ECSE 211 Lab 5

Search & Localize

Oussama Mehdi 260789308
Glen Xu 260767363
Shaluo Wu 260713923
Mael Mugerwa 260805446
Chen Liu 260776963
Sameen Mahtab 260737048

February 2019

Design Evaluation

In this lab, we are aiming to design a robot that localizes itself to the lower-left corner of the prescribed area, search for the can with a specific colour without touching any of the cans with the ultrasonic sensor and the light sensor, then travel to the upper-right corner.

HARDWARE

OVERALL
The robot contains two motors on both sides, two wheels at the front, a light sensor at the back, a small metal ball in front of the light sensor supporting the robot, an ultrasonic sensor at the front detecting the distance to the object, an arm stretching from the front with another light sensor detecting the colour of the can connected to the end of the arm, and a brick with firmware where we can control the robot and read data from. Note that in order to make the cable connection from the sensors and motors to the ports simpler, we have rotated the brick frontside back.

MOTORS
The EV3 and NXT motors seemed very similar and interchangeable. After reading some introductions of these two kinds of motors, we learned that EV3 motors provide higher speed[1], but lower efficiency[2], while NXT motors provide higher resolution with the help of the rotation encoder[2]. In this lab, there were not any requirements on speed, but the robots were expected to run very precisely and accurately. Based on these information, we once decided to use NXT motors. However, when we started to build the robot, we found out that we required a beam to stabilized the back of the robot, but only EV3 motors had three mounting holes in the back. Also, EV3 motors have a cross hole on each side, making the structure even more stable. We finally chose EV3 motors because they make the hardware design more versatile.

INHERITANCE AND IMPROVEMENT

We first approached the design based on the robot we used in lab 4, and added the arm with the light sensor to the front of the robot. However, we found that the ultrasonic sensor was constructed too forward, making the arm reaching out too far way from the centre of the robot in order not to touch the can. Such design was both inaesthetic and oversized, making the robot not able to avoid all the cans. We then redesigned the robot by putting the ultrasonic sensor into the centre of the robot, right underneath the brick. In the way, the arm did not have to reach out for an exaggerate distance. The robot became both attractive and pragmatic. We finally rotated the brick frontside back to simplify the cable connections.

Our final hardware design looked like the following figures.

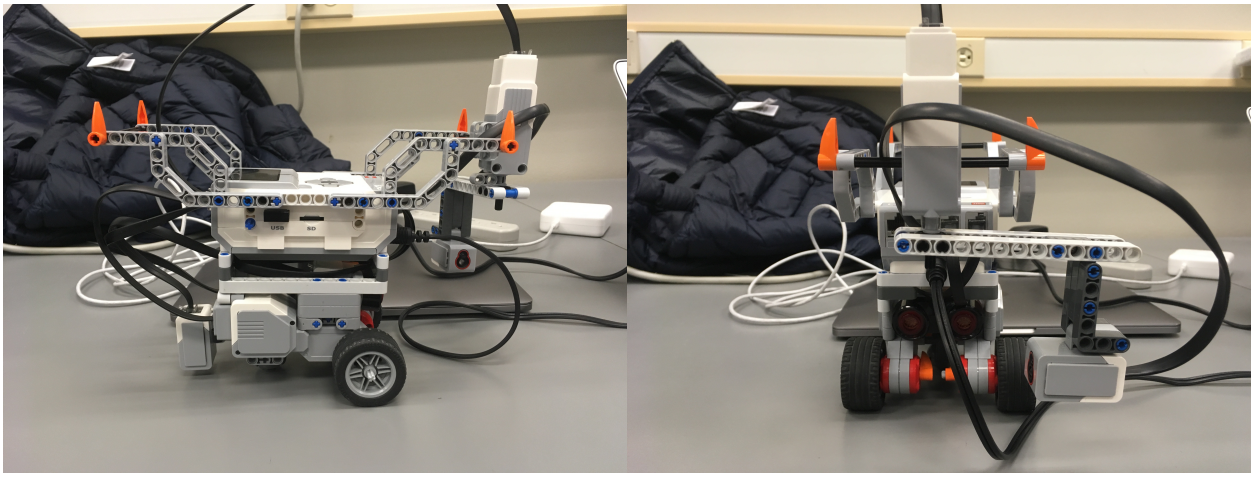


Figure 1 Hardware

SOFTWARE

OVERALL

We applied the classes Odometer, OdometerData and OdometerExceptions from Lab 2, to update the x and y values, and θ value. The displacement of both wheels and the direction the robot was heading towards determined the x and y values. And the difference of the displacement of both wheels determined θ value. To achieve the purpose of navigation, we implemented a method to navigate the robot to the next waypoint, by calculating the change of x, y and θ values. The next theoretical θ value was calculated by the following formula $\theta = \tan^{-1}\left(\frac{\Delta x}{\Delta y}\right)$. The x, y and θ values are calculated by subtracting the current value from the next theoretical value.

LOCALIZE with ULTRASONIC SENSOR

The ultrasonic sensor was used to rotate the robot to 0°. Two methods were implemented, falling and rising edge. The robot would start from facing opposite to the walls. It would continue to turn left until the ultrasonic sensor detected a distance at the threshold. The robot would record

the angle it rotated from the odometer and then start to turn right, until it found another falling edge, and accomplish the same procedure. The rising edge method was similar to the falling edge one, except for the robot started from the position of facing toward the wall. After two angle values were recorded, the final angle for the robot to turn could be calculated.

LOCALIZE with LIGHT SENSOR

The light sensor was used to make the robot travel to the origin, and then rotate the robot to 0°. The robot would first travel to the estimated position, where the grid lines intersect. Then it would do an entire rotation, with the light sensor detecting the positions of the four grid lines. The four angle it traveled to touch the grid lines were recorded into the theta array, as shown in figure 2.

