Autonomous Trash Collection Robotic System

ECE 8843_Final Project Report

LaVonda Brown, Sergio García, Mason Nixon, Giancarlo Valentín, Ivan Walker
Department of Electrical and Computer Engineering
Georgia Institute of Technology

***Abstract* - Picking up the trash of people's homes in a given neighborhood is not the most appealing job. With the advances in technology it is now a possibility to use autonomous mobile robots to traverse a neighborhood with the main goal of retrieving the trash from every house. To prove that this idea is indeed possible and viable, we created a simple small scaled neighborhood. We programmed the commercially available ActivMedia's Amigobot with a lane detection algorithm to stay inside the road, computer vision algorithms to obey traffic signals, and with an algorithm to detect simulated trash containers in order for the robotic arm to pick it up. This paper presents the individual components of the projects together with the conclusions of our work.**

Keywords: autonomous mobile robots, Amigobot, autonomous trashcollection, lane detection algorithm

I. INTRODUCTION

Autonomous trash collection is indeed a viable alternative for our society. The advances in technology allow us to have a team of autonomous mobile robots collaborate with each other with the main goal of following a predetermined path in a given neighborhood and picking up the houses' trash. Having autonomous mobile robots perform these simple tasks is more cost efficient rather than having to pay a human to do it. The initial investment of a system like this may be expensive, but in the long run it's more expensive to pay a human's salary and health benefits.

It is the purpose of this paper to demonstrate the concept behind autonomous trash collection. A simple small scaled neighborhood was drawn with chalk and the commercially available mobile robot Amigobot was used to traverse said neighborhood. The idea is to equip the Amigobot will all the necessary software and hardware needed to make autonomous trash collection a reality. Our small scaled trash collector behaves as if it were implemented in the real world. It stays between the lines of the road, it follows all traffic lights and signals, and it stops at any intersection. Once it detects a trash can, it stops to pick it up, and it deposits it in the onboard trash bed.

All of this is possible thanks to the advances in Computer Vision. Previous work has presented different lane detection algorithms [1] and light filter algorithms. We based our work on these algorithms and wrote our own. The lane detection algorithm is used to detect the lines on the road in order to avoid drifting away and to detect the stopping horizontal line at the intersections. The light filter algorithm is used to detect the traffic light at the main cross-section making it possible for the robot to obey all transit laws.

II. NEIGHBORHOOD MODEL

A. *Simulated Neighborhood*

Because of the Amgiobot's size, implementing one-lane (one way) streets was considered in effort to keep the neighborhood model at a reasonable size. The Amigobot is approximately 11 inches wide, but will require a little extra room to adequately make turns and maneuver to pick up trash. If one-lane/one way roads were chosen, the trash collection neighborhood Sunset Place would have been modeled as shown in Fig. 1.

Sunset Place has been modeled with two main roads that intersect orthogonally: Main Street and Jones Avenue (Fig. 2). Sunset Court is the outer loop that connects these two roads to the trash dumpster where the robots will deposit the collected trash at this local trash repository. The traffic signal is located at the intersection of Jones Avenue and Main Street, and the stop signs can be found at the remaining intersections.

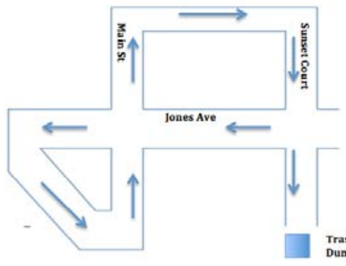


Figure 1. Simulated one way neighborhood.

