ECSE 211
Design Principles and Methods
Winter 2019

Date: 27th February 2019

Lab 5

Search and Localize

Team 18

Preyansh Kaushik	260790402
Alexander Asfar	260771684
Maxime Rieuf	260782116
Dhriti Rudresh	260710210
Marwan Khan	260762251
Theodore Janson	260868223

Section 1: Design Evaluation

Hardware Design:

The hardware design for this lab borrowed upon multiple previous iterations of hardware designs during the first four labs. Since our resources tripled for this lab, with two extra kits, our hardware design for this lab considered the extra resources available.
To this avail, we decided to use three color sensors and one ultrasonic sensor. While this lab does not cater for the need of a claw, we kept the potential design for a claw in the project after in mind and left space for it at the back of the robot.

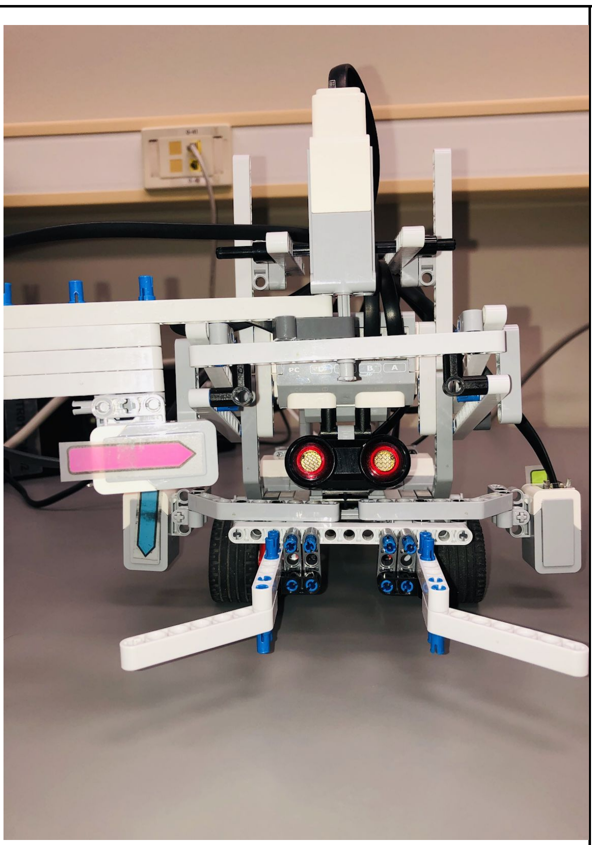
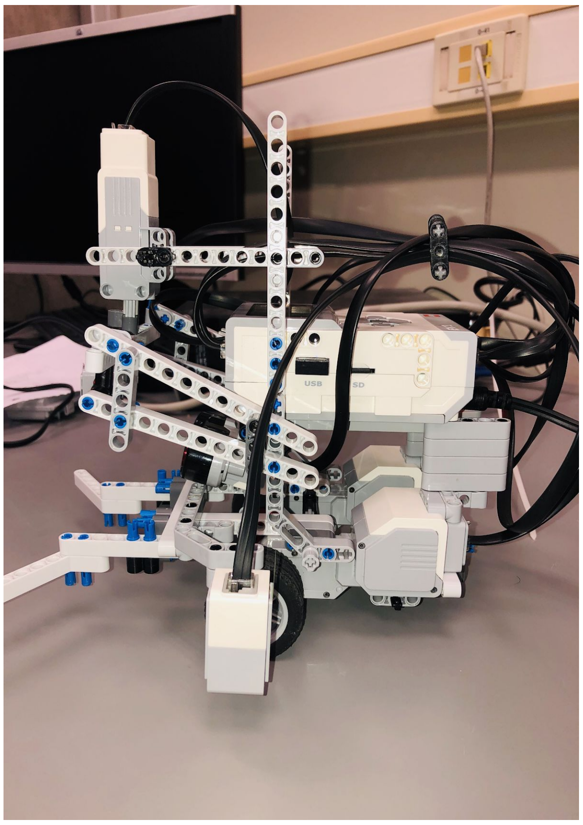
	

Figure 1: Hardware view from the front	***Figure 2: Side view***

Depending on our designs, we decided to use extra sensors such as color sensors on both sides and one to detect the can in front. We characterized the light sensors according to their ability to detect colours and black line in order to determine the location of each sensor. The best sensor was used for can colour detection. The two color sensors most similar characterization were placed on the sides, while the most accurate one placed in front for the can color detection. We attached an extra motor to the top that will move the color sensor to detect the cans, as it rotates around the can and stores all the RGB values detected. This was placed above the can's location to maximise the arc length by which the colour sensor scans (close to the can). Below the ultrasonic sensor is a guider. This ensures that if the robot approaches a can at an offset angle, the can may be guided into place by a set of arms.

Software Design:

We organized our software design by the use of a UML class diagram as shown below to depict how our classes interact:

