Research Track II: Statistical Analysis

Student Claudio Tomaiuolo (5630055)

Introduction

This report is a performance analysis of the algorithm developed as the first assignment during the Research Track I course. The analysis involves the comparison of two algorithms implemented by myself and another student (Nicholas Attolino) that have the same objective: to make a robot move in a virtual environment to pick up silver tokens and place them next to gold tokens to form pairs. To this end, data was collected and statistical analyses were made to check which implementation performed better, in execution speed and also regarding the rate of success or failure in achieving the goal.

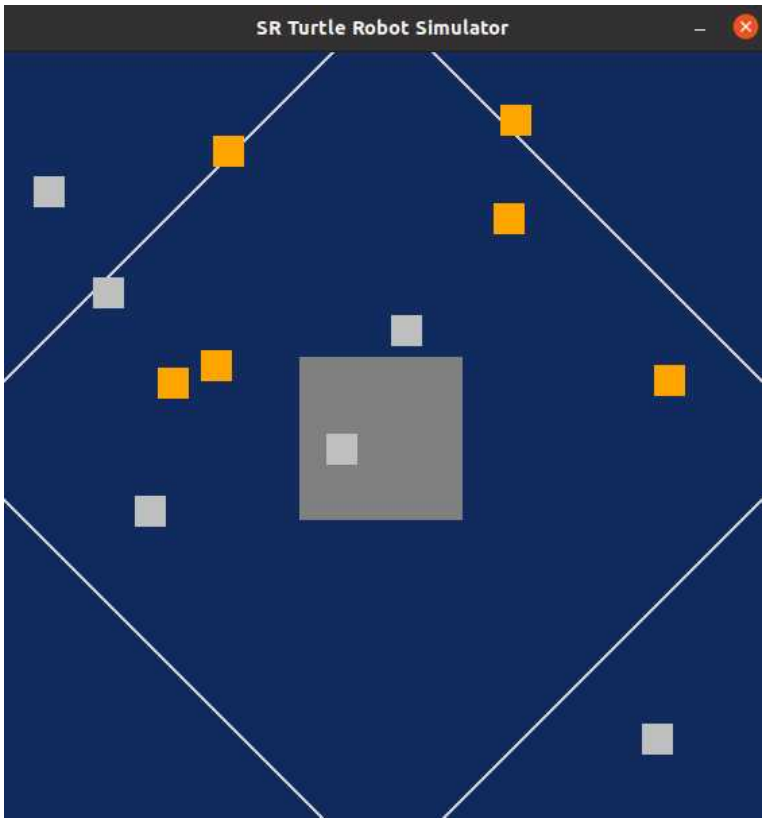

Figure 1: An example of a randomly generated environment

Environment

Because in the first algorithm performed in the Research Track 1 assignment, the environment was states and the blocks were always spawned in the same position, to perform this analysis, lines of code within the file `robot-sim/sr/robot/arenas/two_colours_assignment_arena.py` were modified to make the blocks appear randomly.

First, add:

```
import random
```

Changing the value of this variable will vary the number of blocks produced. The value set for the analysis is 6, it means that 6 silver and 6 gold blocks will be spawned.

```
TOKENS_PER_CIRCLE = 6
```

Then, comment the same line as here and add the one immediately below.

```
def place_token_circle(radius, number_offset=0, angle_offset=0.25 * pi,
                               rotate_silvers=0.0):
            for i in range(TOKENS_PER_CIRCLE):
                if radius == INNER_CIRCLE_RADIUS:
                    token_type = SilverToken
                    rotation_amount = 0
                else:
                    token_type = GoldToken
                    rotation_amount = 0
                token = token_type(self, number_offset + i)
                angle = angle_offset + (2 * pi / TOKENS_PER_CIRCLE) * i
                # token.location = (cos(angle) * radius, sin(angle) * radius)
                token.location = (random.uniform(-2.5,2.5), random.uniform(-2.5,2.5))
                token.heading = rotation_amount
                self.objects.append(token)
```

Hypothesis

Thesis: My algorithm reaches the goal faster than Nicholas' one.