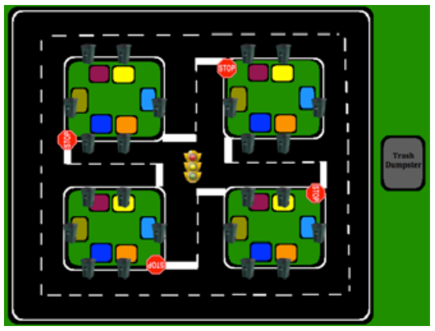


Figure 2. New Model of Sunset Place

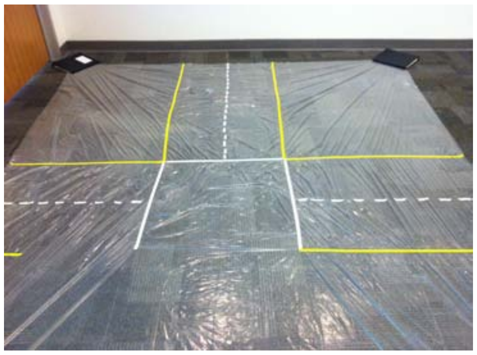


Figure 3. Attempted neighborhood model using the tarp

B. *Actual Neighborhood*

Initially, Sunset Place was going to be modeled using a tarp as its foundation for easy and convenient setup, but the tarp was too loose for the robots to traverse throughout the neighborhood. Instead, a more solid base will be needed for demonstrations, such as the floor of an empty room or multiple poster boards. The attempted neighborhood model using the tarp is shown in Fig. 3.

Considering the few options, sidewalk and chalk was chosen as the best materials to build Sunset Place. Being that the sidewalk is everywhere, there were no limitations in regards to location. In addition, the sidewalk is a very solid foundation, which is ideal for the Amigobot to traverse throughout the neighborhood. Chalk deemed to be a very feasible solution to draw Sunset Place on the sidewalk. Initially the neighborhood was drawn to model a real neighborhood precisely. The lanes were drawn as white and yellow boundaries lines, as seen on a real road. Due to vision complications of the camera when the sun was out, these colors were not able to show enough contrast on the sidewalk and the Amigobot could not identify the lanes. Because of this, Sunset place was drawn with a variation of blue and red chalk. Fig. 4 shows the semi-final model of Sunset Place drawn in blue chalk.

In efforts to beautify our neighborhood, houses were built and each house was required to have the same size and shape garbage bin. The traffic signal was placed at the 4-way intersection and stop signs were placed at each horizontal stop line. Due to lane detection complications, Sunset Place had to be scaled down to only one straight lane as shown in Fig. 5. The 90° angle turns were too much for the Amigobot to handle at this point in our research.

III. Computer Vision

A. *Graphical User Interface*

To analyze the visual data obtain from the Dynex 1.3MP webcam, we used the OpenCV2.0 C++ library. First the image is captured from the live video stream and stored as an

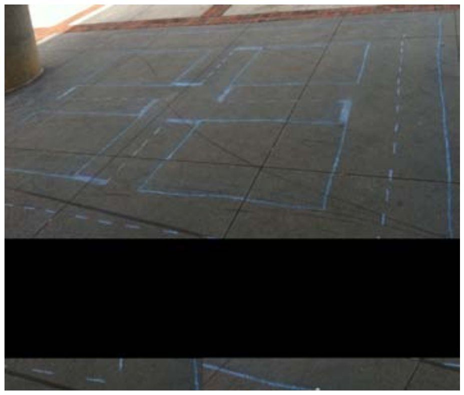


Figure 4. Sunset Place drawn with blue chalk

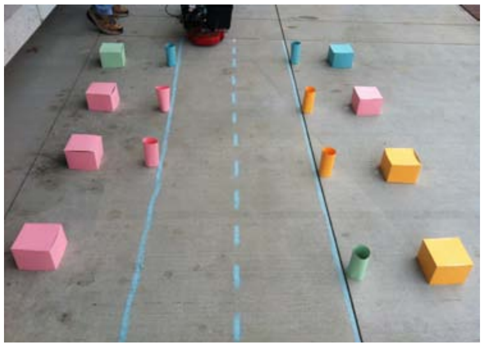


Figure 5. Scaled down Sunset Place with houses and garbage bins

IplImage data type. Next the image is converted from the RGB (red, green, blue) color space to the HSV color space (hue, saturation, value). HSV is a cylindrical-coordinate representation of points in an RGB color model, which rearranges the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the Cartesian (cube) representation (Fig. 6).