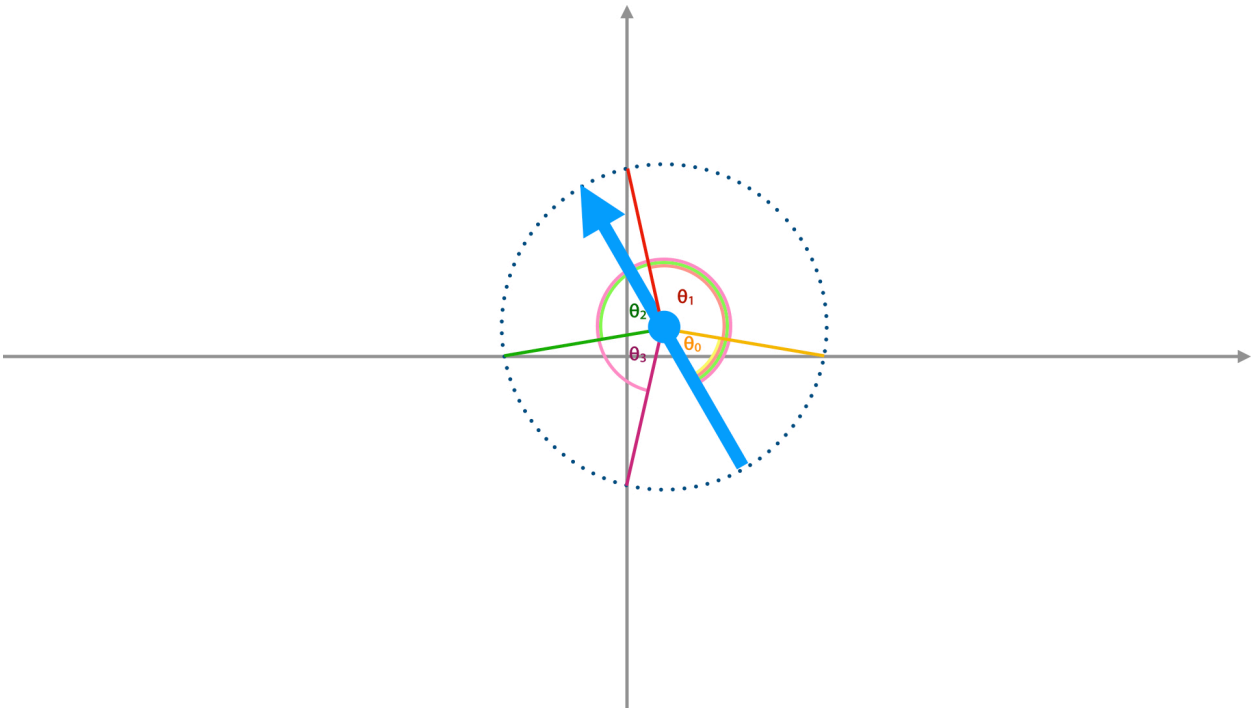


Figure 3 Light Sensor

Then the angles recorded can be used to calculate the distance and direction the robot needed to travel, and the final angle it had to rotate.

The y offset will be used as an example to illustrate how the algorithm would work. There exists a triangle consisting of the y offset, the x axis and the distance from the light sensor to the centre of the robot, which is also the radius of the rotation. The distance, and the angle between the y offset and the distance are known, so the y offset can be calculated with cosine trigonometric function. The equation to calculate the y offset is $offset_y = \cos\left(\frac{1}{2}(\theta_2 - \theta_0)\right) \times$ distance. After both x and y offsets were calculated, the robot would travel to the position using the travelTo method.

The angle it had to turn will be calculated by subtracting θ_0 from the angle in the triangle. The equation is $\theta_{final} = \frac{1}{2}(\theta_2 - \theta_0) - \theta_0$. The robot would rotate accordingly using the turnTo method. The calculation processes are illustrated in figure 3.

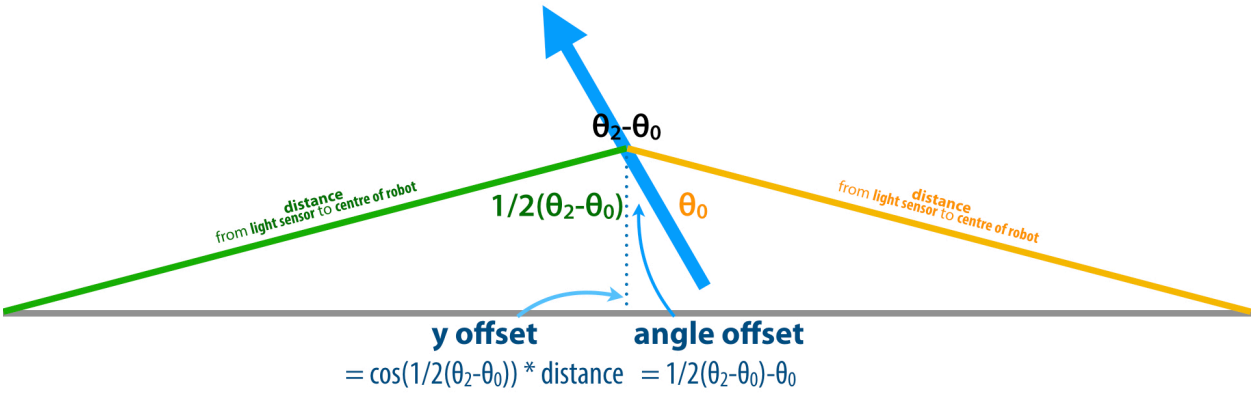


Figure 4 Light Sensor

COLOUR DETECTION

In the colorScanner method, we started with measuring the mean RGB values of different cans, named as YRmi, YGmi, YBmi and such. The sensor then sends back 15 sets of RGB values. The averages of red, green and blue values are calculated respectively. We take the difference between the average of the measured value and the standard mean values. The program will then compare and pick out whichever color can that gives the smallest Euclidean distance value. This is the method introduced in the lab instructions. Through testing, this method is proven to be quite reliable, even if there is a high level of uncertainty due to the absence of filters.

PARAMETERS

We spent a lot of time testing the appropriate parameters.

Speeds of Wheels

The speeds of both wheels should have been the same theoretically. However, the motors were not in very good quality, and the motor on the left actually rotated slower than the one on the right. As a result, the robot would have turn to the left, if we set the speeds to be the same. So after testing, we added the value of speed of the left wheel by 3 to narrow down the negative effect. Actually this did not fully solve the problem, but 3 is the nearest integer. The type of the

variable was set to be an *int*, and that was the only type LeJOS motor could take. We decided to adjust it more precisely with the track.

Track

The track was the distance between two wheels. This value was used to calculate the angle the robot turned, so it should be as accurate as possible. The actual value we measured was 9.35, but the change of the speeds of wheels required a larger turn to compensate the effect. In order to make sure the robot travelled in straight lines we increased the track by 0.61. This was not the actually value of the distance, but the value that worked the best.

Overall software design flow charts are shown in figure 5.

