Automated Garbage Collecting Robot

Submitted To

Yeojoon Kim
Jennifer Jefferson
Dr. Alan Bovik
Supervising Professor

Prepared By

Ruide (Ray) Chen
Scott Chu
Bao Nguyen
Kevin Tan

EE464 Senior Design Project
Electrical and Computer Engineering Department
University of Texas at Austin

Spring 2010
CONTENTS

FIGURES ......................................................................................................................... iv  
EXECUTIVE SUMMARY ................................................................................................. v

1.0 INTRODUCTION ........................................................................................................1  
2.0 DESIGN PROBLEM STATEMENT ...........................................................................2  
3.0 DESIGN PROBLEM SOLUTION ..............................................................................3  
  3.1 ROBOTIC ARM MODULE ......................................................................................6  
    3.1.1 Hardware Components ......................................................................................6  
    3.1.2 Robotic Arm Design .......................................................................................7  
      3.1.2.1 Trajectory Planning ...................................................................................8  
      3.1.2.2 Arm Control ............................................................................................9  
  3.2 NAVIGATION MODULE ......................................................................................10  
  3.2 IMAGE MODULE ..................................................................................................12  
  3.2 INTERFACING .....................................................................................................13  
4.0 DESIGN IMPLEMENTATION ....................................................................................14  
  4.1 ROBOTIC ARM MODULE MODIFICATIONS ......................................................14  
  4.2 NAVIGATION MODULE MODIFICATIONS .......................................................18  
    4.2.1 Path Planning Algorithm Modifications ......................................................18  
    4.2.2 Additional Feature – Mapping .....................................................................19  
    4.2.3 Movement Modifications and Roaming ......................................................20  
  4.3 IMAGE MODULE MODIFICATIONS ....................................................................22  
  4.4 INTERFACING MODIFICATIONS .......................................................................23  
  4.5 INTEGRATION PROBLEMS ................................................................................25  
5.0 TEST AND EVALUATION .........................................................................................27  
  5.1 COMPONENT TESTING .......................................................................................27  
  5.2 MODULE TESTING .............................................................................................31  
  5.3 SYSTEM TESTING ...............................................................................................33  
6.0 TIME AND COST CONSIDERATIONS ....................................................................34
CONTENTS (Continued)

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN .................................................. 35
8.0 RECOMMENDATIONS .................................................................................. 36
9.0 CONCLUSIONS ......................................................................................... 38
REFERENCES .................................................................................................... 40
APPENDIX A – ADDITIONAL FLOW CHARTS AND BLOCK DIAGRAMS ...... A-1
APPENDIX B – GANTT CHART ......................................................................... B-1
APPENDIX C – COST TABLE ............................................................................ C-1
FIGURES

1  High level block diagram........................................................................................................5
2  Design of robotic arm (top view)..............................................................................................7
3  Design of robotic arm (side view)...........................................................................................8
4  Joint solution for the robotic arm with link 1 and link 3 parallel..........................................8
5  Dijkstra’s algorithm ...............................................................................................................10
6  A* algorithm ........................................................................................................................11
7  Modified trajectory plan .........................................................................................................15
8  Plane for orientation of cans ..................................................................................................15
9  Joint 4 at -180°.......................................................................................................................16
10 Difference between desired angle and actual angle due to torque ......................................16
11 New resting position of robot arm .........................................................................................18
12 A sample virtual map .............................................................................................................20
13 A sample virtual map with solution .......................................................................................20
14 Original map with target found (left), goal lost due to movement (middle), pseudo-goal enabled (right). ........................................................................................................................................21
15 Blob detection input and output, including GUI feedback of blob data ................................22
16 Color detection output ...........................................................................................................26
17 IR sensor output detecting paper ............................................................................................30
18 IR sensor output detecting aluminum can ...............................................................................30
EXECUTIVE SUMMARY

The goal of our project was to build a prototype automatic garbage collection robot (AGCR) for household use that featured more robust collecting abilities than currently available commercial cleaning robots. To realize this goal, we designed a robot that was capable of targeting specific objects, collecting them with a robotic arm, and depositing them in a receptacle. The project consists of four main modules: robotic arm, navigation, image processing, and interfacing.

The design of the robot is based on the iRobot Create, and uses a 1 GHz x86 onboard computer as well as a Freescale DP512 Microcontroller. The onboard computer runs the main program, which is written in C++ using the Linux-based Player robot interface, and communicates with the DP512, the Create base, and a webcam. The robotic arm uses five servos to act as joints in the arm, one servo for the gripper, and one short range IR sensor to confirm the successful pickup of an object. The navigation component also uses a servo for camera direction control and two IR sensors, one far range and one close range, to detect the distance of obstacles. Drivers for controlling servos and IR sensors are written in C and run on the DP512 microcontroller. For the software, the arm control used trajectory planning based on forcing the arm to assume a certain general configuration. The joints in the arm were moved in a predetermined order to prevent collisions and also featured provisions for smooth movement. The navigation used camera data to generate a virtual map and applied the D* search algorithm to find a path. The image processing uses a combination of the SURF algorithm for precise object detection and a color filter for sensing the presence of general objects.

When implementing the design, the high level design was unchanged, and the only hardware change was choosing a larger battery to accommodate for power consumption. However, several changes were made in the software, including the arm control, trajectory planning, navigation algorithm, and image processing. For arm control, we introduced timed waits for smoother movement, and we changed the general configuration for trajectory planning. We changed the navigation algorithm to the A* algorithm. In addition, open-source color detection was used instead of our own image processing. During implementation, we also considered potential safety and ethical issues, such as bodily harm to people and shock hazards, as well as standards involved with our project like the Consumer Product Safety Act and industrial robot standards.

We tested our project starting at the hardware components followed by individual module tests and finally system-level tests. Although all components passed, very few modules were able to pass our original tests due to impossibly stringent test criteria. With revised testing criteria, all module level tests except image processing passed. Finally, our overall project was successful because the robot passed system level tests by completing its tasks with a success rate of 78%.

Despite an initially well planned schedule, our project suffered from significant time management and planning problems. To other groups seeking to do similar projects, we would recommend concentrating on the critical path portion of the project and starting implementation as early as possible. We would also recommend stronger servos and a robotic platform with more reliable encoders. We have also considered plans for future work of this project to fully integrate our own image processing. We also have plans to be able to map farther distances away from the robot, essentially increasing the robot's range.
1.0 INTRODUCTION

This report details the design solution, final implementation, and testing results of the Automated Garbage Collection Robot (AGCR). The AGCR is a prototype robot that can automatically find and target specific objects, navigate without colliding into obstacles, retrieve the target with a robotic arm, and deposit the object in a receptacle. Because of the nature of our project goals, our project was split into four main modules: robotic arm, navigation, image processing, and interfacing. Our team consists of four members, Ray Chen, Scott Chu, Bao Nguyen, and Kevin Tan, each in charge of a module. Ray was in charge of the navigation module that will determine how to search for and navigate to targets while avoiding obstacles, as well as returning to the receptacle. Scott was in charge of the design, creation, and software control of the robotic arm. Kevin was in charge of the image processing module that would determine how targets were differentiated from obstacles using information from images. Bao was in charge of interfacing all of the external hardware to the robot’s onboard computer. Our team is sponsored by Professor Al Bovik, who provided the team with insight and guidance in his area of expertise, image and video processing. In addition, we would like to acknowledge the UT Laboratory for Informatics, Networks, and Communications (LINC) and Nicholas Paine for support and loan of the iRobot Create platform.

This report discusses why our robot is different from currently commercial options and discusses the design problems, requirements, and constraints that we faced and how they affected our system-level design solutions consisting of an iRobot Create platform, a 1 GHz x86 onboard computer running Linux, a DP512 microprocessor, a webcam, two IR sensors, and a robotic arm. The report then delves further into module level design solutions including the hardware decisions and software design of the robotic arm, algorithm choice for navigation, choice of image processing and object detection procedures and algorithms, and selection of interfacing data flow. Afterwards, the report delves into the final implementation of the modules as well as the problems and challenges that arose during module implementation and integration. Then, we discuss the testing that was conducted on the component, module, and system levels used to determine the success of our project. We also discuss why our schedule met delays, how our actual costs differed from our estimated costs, and the safety and ethical considerations involving
our project. Finally, this report concludes with recommendations for future projects that are similar to the AGCR, as well as plans for expanding the current design.

## 2.0 DESIGN PROBLEM STATEMENT

Our goal in this project was to create a prototype automatic household cleaning robot capable of targeting specific objects, collecting them with a robotic arm, and depositing them in a receptacle. Current commercial cleaning robots, like the Roomba, are only capable of cleaning dust and small particles off the floor using random movement. We aimed to make the robot more intelligent in its cleaning procedure and make it capable of picking up larger pieces of garbage.