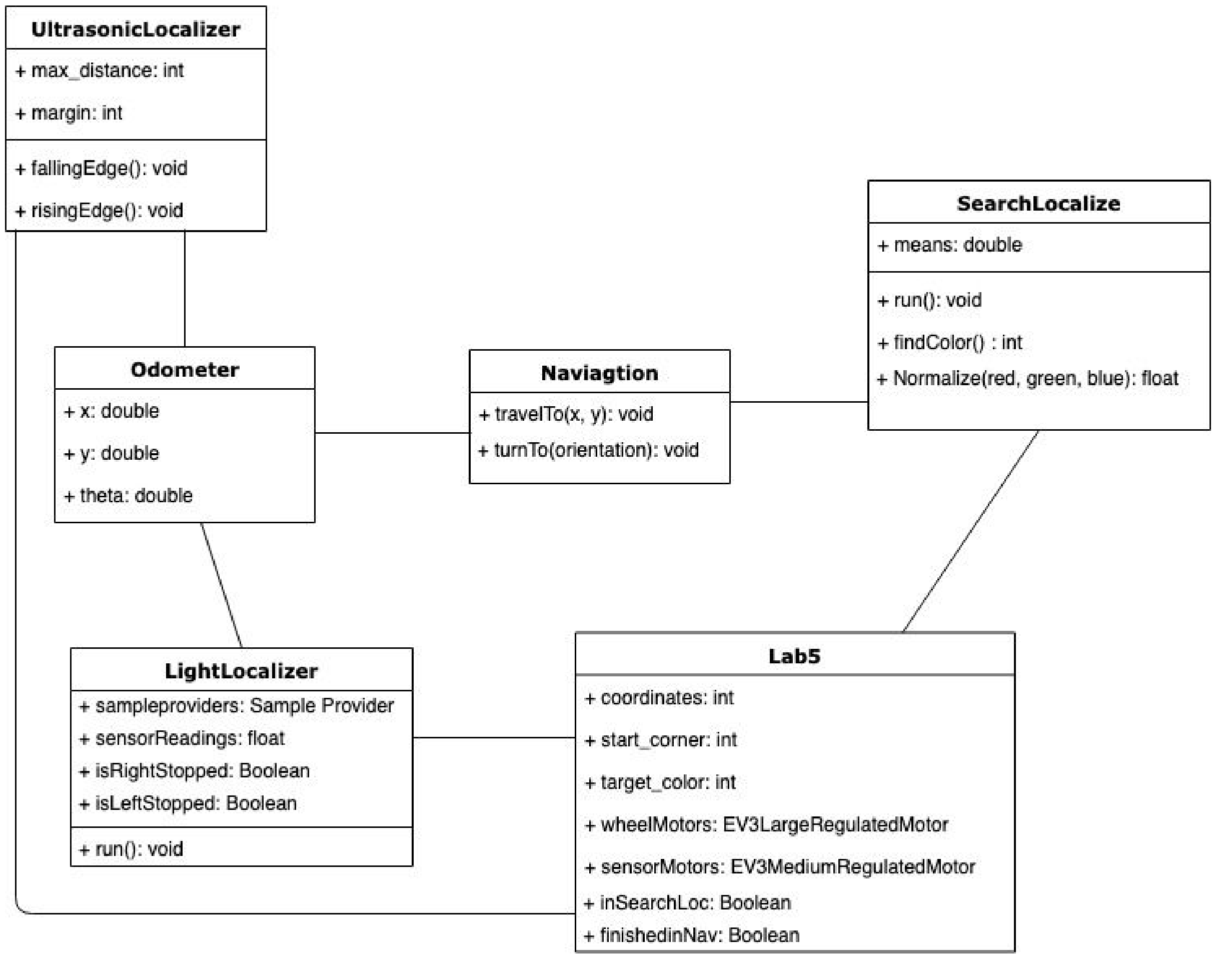


Figure 3: Class diagram

For this lab, we had to keep in mind the algorithms to employ, and the main considerations were given to specifically the search algorithm, the color detection algorithm, and more.

We had to keep in mind the algorithms to employ, and the main considerations were given to specifically the search algorithm, the color detection algorithm, and more.

1. *Search Algorithm*

Our main idea initially, was to sweep around the lines and detect the can with a rotating sensor around. However, upon multiple thought and criticism of this idea, we decided that this was not the most efficient way, especially given time constraints.

Upon perusal, it was decided that the robot should move along one column and turn 90 on every row to detect whether cans lie on that row. Since all the cans are placed on waypoints, this method works.

2. *Localization*

Ultrasonic Localizer:

This class uses the usPoller class to retrieve the distance from the ultrasonic sensor. For the localization of the robot to the starting waypoint, we opted to use the Falling Edge method described below.

Falling Edge:

We implemented Falling Edge, which works best when the robot is facing away from the wall. Therefore, if the robot is facing the wall on initialization, the robot will rotate until it faces away from the wall. After this step, it will rotate until the left wall until the distance reduces significantly on detection of the wall. (i.e. the distance is under the threshold, and so a falling edge is found). At this point, we set the odometer's theta value to 0 to make future computations easier. After this, the robot rotates in the opposite direction, towards the bottom wall. Once it detects this second falling edge, we record the theta value. Rotating the robot by half this theta value would place the robot at 45 degrees and therefore facing towards the starting waypoint.

Light Localizer:

The instance for this class is called upon after the ultrasonic localization terminates, so that the robot is facing in the positive y-axis direction, within a degree of error.
The speed of the robot at this stage is set to be very low to ensure precision for the light sensor reading. Since velocity and sample frequency are intertwined, this helps us achieve more precise samples.
We then use a differential filter, which is explained below in the Noise Filtering section of this document, to detect the black line.

As for the run method, the robot goes forward until one of the sensors detects a black line. If a sensor detects a black line, we stop the motor which is on the same side of this sensor. Through this method, when both sensors detect a black line, we know that the robot is approximately facing the positive y-axis direction and hence correcting errors of ultrasonic localization. At this stage, we set the y coordinate to the tile's size and theta to 0. Then the robot goes backward (to avoid the sensor touching a line while turning) and turns 90° to the right to face the black line with equation x=1. We follow the same process and set the x coordinate to the tile's size. The method concludes by travelling to the waypoint (1,1).