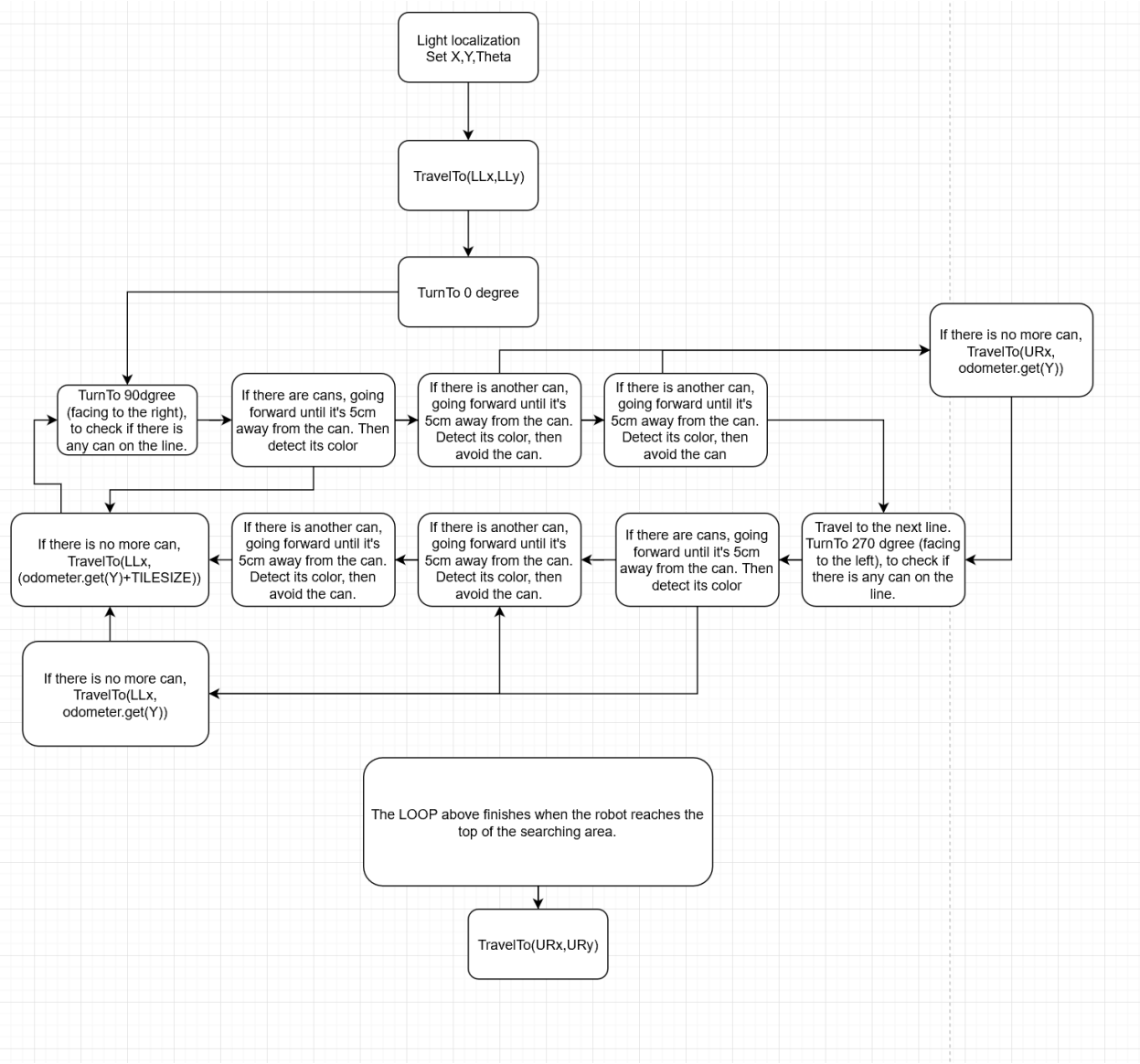


Figure 5 Software Design Flow Chart

Test Data

MODEL ACQUISITION

CONSTANT VARIABLES

The hardware design of the robot, the motor speeds, the track, the board where is robot is running on, the distance from the colour sensor to the can, the illumination conditions, and other parameters.

RAW DATA

The data are shown in table 1 to 4. We could not determine the uncertainty of the reported values, because the values shown on the screen had inconsistent significant figures. Fortunately, there were no analysis on the uncertainty required in the report, so we decided to just keep the data the way they were shown.

Table 1 Raw Data of Red Can

Trail	R	G	B
1	1.4974022	0.1387942	0.34934994
2	0.7201213	0.1223847	0.218847
3	1.4616269	0.6624958	0.0334971
4	0.6413360	0.641348	0.6498728
5	1.3616524	0.862396	0.1343459
6	0.2062832	0.039489	0.1328432
7	1.335039	0.8334982	0.1290564
8	0.3644924	0.0349843	0.6697718
9	0.8163913	0.6119391	0.343928
10	0.034071830	0.1340203	0.86234877

Table 2 Raw Data of Green Can

Trail	R	G	B
1	0.1834891	0.0823842	0.2338274
2	0.2039459	0.323472	0.2947529
3	0.664327	1.9345789	0.122387
4	0.2434854	0.23485889	0.0234981
5	0.923495	0.4348589	1.0342349
6	0.2983478	1.82384755	0.1838478
7	0.2984375	1.334887	0.20239891
8	0.122398	0.43498349	0.662221
9	0.0293842	0.28837	0.2423981
10	1.03407183	1.432849	0.9234919

Table 3 Raw Data of Blue Can

Trail	R	G	B
1	0.34992374	0.2062832	0.1923847
2	0.21239192	0.335039	0.7234891
3	0.0323899	0.3644924	0.7948599
4	0.6423998	0.8163913	0.623920
5	0.1323981	0.034071830	1.9623991
6	0.13230919	0.5974022	0.70623841
7	0.1261398	0.7201213	1.3319238
8	0.6619238	1.9616269	0.3345838
9	0.34891283	0.6413360	0.2843890
10	0.8619382	0.6616524	0.043998

Table 4 Raw Data of Yellow Can

Trail	R	G	B
1	0.1342892	0.13823948	0.18323489
2	0.2237429	1.5223478	0.2023811
3	1.52347882	0.6623874	0.6623747
4	1.3384792	0.1432499	0.24223949
5	0.232384	0.86238481	0.523846
6	0.2349202	1.132391	0.23237471
7	1.52374922	0.12238483	0.2923734
8	1.3399222	0.6239411	0.122342
9	0.223992991	0.24239915	0.0223774
10	0.5234923	0.26234991	1.0344265

COLOUR & POSITION IDENTIFICATION

CONSTANT VARIABLES

The hardware design of the robot, the motor speeds, the track, the board where is robot is running on, the distance from the colour sensor to the can, the illumination conditions, and other parameters.

RAW DATA

The data are shown in table 5 to 8. We could not determine the uncertainty of the reported values, because the values shown on the screen had inconsistent significant figures. Fortunately, there were no analysis on the uncertainty required in the report, so we decided to just keep the data the way they were shown.

