Wall-E Robot Final Report

Johnathon Schultz Peter Clain Jesse Miner

INTRODUCTION

Overview

Robots have always been an object of fascination in our society. They have been portrayed as humble servants of man as well as evil creations that rise to overthrow their masters. But where does this fascination come from? All robots share one thing in common at the root of their design and purpose - they can perform tasks in place of humans. Life is filled with many repetitive tasks, and if robots are able to perform those tasks, they can help to ease an overarching burden.

With that said, robots are optimal replacements for humans in a multitude of scenarios. As simple as it may seem, the primary action in many repetitive tasks is picking up objects and moving them to other locations. Be it picking up garbage from the floor, moving parts along an assembly line, or removing fallen debris, robots that can pick up and move objects will always be useful.

Project Description

With an optimistic attitude toward robots in mind, our group decided to create a robot that could do just this - pick up and collect objects. While the goals of our project were not as advanced as some of the aforementioned examples, we had hoped that our project would expand upon the ever-growing nature of robot servants.

Our robot was designed as a fetching robot. It was to come with a set of objects that it was designed to detect, and it would be able to collect these items from the environment (when they were placed at random). Using an arm mechanism to lift the objects from the floor, it could then place the objects in a container on its back.

The behavior of the robot was to be very similar to that of a dog playing fetch, with the key difference being its ability to fetch and carry multiple objects. Like a robot that collects garbage or debris, the robot would be predisposed to collect objects of a certain type. Within this type, the range of object attributes would be very narrow. If the object were a yellow ball, for example, the robot would only collect objects with the ball's predetermined size and color. The characteristics of the objects would depend on what could be detected reliably using the available sensors.

This limitation served to keep the project within the scope of the class. While we did need to include components not provided by the instructor, such as a camera, we had hoped to make the design of this robot simple yet elegant. The specific design goals we hoped to accomplish were as follows:

1. The robot can detect target objects for collection
2. The robot can pick up objects successfully, and hold at least 3 of them in its payload
3. (Optional) The robot can redistribute the objects randomly and begin searching for them again. Conversely, it could also approach the nearest human when it has collected 3 objects.

RELATED WORK

Three systems were found that had several design goals in common with our robot. They all had mechanisms to move around, find certain objects, and pick them up.

First, the movie Wall-E was an inspiration for this project. In the beginning of the movie, Wall-E goes around collecting interesting objects from the garbage. Although our robots personality was not intended to be quite so developed, the basic function is the same.

Another robot was found on the web that is very similar to our idea [1]. It comes with a few barrels and is able to collect them in the container on its back. It uses a laser sensor to detect the barrels, which are covered in reflective tape. It then grabs the barrels and lifts them onto its back with the large arm on the front (Figure 1).

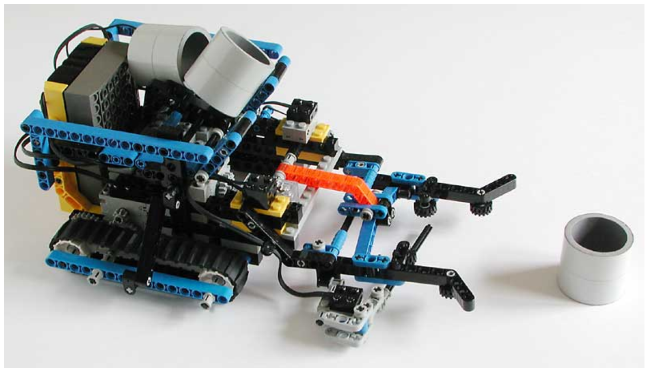

Figure 1. Barell Collector Robot

Collector Robots like ours have been designed as useful tools as well as toys [3]. For example, another Lego robot was found online that cleans up garbage. It uses two conveyor belts positioned at an angle to pick up objects. This method requires less precision than a grabber arm, but it may not provide enough grip for all possible objects. The garbage collection robot uses an ultrasonic sensor to find the nearest object and a light sensor to determine its color.

