

Research track 2, assignment 1

Data Analysis

Introduction:

This last portion of the Research Track assignment 2 consists in going through a statistical analysis of the Research Track 1 first assignment. The aim of this project was to code a **python script** capable of making a simulated holonomic robot sequentially grasp and released some targets (**Silver tokens**) inside of an arena composed by **golden tokens**. This Assignments aims at comparing and testing the Behaviour of the robot considering two different implementations of the code:

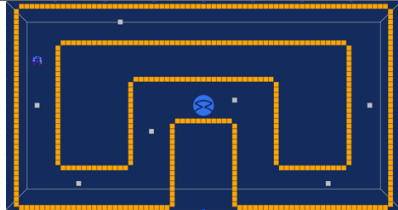
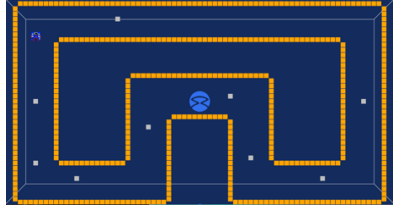
- My own implementation turned in at the end of the project and published on GitHub at the following repo: https://github.com/Fabioconti99/RT1_Assignment_1.
- The teacher implementation Cloned at the following repository: https://github.com/Carmined8/python_simulator/tree/rt2.

To evaluate some differences between the two implementations, I choose to collect the following parameters:

- The **distance between the robot and the closest golden token**. To retrieve these values, I opted for the implementation of an extra function within the code that would register the closest distance to a golden token whenever called. This function (*my_gold_token_list()*) exploits the *R.see* method implemented within the simulation's robot class. A for-cycle embedded in the function will spin all the gold token detected within a 100 units range by the robot. The function will record all the relative distances between the robot and the token and it will save the closest. All the values will be published run time on a *.txt* document.
- The **time the robot takes to finish a lap**. Some lines of code added within the main executed function, will register a current-time value every time the whole number of targets are reached. Once a count matches the number of silver tokens distributed in the scenario the previous current-time value will be subtracted to the current on obtaining the total time the robot took to finish the single lap. This value will later be published on a dedicated *.txt* file.
- The **number of times the simulation failed**. This could potentially happen in different ways: the robot hits a wall of golden tokens; the robot starts going towards the opposite direction of the track, the robot doesn't grasp the reached token at the first try. All this data will also be reported on a *.txt* file.

I decided to manage these parameters in 2 different data sets described by the following tables.

Set 1:

Number of laps (for each run)	Number of tokens	Image of the map:
5	7	
5	9	

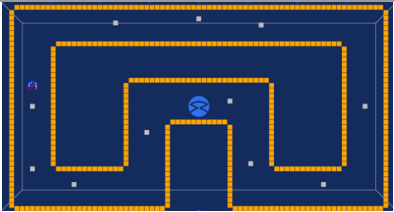
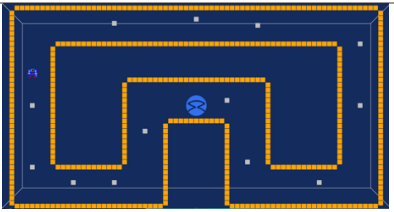
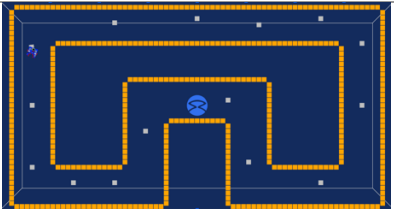
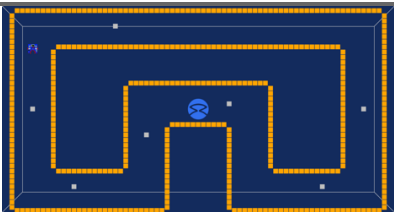
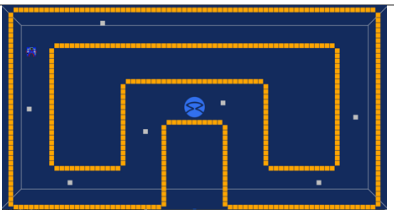
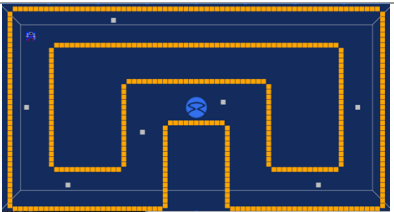
5	11	
5	13	
5	15	

Table 1

The analysis computed through this configuration wants to highlight how the Behaviour of the robot changes with the rise of the number of tokens. Graphs and test will later prove the difference between the two codes underlining which of the two performs better and in what field.