Table 5 Raw Data, Red Can as TR

Can	RGB			Distance			
				Estimated		Actual	
	R	G	B	TPEx	TPEy	TPRx	TPRy
Red	0.8423891	0.4023891	0.3423911	4	4	4	4
Green	0.3813151	0.8176567	0.408172	4	5	4	5
Blue	0.341384	0.6235123	0.69623489	5	6	5	6
Yellow	0.7278991	0.581621	0.346789	5	5	5	5

Table 6 Raw Data, Green Can as TR

Can	RGB			Distance			
				Estimated		Actual	
	R	G	B	TPEx	TPEy	TPRx	TPRy
Red	0.8368126	0.4012188	0.3312317	6	6	6	6
Green	0.391381	0.8289111	0.391911	4	5	4	5
Blue	0.3423114	0.618492	0.68237	5	6	5	6
Yellow	0.8367515	0.4387661	0.3317758	5	4	5	4

Table 7 Raw Data, Blue Can as TR

Can	RGB			Distance			
				Estimated		Actual	
	R	G	B	TPEx	TPEy	TPRx	TPRy
Red	0.8581244	0.3876511	0.3771521	4	6	4	6
Green	0.3761289	0.8197655	0.3887811	5	5	5	5
Blue	0.342119	0.615171	0.679823	3	3	3	3
Yellow	0.8128717	0.4392381	0.377192	4	4	4	4

Table 8 Raw Data, Yellow Can as TR

Can	RGB			Distance			
				Estimated		Actual	
	R	G	B	TPEx	TPEy	TPRx	TPRy
Red	0.912731	0.2001118	0.338118	4	4	4	4
Green	0.3618919	0.8209761	0.4187611	6	7	6	7
Blue	0.351778	0.608754	0.729293	3	4	3	4
Yellow	0.732894	0.576207	0.357956	5	6	5	6

Test Analysis

The processed data are shown in the following table, with sample working for one of the trails.

MODEL ACQUISITION

MEAN and STANDARD DEVIATION

Table 9 Processed Data, Mean

Can	R	G	B
Red	0.843841653	0.40813496	0.352386091
Green	0.400138173	0.832508993	0.392305801
Blue	0.350072728	0.633841653	0.1923847
Yellow	0.7298451031	0.571207538	0.351797019

Table 10 Processed Data, Standard Deviation

Can	R	G	B
Red	0.544391122072522	0.341953314980436	0.281763170049366
Green	0.348144402571896	0.715023921671695	0.35159711639911
Blue	0.28078093074212	0.528131061882098	0.577603236516247
Yellow	0.615041107268602	0.482013412483382	0.304365142912508

SAMPLE WORKING

The R values of red cans would be used to show the working of calculating the mean and standard deviation.

Calculating the MEAN

$$\overline{R} = \frac{\sum_{i=1}^{n} \varepsilon_i}{n}$$

$$\overline{R} = \frac{1.4974002 + 0.7201213 + \ldots + 0.034071830}{10}$$

$$\overline{R} = 0.843841563$$

Calculating the STANDARD DEVIATION

$$\sigma_R = \sqrt{\frac{\sum_{i=1}^{n}\left(R_i - \overline{R}\right)^2}{n-1}}$$

$$\sigma_R = \sqrt{\frac{\left(1.4974022 - 0.843841653\right)^2 + \left(0.7201213 - 0.843841653\right)^2 + \ldots + \left(0.034071830 - 0.943841653\right)^2}{10-1}}$$

$$\sigma_R = 0.544391122072522$$

GAUSSIAN DISTRIBUTION

The graphs are shown in the appendix.

COLOUR & POSITION IDENTIFICATION

EUCLIDEAN DISTANCE of RGB

Table 11 Processed Data, Euclidean Distance

Can	Red as TR	Green as TR	Blue as TR	Yellow as TR
Red	0.011620011293	0.023339861049	0.035170166110	0.219597257832
Green	0.028751260063	0.009475694213	0.027409219634	0.047913128311
Blue	0.013965398987	0.024501082393	0.028489968972	0.038743253103
Yellow	0.011717822886	0.171378415817	0.157969153378	0.008498437478

CAN DISTANCE in ORDER & CLASSIFICATION Returned

Red can: 0.011620011293 RED < 0.023339861049 RED < 0.035170166110 RED < 0.219597257832 RED

Green can: 0.009475694213 GREEN < 0.027409219634 GREEN < 0.028751260063 GREEN < 0.047913128311 GREEN

Blue can: 0.013965398987 BLUE < 0.024501082393 BLUE < 0.028489968972 BLUE < 0.038743253103 BLUE

Yellow can: 0.008498437478 YELLOW < 0.011717822886 YELLOW < 0.157969153378 YELLOW < 0.171378415817 RED

SAMPLE WORKING

The R values of red cans would be used to show the working of calculating the euclidean distance.

Calculating the EUCLIDEAN DISTANCE

$$\varepsilon = \sqrt{\left(s_R - \mu_R\right)^2 + \left(s_G - \mu_G\right)^2 + \left(s_B - \mu_B\right)^2}$$

$$\varepsilon = \sqrt{\left(0.8423891 - 0.843841653\right)^2 + \left(0.4023891 - 0.40813496\right)^2 + \left(0.3423911 - 0.352386091\right)^2}$$

$$\varepsilon = 0.011620011293$$

Precisely describe the method you used to determine can position (TPEx, TPEy).

We used the odometer.x() and odometer.y() methods from the odometer class to get the values of the position of the robot, and then we subtract the multiples of the tile size (0, 30.48, 60.96 etc.) from the values. Whichever result has the smallest absolute value is the number of the nearest grid line. For example, if the value returned from the odometer minus 30.48 had the smallest absolute value, then the first grid line is the nearest to the robot.

EUCLIDEAN DISTANCE of POSITIONS

Fortunately, our robot performed well during the tests, and all the positions calculated by the firmware are the same as the actual values. So all the euclidean distances are 0.

Observations and Conclusions

Q1: Are rank-ordecan Euclidean distances a sufficient means of identifying can colours? Explain in detail why or why not.

Q2: Is the standard deviation a useful metric for detecting false positives? In other words, if the can colour determined using the Euclidean distance metric, d, is incorrect, can this false positive be detected by using μ±1σ or μ±2σ values instead?

Q1 & Q2: We tested our robot using both Euclidean distances and standard deviation. We started with standard deviation. We ran multiple independent trials to collect RGB colour values for each target can from which we computed the mean and their standard deviation. Using these precomputed measurements, we could figure out if a sample can's mean values where in the target range of only 1 standard deviation from the target mean. This method worked for can's of vary different colour (blue and yellow) since their mean values and standard deviation were notably different. However, for cans of similar colour values (blue and green), the sample can would be identified as both blue and green. On the other hand, the method where we took the minimum Euclidean distance gave us the correct can colour in all the trails we ran.

Q3: Under what conditions does the colour sensor work best for correctly distinguishing colours?

Q3: The colour sensor works best for correctly identifying colours when the sensor is placed at a height where when the sensor scans around the can, the amount of labels it encounters is limited.

This height is around the top of the can where you can find the most amount of target colour. Another way to reduce the impact of the problem is to change the fixed threshold value to a relative value. The light sensor would detect the ambient light condition first, and then compare the values later with the ambient value. However, this would still require the light condition of the room to be a reasonable range.

Further Improvements

Depending on how you implemented your colour classifier, can your results be improved by using one or more of the noise filtecan methods discussed in class?

Yes we could improve our results by using some of the methods discussed in class; for example using the median filter seen in class. This method consists of substituting any inconsistent value with the median value. We know that the noise of signal can affect the overall error, so if we want to minimize errors in the colour sensor, we can implement a median filter to remove noise. The basic idea behind is that the median filter runs through every signals and replace each entry with the median of neighbouring entries. Through this process, the noise can be filtered. For example, three continuous R values of RGB detected by the colour sensor are 2.1, 2.1, and 8. By using the median filter, the noise 8.0 can be filtered and replaced by 2.1.