In designing our robot, these related systems were considered in the implementation of movement, object detection, and retrieval. Specifically, we envisioned our robots behavior to be something like the following:

1. Rotate while scanning the environment until it finds an object or it turns a full 360 degrees.
2. If an object is found, go to it, pick it up, and repeat cycle. If no object is found, either redistribute objects, go to the nearest person (wed need a way to see people), or simply turn off.

We noted that LEGOs were a popular building tool, and using them was one possibility for our project. We also found some less useful, but still mentionable related works at [6, 4, 5].

OVERALL DESIGN

Our robot had 4 main features to be designed. They were: one, a drive train to move our robot; two, a mechanism for lifting the objects off of the floor and into the container; three, a hopper for storing objects retrieved from the floor; and four, a camera for identifying objects to be picked up. These features then needed to be controlled using the Arduino microcontroller.

Over the course of development, our robot went through two main iterations of design. In each iteration, the key features were cut out of acrylic using the laser cutter. They were placed on the body of the robot, which was designed to hold the breadboard, the hopper, and any additional electrical components (servos, motors, etc.). Using a flat surface as the body, the features were arranged as follows:

- Movement: Motors and wheels were placed in the rear, underneath the chassis
- Mechanism: Attached to a servo that was mounted inside the hopper through a hole
- Hopper: Attached on top of the chassis in the rear
- Range-finder: Attached to the arm in a position to see balls that entered the forlift
- Camera: Attached to the top-front of the hopper

Both the protoype and the final version of the robot can be seen in Figure 2 and Figure 3 respectively.

Prototype

The first prototype of our robot made good progress towards our final vision. We assembled the body, hopper, and two arms out of laser-cut acrylic and tape. The body was a flat piece long enough to hold the breadboard on the front and

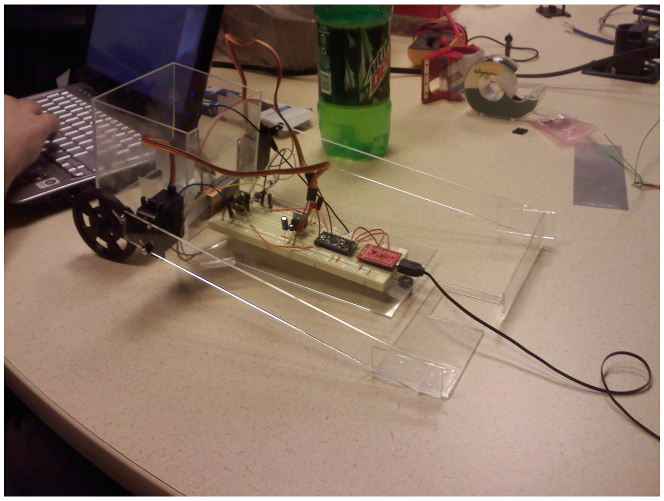

Figure 2. Initial Prototype

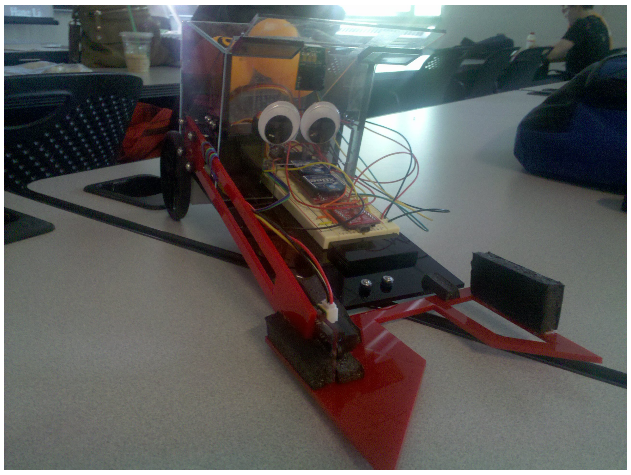

Figure 3. Final Prototype

the hopper on the back. The hopper was just a box with an open top so the ping pong balls could fall inside. We mounted two wheels controlled by motors on the back of the robot, underneath the hopper, and a track ball on the underside of the very front of the body.

