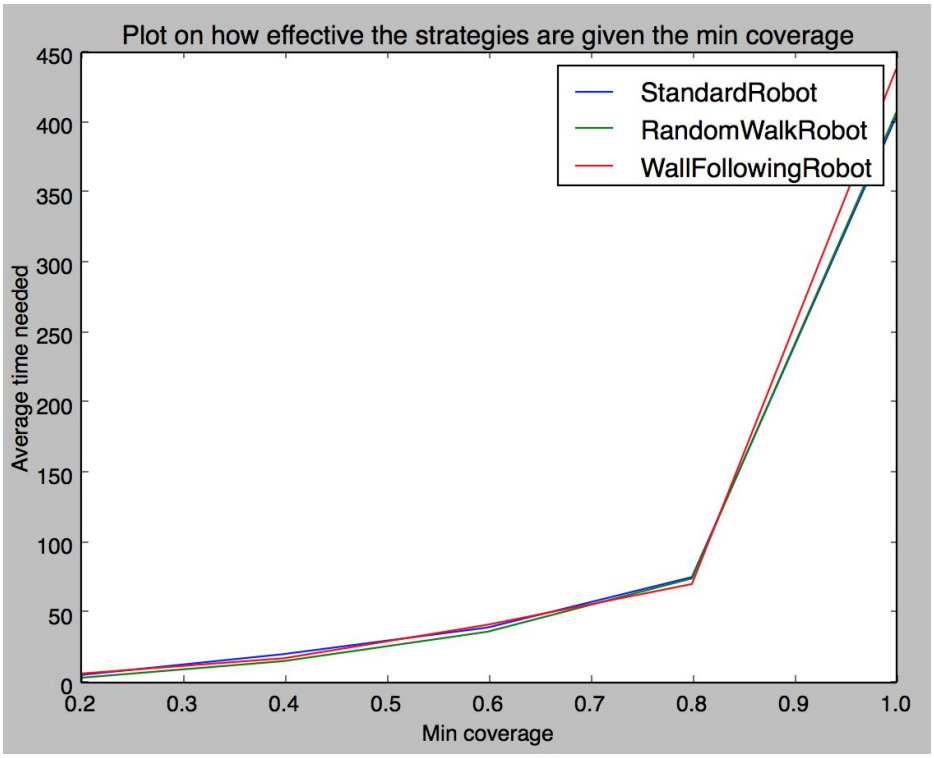

***Figure 2**: How effective different strategies are given minimum coverage of the floor for the robot to reach goal.*

Figure 3 shows us how much time it takes for the robots to reach goal given different proportion of room that is occupied by obstacles. In this simulation, only proportion of obstacles is changing, other parameters are chosen to be {velocity: 1, width: 5, height: 5, minimum coverage: 0.8}. The more obstacles there are, the longer time it takes, because the robots have to navigate itself around the obstacles. As we can see from Figure 2, when there is up to 1/5 area of the room occupied by obstacles, the strategies seem to work equally effective. When there are more obstacles than that, Random Walk Robot needs the least time to reach goal, while Standard Robot needs the longest

time; Wall Following Robot works just slightly better than Standard Robot in this case. This result makes sense because due to more randomness in changing direction, Random Walk Robot could easily get out of a corner when it is bounded by obstacles, while Wall Following Robot and Standard Robot take longer time if they get stuck.