3. Navigation:

This class is used to travel to a point defined by its coordinate.
Getting the minimum angle method: This method adds 360° to an angle if it is below -180° and subtract 360° to an angle if it is above 180°
Avoid Obstacle method : Since we know the dimension of the can (which will be the only obstacle in this project) we implemented a can-specific algorithm. First turn 90° to the left, go forward 20 cm, turn 90° to the right, go forward 40 cm, turn 90° to the right and go forward 20 cm.
TurnTo method : This method takes an angle as parameter. This angle is the new orientation we want the robot to have. First we subtract to this angle, the angle at which the robot is oriented: the result is the amount the robot have to rotate to get to the new orientation. We get the minimum angle of the subtraction and make the robot rotate this amount (with the method provided by the leJOS library).
TravelTo method : This method takes the coordinates of the destination as parameter. We first compute the difference in x-coordinate and y-coordinate between the destination and actual location and compute the arctan (with the method of the Math library that takes coordinates of a point as parameter) to get the new orientation the robot needs to get to. We then call the turnTo method with the new orientation as parameter. For the distance, we compute it using the differences and x and y and make the robot advance by this amount. However, if along the way the robot detects a can in front of him, it goes to it and scans when it is close enough. After the scanning, we avoid the can using the method described above. If the can was the right one we call travelTo recursively to go to the Upper right corner of the search area and conclude the code. If not, then we also call travelTo recursively and go where the robot was supposed to go initially.

4. *Color Detection algorithm*

For our color detection, we implemented an algorithm that initializes the motor to rotate the color sensor around the can after ultrasonic detection. The RGB values are then stored and normalized before detection is confirmed as a yellow, blue, green, or red can.

One main challenge with the color detection is that the cans specified by the client are not consistent with one color in any given section of the can. Due to the “nutritional information” column, most cans have a white section that labels this. At the same time, the yellow can poses a problem since it is a combination of RGB values and not a peak of any of these primary colors alone.

The process of normalizing the data beforehand is to cater for the former problem. As samples are taken around the can using a color sensor attached to a motor, the most prevalent readings give insight onto the color of the can.

As for detecting the yellow can, several tests were conducted of expected rgb values and this was extended to all other cans too. By calculating averages of every rgb value, we can compare euclidean distance for each sample to these averages to enable detection of the color.

Section 2: Test Data

Model Acquisition:

For each TR, we obtained 10 independent trials for their respective RGB values.

	Blue Can			Red Can		
Trial No	Red Median	Green Median	Blue Median	Red Median	Green Median	Blue Median
1	0.267	0.619	0.704	0.987	0.11	0.076
2	0.254	0.636	0.722	0.986	0.117	0.09
3	0.318	0.641	0.725	0.981	0.121	0.106
4	0.239	0.65	0.687	0.985	0.123	0.107
5	0.254	0.641	0.72	0.985	0.133	0.101
6	0.298	0.641	0.731	0.99	0.099	0.075
7	0.287	0.648	0.7	0.985	0.118	0.092
8	0.281	0.628	0.717	0.978	0.135	0.115
9	0.238	0.639	0.635	0.983	0.136	0.101
10	0.315	0.559	0.728	0.977	0.149	0.136

	Green Can			Yellow Can		
Trial No	Red Median	Green Median	Blue Median	Red Median	Green Median	Blue Median
1	0.398	0.808	0.429	0.841	0.496	0.164
2	0.408	0.746	0.333	0.848	0.492	0.204
3	0.402	0.804	0.334	0.852	0.475	0.123
4	0.341	0.762	0.4	0.832	0.483	0.127
5	0.348	0.811	0.333	0.854	0.478	0.123
6	0.447	0.811	0.338	0.821	0.492	0.141
7	0.348	0.744	0.417	0.84	0.492	0.175
8	0.408	0.748	0.408	0.867	0.479	0.128
9	0.353	0.851	0.349	0.874	0.471	0.118
10	0.327	0.886	0.302	0.871	0.468	0.119

Figure 4: Table representing the test results for the RGB values of each can

The can to sensor distance depends on the orientation of the color sensor. From the side, the can to sensor distance is at a measured 1.9 cm, while from the front it is at 0.5 cm.

Color and Position Identification:
(LLX, LLY) = (3,3), (URX, URY) = (7,7), SC = 0.

TR = 1, BLUE
Rgb values found:

Can	R	G	B
Blue	0.254	0.681	0.720
Red	0.291	0.628	0.717
Green	0.318	0.641	0.725
Yellow	0.239	0.650	0.687

Figure 5: Table representing RGB values for blue can

(TPRx, TPRy) = (5,3)
(TPEx, TPEy) = (5.6, 3.2)

TR = 2, RED
Rgb values found:

Can	R	G	B
Blue	0.985	0.133	0.101
Red	0.978	0.135	0.125
Green	0.981	0.121	0.106
Yellow	0.985	0.123	0.107

Figure 6: Table representing RGB values for red can

(TPRx, TPRy) = (12,8)
(TPEx, TPEy) = (10.1, 9.6)

TR = 3, GREEN
Rgb values found:

Can	R	G	B
Blue	0.348	0.811	0.333
Red	0.408	0.748	0.408
Green	0.402	0.804	0.334
Yellow	0.341	0.762	0.429

Figure 7: Table representing RGB values for green can

(TPRx, TPRy) = (7, 2)
(TPEx, TPEy) = (7.9, 3.7)

TR = 4, YELLOW
Rgb values found:

Can	R	G	B
Blue	0.854	0.478	0.123
Red	0.837	0.479	0.128
Green	0.852	0.475	0.123
Yellow	0.832	0.483	0.127

Figure 8: Table representing RGB values for yellow can

(TPRx, TPRy) = (25, 20)
(TPEx, TPEy) = (23.8, 22.4)

Section 3: Test Analysis

Model Acquisition:
Compute the RGB mean and standard deviation values for each of the 4 sample TR sets.