The robot's two arms were each connected by screws to a servo. The servos were taped to the sides of the hopper, one on the right side and one on the left. The fork arm (the one that holds a ping pong ball) was on the right. We originally intended the left arm to have a sort of ”gate” that would come down to prevent the ball from falling out of the fork as it was lifted into the hopper, but we found that that was unnecessary. We ended up keeping the left arm because it was a convenient place to mount the range-finder, which detects when there is an object in the fork arm. The piece of acrylic that was meant to be the gate was used as a lip on the hopper to help ensure that the ping pong balls landed in the hopper.

The prototype accomplished our basic goals quite successfully. It was able to drive forward until it encountered a ping pong ball. When the range-finder detected the ball in the fork, the robot stopped if it was moving, lifted the fork arm to deposit the object in the hopper, and then lowered the arm back to ground level. The left arm never needed to move on the prototype.

Possibly the prototype's greatest limitation was that its wheels could not go backwards, so the robot could not turn or go in reverse. Thus, the ping pong ball had to be placed directly in front of the forklift. In addition, the prototype was physically tethered to a computer due to a requirement for a 5v power supply.

A video demoing the functionality of our initial prototype is available here: `http://www.youtube.com/watch?v=tuc2jwOD6Ng`.

Final Design

Both versions of the robot were very similar, with the final version adding any functionality that was absent in the prototype. The specific hardware and software decisions for the final version are evaluated in more detail below, while a general overview of our design is presented in Figure 4 and a diagram of our circuitry can be found in Figure 5.

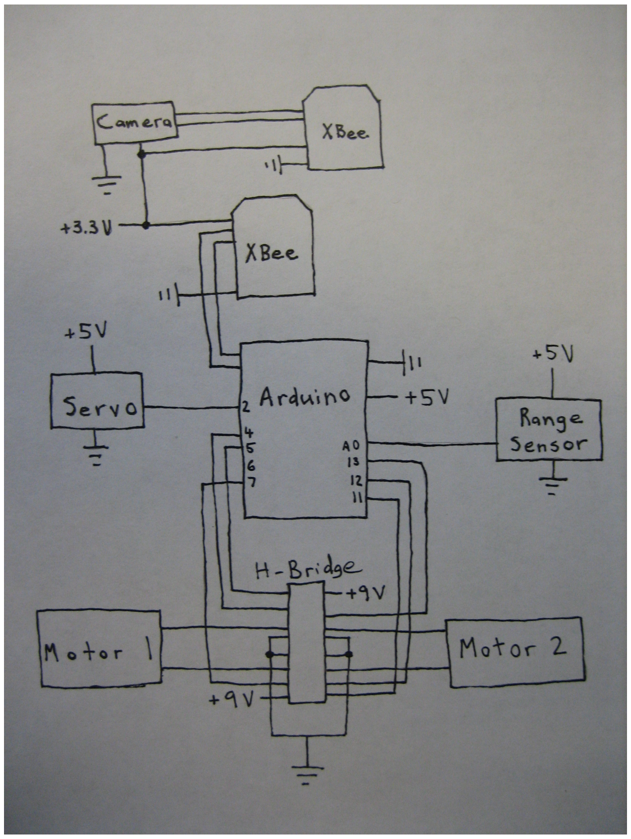

Figure 5. Robot Wiring Diagram

Hardware

The hardware portion of the robot changed visibly over the course of our development, but the overall design remained pretty much the same. Most of the changes were minor and designed to fix small problems or add a small bit of extra functionality, not completely change the capabilities of the robot.