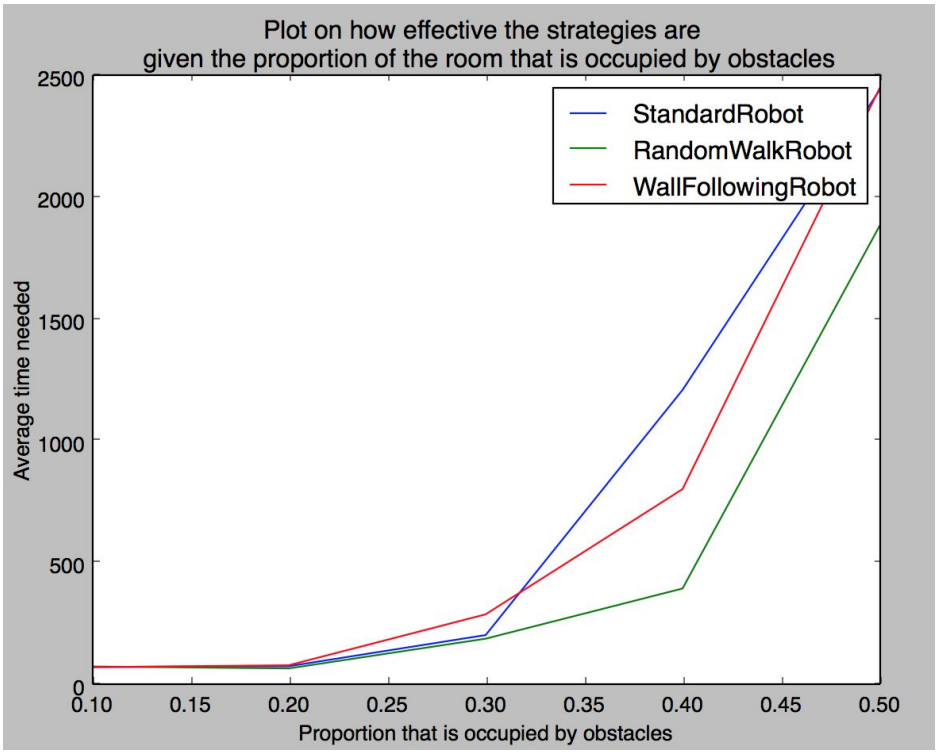

***Figure 3**: How effective different strategies are given the proportion of the room that is occupied by obstacles*

Figure 4 shows us how much time it takes for the robots to reach goal given different aspect ratios of the room. Aspect ratio in my implementation is defined as the ratio of width over height. In this simulation, only the aspect ratio is changing, other parameters are chosen to be: {velocity: 1, number of obstacle: 5, minimum coverage: 0.8}. For Standard Robot and Wall Following Robot, the time it takes to reach goal only slightly increases when the aspect ratio increases, which means that their efficiency is not very sensitive to the change in shape of the room. However, for Random Walk

Robot, the time it takes to reach goal increases significantly when the room changes from very square (aspect ratio nearly 1) to very long rectangle (aspect ratio nearly 30). This makes sense because for Standard Robot, the robot remains its direction if not seeing obstacles, and for the Wall Following Robot, after it hits an obstacle, it tries to divert but still follows the obstacle wall to get back to its initial direction. In the case of long room, even though the time it takes is relatively longer for square room, this behavior still helps because they can potential traverse a greater length until they have to change direction, thereby taking less time to get from one end to the other end. In contrast, because Random Walk Robot changes its direction at each time step, when the room is long, it takes the robot longer time to get from one end to the other end of the room.