To solve this problem, we determined the general components necessary for the robot to be able to complete its goals. A robotic arm is necessary to collect and deposit garbage of considerable size. Navigation was also needed to move the robot to search for and collect garbage as well as to avoid obstacles and return to the receptacle. Image processing is also necessary to differentiate obstacles from targets. Finally, interfacing was also required in order to seamlessly integrate these components together.

Additionally, certain constraints were placed on our project to prevent the system from becoming unrealistic. A major concern was the strength and control of the robotic arm. The actuators that act as the joints of the robotic arm must be able to detect the position of the actuator with accuracy in precision. Accurate actuators are necessary in order to move the robotic arm as planned so we sought actuators that could find its position 100% of the time with no resistance. In order for the robot to search for targets a distance away from the robot, the image processing of the robot must be able to recognize objects at least 4 feet away. In addition, the robot is required to operate in real-time or semi-real-time. This requirement dictates that the robot must not stop and hang up frequently. We assumed that the bottleneck of processing speed would be on the image processing because image processing requires significant amount of processing power. Therefore, we required that the image processing must be capable of running on the robot with a processing speed of at least 5 images processed every 4 seconds.
In addition, constraints were placed on the operating environment to allow for consistent and controlled tests. Adequate and stable lighting had to be present in the environment to produce consistent, clear images for the camera to capture. In addition, the target objects were limited to soda cans that must be empty or closed; else during arm retraction the can would spill its contents onto the robot itself. Obstacles also need be a certain height so that patterns on the ground could be ignored. For our prototyping purposes, we created stages that were of limited size and with an accessible target and receptacle to allow the AGCR to complete its task without running out of batteries or indefinite roaming.

3.0 DESIGN PROBLEM SOLUTION

In this section, we detail our procedure of choosing our current design solution over various design alternatives. First, we look at possible designs and hardware components, focusing several factors including hardware limitations, software complexity, and synergy\(^1\) between components. Next, after we describe the final robot design and its major design alternatives, we discuss the design solutions of the four individual modules – the robotic arm module, navigation module, image processing module, and interfacing module.

One of the possible design solutions that we considered for object detection and recognition was using an infrared (IR) camera instead of a standard color camera. IR cameras function similarly to color cameras but only capture light in the infrared spectrum, not the visible color spectrum. In addition, an IR diode is necessary to send IR light for reflection (to effectively capture visual information). The image obtained by an IR camera would not be affected by the lighting, allowing the robot to operate in light-deprived environments. Since IR cameras do not distinguish different colors, objects with designs would be easier to identify. However, being unable to sense colors may make it difficult to distinguish between objects that share a similar shapes and colors, such as cans of soda, ultimately being a limitation of the hardware. As a team, we decided that retaining the color of the environment would be beneficial to distinguishing objects, keeping the object recognition robust. In addition, an experiment by Bojan Kuljić et al.

\(^{1}\) Synergy - The interaction of two or more agents or forces so that their combined effect is greater than the sum of their individual effects.
stated that using a color camera was found to give "very good results" in the right conditions, which we believe to be acceptable for our purposes [1].

Another possible design solution the team discussed was using two cameras in stereo to extract precise distance data from the stereoscopic relationship of the two cameras. According to an IEEE document detailing the use of a hand-eye coordination system for a robotic arm, stereo vision requires the near perfect calibration of two cameras [2]. In addition, the calculations to solve for object depth required the use of a pseudo-inverse least squares solution along with a Hough transform [2]. After reviewing these details, the team decided that the complexity and scale of the calculations for implementing stereo vision were unnecessarily complex for our task.

Finally, we considered two design solutions for the mobile platform: the iRobot Create and the Proteus. The iRobot Create is a stripped version of the popular Roomba vacuuming robot. This platform is quickly gaining popularity in the robotic research community because of its simplicity, small size, and expandability. The Proteus, on the other hand, is a modular robot developed by the University of Texas at Austin that features high load capacity, speeds up to 8 mph, and compatibility with major off-the-shelf components [3]. The main specifications that we used to compare the two platforms were the payload capacity, the height, and the turning radius – the cost was not an issue as both options were available for loan from the LINC lab. Because both options met the minimum payload capacity (15 lbs) for us to mount the other components, the iRobot Create seemed to be more advantageous; the lower height creates a lower center of gravity and better stability. In addition, the Create is able to turn about its center unlike the Proteus, which maneuvers like a car. This zero turning radius establishes an extra degree of freedom that would require an extra rotational pivot joint on the Proteus [3].

Our complete ideal design solution is a design based on the iRobot iCreate (Roomba Platform) and a robotic arm that contains five degrees of freedom - meaning the arm has five joints in the arm to allow for variable movement. The design uses a single color camera in order to capture an image of the AGCR’s immediate surroundings. In addition, an infrared (IR) sensor determines the distances of objects from the AGCR by directing an IR sensor at an object when the camera recognizes that an object is present. The AGCR will navigate within a given perimeter and avoid
obstacles using the combination of camera and IR sensor data. While navigating inside the perimeter, the AGCR will also actively search for its target object (cans). It uses object recognition to approach an object as a target or avoid it as an obstacle. After the AGCR comes within a certain distance of the target, the robotic arm will reach out and grab the target using a combination of visual information from the camera and a close range IR sensor mounted near the hand of the robotic arm. After the robotic arm secures the object, the AGCR will navigate back to its home position to deposit the object and continue roaming within the perimeter. A visual representation of this information is shown in the block diagram in Figure 1 below.

![Figure 1. High level block diagram](image)

Our chosen solution was the best design decision based on our selection criteria described in the previous section. Our usage of a color webcam and an IR sensor allows for robust object recognition in well-lit conditions and less complexity than the stereoscopic vision option. This combination will be sufficient to recognize objects and determine their distance. In addition, our choice of the iRobot Create platform manages to create synergy with the robotic arm due to the
movement style of the robot to lower the complexity required from the robotic arm. Next, we will cover the detailed design solution of each module that allows the AGCR to carry out its task.

3.1 ROBOTIC ARM MODULE
During the design of the robotic arm, we considered several design problems that ranged from the choice of mechanical hardware components to the design of the arm and the associated trajectory planning and arm control.

3.1.1 Hardware Components
One of the hardware decisions that we considered was the robotic arm's components and design. The arm's components would need to include components for the joints as well as the links of the arm. First, we considered possible joint mechanisms. As different configurations of the arm produce different amounts of torque, the joint mechanisms must move variable amounts of torque accurately and precisely. The two possible solutions that we discovered were to use either servos or DC motors to act as joints of the robotic arm. DC motors can act as powerful and cheap actuators for the robotic arm that increases mechanical power with an increase of DC voltage. However, these DC motors lack feedback to determine how far the motor has actually turned. Due to this lack of feedback, the joints will have difficulty determining how to reach specific positions or hold positions. Servos are a more expensive option that allow for movement to a specific position based on the length of a pulse signal that is sent every 20 ms. Servos can be controlled with precision to rotate to certain points and hold that position, which provides much more precision and accuracy than DC motors. However, most servos can only rotate a maximum of 180° with mechanical hard stops at 0° and 180°, though it is possible to modify servos so that they are continuous and can rotate for 360°. Continuous servos become similar to DC motors with lower precision and accuracy. Despite the cost effectiveness of the DC motors, we ended up designing our robotic arm with servos due to their higher accuracy and precision.

Next, we considered possible materials for the links of our robotic arm. We sought a light-weight material that would not bend due to the stress of the arm. The two materials we considered for this were carbon fiber and aluminum. Between the two materials, carbon fiber is lighter and stiffer, but more expensive, while aluminum is slightly heavier and more malleable, but
significantly cheaper. We chose aluminum for a cheaper material that would still be able to support the weight of the arm.

Finally, the problem of retrieval confirmation was solved by using a short range IR sensor to detect whether or not the gripper was empty after it closed. Other possible solutions included a button that would only be touched when a can was inside the gripper. However, buttons have a high probability of false negatives due to dents in the can. In addition, buttons do not allow for easy expansion capabilities if the robot were to pick up other items. In comparison, a short ranged IR sensor will be able to detect an object in front of it and would be simple to integrate with the robot due to other modules requiring IR sensors.

3.1.2 Robotic Arm Design
Since the arm has already been determined to have five joints, the arrangement of the joints must maximize arm range with respect to angle and distance. With these design goals in mind, the design of the arm can be seen in Figure 2 below and Figure 3 on the next page. We define the position of the arm in these figures as being 0° for every joint.