Performing this conversion allows us to then perform an HSV range threshold. This threshold isolates every part of the HSV image that fits in the specified range and then outputs a binary image where the white spots are within the range and the black is outside of the thresholded range (Fig. 7).

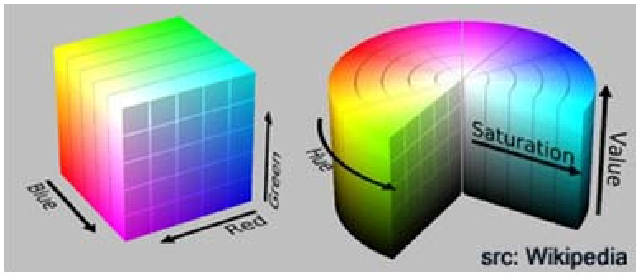


Figure 6. RGB color space and HSV color space

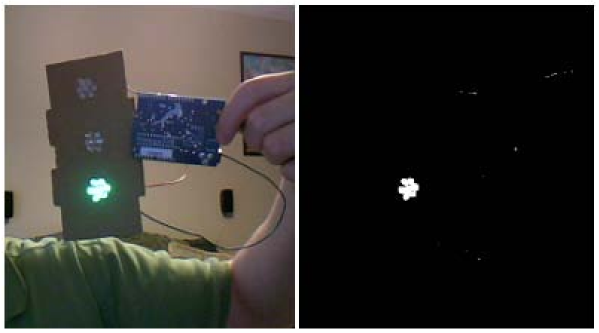
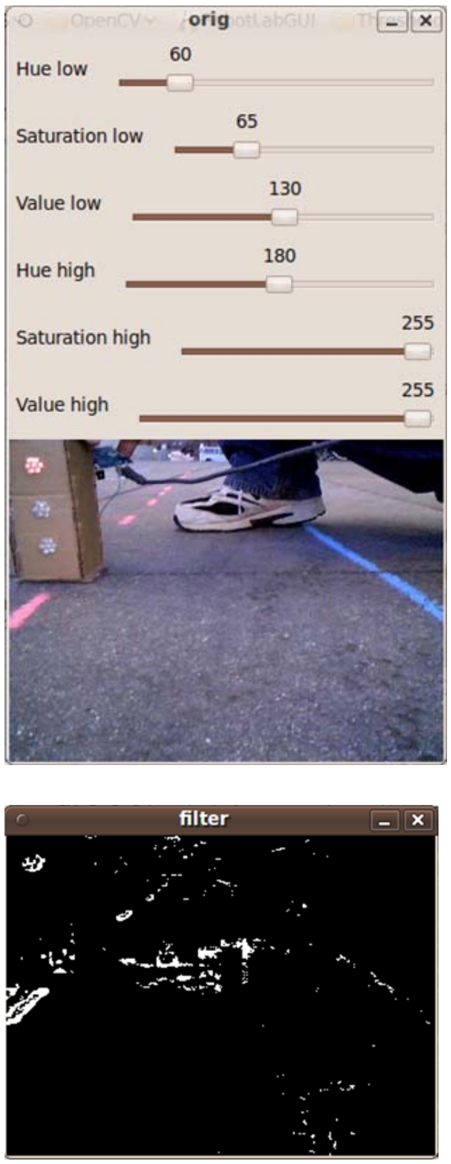


Figure 7. HSV filter of the green traffic light



Hue low 60

Saturation low 65

Value low 130

Hue high 180

Saturation high 255

Value high 255

Figure 8. HSV filter GUI

Optimizing the values was highly dependent on lighting of the surroundings and similar colors in the environment. These similar colors showed up in our filtered image as visual “noise,” which made a very narrow range over which we could filter. This was overcome by using weighted pixel areas via OpenCV structures called moments. As defined in the OpenCV library, “the function computes moments, up to the 3rd order, of a vector shape or a rasterized shape”. Equation (1) presents how the spatial moments (m_{ji}) are computed in case of a raster image [2].