The mean and standard deviation are respectively calculated by the following:

$$\text{Mean} = \frac{\sum fx}{\sum f}$$

$$\text{Standard deviation} = \sqrt{\frac{\sum fx^2}{\sum f} - \left(\frac{\sum fx}{\sum f}\right)^2}$$

The results we obtained are shown below:

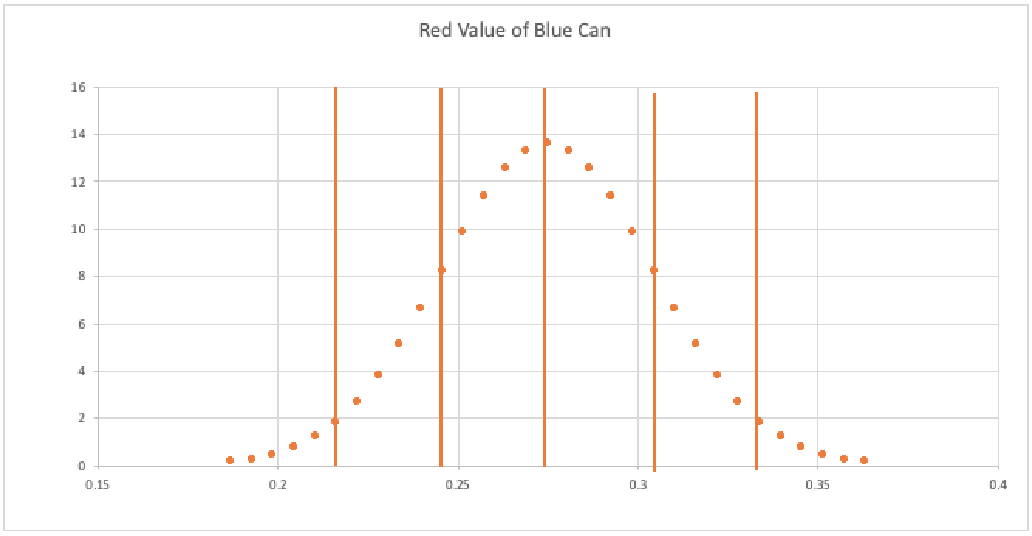
Can	Red Mean	Green Mean	Blue Mean
Blue	0.275	0.630	0.707
Red	0.984	0.124	0.100
Green	0.378	0.797	0.364
Yellow	0.850	0.483	0.142

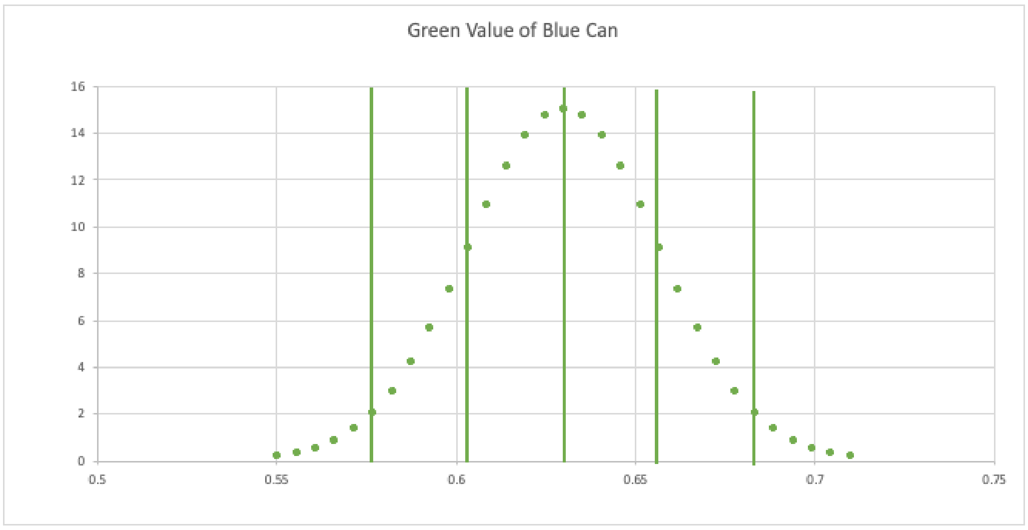
Figure 9: Table representing the means of RGB values for each can

Can	Red Standard Dev	Green Standard Dev	Blue Standard Dev
Blue	0.029	0.027	0.029
Red	0.004	0.014	0.018
Green	0.039	0.048	0.045
Yellow	0.017	0.001	0.029

Figure 10: Table representing standard deviation of RGB values for each can

For each sample TR set, plot the corresponding RGB normalized Gaussian distributions and draw vertical lines at μ, ±1σ from μ, and ±2σ from μ.