![Figure 2. Design of robotic arm (top view)]
In this design, joints 1, 2, and 3 control the distance that the gripper reaches. Joint 4 controls how the arm can pick up objects with different orientations by moving the gripper to different angles of orientation. Joint 5 would then adjust for minor distances and finalize how approaches should be made. After the arm was designed, the issues of trajectory planning with unique arm positions and arm control needed to be solved.

3.1.2.1 Trajectory Planning
For a can at any given distance and orientation, there could be multiple trajectories (configurations) of the arm that could pick up the object. In order to reduce the possible trajectories of the arm, we created a single initial trajectory that the arm employs. Next, we looked at the problem of which configuration the arm should seek. We determined that the arm should assume a position similar to Figure 4 below.

Figure 3. Design of robotic arm (side view)

Figure 4. Joint solution for robotic arm with link 1 and link 3 parallel
This configuration of the arm allowed links 1 and 3 to be parallel to the ground by making joint 1 0° and the angle of joint 3 the inverse of joint 2. Then, link 2 will determine the distance and height that the arm reaches. We determined this configuration by considering that joints 1, 2, and 3 are parallel, meaning that these joints move the arm along the same plane.

In addition, when link 3 is parallel to the ground, link 4 is the sole determinant of both the orientation of the gripper and the distance of the object from the center of the arm. For this configuration, having link 4 determine the orientation of the gripper was of greater importance, since the Roomba base could turn to compensate for distance away from the center of the arm. Finally, joint 5 determines how the gripper approaches the object, and can compensate for varying heights due to the height of the arm varying with the distance of the object.

3.1.2.2 Arm Control
Another design problem we faced was how to handle arm control. This problem consisted of a number of parts, including preventing the arm from colliding with the robot and itself and producing smooth movement for the arm. In order to prevent the links of the arm from colliding with itself, we found specific waypoints and a joint movement order. The waypoint that was used when the arm was fully retracted consisted of moving joint 1 to 90°, joint 2 to 135°, joint 3 to -135°, joint 4 to 180°, and joint 5 to -90°. This waypoint allows for the most arm servos to be turned to an idling state for having the heavier joints of the arm resting on the robot. In order to extend the arm without collisions, the arm would be required to unravel by moving joint 1 first, then joints 2 and 3 simultaneously, followed by joint 4 and leaving joint 5 to be positioned last. After the object is picked up, the arm should return to the fully retracted waypoint by moving the joints in the opposite order to when the arm was extended.

We sought to make smooth arm movements by slowly increasing or decreasing each joint's output power. Since we did not have direct control over the servo's output power, we systematically adjusted joint positions to change the amount of power outputted. This technique is based on the knowledge that a servo’s output power increases with the difference between the servo's current and desired position.
3.2 NAVIGATION MODULE
The navigation module is responsible for moving the robot towards the appropriate object while avoiding obstacles, searching the area for targets, positioning the robot within the appropriate ranges for the robotic arm, and returning to a bin for proper garbage disposal. Initially, the only item we considered for this module was the search algorithm. This algorithm should analyze all possible paths and efficiently find the shortest path. We researched several path planning algorithms for the navigation module, including Dijkstra’s algorithm, A*, and D*.

Dijkstra’s algorithm provides a simple solution for path planning and obstacle avoidance by searching in all directions until it finds a solution path to the goal [4]. This can be seen in Figure 5 below. The red square represents the starting position while the dark blue represents the goal. The highlighted area is the area that the algorithm had to search before a solution was found. As we can see, it is relatively inefficient since it searches a large amount of sectors before finding the goal.

![Figure 5. Dijkstra's algorithm [4].](image)

In order to prevent the waste of limited processing power and memory, we also considered the popular A* algorithm which uses a more directed search technique. The A* algorithm estimates the distance between a sector and the goal, allowing it to single out the solutions which would
provide shorter paths and ignore solutions going the opposite direction. This is shown in Figure 6 below. Using the same starting points as the previous figure, we can see that this algorithm is much more efficient. Despite its robustness, the A* method can still be unreliable; when the map environment is unknown or if a goal does not exist, the algorithm would fail [4].

The D* algorithm, a variation of A*, builds upon the A* algorithm by eliminating this weakness; this is because D* technique includes map learning capabilities [5]. The D* algorithm's map learning feature not only creates a virtual map but also incorporates means for intelligent roaming. Since our robot will be operating in an unknown environment, the D* algorithm should be able to completely suit our navigation needs.

Once the AGCR has reached its destination, it will call a function to allow the arm module to perform its function. The arm module will then return a flag once it has completed its tasks. Once the completion flag is received, the robot will rotate towards the general direction of the drop-off receptacle. The AGCR will repeat the path finding process, except this time representing the bin as the goal and red cans as obstacles.
3.3 IMAGE MODULE
The image module must correctly identify obstacles and targets using a form of color detection combined with keypoint object detection. When designing the image processing, we considered the robustness of our algorithms and processing time required since we had constraint of limited processing power on the 1 GHz x86 board. The color detection would use custom image processing techniques and the object detection would use David Lowe’s scale-invariant feature transform (SIFT) algorithm, which returns keypoints that resemble patterns that the program has been trained to look for [6]. SIFT was chosen due to the robustness of the algorithm in its ability to recognize patterns through changes in scale, orientation, and slight occlusion. Both the color detection and SIFT algorithms would run on every image taken and look for similarities in the results. For instance, if red was detected in the image and multiple keypoints were matched using SIFT in the bottom-right area of the image, the program would return data relating a target in the bottom-right. This minimizes false positives in the results by checking multiple attributes, since we felt that using only one method was insufficient.

Using a color detection program, the robot could identify color, approximate size, and location. Color can be identified from an image as long as the image contains red-green-blue (RGB) or some other form of color data, such as YUV\(^2\). By only looking for a specific range of colors within the image, we are able to single out objects that contain the color of our target. The size can be determined by the number of pixels that represent the object along with distance data from the IR. Finally, the location of the object is determined by the position of the target object in the image along with distance data from the IR. However, we feel that this information is not enough to single out a target from an obstacle. Therefore, we also sought to identify shapes and patterns within the image. Identifying the shape is more complex and requires a form of keypoint detection. By being able to identify shapes, we can identify distinctly unique attributes such as logos. This eliminates the possibility of having an obstacle that is red be mistaken for a target that is red.

\(^2\) YUV is also a color space that takes human perception into consideration. Colors are defined in luma, or brightness (Y) and chrominance (U and V). YUV and RGB are directly related and can be derived from each other through linear transforms or matrix algebra.
The object detection method was always planned to be a modification of an open source program. David Lowe’s SIFT program was readily accessible in MATLAB format with documentation. The team intended to utilize this program by customizing the inputs and outputs for the AGCR. The input image was to be modified through a combination of histogram flattening and morphological erode and dilate operations. These operations are image processing techniques which would pre-filter our image data to prepare it for the keypoint detection of SIFT. Histogram flattening was intended to increase the likelihood of SIFT finding keypoints due to higher contrasting edges to bring out patterns. Eroding and dilating the image was meant to eliminate noise and artifacts in the image that we wanted to filter out. Eliminating noise meant the removal of false positives due to the quality of the image taken from the webcam. The output would be customized to indicate results from the color detection program and object coordinates that would be relayed to the robotic arm and navigation modules.

Finally, we also considered the speed and amount of time and processing power that would be used by the image processing. In order to reduce the processing time, we considered analyzing only the top and side portions of the image under the assumption that objects would only be approached from certain angles. However, we found another method of reducing processing time by only running keypoint detection if a significant size target blob candidate was detected. This allowed for complete scans of every image, ensuring thoroughness, while also remaining efficient.

3.4 INTERFACING
The navigation and robotic arm modules require usage of both the DP512 microcontroller and the processing power of the on-board computer. Originally, the team planned to utilize the serial connection between the x86 and DP512 to write custom code on both sides to send and receive data. Since Bao had not previously worked with the Player environment nor the iRobot Create platform, there was no specific implementation plan at the beginning of the semester. However, the team defined a set of requirements – the main program on the x86 needed the ability to poll the IR sensors and control the servos that were connected to the DP512.
4.0 DESIGN IMPLEMENTATION

With the ideal design solution in mind, the team began implementing the various modules of the AGCR. As expected, many design changes were needed due to the transition of the AGCR from design concept to real-life hardware and software. We cover the problems encountered and respective modifications in the robotic arm, navigation, image, and interfacing modules.