Another method we could use is having our color sensor ignore the input when reading a value that is very close to black or white. Since we want to know if the cans are Green, blue, red or yellow, a reading that refers to some black or white is unnecessary and can only interfere with our process.

How could you improve the accuracy of your target can's position identification?

We could improve our our target can's position by modifying our hardware and having something that will bring the can closer to our robot and permit a better rotation of our colour sensor around the can. This could be done by adding two beams in a "V" shape at the front of our robot, so that when a robot reaches a can, the can slides to the front of the robot. To leave the robot nearest possible to its initial position we would make the robot move in reverse, reducing the damages done to initial form of the search region.

We can also use the colour sensor to determine the position, in additional of using odometers. The colour sensor can be use to correct the angles.

When the robot has two light sensors.

There are two light sensors: Installing the sensors right next to the two wheels, then by taking the time difference between feedbacks sent from the two sensors, one can easily calculate the offset angle. The distance between two wheels equals the speed multiplies the time difference and the angle calculating algorithm remains the same as what was described in the code.
i.e.

```
Math.toDegrees(((distanceLeft - distanceRight) / TRACK)).
```

When the robot has only one light sensor.

Ideally, if the robot goes in a straight line, then the angle is zero. The distance of which the sensor detects two black line should be exactly 30.48, aka the tile size. If the distance measured by the odometer is not equal to a tile size, then there is an angle offset. The angle can be calculated by:

$$cos^{-1}(TILESIZE \ / \ distanceMeasuredByOdometer).$$

Figure 6 can be used to illustrate the situation.

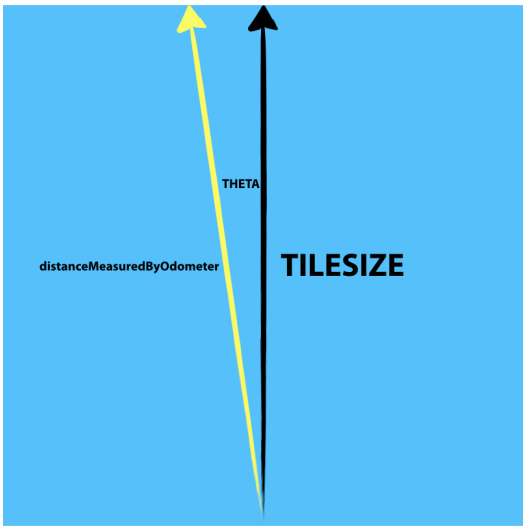


Figure 6

Bibliography

[1] Giles, M. (Feb. 24, 2017). Motors: EV3 vs NXT. Retrieved from http://www.mindsensors.com/blog/news/motors-ev3-vs-nxt.

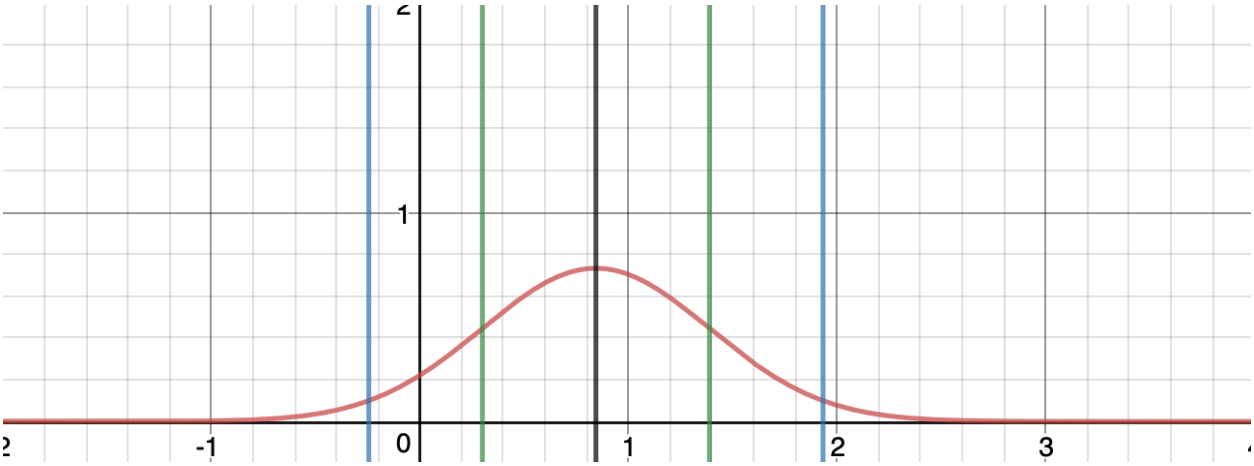
[2] Hurbain, P. (Dec. 15, 2002). LEGO® 9V Technic Motors compared characteristics. Retrieved from http://www.philohome.com/motors/motorcomp.htm.

Appendix

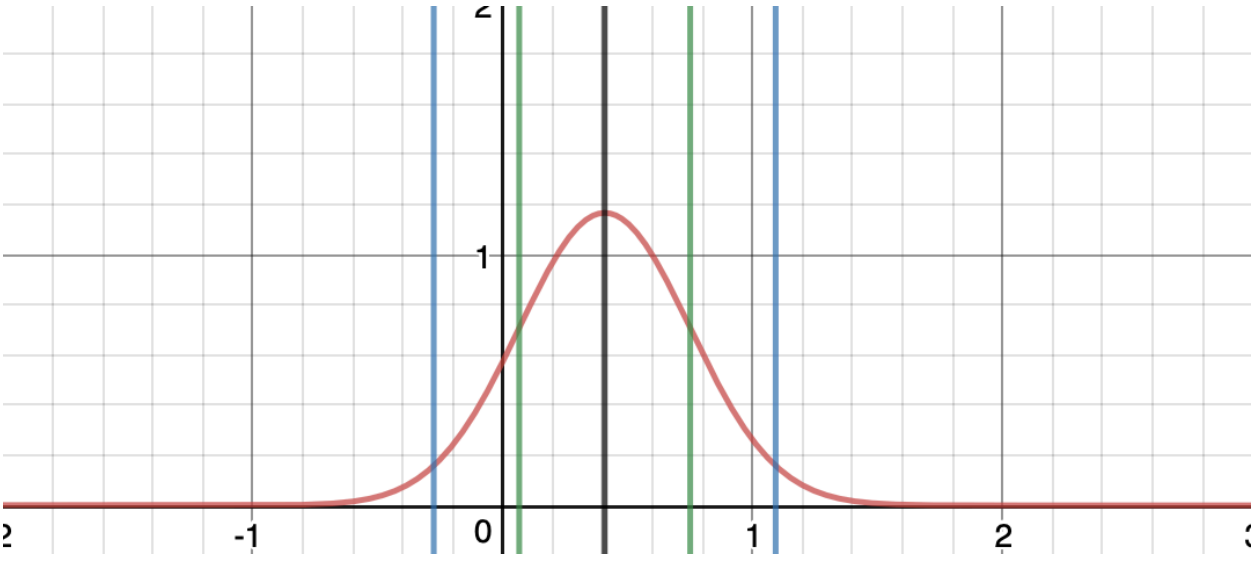
The following figures are the gaussian distribution graphs of the RGB values of all red, green, blue and yellow cans. In these graphs, the distribution curves are sketched in red, μ values are sketched in black, μ±1σ values are sketched in green, and μ±2σ values are sketched in blue.