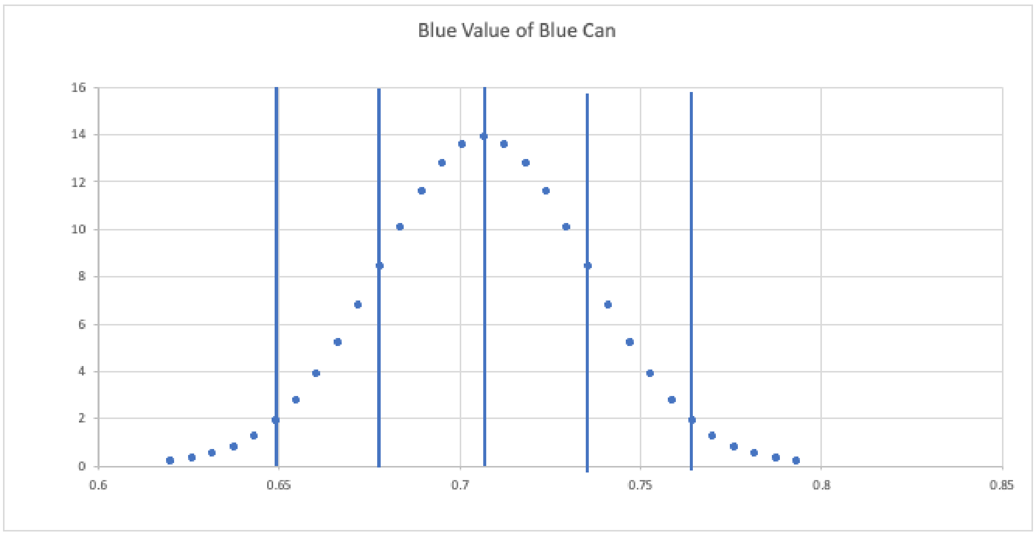
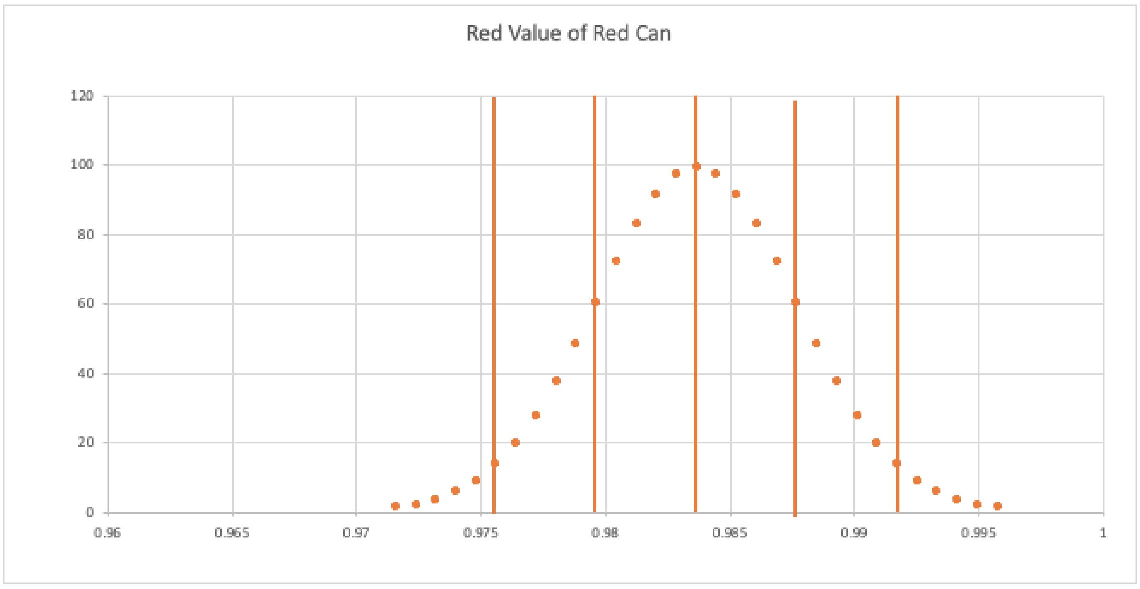
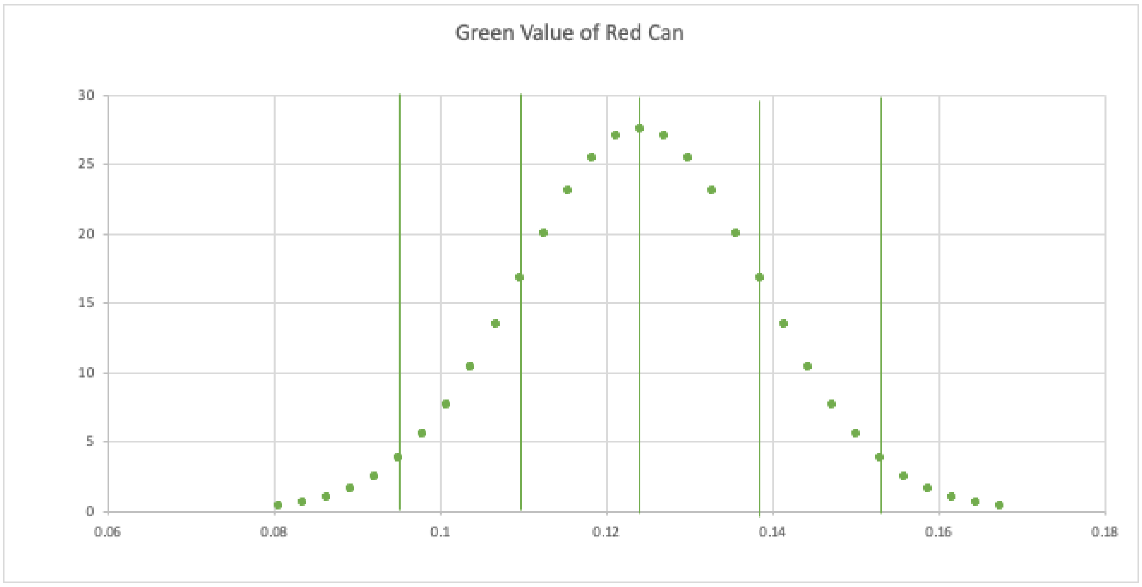