4.1 ROBOTIC ARM MODULE MODIFICATIONS

When implementing the components of the robotic arm, the mechanical construction of the arm was successfully completed with the planned design as seen in Figure 2 and 3 on pages 7 and 8, respectively. The arm was built using five servos as the joints of the arm and a servo to facilitate the gripping of the arm. Specifically, we used a Hitec HS-985MG servo for joint 1, an HS-645MG servo for joint 2, HS-322HD for joints 3, 4, and 5, and a HS-425BB for the gripper. The aluminum links and servos’ mounting pieces were drilled with matching mounting holes. We also used epoxy to affix the aluminum links to the non-mounting end of the servo.

However, trajectory planning and arm control both faced numerous changes due to unexpected problems. One of the earliest and most noticeable problems was that the arm was mounted lower than expected, and thus the planned configuration for trajectory planning was unable to work properly because the gripper dragged along the ground. In order to correct this issue, the configuration for trajectory planning would now angle link 2 above link 1 and leave link 3 pointing toward the ground as shown in Figure 7 on the next page. This new configuration prevented the arm from colliding with the robot and the ground. However, this configuration made picking up cans with orientations between 30° and 150° difficult according to the orientation framework on Figure 8 on the next page. This difficulty occurred because when joint 4 was greater than 30° and joint 2 was greater than 45°, the gripper could not reach the ground at a sufficient angle, causing the arm to be unable to pick up the object.
To accommodate these new problems, we simplified the area from which the robotic arm could grab and placed restrictions on the can's orientation. We divided the area into a left and right sector, and required the can to be oriented between $-30^\circ$ and $30^\circ$. 

Figure 7. Modified Trajectory Plan

Figure 8. Plane for orientation of cans
Another problem arose when joint 3 was frequently unable to lift the gripper to the desired height. This problem was caused by joint 3’s inability to handle the torque placed on it by the gripper. However, after significant testing, we determined that when joint 4 was positioned at -180°, as seen in Figure 9 below, joint 3 was capable of lifting the gripper from about 0° to about -15° as shown in Figure 10 below. When joint 4 is positioned at -180°, the weight of the gripper is moved closer to joint 3. Therefore, the torque on joint 3 is reduced, allowing it to have a greater effect on the gripper and lift the gripper up to -15°.

![Figure 9. Joint 4 at -180°](image)

After we determined the relationship between the desired and the actual angle of joint 3, we adjusted the calculations for the arm joints when the object was on the left side of the robot. We
increased the angle that the joint would attempt to move to, which worked because servos output more power when the desired goal of the servo is farther away. When the gripper was on the robot's right side, the torque on joint 3 became even greater, and the gripper would hang very loosely on the arm due to joint 3 failing to provide enough power. This problem was solved by turning joint 3 off to allow it to hang straight down so that at a minimum, the joint’s movements would be predictable. However, this solution reduced the area that the gripper could reach without sliding on the ground. The sum of these modifications resulted in a robotic arm that was unable to pick up cans that had an orientation that was not perpendicular with the arm. In addition, the arm would usually pick things up only from the left side, due to the larger area that the arm was capable of picking up objects on the left. As such, we compensated for this by altering the navigation algorithm to place greater emphasis on the left side by changing the method of approach.

A problem that came up with the retracted waypoint of the arm control is that the arm would often make contact with the camera when retracting. This contact caused the camera to move and lose its calibration, causing a number of problems associated with mapping and navigation. This problem was solved by changing the resting position of joint 5 to -180° and is shown in Figure 11 on the next page along with the other joints in their resting position. This position left the gripper standing straight up on the robot and took less space on top of the robot. However, the new position also required that joint 5 be powered constantly in order to keep the gripper standing upright, which caused more power consumption by the robotic arm than originally planned.

Another problem that affected the arm control was the granularity of the PWM signals that controlled each servo. Due to DP512 code for creating PWM signals at the duty cycle of 20 ms, the output PWM signals suffered from relatively low resolution for the servo. This low resolution caused a large granularity in movement where the minimum change of servo angle was 20° of rotation. This granularity made it impossible to implement the planned arm control of having the servos slowly ramp to appropriate values and caused the arm to have jerky, rough movement. The jerky movement caused observable swinging and instability in the arm. We fixed this problem by inserting timed waits after each individual movement so that the arm had time to
stabilize before moving to the next position. These wait statements caused a severe delay in execution time, but were necessary to stabilize the arm and maintain control.

![New resting position of robot arm](image)

**Figure 11. New resting position of robot arm**

### 4.2 NAVIGATION MODULE MODIFICATIONS

There were several issues that altered the original design of the navigation module during the prototype construction phase regarding the path finding algorithm, inaccurate encoders, roaming, image module inputs, and inconsistent IR sensors. The modifications and additions are separated into three major sections – path planning, mapping, and movement.

#### 4.2.1 Path Planning Algorithm Modifications

First, although we initially thought that D* would be the best algorithm to implement, we quickly found out that it was not suited for our particular application. The D* method added unnecessary complexity to our implementation, and the map learning feature was rendered useless by the inaccuracies of the rotary encoders, which are devices built-in to the mobile base that keep track of wheel position. The errors in the encoders would compound after each movement, which would result in an ill-defined and ultimately unusable map. As a result, we decided to apply the A* method supplemented with our own mapping and roaming techniques to supplement the algorithm's weaknesses.
We were able to find open-source code for A*, but we had to modify it in order suit our needs. For example, the open-source code did not make provisions to return an actual path list; it was only able to evaluate whether or not such a path exists. In order to fix this, we had to write our own program that generated the final path. This was done by initially calling the open-source function and checking if a path existed; if so, we traversed backwards from the goal, keeping track of the backwards path, and reversing the list. If the function determined that no path existed, we would then call our roaming program. Furthermore, there were also a few bugs in the open-source A* search algorithm. For example, the algorithm did not always find the shortest path. Upon examining the code, we realized that it was due to an error in sorting. However, we could not fix this problem without changing the main data structure in the code. This was because the open-source code used a type of data structure called a min-heap, a binary tree arranged in a way so that the parent node is always the smallest, meaning that the sorting logic was intertwined with the data structure itself. To fix this, we changed the main structure into an array and wrote our own sorting algorithm to gain more control over the data flow. These changes made the A* search algorithm much easier to integrate with the rest of the navigation program. A flow chart of the A* search algorithm is show in Figure 1 of Appendix A.

4.2.2 Additional Feature – Mapping
Now that we are no longer using D*, we have also lost the mapping and roaming features that the D* algorithm provided. As such, we have decided to implement our custom mapping program. This virtual map should represent the area in front of the robot as a number of sectors corresponding to specific predetermined areas relative to the robot. Each sector can be labeled with either "X", "0", "1", or "2" as seen in Figure 12 on the next page. The "X" represents the starting location, "0's" represent free space, "1's" represent obstacles, and "2's" represent the goals. In addition, each sector in the map will automatically be labeled based on the data from both the camera and the IR sensors. The IR sensors sweep the front of the robot and mark all objects within a 2 feet range as obstacles. Next, the image data from the camera is color filtered and the midpoint and corners of each color object would be correlated to the appropriate sector of the map. Using the corner coordinates of the object was necessary in order to account for varying object sizes. Bigger objects that would span multiple sections in the map would fill all associated sectors of the map.
After the map is successfully generated, it is used by the A* search algorithm to determine a list of movements. Figure 13 below shows how the search algorithm works when a target is found. The red sectors in the map indicate the sectors that the robot has chosen to move through. Although the virtual map seems very simple, the small 3 by 5 map worked very well for our application.

4.2.3 Movement Modifications and Roaming
Even with the A* implementation and custom map, the team confirmed that the rotary encoders were still not accurate enough, even for a simple 3 by 5 map. In order to remedy this situation, we altered the navigation module so that it would stop the robot after every foot of movement and remap its environment. This rendered the original movement list somewhat useless, as it would only be necessary to execute the first item in the list. On the other hand, this simplified the mapping algorithm; since the robot only moves one foot at a time before remapping, we can simply place greater emphasis on the accuracy of the mapping for the area two feet in front of the robot. Objects farther away would no longer have to be accurately represented as the map would be updated once the robot moves closer. Because of this, we were able to retain a simple 3 by 5 map while at the same time maintaining the mapping accuracy needed for robot movement.
However, remapping after each movement would also mean that there is an increased possibility that the AGCR would lose sight of the target. In order to compensate for this, we had to implement a "pseudo-goal" system. This system activates after the AGCR initially finds a goal and loses track of it after remapping. In order to implement this system we first added a border of obstacles to the virtual map, as shown in Figure 14 below. The navigation module was then modified to continuously keep track of where the most recent goal was. Once the pseudo-goal system activates, we simply add a fake goal to the border at a location determined by the most recent "real" goal. The pseudo-goal described here should be sufficient to guide the AGCR to the general direction of the real goal. The pseudo-goal system is deactivated when a real target is found through the camera data.