$$m_{ji} = \sum_{x,y} (array(x,y) \cdot x^j \cdot y^i) \qquad (1)$$

The central moments (mu_{ji}) are defined by (2).

$$mu_{ji} = \sum_{x,y} (array(x,y) \cdot (x - \bar{x})^j \cdot (y - \bar{y})^i) \qquad (2)$$

The mass center $(\bar{x}, \bar{y})$ is defined as (3)

$$\bar{x} = \frac{m_{10}}{m_{00}}, \qquad \bar{y} = \frac{m_{01}}{m_{00}} \qquad (3)$$

After isolating our desired object using a color HSV threshold we first calculate the moments to estimate the position. Next we divide these calculated moments by the moment area and find the center coordinate of the detected pixels. Finally, a circle is drawn with the center at this point for debugging. At first, we tried to optimize this process by choosing different range values and recompiling after each small change. This proved extremely tedious and inefficient, fortunately OpenCV has a built in graphical user interface (GUI) library to cater to this exact situation. We were able to successfully create a GUI for optimizing these thresholds (Fig. 8).

The sliders in Fig. 3a enabled us to dynamically set the HSV low side of the threshold and the HSV high side. This GUI saved a tremendous amount of time and was also modified later and was used for brightness and contrast filtering as well as Hough transformations for line detection.

B. *Traffic Light Construction*

We were able to design a working traffic light. To do this, we utilized 3 clusters of 7 LEDs. Each LED is powered and controlled by a single I/O pin on an Atmel microcontroller with the Arduino boot loader. The code used for this is a simple infinite loop with a set delay between each light array. Using a single Atmel328 microcontroller we are able to control and power up to 4 traffic lights at once. However, due to the scale of this prototype, it was unnecessary to make more than one, although it would be a trivial matter to orchestrate a single intersection of 4 lights (Fig. 9).

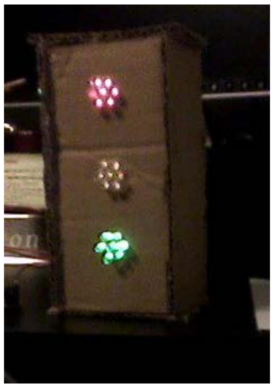
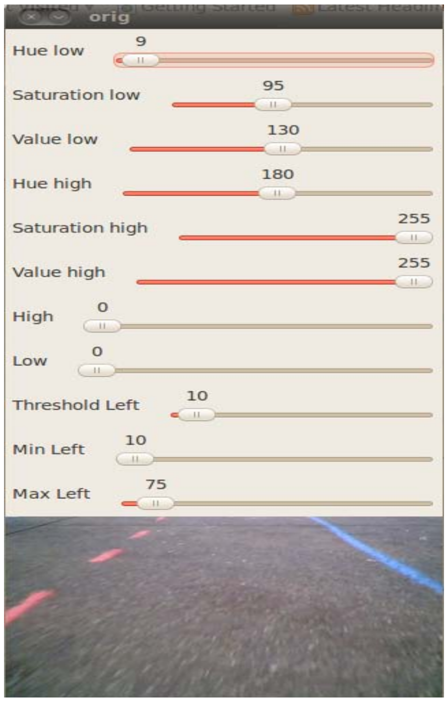


Figure 9. Traffic light


orig

Hue low 9
Saturation low 95
Value low 130
Hue high 180
Saturation high 255
Value high 255
High 0
Low 0
Threshold Left 10
Min Left 10
Max Left 75

Figure 10. Modified HSV GUI with cropped video feed

C. *Traffic Light and Stop Sign*

Our robot is designed to be able to detect the traffic light as well as various stop signs. Both of these tasks are achieved by having the robot filter the image for each specified color. If the robot detects red, as in the stop sign and the traffic light, then we know that we must stop. Our traffic signal subroutine is repeatedly called by the main robot instruction to determine whether or not a signal is present (the word “signal” is used broadly here to mean both light and sign). To eliminate false detections due to noise, we average 10 frames and if all 10 are consistently green or red then we send the appropriate command. Also, we should note that our camera has a frame rate of 25 frames per second so we could determine detection in less than half of a second which is appropriate for the speeds our robot travels.