RED CAN

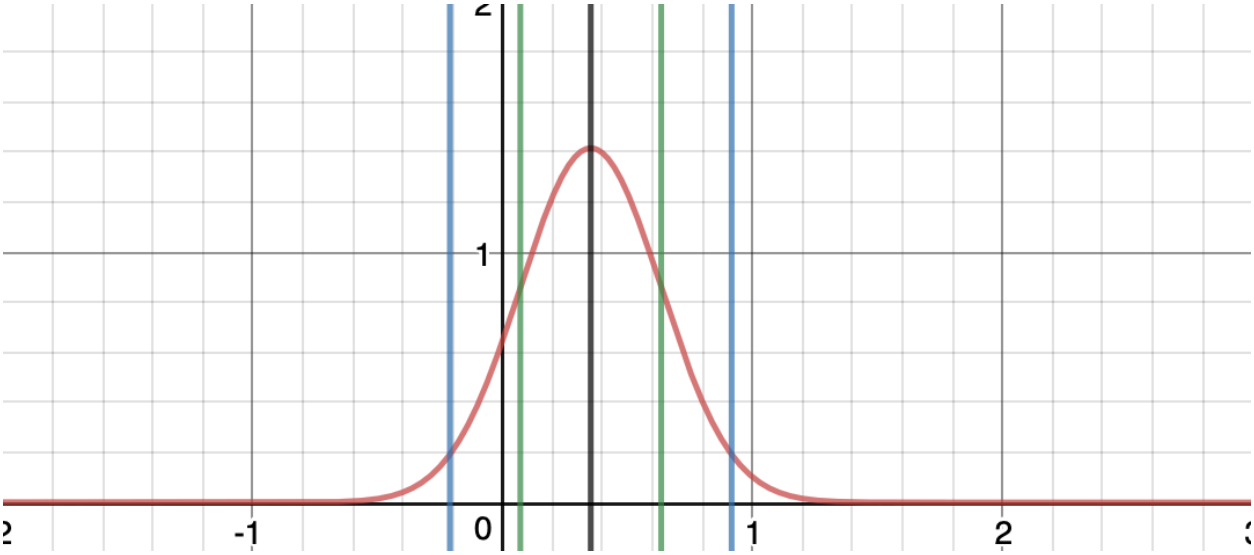
R



G

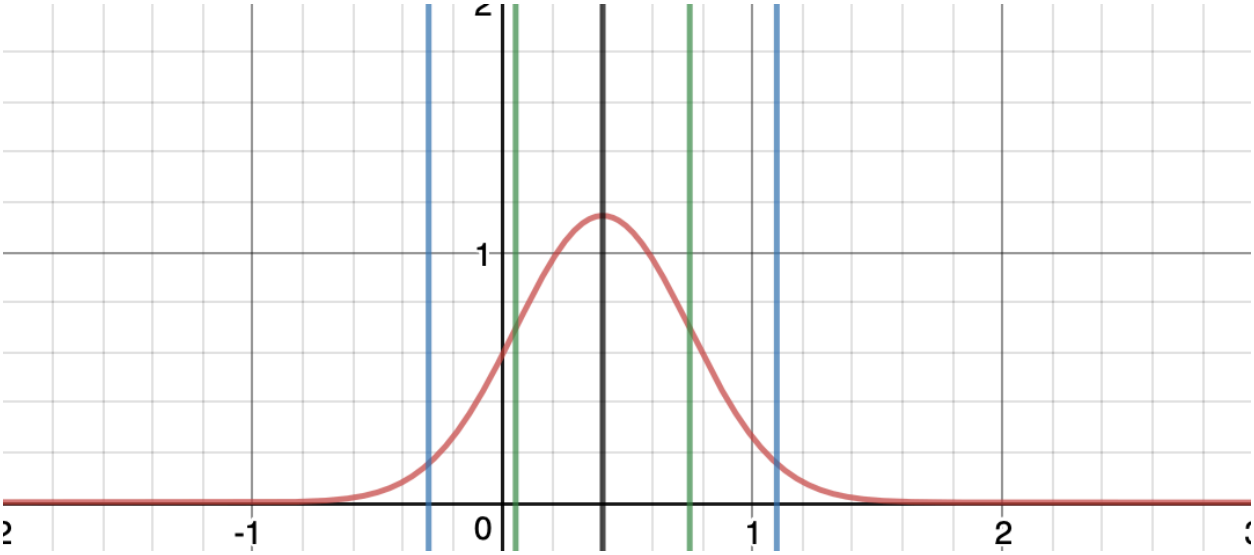


B

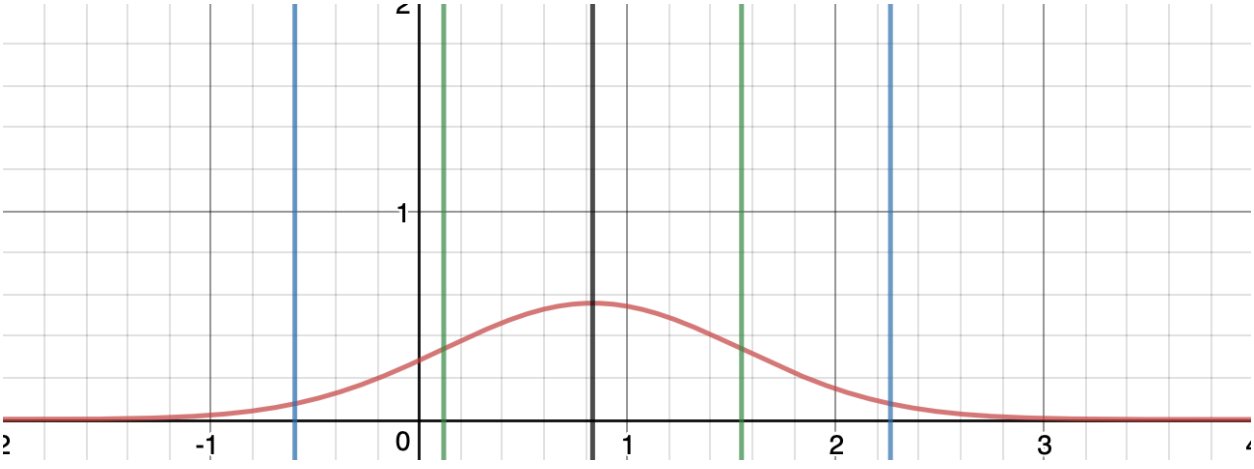


GREEN CAN

