AU Smart Cart

Team 3

Cycle 2 Report

12/03/2012

Senior Design

Fall 2012

Dr. Roppel

Brandon Poole

Geoffrey Bennett

James Liner

Joshua Chumbler

John Johnson

Harshit Goyal

Executive Summary

Team 3 consists of six members working together to design and build a fully autonomous shopping cart. The purpose of the cart is to perform as a personal shopping assistant to the user as he/she shops in different environments. The project must be able to meet the following requirements in order to be considered a success:

- Follow the user and maintain a safe distance
- Perform as a cart by carrying a fair amount of objects inside
- Respond to voice commands such as: stop and go
- Respond to hand gestures

As a result from a great deal of hard work put in by each member this semester the team has been successful in developing a working product. The AU SmartCart meets all the requirements set by the team and operates safely and accordingly to all available regulations.

In a day and age when humans are becoming increasingly lazy a robotic shopping cart is just the ticket for your everyday shopping desires. This product can be targeted to everyone from lazy young college students, which are too busy text messaging to merely push a cart, all the way up to older citizens that are unable to do so. For a successful company this project could prove to be a very good investment. The endless possibility of potential clientele would leave a company with a wide range of opportunities to cash in on its investment. Overall the AU SmartCart is a wonderful piece of equipment and should prove to be both useful and profitable for any company willing to invest.

Table of Contents

<u>Project Overview</u> – Joshua Chumbler

Introduction

For this year's senior design project Team 3, consisting of Brandon Poole, Joshua Chumbler, Geoffrey Bennet, John Johnson, James Liner and Harshit Goyal, decided to design and build an autonomous robotic shopping cart known as The AU Smart Cart.. The purpose of The AU Smart Cart is to perform as a personal shopping assistant to the user and follow him/her around as he/she shops. To make the cart useful as well as safe the team decided to set certain criteria on the project. First the cart would obviously have to be capable of following the user around and maintaining a safe distance. To do this the team soon discovered that another requirement needed to be that the cart could accurately navigate around additional objects and people without running into anything. Next the cart needed to still function as a complete shopping cart and be capable of carrying a reasonable amount of goods. Finally it would need to be safe and easy to shut down in case of an emergency to make it complete.

Requirements

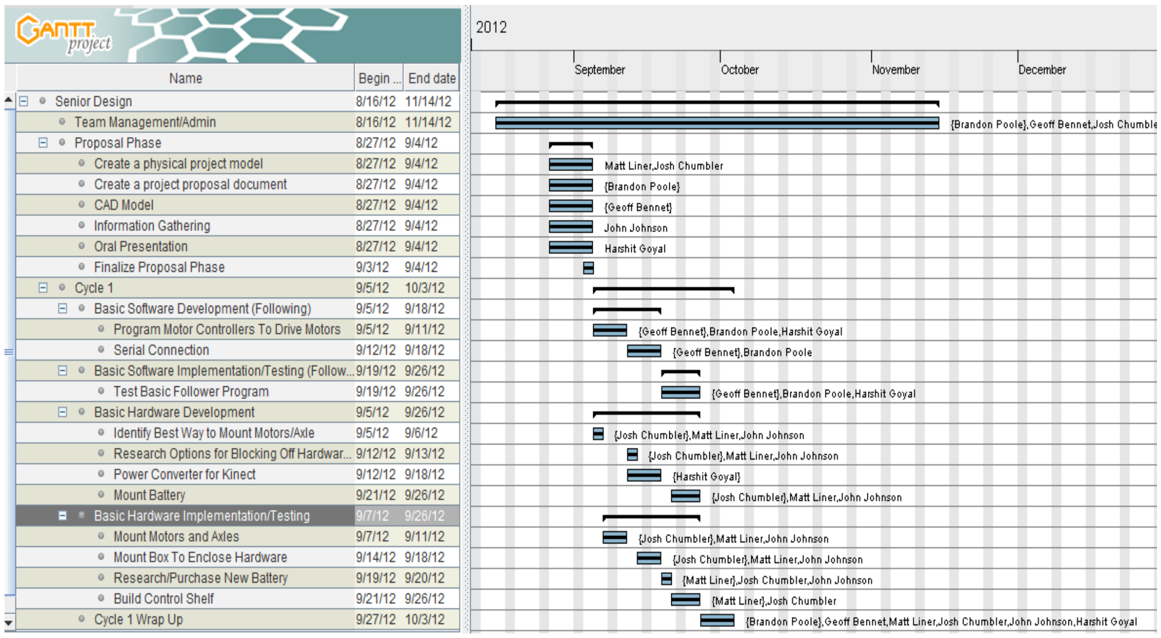
Once the criteria had been set the team began working on the design portion of the cart. First the team determined that to keep the cart fully functional a medium sized cart would be the best choice so that it would remain easy to maneuver yet still be useful. Next the team started working on ways to control the cart. The simplest most available way to make the cart follow the user seemed to be to use a Microsoft Xbox Kinect and communicate with it using a laptop that would transmit the signal to a microcontroller. Next to make the cart easy to use the team decided to give it the ability to respond to hand gestures as well as voice commands. The team decided this would be an easy way to make the cart start and stop when picking up items to go in the cart. In addition to the start and stop function provided by the voice commands, the hand gestures would give the user the ability to manually control the cart. All of this combined would give the user the power to maneuver around obstacles that could get in the way and stop the cart from advancing.