Set 2:

Number of laps (for each run)	Number of tokens	Image of the map:
10	7	
10	7	
10	7	

10	7	
10	7	

Table 2

Thanks to this setting, the analysis will be able to compare how the two different implementations managed the speed of the task and the safety distance between obstacles. The numbers of tokens don't change between executions, only their positioning does. The silver tokens will move closer and closer to the edge of the arena. The analysis will evaluate how the robot deals with the grasping of the targets. The following graphs and tests will compare the performance of the two computations, concerning differences in timing and distances from the obstacles.

Analysis and evaluation of the compared parameters

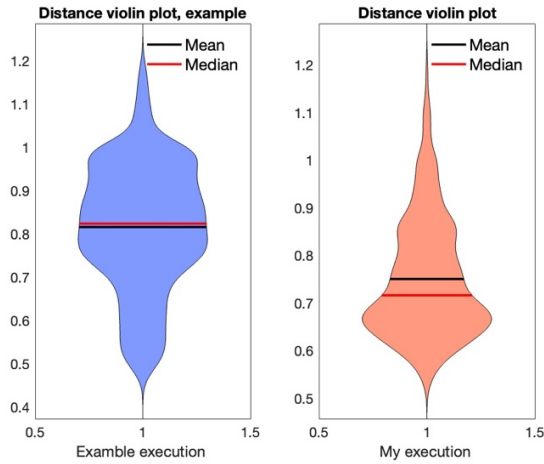
The software used to process the data and conduct the whole statistical analysis is *Matlab* (version: 2022a). This software is a programming platform designed specifically for engineers and scientists to analyze and design systems.

Gold distance:

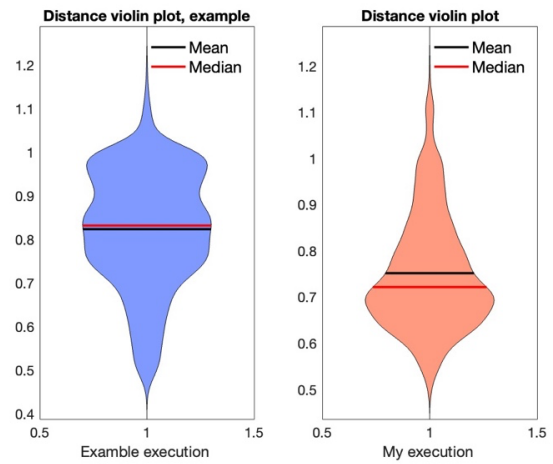
For the statistical analysis of this parameter, I took advantage of both the data sets retrieved from the simulation. Concerning the first set, I collected all the data related to the single simulation in a unique array. This array divides each of the runs with the number 20000. This value is exploited within the *get_gold_dist.m* *MatLab* function to divide the different executions in different arrays collected in a specific cell.

Set 1 evaluation:

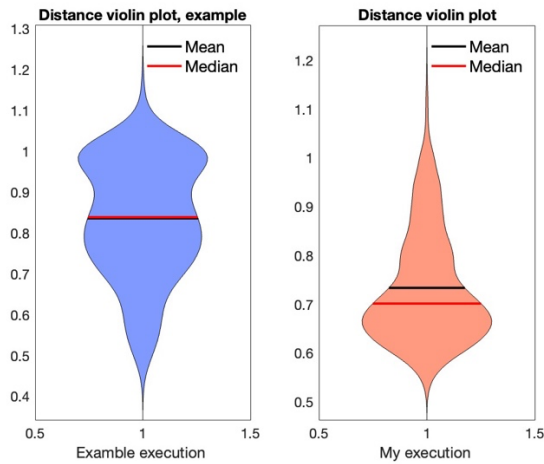
The Name of the following graphs is **violin plot** (*violin.m*). They compare the two sets of runs by showing the different distributions of values of registered for each run. Also, the values of **median** (: The median is the middle value of a set of data containing an odd number of values, or the average of the two middle values of a set of data with an even number of values) and **mean** (: aka average, is obtained by dividing the sum of observed values by the number of observations, n.) are shown in the graphs.