The first thing we did the with the final version of the robot was to correct the three most obvious flaws in the prototype. We incorporated an H-bridge to allow the motors to move forward as well as backward. This gave the robot the ability to go in reverse as well as turn. The second thing we corrected was the fact that the robot had to be tethered to a computer in order to receive the 5v power it needed. We fixed this by adding an additional 9v battery and hooking it up to the raw input of the USB converter. Finally the third problem we corrected was removing the vestigial left arm. The range-finder was moved from the left arm to the right arm.

Initial testing on the prototype also showed that the arm mechanism was a little shaky and not very stable. Due to the difficulty of mounting the forklift to the arm at the proper angle, we decided that the best way to deal with the stability issues was to make the arm lighter and shorter as well as actually screwing the servo into the hopper. In order to shorten the overall chassis the hopper was modified to allow the breadboard to partially fit inside of it. This freed up extra room on the front of the robot and allowed us to make the entire chassis smaller. To do this, we cut a hole in the front of the hopper as well as another piece of acrylic to act as a floor for the hopper. We also stored the two 9v batteries that we were using inside the hopper, underneath this new floor we had constructed.

Over time, we noticed a problem with the arm where it would become misaligned with the floor. The arm would start scraping along the floor where previously it would not. When the arm was dragging along the floor, either the motors would be unable to turn the robot or the arm would pop off of the servo. To fix this, we added wedges of foam to the top of the forklift that would hit the front of the chassis as the arm came down. This was mostly successful in stopping the arm from going below a set minimum height. On certain terrain, such as carpet, this still became an issue periodically. However, this partial problem was solved by implementing a "carpet" setting for the motors, where they would drive with greater force than usual.

We were also worried about accuracy with the forklift, especially considering the problems we were having with the camera (more on this later). Thus, for the final version, we had the arm re-cut using some scrap light-weight red acrylic that we found. This new version of the arm had an angled front, with the goal being to push balls inwards towards the forklift. This worked surprisingly well, as most balls that hit the angled rolled right in.

Software

The software side of our robot changed substantially over the course of development. This was mostly due to the many many problems we encountered with the camera we were attempting to use. We initially imagined a much more com-