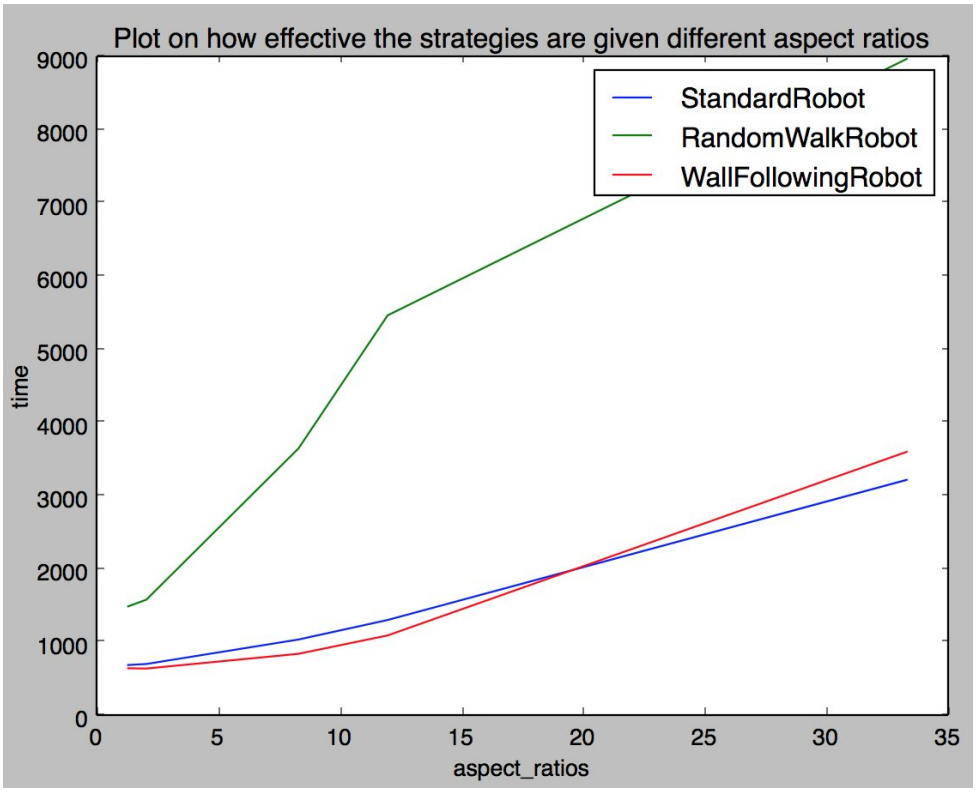

***Figure 4**: How effective different strategies are given different aspect ratios of the room.*

Based on the analysis of the results, Random Walk Robot seems to work better than other two strategies in handling obstacles. Therefore, if we are given a room with a lot of obstacles, I would advise to go for this option. However, if the room is too long, then I believe that we should choose either Standard Robot or Wall Following Robot.

2/ Distribution of output:

I ran the simulation for 100000 trials with the following parameters: {velocity: 1, width: 5, height: 5, number of obstacle: 5, min coverage: 0.8} and got the distributions over the total number of time steps it takes for robot to reach goal, as illustrated in Figure 5, 6, and 7. All three of the distributions corresponding to the three strategies are skewed to the right. There are more extreme outlier in the case of Wall Following Robot in comparison to Standard Robot and Random Walk Robot. This is because for Wall Following Robot, there is less randomness involved in the rule of changing direction, it is easier to get stuck in one corner (the area that is bounded by a lot of obstacles) and takes much time to get out of that corner.