Hypothesis:

H_0: $\alpha_m = \alpha_g$
H_a: $\alpha_m > \alpha_g$

where:

- H_0 is the Null hypothesis, meaning that there is no relevant distinction between the two algorithms. The averages of the time required for completion are similar.
- H_a is the Alternative hypothesis, stating that there is a significant difference between the two implementations. One algorithm is faster than the other in reaching the goal.
- α_m is the average time taken to reach the goal by my algorithm.
- α_g is the average time taken to reach the goal by Nicholas' algorithm.

Paired T-Test

Since the analysis deals with the comparison of two different approaches applied to the same scenario, the method I used is the Paired T-Test.

A Paired T-Test is used to compare two population means where you have two samples in which observations in one sample can be paired with observations in the other sample. To test the null hypothesis that the true mean difference is zero, the procedure is as follows:

1. Calculate the difference ($d_i = y_i - x_i$) between the two observations on each pair, making sure you distinguish between positive and negative differences.
2. Calculate the mean difference, $\bar{d}$.
3. Calculate the standard deviation of the differences, s_d, and use this to calculate the standard error of the mean difference, $SE(\bar{d}) = \frac{s_d}{\sqrt{n}}$
4. Calculate the t-statistic, which is given by $T = \frac{\bar{d}}{SE(\bar{d})}$. Under the null hypothesis, this statistic follows a t-distribution with n-1 degrees of freedom.
5. Use tables of the t-distribution to compare your value for T to the t_{n-1} distribution. This will give the p-value for the paired t-test.

Data and Graphs

The code was executed 30 times each and data was collected on the execution time and also on the success or failure of the execution.

Test N.	MyCode Time	Nicholas' Time	Difference	S/F MyCode	S/F Nicholas'
1	48.5667300224	65.9046080112	-17.3378779888	Success	Success
2	49.544549942	69.5341389179	-16.3600580692	Success	Success
3	52.220938921	74.2226400375	-22.0017011165	Success	Success
4	48.8914608955	79.5315189362	-30.6400580407	Success	Success
5	96.0560538769		96.0560538769	Success	Failure
6		112.98900485	-112.98900485	Failure	Success
7	53.5389080048	90.4100699425	-36.8711619377	Success	Success
8	53.146930933	80.6049060822	-27.4579751492	Success	Success
9	43.9457170963	48.3839590549	-4.4382419586	Success	Success
10	46.7199590206	89.23508811	-42.5151290894	Success	Success
11	52.3510990143	72.4539389619	-20.1028399476	Success	Success
12	46.0963249207	59.0012312034	-12.9049062827	Success	Success
13	54.0752658844	63.3274832777	-9.2522173933	Success	Success
14	60.9039108753	83.3247327437	-22.4208218684	Success	Success
15	58.5679700375	59.918593273	-1.3506232355	Success	Success
16	62.009499073	52.7438574374	9.2656416356	Success	Success
17	44.222648859	72.3247328738	-28.1020840148	Success	Success
18	46.9353940487	68.3728321138	-21.4374380651	Success	Success
19	48.9896190166	77.3274627346	-28.337843718	Success	Success
20	55.6576600075	97.732437423	-42.0747774155	Success	Success
21	47.3572800159	81.4701147327	-34.1128347168	Success	Success
22	60.3566908836	57.3701293111	2.9865615725	Success	Success
23	50.3458020687	55.4217804284	-5.0759783597	Success	Success
24	48.9160840511	50.1434410348	-1.2273569837	Success	Success
25	50.217936039	91.7238436423	-41.5059076033	Success	Success
26	50.2937259674	65.7234611146	-15.4297351472	Success	Success
27	44.4534449577	80.3274327483	-35.8739877906	Success	Success
28	54.4650080204	61.4237542239	-6.9587462035	Success	Success
29	47.6032290459	72.8439734857	-25.2407444398	Success	Success
30	45.9435019493	66.4357551043	-20.492253155	Success	Success