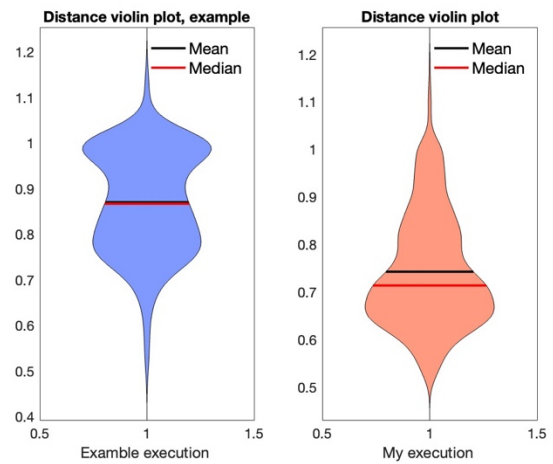
Picture 1: violin distribution graph



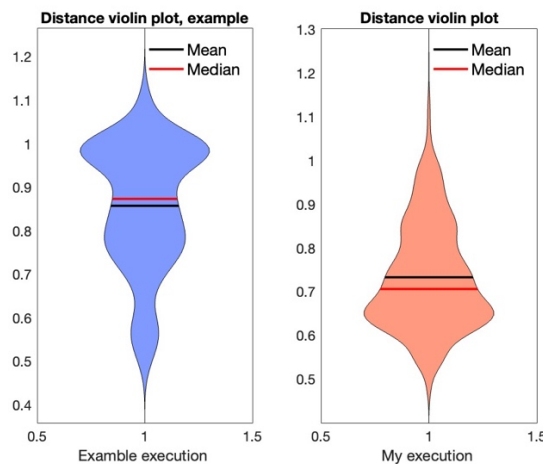
Picture 2: violin distribution graph



Picture 3: violin distribution graph



Picture 4: violin distribution graph



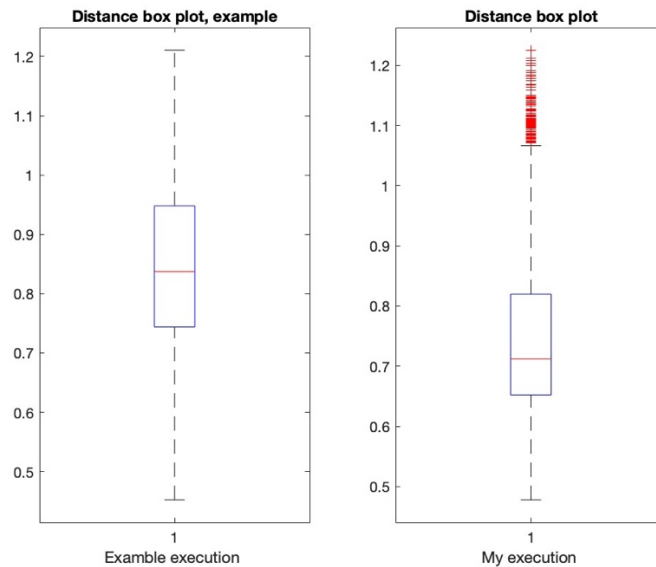
Picture 5: violin distribution graph

At first glance we can notice how similar the distributions coming from the same codes are. The algorithms implementing the robot's "obstacle avoidance" are determinant for the evaluation of this parameter. From these graphs it's easy to understand how different my assignment code is from the example one. My implementation for the gold walls avoidance aimed at turning away from the walls any time the front sensors would see one closer than .7 units. Since every time the robot must get aways of the trajectory of a

gold token it stops until the front sensors do not detect any gold tokens, the majority of the distances retrieved stays a little above that threshold.

Since the distributions were similar throughout all the different maps regardless of the number of tokens, I decided to also plot the distances for both the programs.

The following plot is called **box plot** (*boxplot.m*) and it gives a clear idea of the distance values distribution through a simple graphical representation.

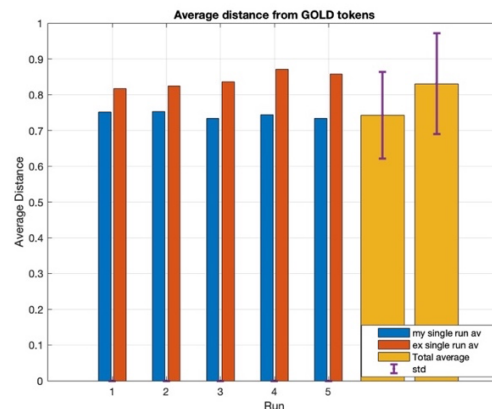


Picture 6: box plot distribution graph

The function creates a box plot of the data in x . If x is a vector, `boxplot` plots one box. If x is a matrix, `boxplot` plots one box for each column of x . On each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points not considered outliers, and the outliers are plotted individually using the '+' marker symbol.

Even in this case we can easily notice that the example code keeps a higher distance from the gold tokens making the robot's motion in the space generally safer.