D. *Point of Stop Line*

Sending a signal at the moment we detect a signal proves problematic because we do not know how far away the robot is from the stop signal. To make the solution more “in the box” and modular from the main program, we borrowed the algorithm used for lane detection. Basically this is a Canny edge detection followed by a Hough transformation to determine the presence and position of a line. This line is decomposed into points and a slope. We determine which line is the stop line by isolating only the lines detected in the image with a slope in a range of ± 0.025 (approximately a horizontal line). If the points of the edges of the line are close enough to the robot (essentially we are counting pixels out to the line from the robot), then we send the command back to the main program for the robot to halt. The number of pixels was optimized so that the robot will have enough space to stop without crossing over the line. For the green light, we continue as usual with no interruption to the main program flow. If there is a yellow light present, then we simply allow the robot to act as if the light is green.

IV. Lane Detection Algorithm

A. *Graphical User Interface*

As mentioned in section A of Vision, the OpenCV library was used to perform vision processing for lane detection along with a modified version of the GUI mentioned there (Fig. 10). The GUI was implemented to determine the appropriate HSV thresholds necessary to filter the lanes in the image and rapidly determine the effects of changing thresholds for individual components of the algorithm, discussed in more detail later. The HSV color space was chosen because after weeks of tweaking a lane detection solution in the RGB color space it was discovered to be a less than optimal solution, maybe less than trivial, after failing to detect any lanes in the second image processed using it.