Figure 2: Timing and success of executions

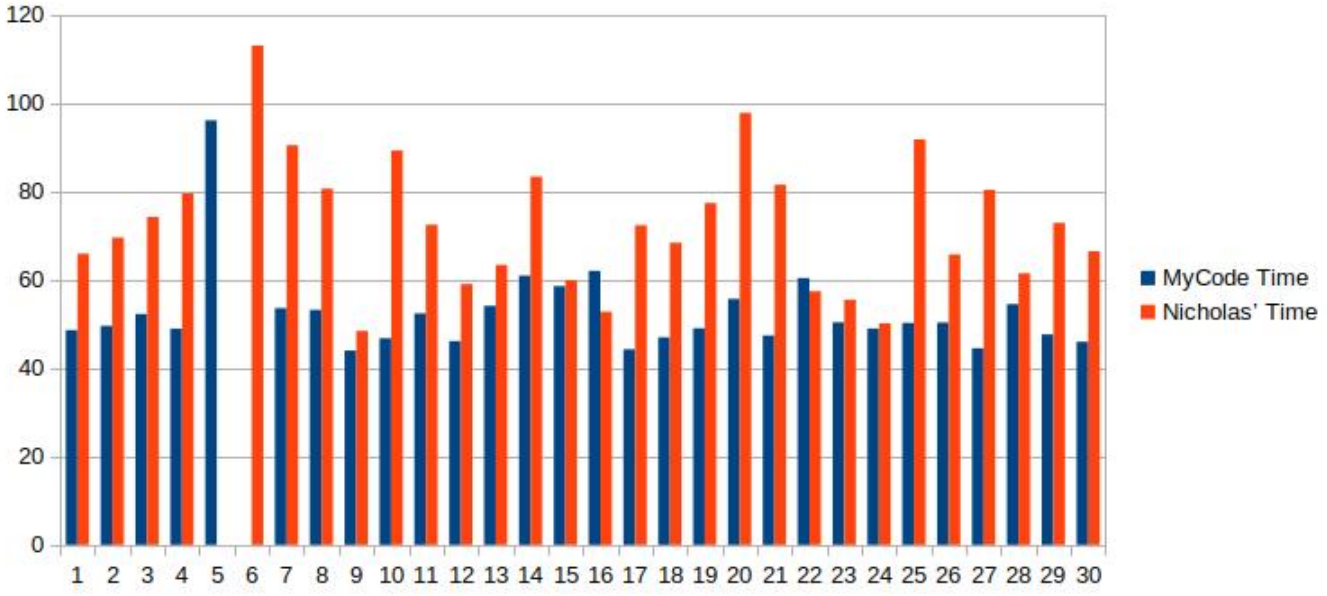

Figure 3: Vertical column chart comparing algorithm execution times

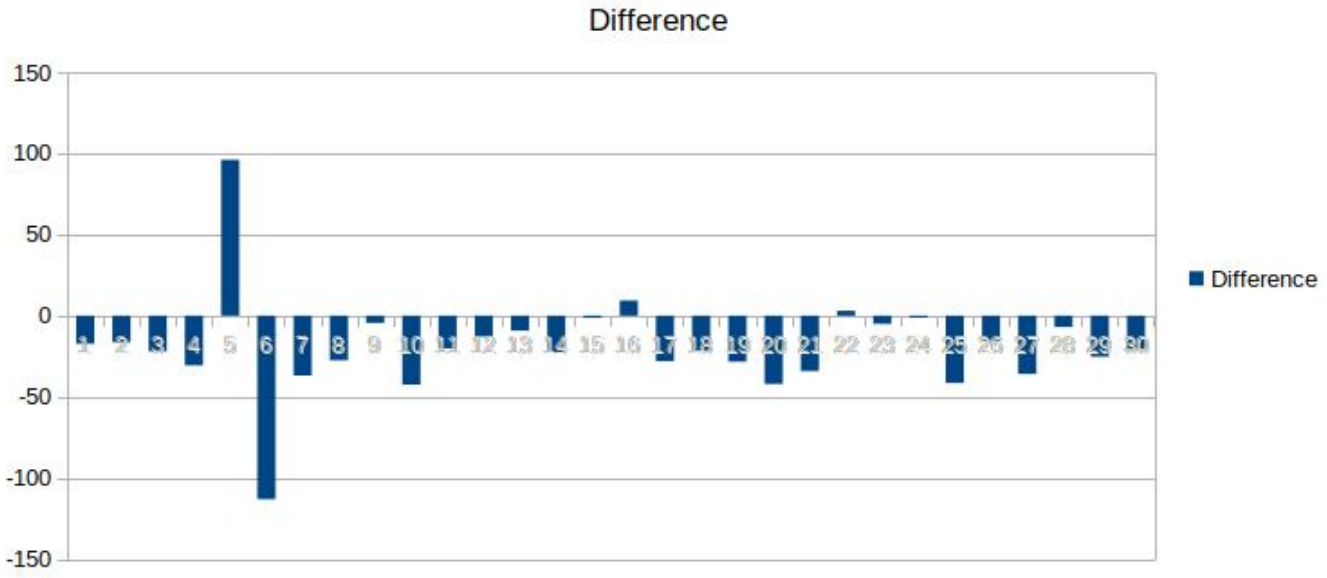