Figure 11: Gaussian Distribution of the RGB values for the blue can





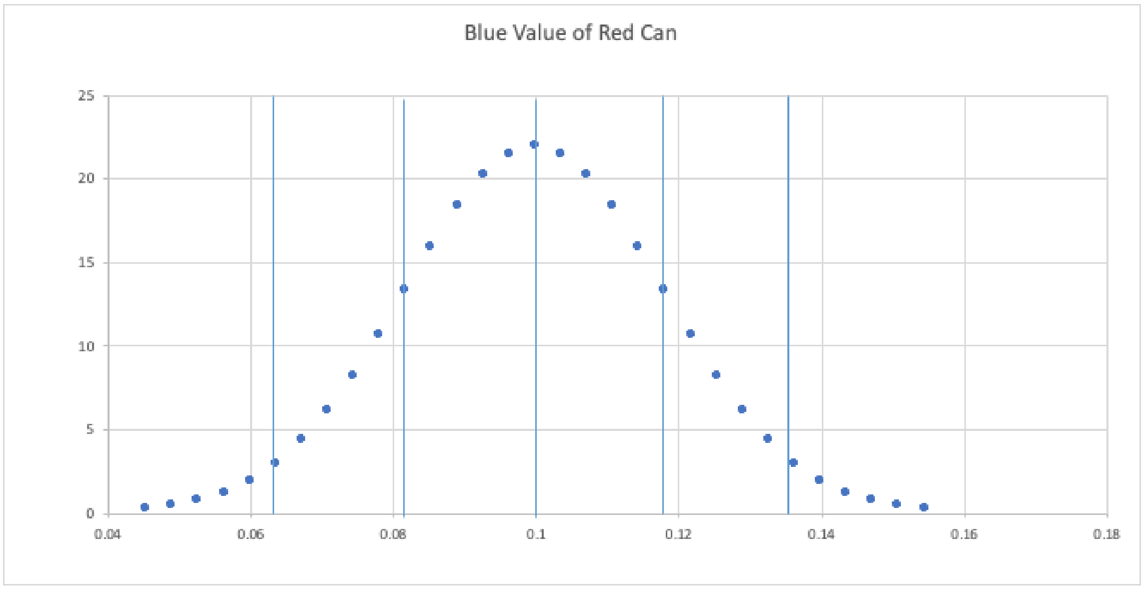


Figure 12: Gaussian Distribution of the RGB values for the red can

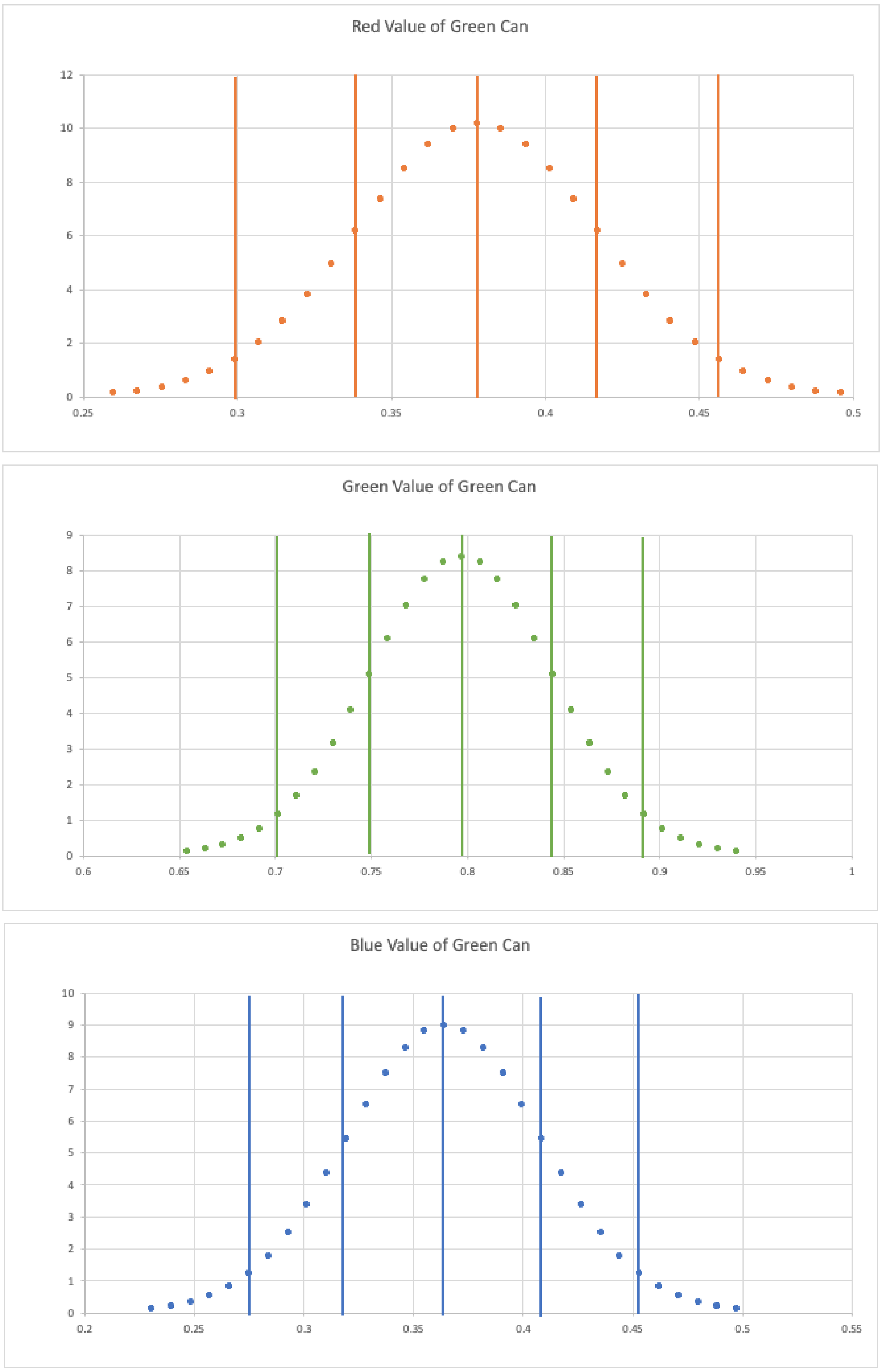