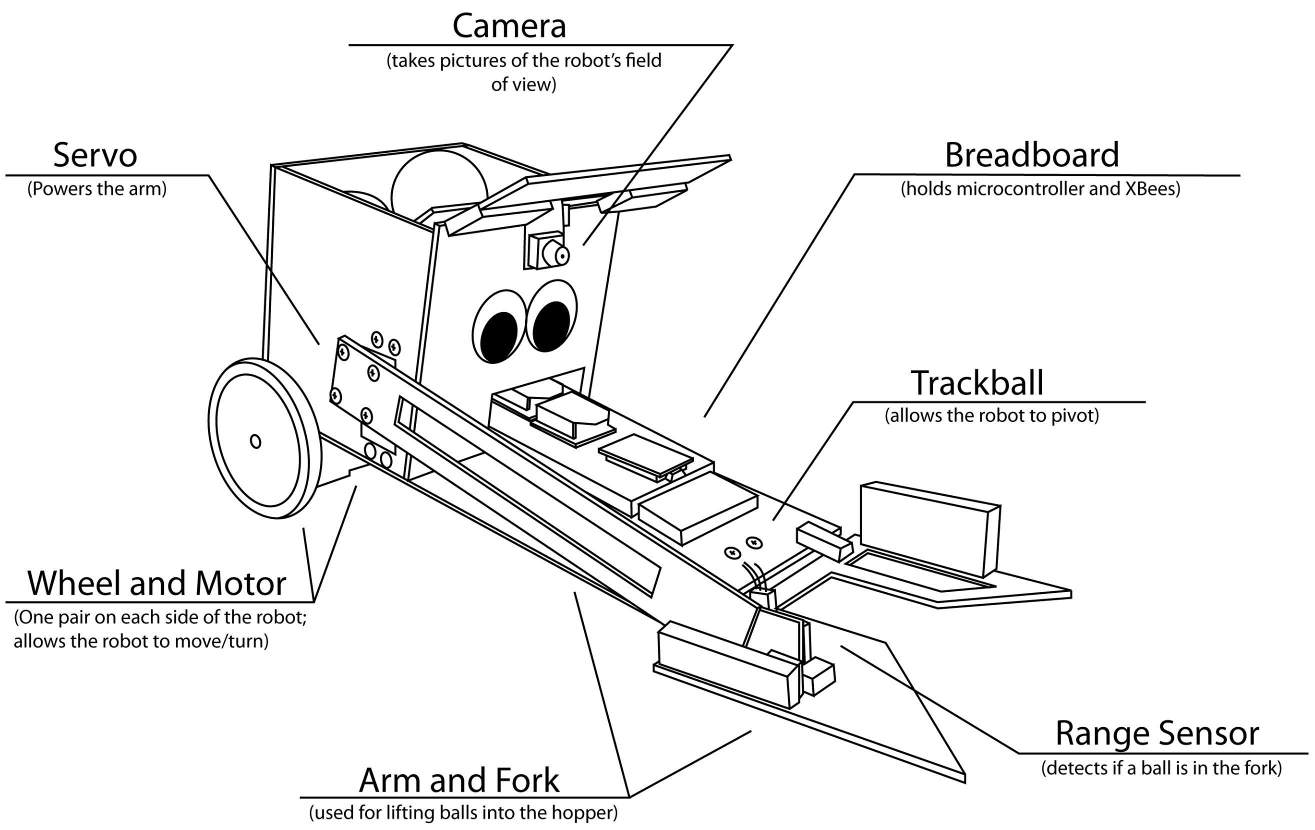
Camera
(takes pictures of the robot's field of view)
Servo
(Powers the arm)
Breadboard
(holds microcontroller and XBees)
Trackball
(allows the robot to pivot)
Wheel and Motor
(One pair on each side of the robot; allows the robot to move/turn)
Range Sensor
(detects if a ball is in the fork)
Arm and Fork
(used for lifting balls into the hopper)

Figure 4. Robot Design Outline

plex code-base, but eventually arrived on an autonomous mode outlined in Figure 6.

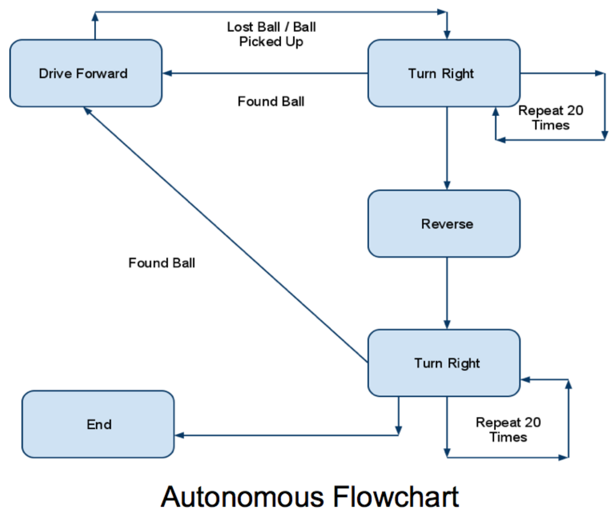

Figure 6. Software Flowchart

There are two software components to the robot. A small set of arduino code, and a larger C# code base for the C328R camera mounted on the robot. The C# code base was heavily modified, but was originally based on code from [2]. In general, the C# camera code retrieved an image from the camera, looked for a given color within a certain tolerance, and if it found the color, then it told the robot to move forward. If it didn't find the color, it told the robot to turn to the right. If the robot turned 360 degrees (about 20 turn commands) then the robot would go into reverse and try to look for the ball for another 20 turns. This was done so that in the unlikely event that the ball was perpetually stuck next to the robot as it turned, it would eventually move to a position that it was able to see the ball. If after the second 360 degree turn it did not see a ball, the robot would assume that there were no more balls to pickup and would end. A video demoing part of this functionality is available at: `http://www.youtube.com/watch?v=oFJHnSWJ17s`.