Figure 4: Vertical column chart representing the difference between the algorithm execution times

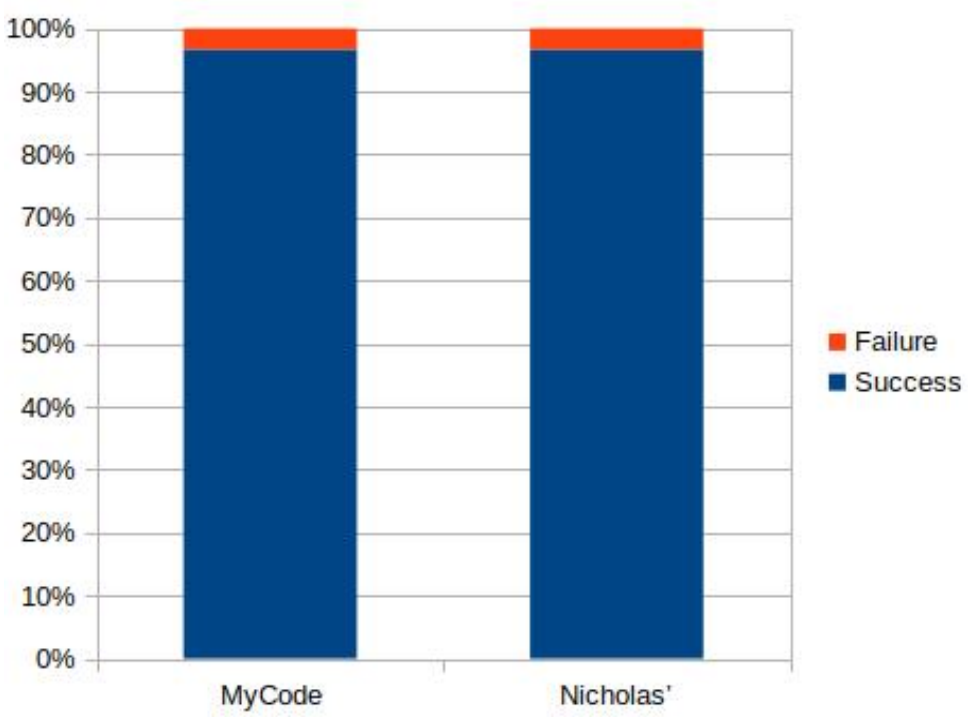

Figure 5: Stacked column chart in percentages representing execution successes and failures