The following **bar plot** (*barplot.m*) compares for every map the average distances from the walls and the last two columns show the total average and standard deviation (The standard deviation gives an idea of how close the entire set of data is to the average value. Data sets with a small standard deviation have tightly grouped, precise data. Data sets with large standard deviations have data spread out over a wide range of values) for each implementation.



Picture 7: average distance from gold tokens (*bar plot*)

T-Test

To ultimately conclude on this difference, I decided to compare the two implementations through a **T-test**. The T-Test, also known as Student's Test, is based on t-distribution, and is considered an appropriate test for judging the significance of a sample mean or for judging the significance of difference between the means of two samples in case of small sample(s) when population variance is not known (in which case we use variance of the sample as an estimate of the population variance). The relevant test statistic, t , is calculated from the sample data and then compared with its probable value based on t-distribution at a specified level of significance for concerning degrees of freedom for accepting or rejecting the null hypothesis.

I conducted this kind of test on the second dataset I retrieved from the simulation. For each map I took the average of every lap's wall distance obtaining two sets of 10 values. With this dataset I conducted a *one* tailed, paired T-test with 18 degrees of freedom.

A **one-tailed** test would be used when we want to conclude whether the population mean is either lower than or higher than some hypothesized value or if the statistics of our experimental group are either lower or higher than the statistics of the control group.

The **paired t-test** is a method used to test whether the mean difference between pairs of measurements is zero or not.

You can use the test when your data values are paired measurements.

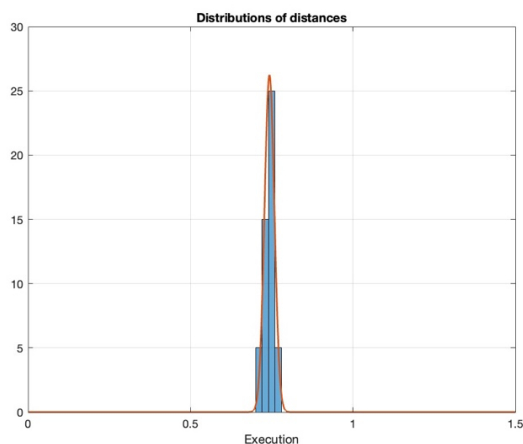
Before conducting the tests, we first must verify the assumption of normality distribution of each of the sets of 10 distance average and of the difference between the two sets to be compared.

This assumption can be tested with multiple tests such as the **Lillefors Test**. The Lilliefors test is aimed at proving the null hypothesis that the sample belongs to a normal distribution, however, without specifying which normal distribution, it does not specify the expected value and variance of the distribution.

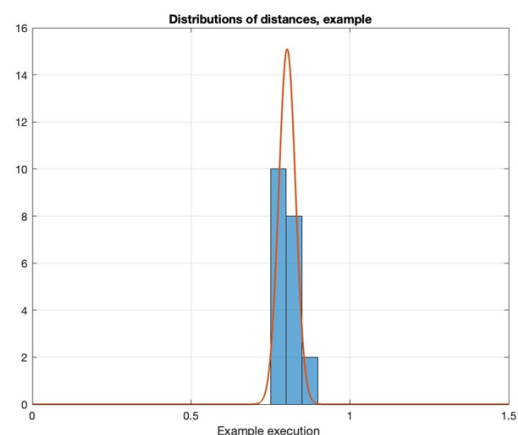
Map	H value
1	0
2	0
3	0
4	0
5	0

Table 3

The representations of the distributions are here expressed using the **histogram plot** (*histogram.m*). These graphs show the distributions of the elements of the studied array using bars that specify the number of occurrences. On every graph There's a *hold on* of the continuous normal distribution the dataset should be part of.



Picture 8: map 5 of my implementation



Picture 9: map 5 of example implementation

The results of the tests are the following:

Map	H value	P value	DoF
1	0	$1.6938 \cdot 10^{-6}$	9
2	0	$1.9517 \cdot 10^{-4}$	9
3	0	0.0011	9
4	0	$2.1073 \cdot 10^{-4}$	9
5	0	$1.3862 \cdot 10^{-4}$	9

Table 3

Where:

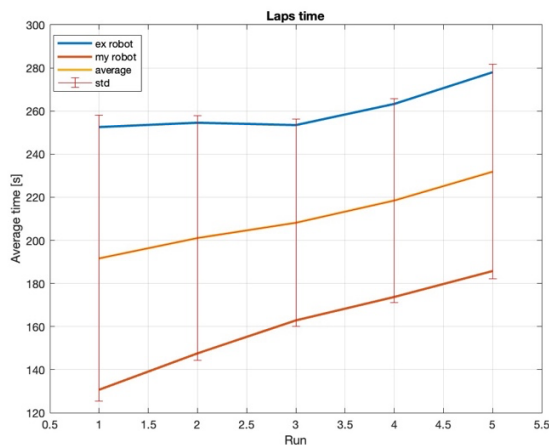
- The **H value** represents the *acceptance* or *rejection* of the **null hypothesis**. In the case of these tests, the null hypothesis is confirmed if the statistics of our experimental group are either higher than the statistics of the control group.
- The **P-value** is instead the is the probability of obtaining test results at least as extreme as the result observed under the assumption that the null hypothesis is correct.
- The **DoF** (Degrees of Freedom) are the amount of information your data provide that you can "spend" to estimate the values of unknown population parameters and calculate the variability of these estimates. This value is determined by the number of observations in your sample. Increasing your sample size provides more information about the population, and thus increases the degrees of freedom in your data. In our case the amount of the degrees of freedom is 20 (*laps/observations*) – 2.

Laps time:

The evaluation of this statistics is based on a comparison of the time it takes for each of the two robots to complete a full lap of the map. For the first dataset I plotted, using the function *plot.m*, the average time my code and the example code took to complete a lap for each of the maps.

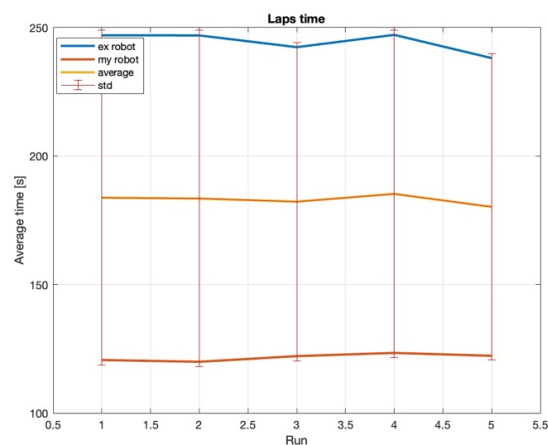
The graph compares how the lap time grows at the rise of the number of tokens for the two codes. From this data representation is clear how faster my implementation of the code is regardless of the number of tokens inside the map.

Set 1:



Picture 10: first set timing graphs

Set 2:



Picture 11: map 5 of example implementation

The same statistics is represented in the same way for the second dataset. This graph shows the same differences in time takes by the two robot implementations. To confirm this difference I used the *ttest2.m* to compute a one tailed **two sample T-test** between the time average of each map. This kind of test is meant to compare two independent groups and check whether their means are different. “Independent” implies that the two samples must have come from two completely different populations.

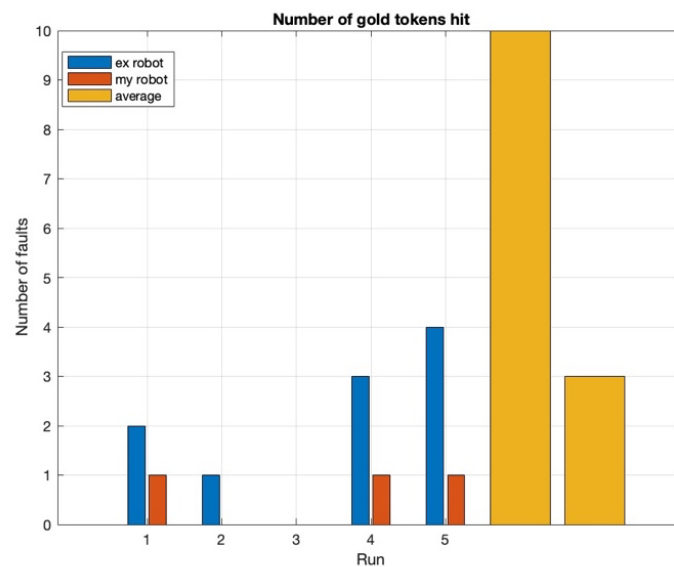
The test retrieved the following result:

H value	P value	DoF
1	$4.8952 \cdot 10^{-7}$	8

Table 4

Errors:

While running the simulation I didn’t encounter that many errors to make a clear statistic out of it. The only data I retrieved for each run is how many times the robot got stuck while running around the arena, how many times it touched the walls, and how many times it changed direction. Each of these errors are represented in the following **bar plot**.



Picture 12: errors for the first set of runs.