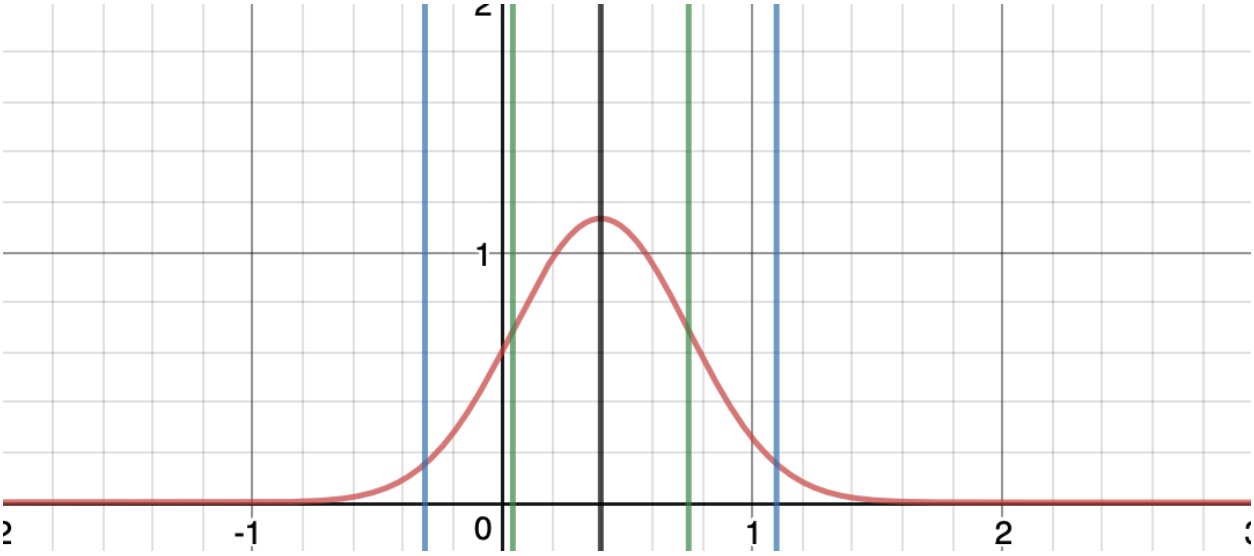
R



G

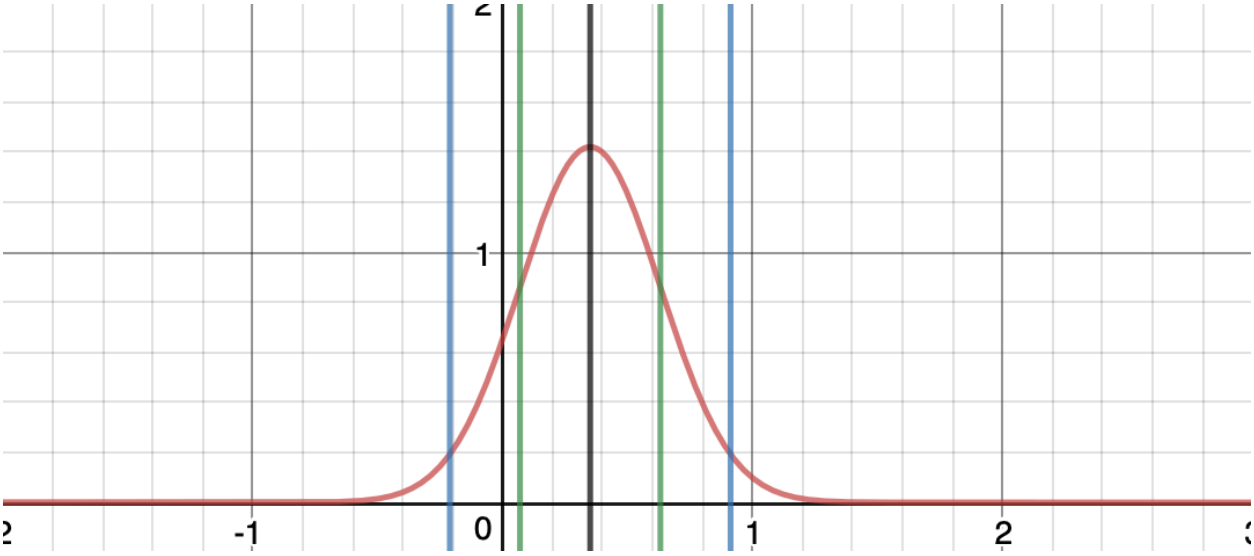


B

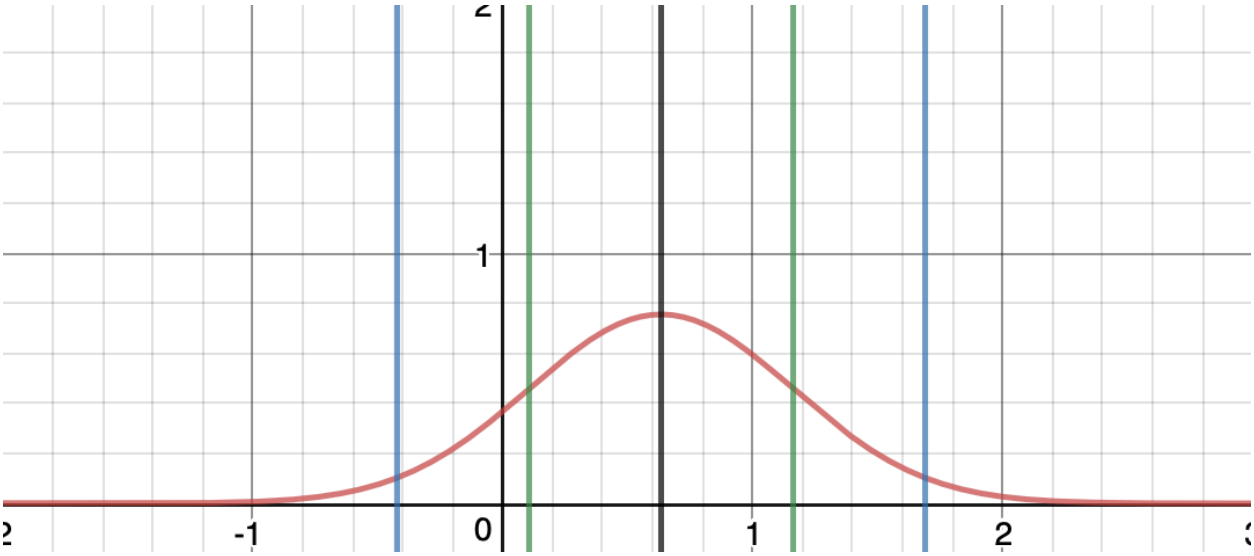


BLUE CAN