Results

Once all the differences among the samples have been calculated (see Figure 2), we go on to calculate the mean of the differences:

$$\overline{d} = \frac{\sum_{i=1}^{n} x_i}{n} = -19.26$$

Then, the standard deviation of the differences:

$$s_d = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n} = 30.82$$

Therefore, the standard error of the mean difference:

$$SE(\bar{d}) = \frac{s_d}{\sqrt{n}} = 5.63$$

Finally, the t-statistic:

$$t = \frac{\overline{d}}{SE(\overline{d})} = -3.42$$

Conclusions

As can be seen from the graphs and results, my implementation of the algorithm is clearly faster than the implementation done by my colleague. There was a failure in a sample from my population, as well as a failure in samples from the counterpart population. This is because in the assignment of the previous course, the cubes were arranged in an orderly and static way and we were not required to implement a function to try to avoid blockages during the movement of the robot. In the scenarios of this test, the blocks that were made to appear randomly sometimes represented an obstacle in the robot's movement and it could happen that a group of blocks would pile up in a corner and prevent the robot from picking up the designated block.

The t-value obtained must be compared with the values available in the t-table. The row with Angle of Freedom given by $DoF = N - 1 = 29$ (the sample population is $N = 30$) is taken and the value closest to that obtained from the calculations is looked for. In this case $t = -3.42$. Since the thesis opines that my algorithm has shorter run times than my colleague's, the one-tail test is used. It thus turns out that we can reject the Null Hypothesis, with a confidence of 99.9%.

t Table

cum. prob	$t_{.50}$	$t_{.75}$	$t_{.80}$	$t_{.85}$	$t_{.90}$	$t_{.95}$	$t_{.975}$	$t_{.99}$	$t_{.995}$	$t_{.999}$	$t_{.9995}$
one-tail	**0.50**	**0.25**	**0.20**	**0.15**	**0.10**	**0.05**	**0.025**	**0.01**	**0.005**	**0.001**	**0.0005**
two-tails	**1.00**	**0.50**	**0.40**	**0.30**	**0.20**	**0.10**	**0.05**	**0.02**	**0.01**	**0.002**	**0.001**
df											
1	0.000	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	318.31	636.62
2	0.000	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	22.327	31.599
3	0.000	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	10.215	12.924
4	0.000	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	7.173	8.610
5	0.000	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	5.893	6.869
6	0.000	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	5.208	5.959
7	0.000	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.785	5.408
8	0.000	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	4.501	5.041
9	0.000	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	4.297	4.781
10	0.000	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	4.144	4.587
11	0.000	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	4.025	4.437
12	0.000	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.930	4.318
13	0.000	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.852	4.221
14	0.000	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.787	4.140
15	0.000	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.733	4.073
16	0.000	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.686	4.015
17	0.000	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.646	3.965
18	0.000	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.610	3.922
19	0.000	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.579	3.883
20	0.000	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.552	3.850
21	0.000	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.527	3.819
22	0.000	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.505	3.792
23	0.000	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.485	3.768
24	0.000	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.467	3.745
25	0.000	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.450	3.725
26	0.000	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.435	3.707
27	0.000	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.421	3.690
28	0.000	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.408	3.674
29	0.000	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.396	3.659
30	0.000	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.385	3.646
40	0.000	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	3.307	3.551
60	0.000	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	3.232	3.460
80	0.000	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	3.195	3.416
100	0.000	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	3.174	3.390
1000	0.000	0.675	0.842	1.037	1.282	1.646	1.962	2.330	2.581	3.098	3.300
z	0.000	0.674	0.842	1.036	1.282	1.645	1.960	2.326	2.576	3.090	3.291
	0%	50%	60%	70%	80%	90%	95%	98%	99%	99.8%	99.9%
	Confidence Level										

Figure 6: T-Table