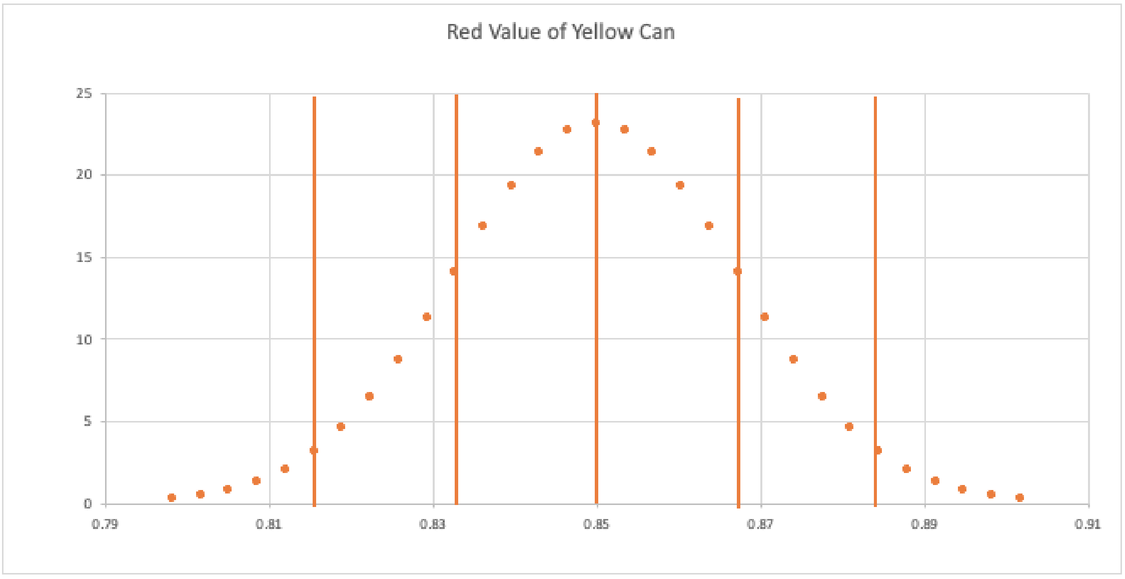
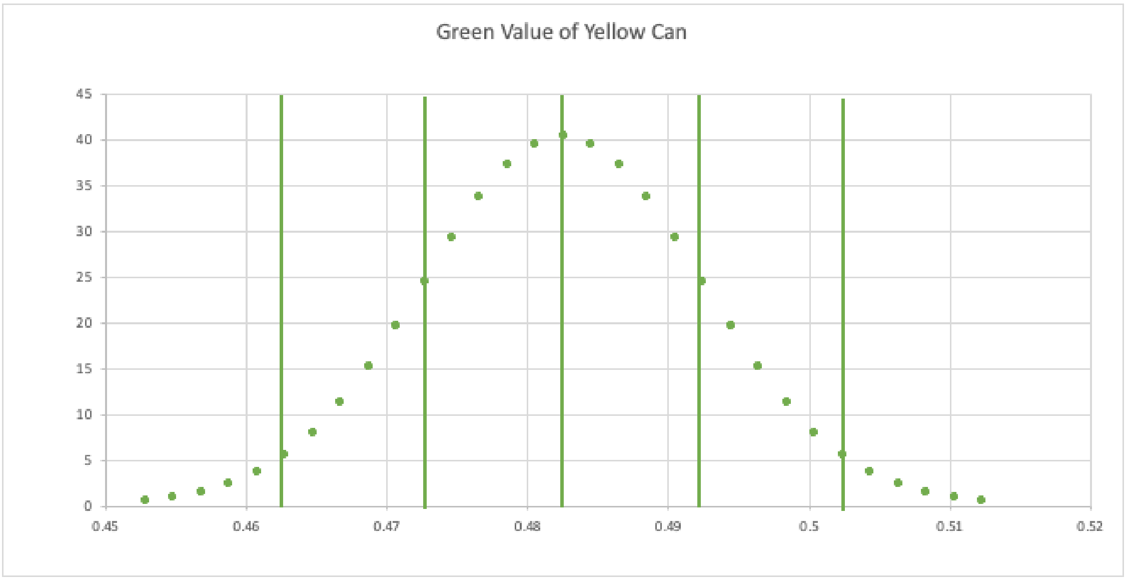


Figure 13: Gaussian Distribution of the RGB values for the green can





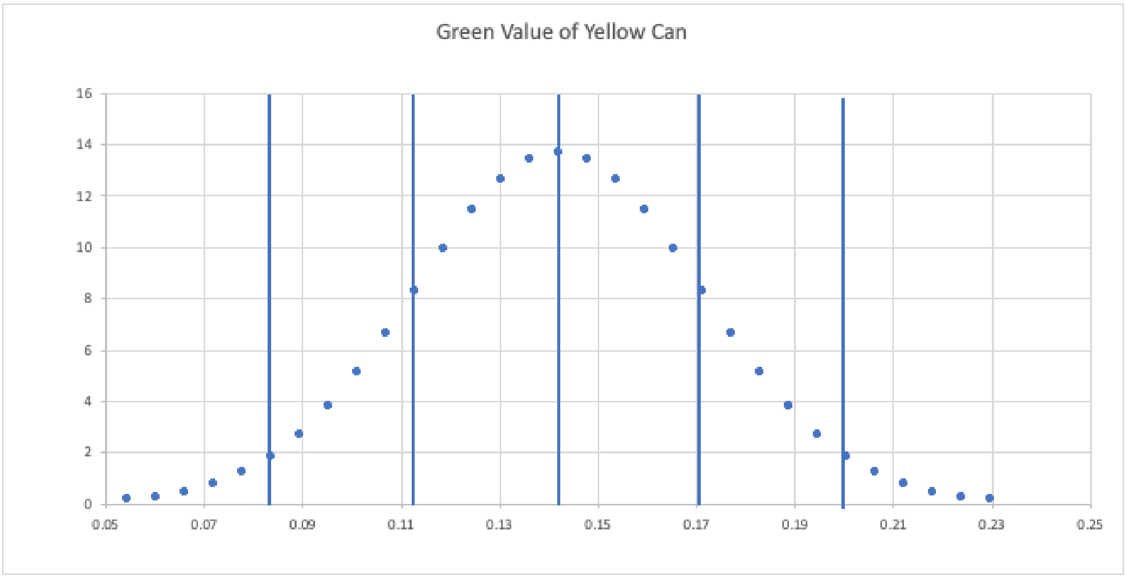
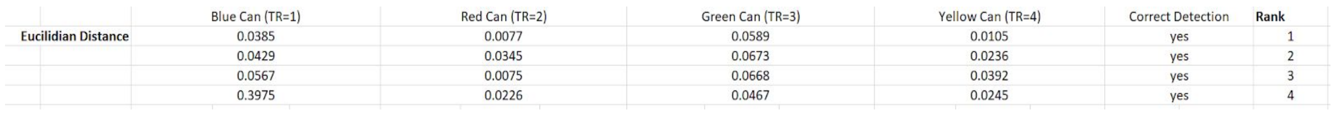