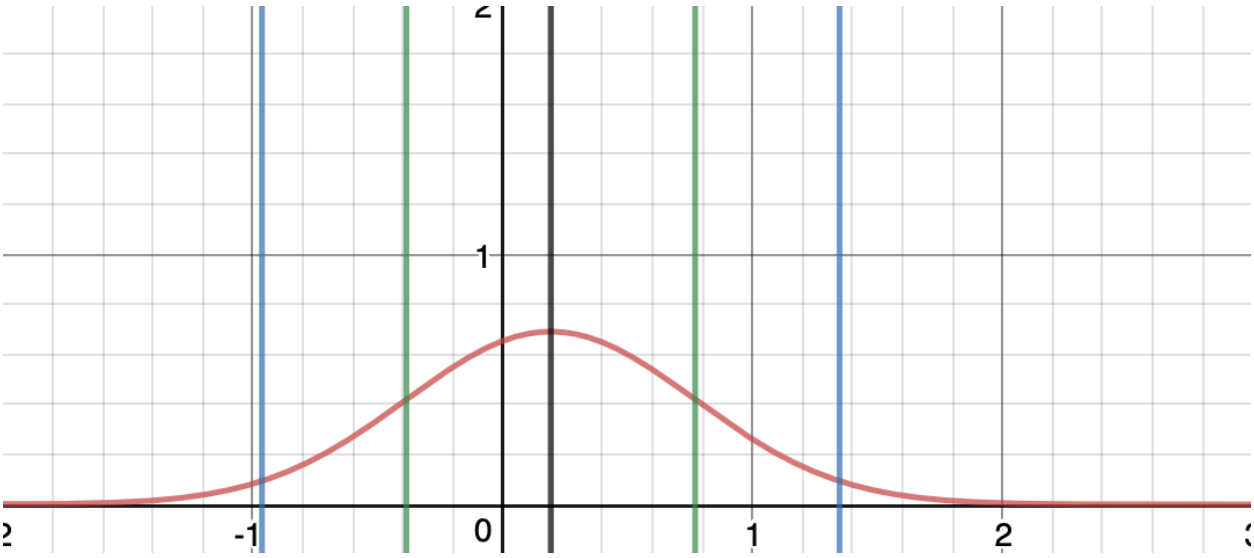
R



G

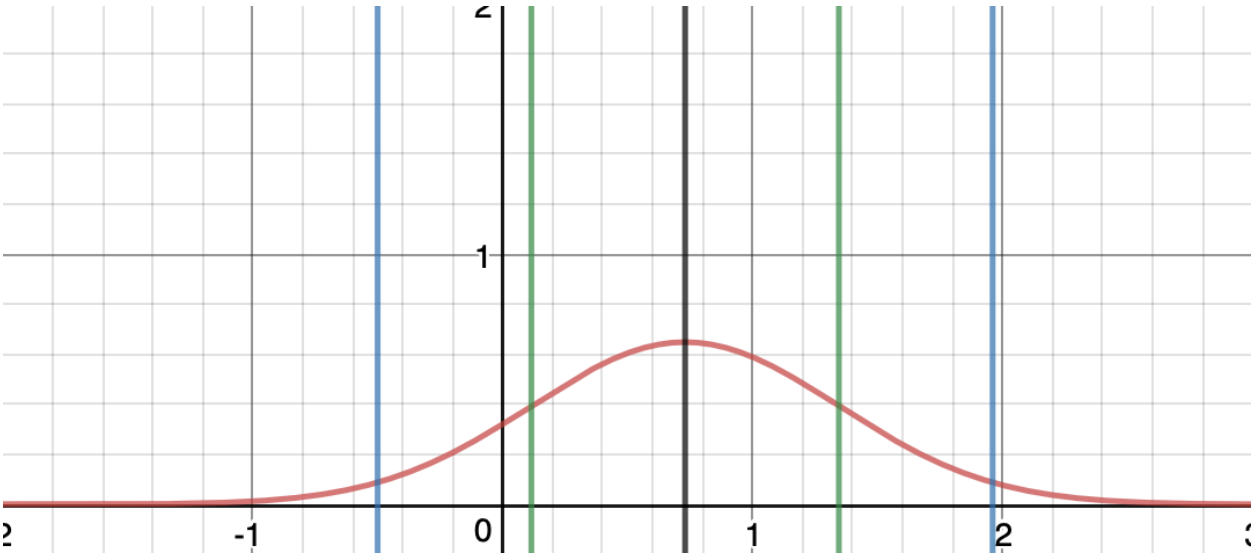


B

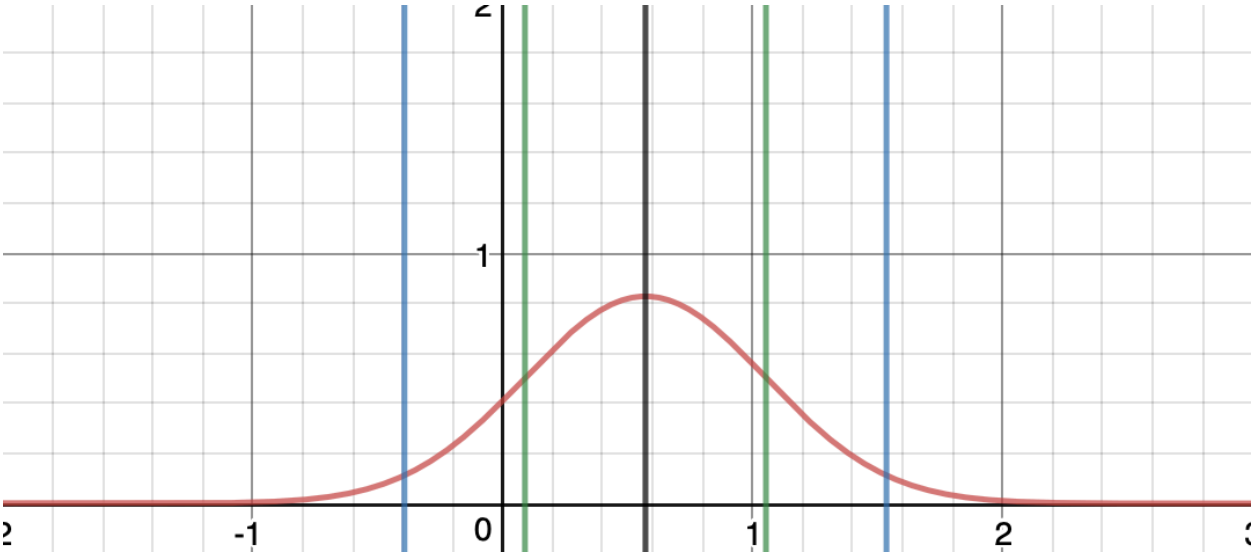


YELLOW CAN

R



G



B