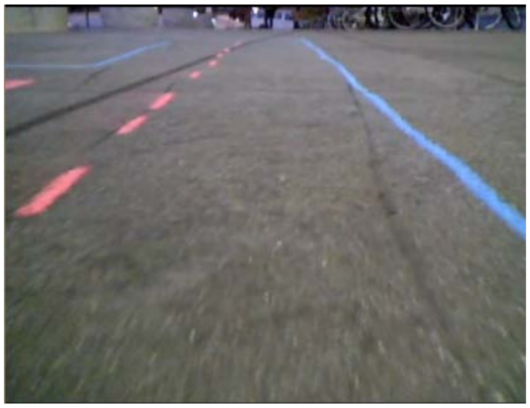


Figure 11. Original image, live video feed

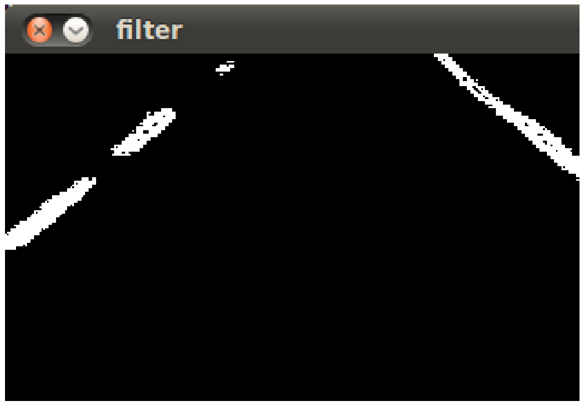


Figure 12. HSV filtered image

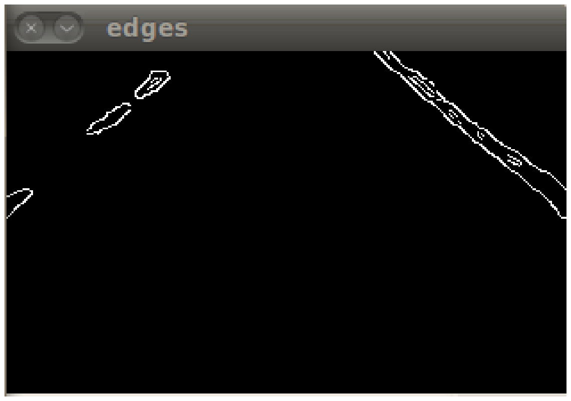


Figure 13. Image after Canny edge detection

B. *Implementation*

After several crash and burn attempts were made for lane detection, it became apparent that a little more thought would be required. The images recorded by the webcam demonstrated that it was difficult to place the camera on the Amigobot with identical poses between successive test runs. It became necessary to crop the image (Fig. 11) to remove as much external noise from the image as possible before processing. External noise in the image refers to the noise that is present outside any region of interests in the image, i.e. the roadway.

After removing the external noise the image show in Fig. 10 was filtered in the HSV color space using the threshold obtained from the first six sliders of the GUI shown. The resulting filtered image is in Fig. 12. The left and right lanes are easy to detect using visual inspection.

The HSV filtered image was then processed using a Canny edge detector. The Canny edge detector uses three parameters to determine edges. The first is an aperture size. The aperture size is used by the Sobel operator in the underlying implementation of the edge detector. The Sobel operator essentially performs image smoothing therefore reducing noise. An aperture size between 5 and 7 were determined to give the best results from the edge detector. The edge detector uses hysteresis to find edge; therefore, it is necessary to specify two thresholds to determine edges. The larger threshold is used to determine the presence of "strong" edges in the image. Then the lower threshold is applied using the information from the previous threshold to find smaller line segments. Using experimentation we were unable to determine optimal thresholds for the algorithm. It was only possible to determine non-optimal thresholds.

Fig. 13 was processed using a Hough Line transform to convert the edges shown in the image into lines in a 2D space. The Hough Transform chosen for this project was probabilistic. To assist the algorithm in classifying line appropriately for a target application, three key parameters must be modified. One such parameter is the accumulator threshold. Because the Hough transform uses a voting scheme, it is necessary to specify the number of votes needed to consider a point a part of a line. The other two parameters are the minimum line length and maximum allowed gap. The minimum line length is used to filter out noise that appears as lines in the image. The maximum allowed gap specifies how much space can be between points that lie on the same line. Although it detects lines in an image, the Hough transform provides no information to classify a line as part of an image.

Classification of lines as a lane was done by setting a slope threshold for lines to be filtered into left and/or right lane. The threshold for the left lane was $0.3 < \text{slope} < 0.6$. This threshold for the left lane was $-0.3 < \text{slope} < -0.6$. The average slope of all lines for each lane was taken along with the maximum and minimum x/y value. The left and right lanes were then connect at their endpoints to form a trapezoid. This trapezoid was then overlaid on the cropped video feed.

V. MECHANICS

A. *Mechanical Arm*

The mechanical arm used is made of five servomotors. However, we only used four of them given power limitations. Three of the motors were used to provide the arm with three degrees of freedom (DOF), while the other servomotor was used for the claw's end-effector.

The three DOF provided by the three servomotors allow the robotic arm to move in the plane perpendicular to the ground. We built the system in a way that both the trash bed and the potential trash can are located in the same plane as the mechanical arm at the time of picking up the trash can. This way, once the trash can is detected, the arm can move freely to pick it up, place it in the robot's trash bed, and to return to the original position waiting for the next detection.