Before this could be done however, the camera had to be initialized. Initializing the camera was a tricky process as the camera was a temperamental beast (more on this later). When it worked, the camera would be initialized and then send an image to the C# code. The C# code would then display the image to the operator, who could then click on a certain portion of the image in order to set that color as the color that the code was to search for. Once this was done, the operator merely had to hit a button that spawned a separate thread that continuously took pictures, scanned for the color, and sent the commands to the robot in the order detailed above.

The messaging system between the camera code and the robot was an incredibly simple. It consisted only of single letter commands. There were commands for forward, reverse, left, right, stop, and change drive mode. The final command changed the power sent to the motors for any given command, allowing for the robot to drive better on carpet. The robot, upon receiving a forward or reverse command would continue to do that until it received another command. If it received a left or right command, it would turn a predetermined amount and then stop. If it any point the robot's range-finder read an object in the forklift, the robot would stop. It would then raise the forklift to put the object in its hopper and then lower the forklift. Finally, it would pause for two seconds, allowing the camera code to take another picture before it flushed its input buffer and awaited new commands.

In addition, due to the simpleness of the command system, and the difficultly of getting the camera to do what we wanted, we decided to implement a manual control mode where the operator could press keys and the robot would operate in response to those key presses. This was known as manual mode and simply allowed the operator to steer the robot instead of the code. There was no change on the robot's end in manual mode, the commands seemed to be coming from the same place.

Due to the code from [2] that we ended up working from, we were a little limited in some of the features we were able to implement. For example, while we had originally thought it might be nice to allow the robot to respond to different colors, it soon became obvious that it would be infeasible to calibrate to multiple colors, just due to the way the code was setup. However, for the same reason it was also easy to allow us to pick any color as the color that we were calibrated too.

Integration Problems

As mentioned before, the range-finder for the final version was moved off of the vestigial second arm, and onto the remaining right arm. It was attached more securely, and cushioned with pieces of foam. During testing, we noticed that some highly reflective environments such as linoleum caused the sensor to report false-positives. This was remedied by placing another piece of foam on its end at the other side of the forklift across from the range-finder. This stopped the reflectiveness of the floor from interfering with the operation of the range-finder.

Originally we attempted to use a small AA battery to 5v converter that we had purchased to supply 5v power to the Arduino. However, this converter ran through batteries very quickly and produced variances in voltage that adversely affected the camera. We remedied this by replacing the 5v converter with a 9v battery and feeding the power in through the RAW port USB hub.

One of the major problems we had with the robot from day one, was the problem of it "flipping out." Flipping-out was characterized by the robot running one or two of its motors and moving its servos up and down anytime it was powered on. We originally thought this wasn't a big issue, just an annoying one. However, we soon discovered that the flip-outs were causing the arm to become unaligned with its correct location relative to the ground. Though we inquired about using a relay or manual switch to prevent the flip-out occurrences we never implemented these solutions. In the end we solved the problem by adding foam to the arm to prevent it from moving below a certain minimum height. We believe that even if we prevented the robot from flipping-out the arm is still slightly too heavy and that this is causing the arm to sag over time.

Finally, the largest integration problem we faced was that of the camera. The camera had a lot of problems and was generally unpleasant to use. The biggest issues for us with the camera were those relating to the speed of the picture taking and maintaining a sync with the camera. The speed problem was two-fold. First, the camera was running at too low a baudrate. Second, the camera refused to take any RAW photos at all. The only format we were able to use was JPEG, which meant longer transfer times as the photos had more metadata, checksums, and compression, as well as JPEG artifacts in the image itself. We tried for a long time to get the camera to use a higher baudrate than the standard 9600bps. Eventually we discovered that the problem was that the XBee's had to be set to a higher baudrate as well. However, even after setting the XBee's to 115200bps, the camera still wouldn't respond. Despite our best efforts, we were unable to get the camera to respond to a baudrate of 115200bps, even though this was listed as a supported rate in the camera's documentation. We eventually settled on 57600bps as the highest baudrate that the camera would respond to. This allowed us to achieve an average photo capture and transfer time of about 1.6 to 2.0 seconds, down from 8-10 seconds.