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 2 1 1 0 0 0 1 1 0 0 0 2 1 0 0 1 1
1 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1
1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 1
1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 0 1
1 1 X 1 1 1 1 X 1 1 1 1 X 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

**Figure 14. Original map with target found (left), goal lost due to movement (middle), pseudo-goal enabled (right).**

Special cases, such as if no goal was initially found or if no path exists, are handled by the roaming section of the navigation module. The first step the roaming algorithm will execute is to rotate the camera to check all sides of the robot for targets. When a target is found, the robot should rotate accordingly so that it is facing the target. If no targets are found, we would force the robot to go forward (after checking for obstacles in front). We also needed the robot to cover more area while simultaneously limiting the range. Therefore, we added code so that the robot would limit the roaming to a maximum of four feet in one direction before turning. A detailed flow chart of the entire navigation module can be found in Figure 2 of Appendix A.
4.3 IMAGE MODULE MODIFICATIONS

When implementing our image processing design, we used MATLAB to test object detection methods using the image processing toolbox [7]. We used histogram and morphologic filtering operations to create a color detection method that also modified the image to make attributes more distinct for SIFT to identify. In addition, we also attempted to mitigate noise caused by small blobs by filtering out objects using a size threshold mask to limit the blobs that were taken into consideration by the color detection. However, our color detection was fundamentally flawed because it assumed there was only one target or obstacle object in the image and made all of the pixels within the threshold values one giant blob. In order to remedy this problem, we used color blob detection instead of color detection. Color blob detection is different from the original color detection because it allows for the identification of multiple objects as opposed to assuming only one is present. Multiple objects are found by finding groups of similarly colored pixels of the desired color and separating them from other objects with the same color if they are too far apart. After experimenting with various algorithms, a row-by-row lineblob detection method was chosen due to its simplicity and reliability. This algorithm is precise, since it scans every pixel in an image, and accurate if the data image being read was a good representation of the colors we wanted to detect [8]. A flow chart of the color blob detection can be found in Figure 3 of Appendix A and Figure 15 below shows sample inputs and outputs for the color blob detection.

In addition, the outputs of our image processing were changed because other modules had begun testing using the Player Project's blobfinder. In order to facilitate integration, the image processing output was changed to the same outputs of blobfinder, which returns the pixel coordinates of the midpoint and corners of the blob, as well as size of each blob detected.

![Image](image_url)

**Figure 15.** Blob detection input and output, including GUI feedback of blob data
Additionally, we also chose to use a modified form of SIFT called speeded up robust features (SURF) instead of SIFT [9]. SURF is similar to SIFT since both identify keypoints and match them. However, while SURF can run up to three times faster, it also sacrifices some diligence in detecting keypoints. During testing and simulation, however, keypoint detection seemed impractical due to the required processing overhead as well as the resolution and clarity of images taken from the webcam. False positives and errant results proved too unreliable to have any consistent impact on finding the correct target objects. As such, we placed the completion of a working color blob detection algorithm at a higher priority than the keypoint detection method.

After the algorithms were written and tested in MATLAB, we attempted to convert the MATLAB code to C++. Conversion of the MATLAB code to C++ proved to be one of the most difficult and time consuming tasks. There is no clear and direct way to convert the code, especially since we used the image processing toolbox’s functions extensively. We originally hoped that using MATLAB’s automatic code converter would work, but it was a very limited tool that could not fulfill our image processing needs [10]. Next, we aimed to reverse engineer open source programs such as CMVision, but this was time consuming and was not the best solution to understanding how to write our own C++ program. Professor Bovik suggested that we use OpenCV, an open source computer vision library. OpenCV contains hundreds of samples and functions for users that assist in programming image processing programs in C, C++ and Python. After completing testing in Microsoft’s Visual Studio 2008 environment, we transferred the code to a Linux system using OpenCV [11].

However, due to poor planning, frequent changes to algorithms, and difficulties converting code to C++, we were unable to integrate the image processing with the robot by open house. Alternatively, we used CMVision in conjunction with the Player Project's blobfinder proxy because the output of the image processing module was based on blobfinder.

4.4 INTERFACING MODIFICATIONS
The interface between the x86 and DP512 was simply defined with no specific implementation for the AGCR as a way for the robotic arm and navigation modules to poll data from the IR sensors and control the servos connected to the DP512. As such, the code on both sides went
through heavy modification throughout the implementation process to increase efficiency and flexibility for the team.

The iRobot Create platform was previously interfaced to the DP512 from the LINC lab – therefore, there was a serial driver already available. The team decided to utilize this serial driver to save time so that other portions of the project could be worked on. On a high level, the x86 sends data to the DP512 in the following format: <begin><opcode><data><end>. The DP512 receives this packet, parses it out, and interprets the data according to the opcode. At first, functions were coded on to carry out one specific task. For example, the team wrote separate functions to extend and retract the arm. For other arm positions, the robotic arm module would send a list of servo angles to the DP512 in one single packet. The DP512 would take this set of servo angles and would move the servos in a specific order. However, during robotic arm module testing, since the order of servos was constantly changed to compensate for the weak servos, the team would have to constantly modify the DP512 code and re-program it. This was inconvenient for two reasons. First, the team had to restart the computer to move from Linux to Windows (Player environment of the x86 to Metrowerks Codewarrior for the DP512). Second, the team had to unplug/replug the serial cable from the x86 to an external laptop. To increase efficiency of testing and debug and reduce the risk of damaging the connector on the DP512, the team coded a more flexible function that allowed the robotic arm module to control individual servos in any order. The final communication between modules can be seen in the block diagram in Figure 4 of Appendix A. Next, each function of the interfacing module will be described separately.

As previously stated, the robot used three IR sensors – two mounted on a servo on the front of the iRobot Create, and one mounted on the gripper of the robotic arm. The proteus_get_sensors() function was modified on the x86 side that sent a command to the DP512 requesting data from the IR sensors. This distance in meters to be stored into a set of variables for use in the robotic arm and navigation modules. On the DP512 side, the calibration data (relating distance to the ADC output of the DP512) was integrated along with a linear interpolation function that would send IR data in distance across the serial interface to the x86. Proteus_camera() and proteus_frontir() allowed the navigation module to rotate the camera and the IR sensors mounted
on the front to a specified position. Proteus_opengripper() and proteus_closegripper() sent a direct signal to close and open the gripper of the arm. Last, proteus_servos() allowed the robotic arm module to control the servos connected to the DP512 with just the servo number and requested servo angle. Calibration that took the data relating the duty cycle of the PWM signal to the servo angle was integrated into the control code with a linear interpolation function on the DP512. A complete flow diagram of the DP512 interface can be seen in Figure 5 of Appendix A.

4.5 INTEGRATION PROBLEMS

There were also a number of problems that arose during our final integration stage. First, there were issues with camera placement. We originally planned to mount the camera at an elevated position at the center of the robot. With the right servo, this would give the camera more range and the ability to rotate a full 360 degrees. First, proportional 360 degree servos are incredibly rare; we were not able to obtain this particular part. Next, mounting the camera one foot above the top of the robot did not provide the proper range—we needed the camera to be able to see both right in front of the robot as well as farther away. Finally, we did not have enough room on the AGCR to mount the camera at the center of the robot since that would cause interference with the arm. We decided to mount the camera on far left, on top of a 180 degree rotation servo, and the arm on the far right. While the arm blocks the camera's view of the right side of the robot, we simply modified the roaming code to compensate for the camera's dead-zones by rotating the base of the robot.

Due to some delays in the completion of the image module, we had to use an open-source color detection algorithm (CMVision in conjunction with the Player Project's blobfinder proxy). This open-source color detection code was somewhat limited in its capabilities. For instance, we added a continuous border in order to prevent the robot from wandering into certain areas. Ideally, the color detection would return multiple objects, which would be fairly simple to insert on the virtual map. However, as seen in Figure 16 on the next page, the color filtering connects the border and returns it as a single object, ultimately causing the mapping program to label the open spaces as obstacles. The only way to work around this was to cut the border into smaller pieces with separation between each piece in order to force the color detection algorithm to return several objects.
The lack of a fully-functioning image module also meant that there was no way to detect edges and walls using the camera. We thought that this would cause no problems since the 2 IR sensors mounted in the front of the robot were more than enough to accommodate for the potential obstacles. That is, until we realized that the IR sensors could not properly read distances on shiny metal surfaces, such as aluminum cans, due to dispersion of the IR beam. This produced errors in the mapping algorithm because the IR sensor data would not match the image data, causing most red cans to be mapped as an obstacle. This problem was remedied by giving higher priority to the camera data.