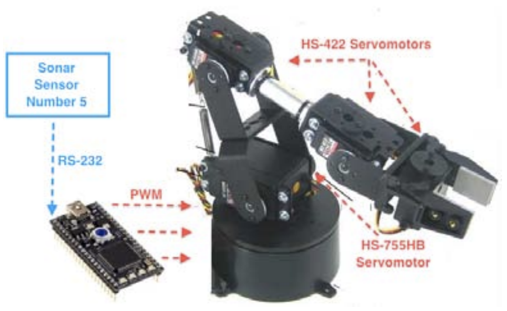


Figure 14. Connection between the mbed and the mechanical arm

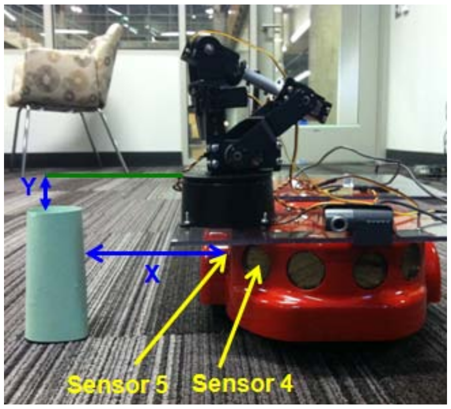


Figure 15. Trash detection via sonar

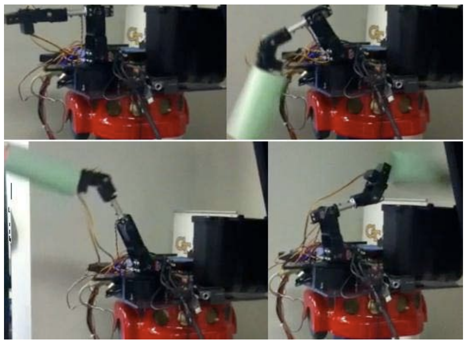


Figure 16. Implementation of the trash collection

B. *Trash Detection Algorithm*

It was necessary for us to use the onboard sonar sensors (sensors numbers 4 and 5) on the Amigobot to detect the trashcan (Fig. 15). In our main program loop we continually poll the forward sonar (#4) until we detect an object at a range of less than 270 mm. At this point we step into a subroutine that sets the speed to about a fourth of the normal and wait for sonar 5 to detect a trash can at a range of less than 270 mm. Once this happens, we tell the robot to stop and send the

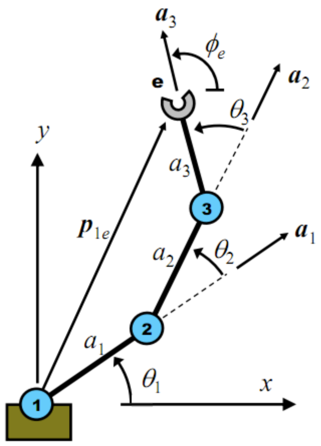


Figure 17. Mechanical arm angle references

distance (Fig. 15) to our robotic arm subroutine. This subroutine formats the distance as individual bits for a serial data transfer to an ARM-based mbed MCU on the robotic arm.

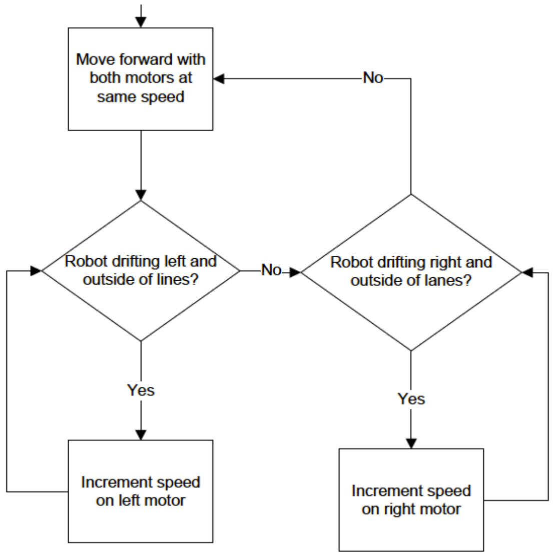
The coordinates are sent sonar sensor 5 as millimeters and are subsequently converted into angles for each of the threejoints of the mechanical arm. The conversion was achieved by programming the inverse kinematic equations using the arrangement in Fig. 17 [3]. These equations were implemented as a function call in the mbed's main program. The angle of pickup was set to -90 degrees since it was determined this angle would result in an optimal grip. The resulting angles are then converted into a 1 to 2ms pulses within a 20ms frequency range. These pulses were generated from 4 of the mbed's PWM outputs.