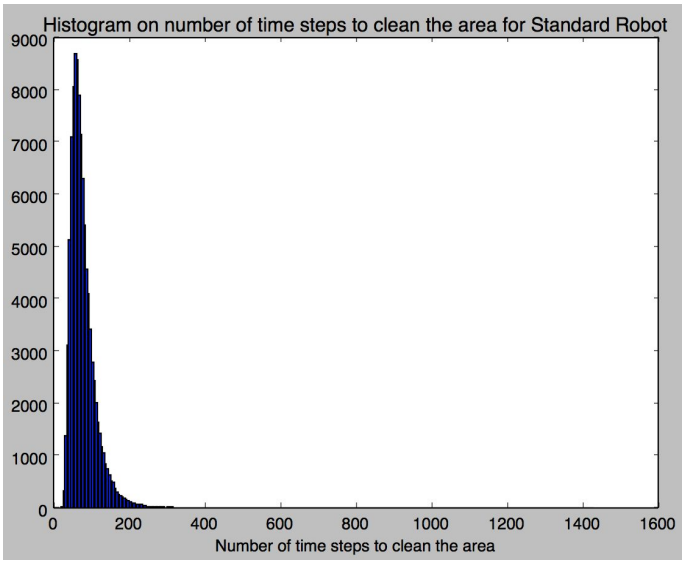

***Figure 5**: Distribution of time for Standard Robot*

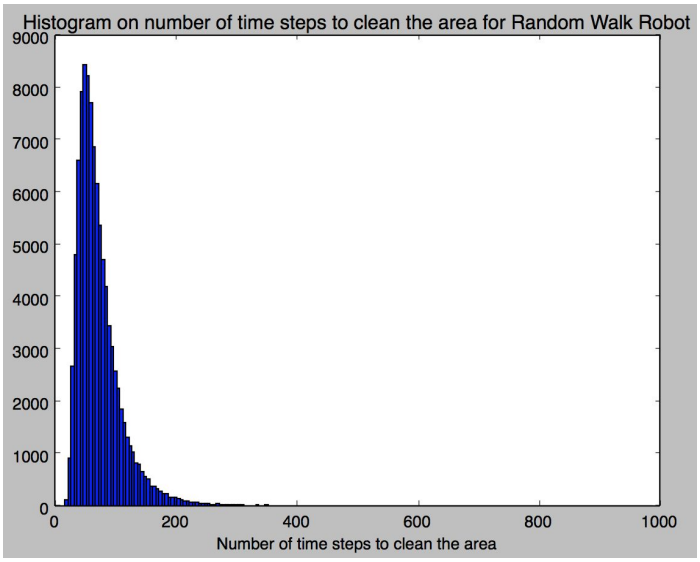

***Figure 6:** Distribution of time for Random Walk Robot*

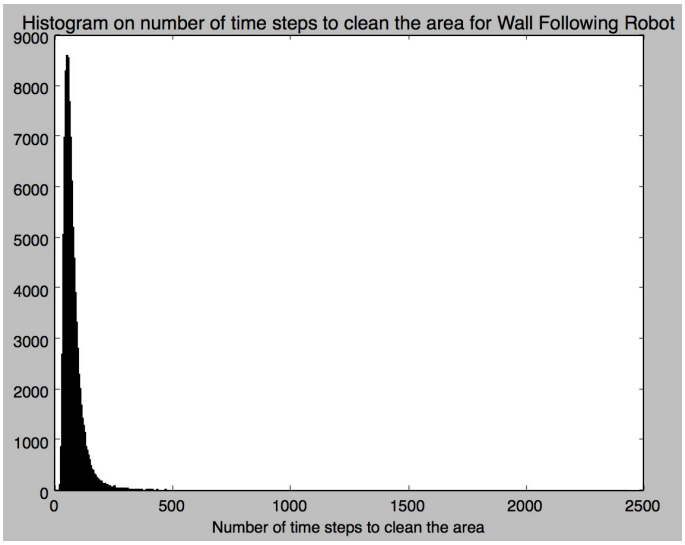

***Figure 7:** Distribution of time for Wall Following Robot*

3/ Uncertainty in result:

The distributions of the sample means are supposed to be normal. However, because I didn't have enough time to run for more number of simulations, they do not look as normal as expected. To

produce figure 8, 9 and 10, I ran the simulation 100 times, and computed the mean. I repeated that same process for 1000 times to get the distribution over the sample means. One interesting thing is for the Wall Following Robot, the distribution looks bimodal, and for the Standard Robot and Wall Following Robot, we also see some of the outlier values around 180. These values make the standard deviations to be very large, which leads to wide confidence intervals.

It makes sense that the larger number of samples we take (i.e. the number of simulations run), the more likely it is that we get to the true mean of the population at each repeat. Therefore, the larger number of samples, the smaller standard error is. Based on statistics, specifically the central limit theorem, the variance of the sample means distribution is equal to the variance of the original distribution divided by number of samples.

$$\sigma_{\bar{x}}^2 = \frac{\sigma^2}{n} \Leftrightarrow \sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

Therefore, for example in the case of Random Walk Robot, if I want to reduce the standard deviation to, say around 1 (which means 1/10 of the original standard deviation), thereby reducing the width of our confidence interval, we have to increase the number of simulations by 100.

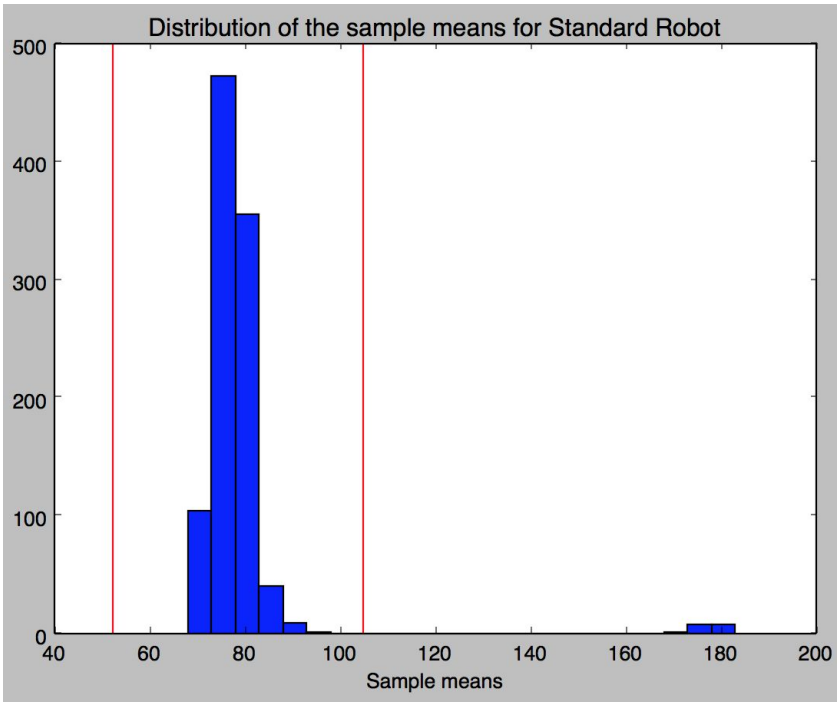

Figure 8: *Distribution of sample means for Standard Robot*

Mean: 78.506

Standard deviation: 13.452

Confidence intervals:

(52.141, 104.871)

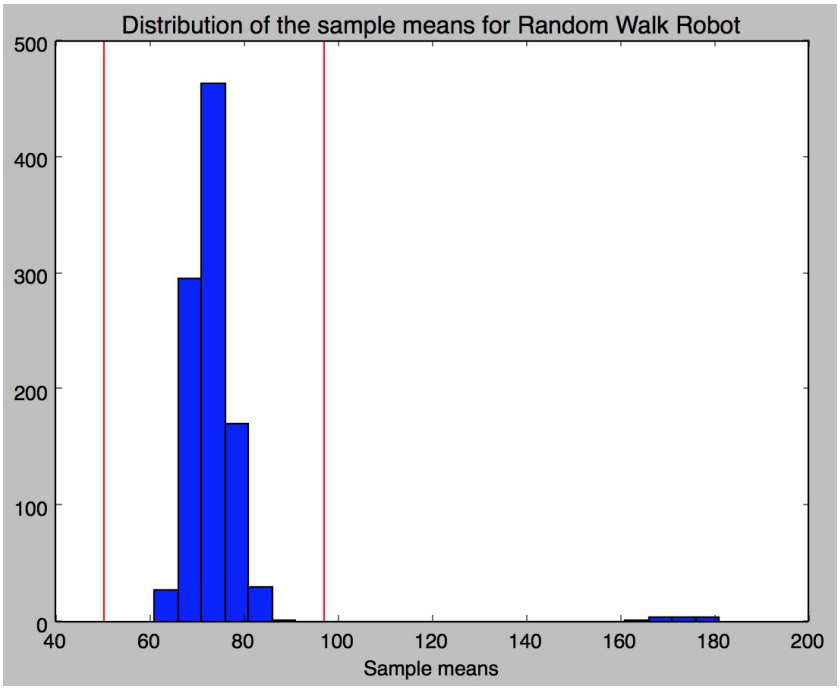

Figure 9: *Distribution of sample means for Random Walk Robot*

Mean: 73.732

Standard deviation: 11.943

Confidence Interval:

(50.324, 97.14)

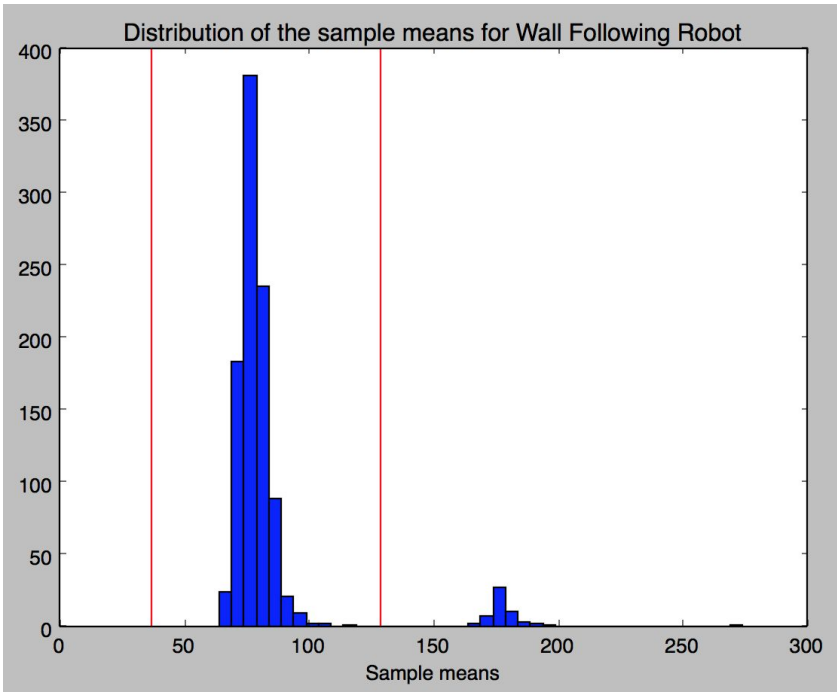

Figure 10: *Distribution of sample means for Wall Following Robot*

Mean: 82.951

Standard deviation: 23.536

Confidence Interval:

(36.82, 129.082)

REFERENCE

raun/Roomba-Vacuum-Robot. (2012). GitHub. Retrieved 20 April 2018, from https://github.com/raun/Roomba-Vacuum-Robot/blob/master/ps7_visualize.py