There were also some additional modifications required in the navigation module to facilitate the operation of the arm. First, the sections of the map that marked the completion of the navigation module did not exactly match ranges possible for the arm. To solve this, the navigation algorithm, after successfully reaching its destination, would perform a final calibration that would shift the target object into the possible arm ranges. Next, the arm module also required distance data about the target object. Since the coordinate systems for the arm and the navigation
were different, we could not directly translate IR sensor data into useful distance data. We compensated for this by adding a function that converts the distance values from the navigation's coordinates to the arm module's coordinates. However, this function was quickly rendered useless after we found out that the IR sensors could not accurately read distances for aluminum cans. As the arm module still needs distance data, we decided to approximate the distances solely using data from the camera. We also added an additional communication flag from the arm module that alerted the navigation if a pick up attempt was unsuccessful. If this flag was set, the navigation would reposition the robot to allow for another pick up attempt.

Finally, we had to change our definition of the drop-off bin. Since we ended up implementing only color detection, searching for blue objects created a high amount of false positives. Most notably, the robot often detected blue jeans as the target. To fix this problem, we decided to use a rare combination of color – red inside blue. However, this change also caused some problems for the rest of the navigation code. For example, introducing extra red objects into the environment causes false positives when the robot is searching for red cans to pick up. Furthermore, the mapping system records red as an obstacle when searching for the drop-off bin. We compensated for these situations by programming more filters that take into account these special cases.

5.0 TEST AND EVALUATION

The team modularized the testing and evaluation of the Automated Garbage Collection Robot. First, we tested individual components and hardware. Next, we tested the functionality of each module – robotic arm, navigation, and image. Finally, once the modules were completed, we conducted complete system testing that tested the full functionality of the AGCR.

5.1 COMPONENT TESTING

Our Automated Garbage Collection Robot contains several off-the-shelf and pre-built components, including an iRobot Create base with on-board computer, a webcam, a microcontroller, IR sensors, servomechanisms, and aluminum beams.
The first component we tested was the iRobot Create platform. As stated previously, we borrowed this component from the UT Laboratory for Informatics, Networks, and Communications (LINC) with an on-board computer already integrated. Therefore, for this section, we will treat the iRobot Create and the on-board x86 computer as a single component and refer to it as the “navigation platform.” The navigation platform must succeed in three tests in order to pass the component testing stage. First, it must be able to run the Player Project software (which provides hardware device interacting) and run compiled C++ code. Since the on-board computer runs the Ubuntu 9.04 Linux operating system and has wireless capabilities, we were able to install the Player software and run programs through Secure Shell (SSH), which is a network protocol that allows external computers to communicate with the navigation platform. Next, the navigation platform must also be able to move accurately with a 15 pound load. We wrote simple programs to test the movement capabilities of the platform – forward and backward motion and dual direction rotation about its center. The platform moved effortlessly with the on-board x86 computer, the arm, and several peripherals such as the camera and IR sensors. Finally, the platform must be able to operate continuously for at least 30 minutes on battery power. This includes providing power to the on-board computer, microcontroller, and iRobot Create base, as well as supporting the potential 5A total current that the joints of the arm require. Our initial battery could not handle this current draw, so the LINC lab graciously loaned us a much bigger battery. Our tests with the upgraded battery showed that it could continuously power the entire robot for more than one hour, which far exceeds our specifications.

The next component was the webcam, with minimum requirements of 20 frames per second and a resolution of 640 by 480 pixels. Furthermore, the webcam must be able to communicate with the on-board computer and be compatible with the Player software. We chose the Logitech Quickcam Pro 4000 for this task because its specifications include a 1280 by 960 pixel resolution and a 30 fps frame rate [12]. This webcam was automatically detected by the on-board computer when connected through a USB interface and was able to be accessed by the Player software when using the “camerav4l” library. However, despite the automatic detection, there were also some compatibility issues with the Player software; the 640 by 480 pixel resolution could not be used because it caused errors in Player. Decreasing the resolution to 320 by 240 pixels fixed the problem, but negatively affected our object recognition capabilities. We were able to remedy this
by upgrading the camera to a Logitech Quickcam Pro 9000. The only thing we had to change 
modifying the library to the "camerauvc" library for the Player software. Overall, this particular 
model provided us with much better compatibility and allowed us to meet our requirements.

Another crucial component we tested was the Freescale 9S12DP512 microcontroller, which 
provides our robot with an interface for the servos and IR sensors. This means that the 
microcontroller is required to communicate with the Player software running on the on-board 
computer and simultaneously control the servos and read the IR sensors. We wrote code to test 
the outputs (pulse-width-modulation signals) to the servos and used an oscilloscope to measure 
the accuracy of the signal; although the measured signal varied up to an error of 2%, this was 
insignificant and should not affect the behavior of the servos. We also tested the serial interface 
between the on-board computer and the microcontroller. The serial ports successfully synced and 
we were able to send and receive data with 100% accuracy.

We also tested two types of IR sensors, long-range and short-range, by providing a +5 VDC 
power supply to the sensors and measuring the analog output using a voltmeter or an 
oscilloscope. We placed a piece of paper in front of the sensor and analyzed the output. Both 
long and short range sensors accurately outputted values that matched their specifications. As the 
output is non-linear and depends on the reflectance ability of the object it is detecting, we 
allowed for a 10% margin of error. However, when we tested with aluminum cans, the outputs of 
the IR sensors were much more inconsistent – the reflective nature of aluminum and round shape 
of the can negatively affected our results. This can be seen in Figure 17 and Figure 18 on the 
next page. Ultimately, the IR sensors did not meet our specifications as the margin of error was 
too significant for the IR sensors to be used for exact distance measurement. However, we still 
used the IR sensors in conjunction with webcam distance data for obstacle detection.
The aluminum beams were also tested. The aluminum beams must be rigid enough to hold the weight of the entire arm without bending. This weight included six servos (approx. 2 oz. each) and six aluminum beams with 20 inches of total length (4.5 oz. per foot) [13 ; 14 ; 15]. In order to test this, we bought five pieces of foot-long aluminum angle beams with 1 inch wide and 0.125 inch thick sides. We simulated an extended arm by temporarily fastening the ends together to make a long beam. Since each beam held up and did not bend, we were able to confirm that the aluminum beams met the rigidity requirements.

Figure 17. IR sensor output detecting paper

Figure 18. IR sensor output detecting aluminum can
Finally, we tested the servos. These servos act as the joints in the robotic arm, so they must be able to mount to the aluminum beams and have enough torque to handle the weight of their respective sections. Furthermore, the servos will need a minimum operating angle of 180 degrees. In order to test the holding power required, we obtained two servos with different torque specifications – a Hitec HS-322HD servo with 51 oz-in of torque and HS-645MG with 133 oz-in of torque [13; 14]. We mounted pieces of the foot-long aluminum beam onto the servos in order to test the capabilities, and found that the HS-322HD was only capable of holding one beam of aluminum while the HS-645MG maxed out at two beams. As the arm will have three major sections, both servos failed to meet the minimum requirements of the base joint. However, these servos do possess enough torque to be used for the higher joints in the arm. We acquired a more powerful servo, a Hitec HS-985MG, with 172 oz-in of torque for the base joint, and it was able to move the entire arm when it was in the retracted position. Although all the servos passed all the initial testing phase, we found that some of the servos begin to lose torque after extended periods of use. As mentioned before, we compensated for this problem by assuming that some of the joints would fail under certain conditions.

5.2 MODULE TESTING

The robotic arm, image, and navigation modules all required individual testing before complete integration on the system level. The robotic arm’s function is to pick up the target can based on a position and orientation calculated based on data from the image module. We mounted the arm and observed the success of the robotic arm in moving and grabbing the target. Although we originally planned to test this module by placing the cans at varying locations within the range of the arm, we ran into some problems when we realized that our distance and can orientation data were unreliable. Again, the IR sensors returned inconsistent data when attempting to read from the target aluminum can. Furthermore, due to the lack of a fully functioning image module, we had to depend solely on color filtering to determine the orientation of the can. This method proved to be highly unpredictable as too many environmental factors such as lighting and the section of varying colors in the design of the can affected our results. We initially specified that the arm would complete the test successfully if it picked up the object at least 95 times out of 100 trials; however, due to the limitations in orientation matching, we have lowered the standard to a minimum of 50 out of 100 successful trials. With the distance data computed by the webcam
and IR sensors, we have to assume the can is within the -30 to +30 degree orientation range. After several trials, we concluded that the arm was successfully able to pick up the object more than 60% of the time, ultimately exceeding our new standard.