Though we would have preferred to take pictures in RAW format, we were unable to get the camera to do so. We read the manual, and searched the internet for hours, but it appears that no one has gotten this functionality to work. No matter what, whenever we issued a command to take a RAW photo, the camera would just stop responding. It would have to be power-cycled before it would respond to commands again.

Speaking of power cycling, the last issue we had with the camera was that of maintaining sync. When the camera is first powered on, we have to issue a sync command to the camera up to 60 times until the camera figures out the baudrate we are attempting to communicate at. The problem is that sometimes the camera will never respond to these sync requests until its power is cycled. The more frustrating problem is that sometimes the camera will randomly decide that it no longer wants to respond to commands in general and we are forced to re-sync with it. Even then, it will sometimes never respond to the sync and we will be forced to cycle its power. Despite our best efforts, we were never able to figure out this cause of this behavior, and our robot suffered greatly for it. Without a consistently working camera, the autonomous mode of our robot is nearly useless.

FUTURE WORK

Although our project met our original design goals, there is certainly room for improvement if work were to continue. The final version of the robot is very object and environment dependent. Accounting for these variables simply made the project too complicated in the time allotted, and a simplistic design was used to bypass these issues. However, given more time and the knowledge learned from the first two iterations of the project, the robot could be redesigned to address these issues. It could also be redesigned in order to fulfill our optional design goals and improve upon construction.

In order to make the robot less environment dependent, it would require redesigns in the following areas: movement, the body, and the arm. For movement, the current motors would need to be replaced with more powerful motors. This would allow the wheels to overcome ridges in the floor. The trackball, which had a habit of getting stuck in rugs or ridged carpeting, would need to be replaced with a sturdier trackball or a small wheel. The body would need to be elevated to account for these changes, and the arm would need to be positioned at a higher elevation from the floor.

Making these changes would break the current ”skim the floor” technique that the robot uses to pick up ping-pong balls. However, the arm would need to be redesigned if it were to become less object dependent. In place of the current ball-catcher, two arms could be designed to pick-up a wider variety of objects. The design would be entirely dependent on the variety of objects, but they could be elevated higher from the floor.

Changes to the image processing would also go a long way towards improving the accuracy/ object dependency of the robot. In addition to replacing the camera with a faster, less buggy alternative, additional image processing techniques could be implemented. Blob detection and image segmentation could be used to calibrate according to specific region-bounds instead of simply relying on color, which would help in identifying a wider variety of objects more accurately. Kalman filters would also help to reduce noise and inaccuracies.

Lastly, the optional design goal could also be met, and the construction could be improved. Implementing the design goal could be as simple as storing a counter for each successful pickup, and an infrared sensor could be used to identify the nearest human. Construction could take advantage of 3D printing for the arm design, and the hopper and motors could be screwed to the rest of the build, similar to the current servo, arm, and trackball.

REFERENCES

1. Barrel Collector. `http://www.philohome.com/picker/picker.htm`.
2. C# Camera Code. `http://www.codeproject.com/KB/recipes/C328R.aspx`.
3. Garbage Collector. `http://roboticdnt4qut2.wordpress.com/2009/03/05/trash-collector-robot-tcr2009/`.
4. Golf Ball Collector. `http://www.belrobotics.com/robot/mower/products_ballpicker.html`.
5. Red/Blue Ball Collector. `http://www.youtube.com/watch?v=VRMF_THNwnI`.
6. Tenis Ball Collector. `http://www.seattlerobotics.org/encoder/aug99/ballbot.html`.