Figure 14: Gaussian Distribution of the RGB values for the yellow can

Color and Position Identification:

For each field test run, compute the Euclidean distance, d, of all the recorded RGB values. This should result in 4 distance values for each recorded sample. Then, rank the order the distance for each can index in ascending order.



	Blue Can (TR=1)	Red Can (TR=2)	Green Can (TR=3)	Yellow Can (TR=4)	Correct Detection	**Rank**
Eucilidian Distance	0.0385	0.0077	0.0589	0.0105	yes	1
	0.0429	0.0345	0.0673	0.0236	yes	2
	0.0567	0.0075	0.0668	0.0392	yes	3
	0.3975	0.0226	0.0467	0.0245	yes	4

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

To determine the can position, we initiate the ultrasonic sensor in accordance with the “search algorithm” we described earlier. If there is a can at most two tiles away, we go forward until we are only 3 cm away. The color is detected using the color sensor, and we read off the odometer values on the lcd screen using the leJOS EV3 Control plugin on Eclipse.

Calculate Euclidean distance between estimated position and real position.

The Euclidean distance can be calculated by the following formula:

$$\sqrt{(TPEx - TPRx)^2 + (TPEy - TPRy)^2}$$

Target Can	Euclidean distance calculated
Blue	0.633
Red	2.483
Green	1.924
Yellow	2.683

Figure 15: Table representing the euclidean distance between expected and real position of target can

Section 4: Observation and Conclusions

Are rank-ordered Euclidean distances a sufficient means of identifying can colors? Explain in detail why or why not.

Euclidean distances are not a sufficient enough means of identifying can colors, because of the existence of false positives. Since the cans are composed of a multitude of values: i.e. red intensity, green intensity, and blue intensity, it is rather likely that despite being a red can, the blue intensity can peak due to sensor accuracy, ambient light, or other factors. In this case, the euclidean distance would identify the can as blue due to small euclidean distance to blue mean. Therefore, it is better to normalize data and compare to distinguish outliers effectively.

Is the standard deviation a useful metric for detecting false positives? In other words, if the can color determined using the Euclidean distance metric, d, is incorrect, can this false positive be detected by using $\mu\pm1\sigma$ or $\mu\pm2\sigma$ values instead?

Yes. It is expected to have values that are outliers or false positives due to limitations of sensor accuracy and other environmental conditions (ambient light, etc). However, since we normalize all data, the outliers can be filtered out by only looking at values that are within one or two standard deviations away from the sample mean.

Under what conditions does the color sensor work best for correctly distinguishing colors?

The color sensor works best when the can is centered right in front of the robot, which is why we have “guider” rods on the front of our hardware design to ensure the can is stuck in one position, right infront of the robot. The color sensor works best in conditions where ambient light is kept to a minimum, since intensity from ambient light could affect reading as it is not only the reflection of the illumination of the color sensor that is recorded. This could easily be fixed by having a skirt on the sensor so that only the illumination is used to record data and ambient light is minimized.
The color sensor also only works best when testing and sampling has been carried out before to normalize data and compare it to a pre-tested mean.

Section 5: Further Improvements

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

Currently, the values are normalized. We could improve the results by using a median filter to filter out the noise due ambient lighting. The filter would run through the data in the arrays corresponding to the red, green or blue light. The array index represents the sample number.

In each of the arrays, each element (value for red, green, blue) is replaced with the median of the values of the element’s around it. The “window” would include the a few previous and following array elements. Median filters are commonly used for image processing, by filtering out noise of each pixel, which smoothes out its surrounding colour. Applying such a filter to the data collected by the light sensor would similarly smooth out the colours detected by the filter. This is advantageous because it would remove outliers due to the detection of alternate colours on the can and errors due to ambient lighting. Since we do not need to discern between types of yellow or red, and each colour is distinct, the use of a median filter would increase the likelihood of correct colour classification.

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

The can’s position is given by the sum of robot’s current location and the distance between the offset between the robot’s centre and the can. This offset is defined as the distance between the location of can colour scanning (ultrasonic distance ~ 5) and the centre of the robot. We now figure out what direction this offset is acting on using the heading of the robot. Since offset is constant, the only variable in improving the odometer’s accuracy, so that the robot’s location is more accurate.

This could be done by using light localization more frequently during the routine. Every time the robot crosses three tiles, it should perform light localization using the same 2-sensor technique currently used. This would make ensure that the robot is following the grid lines. The odometer’s values would be updated as such, which would increase the accuracy of the robot’s location.