The next module is the image module that consists of two parts – color blob detection and object identification. The color blob detection was tested with multiple images taken in an environment to simulate the open house setting, on a laboratory tile floor with cans and obstacles. If the program was able to correctly identify what the human eye recognized as green, blue, and red and return those as blobs with accurate size and coordinate values, the tests would be considered a success. These tests were simulated in a Visual Studio C/C++ compiler. Modifying the threshold values for the color detection was a matter of trial and error. Also, testing had to be done to eliminate false positives, such as shadows and splotches on the floor. Anything that was not meant to be an object should have returned nothing at all, no color read out or blob detected. Next, object identification using SURF was tested using similar images in the lab environment. SURF would be considered a success if it was able to find at least five matches within a designated blob. While some tests were successful, the object identification part was incapable of consistently returning correct matches and therefore failed. The results mostly depended on clarity of image, distance of object from robot, orientation of object, and lighting. Reflections on aluminum cans as well as curved surfaces made slight irregularities, which were difficult for the program to identify. Most test results from object detection research were based on simpler targets, such as a painting or design on a flat surface, like a cereal box, that is unlikely to be warped or skewed intensely. However, simplifying our test conditions any further seemed impractical.

The last module is the navigation module, which relied heavily on the image module. The navigation module must avoid obstacles while maintaining a path towards a target. We first began testing simple object-following functionality. This was done by placing a red object within five feet of the AGCR, and observing if the AGCR approached the object. The AGCR was consistently able to follow the object and stop when it was within the range threshold of 15 inches (10 out of 10 trials). The next test was to trial the obstacle avoidance system. We originally specified that our obstacles will be random, but the lack of a functioning image
module forced us to use green objects. The AGCR must completely avoid the obstacle and come within 15 inches of the target on all 10 trials to pass the test. From our testing results, the AGCR failed to meet these requirements – the AGCR was able to come within 15 inches of the target 9 out of 10 times and the robot was only able to avoid the obstacle completely 6 out of 10 times. However, in the instances that the AGCR failed to avoid the obstacle completely, the AGCR recognized the obstacle and attempted to avoid it; contact with obstacles, if any, were limited to a slight scrape on the sides of the robot.

5.3 SYSTEM TESTING
The first step of system testing was to check the integration of all modules. For example, there were some restrictions the robot must meet: the arm should not be moving while the robot is navigating to the next location, the robot should be limited to rotating about its axis while the arm is in operation, and the robot must begin the tasks as soon as it is powered on. From our observation during test trials, the AGCR passed all of the tests with the exception of last one. There were some issues with the startup script and we could not make the robot automatically begin the tasks immediately after power-on. This, however, did not greatly impact the overall functionally of our system.

After observing the integration of the modules, we moved on to final testing. The final testing stage was fairly simple. The robot should search for red cans, change directions to move towards the can when it is found, and stop within 15 inches of the can, all while avoiding obstacles in the process. The robot arm will then proceed to pick up the can, and the AGCR should navigate back to the starting position where it will drop the can inside the red-blue receptacle. The entire team will then assess this process through observation; no external equipment will be needed to measure the results. In order to determine the success of the system, we will count the number of times the robot is successfully able to pick up the can and drop it off in the receptacle and compare it to the number of times we executed the program. We originally planned to deem the project a success if there was less than 5% failure rate. However, due to the large amount of setbacks and modifications, we lowered our minimum overall success rate to 70%.
We repeated this system-level testing process approximately 200 times with randomly placed obstacles and red cans to ensure thorough testing. Despite the fact that we lacked a fully-functioning image module, these errors were mainly small problems related to calibration and were not major failures in operation. This was because we realized that there was a risk of failure in the image module and we made contingency plans that involved simplifying the testing environment to account for performance failures. Since the system did not meet our original specifications, we created a more controlled testing environment by altering the key objects in a way that allowed for more accurate object recognition. This included removing shadows for more consistent lighting, creating uniform green obstacles, and placing cans in a way so that the camera would be able to detect the most amount of red. With these constraints in place, our system picked up the can and dropped it off at the receptacle at a success rate of 78%, ultimately exceeding our minimum requirements.

6.0 TIME AND COST CONSIDERATIONS

Although various problems were encountered during implementation, the AGCR prototype was completed in time for a successful open house demo. In addition, the group stayed within the projected budget that was created at the beginning of the year. First and foremost, the code conversion problems and frequent algorithm changes of the critical image pushed the schedule at least, as can be seen with the red boxes in the Gantt Chart in Appendix B. However, one week after the milestone for early image processing was missed, the team began to use an open-source image processing program. Although the program had limited functionality and increased the complexity of the navigation module, it saved the team from losing more time. Next, other problems that pushed back the schedule included the unanticipated problem of powering the servos. So that no time was lost, the team researched and created the power circuit while temporarily utilizing the external power sources available in the ENS labs. In addition, as an alternative to buying a custom power connector and waiting for it to arrive, the team constructed a custom connector to interface the breadboard to the female Molex power connector that was available on the iRobot Create platform. Besides these two major problems, the original schedule accounted for unforeseen obstacles with extra time devoted to system integration and debug.
In terms of budget, the quantities of the IR sensors increased from two to three, and the number of servos required increased from six to eight. However, the team spent $337, which was well within the projected budget cost of $369. The cost savings was due to minimizing servo costs by adjusting arm configuration and optimizing the order of movement to reduce torque on certain joints. An itemized list of costs is available in Appendix C.

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN

During the design of the AGCR, we looked at integrating safety features for our prototyping purposes, and also at regulations that would need to be followed if this product was manufactured for public use. We attempted to integrate several safety features during our design and construction phases. For example, we chose the servos in the arm in a way that would only provide enough force to move the arm. This means that the joints and gripper do not have enough strength to cause injury; in fact, if too much force is exerted on the servos, they would automatically shut off. Next, we limited the movement speed of the robot. Even though the IR sensors and camera should prevent the AGCR from directly running into objects and people, the forward force that the robot exerts would be insufficient to cause safety concerns. Finally, we have also shielded most of the wiring with corrugated fiberboard, an insulating material. This is not only to protect the wires from external factors, but also to prevent potential shock hazards. Although these features may seem primitive, they were sufficient to provide use with enough elements to allow us to safely test our prototype.

If, in the future, the robot is manufactured and sold for consumer use, we must consider a few standards on which the product is required to conform. One set of applicable standards is the Consumer Product Safety Act, Public Law 92-573, 86 Stat. 1207 [16]. This document lists a variety of standards that all consumer products must follow. As our automated garbage collection robot can definitely be applied to consumer usage, we must abide by these requirements. The first major requirement is labeling. Our product must appropriately warn the user of operational safety hazards through the use of visual warning labels stated in English. This will include warning for electrical shock when charging the battery and a warning to keep clear of the robotic arm in order to prevent injury. Furthermore, our product must not be made of hazardous
materials such as lead-based paint, or small parts. Finally, the product must be certified through independent third-party testing. If any of these standards are not met, or are violated after the product has been delivered to the consumer, then we must stop all production with the risk of incurring civil and criminal penalties.

The Occupational Safety and Health Administration, an agency of the United States Department of Labor, also provides some design and training restrictions that may be applicable to our robot if the product is used in an industrial setting. These restrictions are documented in OSHA Instruction PUB 8-1.3, directive number STD 01-12-002 [17]. The document first elaborates on their definition of a robot: “A robot can have one or more arms which are interconnected sets of links and powered joints. Arms are comprised of manipulators which support or move wrists and end-effectors. An end-effector is an accessory tool specifically designed for attachment to a robot wrist to enable the robot to perform its intended task. Examples of end-effectors include grippers, spot-weld guns, and spray paint guns.” Next, [18] refers to the ANSI R15.06-1986 Standard [19] to specifically define an industrial robot as “that which includes industrial robots, end-effectors, and any equipment, devices and sensors required for the entire robot system to perform its tasks.” This OSHA document lists a number of methods of guarding against human errors, control errors, unauthorized access, mechanical hazards, and power sources. These include using physical barrier guards, awareness barrier devices, presence sensing devices, emergency robot braking, and audible and visible warning systems. Of course, although these standards are more directed towards industrial robots, we must still consider these safety precautions since our robot may ultimately be expanded to apply to industrial settings.

8.0 RECOMMENDATIONS

Although the final AGCR was a relative success, the team has created an outline for future improvement in terms of research and theory, implementation, and program management. On a high level, more research should have been done regarding the interfacing between modules so that a specific implementation plan could have been created. This would have allowed the team to avoid on-the-fly creation of code with limited functionality, leading to a greater efficiency of
testing and debugging. In addition, more time should have been spent researching mechanical design of the robotic arm, whose problems led to a limited functionality of the robot.