Once the servomotors receive their respective pulses, they move to the indicated location and the arm's claw is closed. With the object secured, the arm retrieves it and deposits it in a trash bin located behind it. The arm then resets to a standard position and waits for further commands.

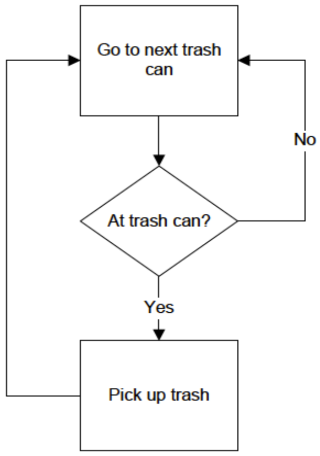
To increase the robustness of the trash picking function al trash objects were composed of paper cylinders with a height of 15 cm.

VI. METHODOLOGY

All of the mentioned components then come together for a completely autonomous trash collection system. At all times the system is polling for three different inputs related to the three main algorithms: the lines of the road for verifying if the car is drifting away, the traffic lights, signals, and the stop line to obey transit laws, and the presence of a trash can in order to pick it up. These three algorithms can be represented by three separate Flowcharts (Fig. 18).

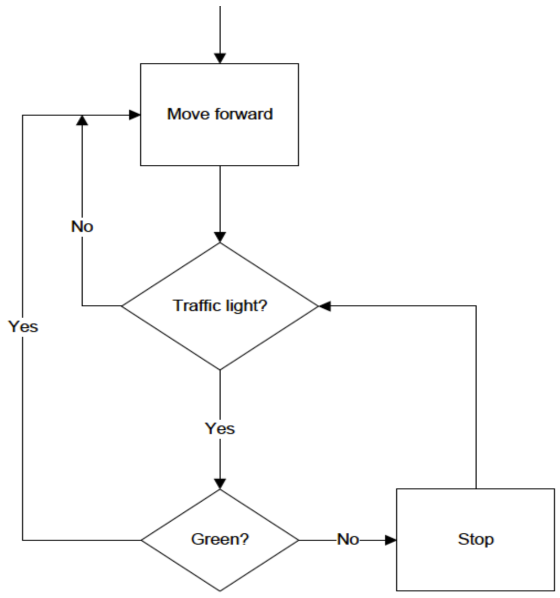

Move forward with both motors at same speed
No
Robot drifting left and outside of lines?
No
Robot drifting right and outside of lanes?
Yes
Yes
Increment speed on left motor
Increment speed on right motor

(a)



(c)

Figure 18. (a) Flowchart to detect the lanes of the road and correct possible drifting, (b) Flowchart to detect the traffic light and act upon red or green light, (c) Flowchart to detect a potential trash can and pick it up if detected.



(b)

At any point in time, the three algorithms are working at the same time. The process of one doesn't interrupt the process of the other two.

VII. Conclusion

Even though we weren't able to implement all the parts together due to time restriction, we were however able to have each individual part running.

At the end, we were able to prove that developing an autonomous trash collection system is indeed efficient and viable. The fully implemented system using our computer vision algorithms will be able to autonomously picking up the trash of the houses on a given neighborhood.

References

[1] A. S. Huang, S. Teller (2011). "Probabilistic Lane Estimation for Autonomous Driving using Basis Curves". Autonomous Robots, vol 31, no. 2-3, 269-283.

[2] Open CV Webpage (2010) "Structural Analysis and Shape Descriptors", http://opencv.willowgarage.com/documentation/cpp/structural_analysis_and_shape_descriptors.html

[3] Lipkin, Harvey. "Displacement Analysis Serial Robot slides". Slides obtained from the Introduction to Robotics course (Fall 2011) at Georgia Institute of Technology.