The bulk of the recommendations lie in the implementation phase of each module. With regards to the robotic arm module, the team would recommend using more powerful servos or DC motors with encoders as joints and use a PID controller to control the arm movements. Encoders are necessary to receive feedback about the position of the motor and the PID controller would be used to control the amount of power that a servo needed to output in order to reach a position. These recommendations decrease the risk of arm malfunction and failure as well as smoothen the movement of the arm. In addition, the mechanical design of the attachment of arm pieces to the servos should be improved to solve the problem described that after extended periods of testing, the screws and nuts attaching the aluminum beams to the plastic headpieces of the servos began to fall apart. Finally, a better solution should be created for confirmation of object retrieval due to the unreliability of the IR sensors. With regards to the navigation module, the team would recommend using a movement platform with higher performance encoders.

Finally, we cover recommendations in terms of program management of the project. Although the original schedule did account for an early implementation of the image module for use in the robotic arm and navigation modules, the team was unable to meet several milestones due to implementation difficulties. The team would recommend devoting more manpower to the image module – the critical path. In addition, very little time was given for the robot's wire management and improving the appearance of the robot. Therefore, more time should be reserved for aesthetic modification of the robot. With these recommendations, the team believes that the AGCR could be improved in terms of efficiency and success rate.

We have also planned for future work for the AGCR to consist mainly of adding all of the originally planned functionality to the robot and making it more robust by adding our own recommendations. This includes complete integration with the image processing module along with a more robust image processing module that can detect objects more efficiently and accurately by feature detection combined with edge detection for object recognition. Color filtering should also be improved to be adaptive. Additionally, we could increase the speed of the robot by increasing the accuracy of the mapping at farther distances and increasing the encoder
accuracy. One solution to increase the accuracy of the mapping would be to use a higher resolution camera so that details can be seen more clearly. These two modifications would allow the robot to travel farther distances before needing to remap the room. In addition, the current roaming algorithm is relatively simple, and could be expanded upon to make sure that the AGCR covers the entire area of the room in the most efficient way. Additional work on the robotic arm could also be done by replacing the joint servos with motors attached to encoders and creating a PID control loop to control the power output by the motor. This change to DC motors will make the arm movements more smooth and controlled, due to the ability to control the amount of power the DC motors output.

9.0 CONCLUSION

This report details the design and implementation of a prototype automated garbage collection robot that uses visual information to navigate and a robotic arm to collect and deposit cans. The AGCR was considered a success based on the system-level test results and a fully working product at open house. However, our system was unable to pass all module level tests and image processing was not ready by open house. Fortunately, we mitigated the damage that a failing module dealt by using open source image processing. Our project could have been more successful given more time, resources, and better project management. Specifically, obtaining a platform with better rotary encoders and fully implementing the image processing module to better distinguish targets from obstacles would have made a significant difference. Despite these setbacks, our robot was capable of completing its desired tasks and was able to successfully do so in 78% of our tests.

During the implementation of the AGCR, the team learned many things about program management and engineering. We learned that while creating a schedule and following it closely is difficult, it is essential to the completion of a project. We should have begun testing for each module according to the original schedule and relied less on the additional time allotted for troubleshooting. We also gained significant amounts of knowledge dealing with navigation algorithms, image processing algorithms, and robotic arm construction and control. We have also learned significant amounts about specific parts and how to use them. Through unexpected
results, we learned that IR sensors were unable to give consistent readings off aluminum cans due to the curved, reflective nature of the can. This can be expanded to apply to all curved and reflective surfaces as well. We also learned how servos function as well as how torque limitation affects the functionality of the servo.

Based on the things we have learned during implementation, we have offered recommendations for future groups that plan to work on a similar project. In addition, we also recognize future work that could be done on the ACGR, such as fully integrating the image processing module, making the image processing module more robust, taking larger navigation steps, and smoothing out the movement of the arm.
REFERENCES


APPENDIX A – ADDITIONAL FLOW CHARTS AND BLOCK DIAGRAMS
Figure 1. Navigation Module – A* Search Algorithm Flow Chart
Initiate variables and connect to robot and peripherals

Set map to all 0’s, assign 1’s at the sides of the robot

Use long and short IR sensors to check if an obstacle is in front

Use camera to check if an obstacle is in front

Add a border of 1’s to the map

Check if goal is close enough

Position robot for arm

Compute distance and orientation

Set bin as goal, change cans to obstacles

Pick up/drop object using arm control

Run A* algorithm

Turn 90 degrees to the left

Turn robot in corresponding direction

Does path exist?

Yes

No

Yes

No

Does real goal exist?

Yes

No

Traverse list all the way back to beginning

Arrived at goal?

Yes

No

Move accordingly to the next open node

Use camera to check left and right sides

Found?

No

Found?

No

Robot find a goal before?

Yes

No

Figure 2. Main Navigation Flow Diagram
Figure 3. Image Module – Color Detection Flow Diagram
proteus_get_sensors() Opcode: SENSORS Request IR Packet
proteus_camera() Opcode: CAMERA Data: Position
proteus_frontir() Opcode: FRONTIR Data: Position
proteus_opengripper() Opcode: GRIPPER
proteus_closegripper() Opcode: GRIPPER
proteus_servos() Opcode: SERVOS Data: Angle, Servo#

Figure 4. Interfacing Block Diagram
Figure 5. DP512 Interfacing Flow Diagram
APPENDIX B – GANTT CHART
# APPENDIX B: GANTT CHART

## Table 1: Gantt Chart

<table>
<thead>
<tr>
<th>Tasks</th>
<th>1/22/10</th>
<th>1/29/10</th>
<th>2/5/10</th>
<th>2/12/10</th>
<th>2/19/10</th>
<th>2/26/10</th>
<th>3/5/10</th>
<th>3/12/10</th>
<th>3/19/10</th>
<th>3/26/10</th>
<th>4/2/10</th>
<th>4/9/10</th>
<th>4/16/10</th>
<th>4/23/10</th>
<th>4/30/10</th>
<th>5/7/10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reports</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design Implementation Plan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing and Evaluation Plan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oral Progress Report 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Written Progress Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oral Progress Report 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open House Demonstration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Oral Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Written Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Part Accumulation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Servos - Test Parts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Parts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IR Sensors</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Aluminum Links</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Programming</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robotic Arm Module</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arm Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trajectory Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Navigation Module</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic Point to Point Navig</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obstacle Avoidance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Image Module</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Interfacing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Early Object Recognition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalized Object Recognition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Interfacing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Servo Drivers for DP512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR Sensor Drivers for DP512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serial Interfacing DP512 with Main Computer Drivers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DP512 Driver for the Main Computer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hardware Construction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robotic Arm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IR Sensor Mount and Pivot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Mount and Pivot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Integration and Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Original time blocks         |         |         |        |         |         |         |        |         |         |         |        |        |         |         |         |        |
| New time blocks              |         |         |        |         |         |         |        |         |         |         |        |        |         |         |         |        |
APPENDIX C – COST TABLE
### APPENDIX C: COST TABLE

#### Table 1: Cost Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Price Range</th>
<th>Quantity Needed (Original)</th>
<th>Extended Cost (Approx.)</th>
<th>Quantity Ordered</th>
<th>Actual Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR Sensor</td>
<td>$15</td>
<td>3</td>
<td>$45</td>
<td>4</td>
<td>$44</td>
</tr>
<tr>
<td>Webcam</td>
<td>$0 (Obtained)</td>
<td>1</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td>Mobile Platform</td>
<td>$0 (Obtained)</td>
<td>1</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td>CPU &amp; Motherboard</td>
<td>$0 (Obtained)</td>
<td>1</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td>Embedded Processor</td>
<td>$0 (Obtained)</td>
<td>1</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td>Standard Analog Servomechanisms</td>
<td>$50</td>
<td>6</td>
<td>$300</td>
<td>8</td>
<td>$278</td>
</tr>
<tr>
<td>Aluminum Tubing</td>
<td>$4 per foot</td>
<td>6</td>
<td>$24</td>
<td>7 feet</td>
<td>$15</td>
</tr>
<tr>
<td>Software: Player</td>
<td>$0 (GNU)</td>
<td>N/A</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td>Software: Matlab</td>
<td>$0 (LRC Labs)</td>
<td>N/A</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td>Powered Mechanical Tools</td>
<td>$0 (EE Machine Shop)</td>
<td>N/A</td>
<td>$0</td>
<td>N/A</td>
<td>$0</td>
</tr>
<tr>
<td><strong>TOTAL EST. COST:</strong></td>
<td></td>
<td><strong>$369</strong></td>
<td></td>
<td></td>
<td><strong>$337</strong></td>
</tr>
</tbody>
</table>