Team Management

As the cart began to come together it quickly became apparent to the team that the project consisted of two distinctively different sides. Based on the size of the project as well as the size of the team, the main leader Brandon Poole, in agreement with co-leaders Joshua Chumbler and Geoffrey Bennet along with the rest of the team, decided to break the project into two separate teams in order to divide and conquer so to speak. The first team consisted of Brandon Poole, Geoffrey Bennet and Harshit Goyal, and this team was faced with the task of working on the software development and design. This involved everything from working on the software for the microcontroller and Kinect to actually finding and purchasing the devices necessary to operate the cart. The second team consisted of Joshua Chumbler, James Liner and John Johnson and this team was focused on the hardware aspects of the cart. This included finding and purchasing the cart along with designing and building other parts of the cart such as the shelf for the Kinect and the control box located in the bottom of the cart.

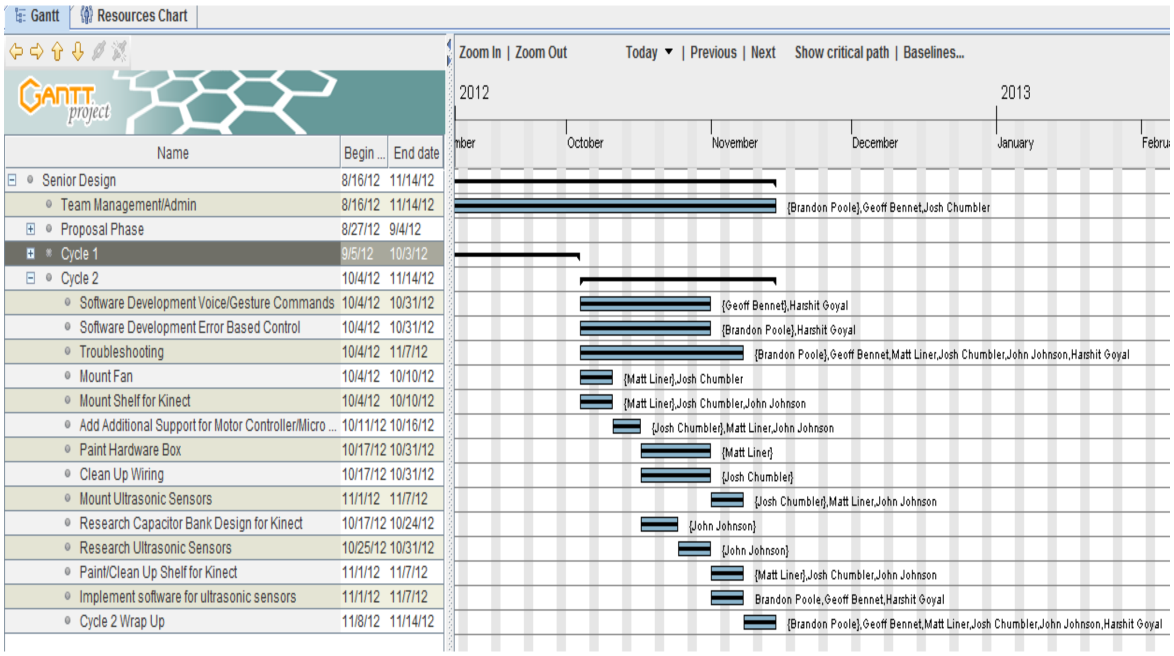
Resources Management and Expenditures

Based on the requirements of the class, each team member was expected to be willing to spend a minimum of one hundred dollars apiece as well as to work an expected ten hours per week on the project. For a project timeline as well as the individual time resources allocated to each aspect of the project see figures 1 and 2 on the following page.



Name	Begin ...	End date	
Senior Design	8/16/12	11/14/12	
Team Management/Admin	8/16/12	11/14/12	{Brandon Poole},Geoff Bennet,Josh Chumble
Proposal Phase	8/27/12	9/4/12	
Create a physical project model	8/27/12	9/4/12	Matt Liner,Josh Chumbler
Create a project proposal document	8/27/12	9/4/12	{Brandon Poole}
CAD Model	8/27/12	9/4/12	{Geoff Bennet}
Information Gathering	8/27/12	9/4/12	John Johnson
Oral Presentation	8/27/12	9/4/12	Harshit Goyal
Finalize Proposal Phase	9/3/12	9/4/12	
Cycle 1	9/5/12	10/3/12	
Basic Software Development (Following)	9/5/12	9/18/12	
Program Motor Controllers To Drive Motors	9/5/12	9/11/12	{Geoff Bennet},Brandon Poole,Harshit Goyal
Serial Connection	9/12/12	9/18/12	{Geoff Bennet},Brandon Poole
Basic Software Implementation/Testing (Follow...	9/19/12	9/26/12	
Test Basic Follower Program	9/19/12	9/26/12	{Geoff Bennet},Brandon Poole,Harshit Goyal
Basic Hardware Development	9/5/12	9/26/12	
Identify Best Way to Mount Motors/Axle	9/5/12	9/6/12	{Josh Chumbler},Matt Liner,John Johnson
Research Options for Blocking Off Hardwar...	9/12/12	9/13/12	{Josh Chumbler},Matt Liner,John Johnson
Power Converter for Kinect	9/12/12	9/18/12	{Harshit Goyal}
Mount Battery	9/21/12	9/26/12	{Josh Chumbler},Matt Liner,John Johnson
Basic Hardware Implementation/Testing	9/7/12	9/26/12	
Mount Motors and Axles	9/7/12	9/11/12	{Josh Chumbler},Matt Liner,John Johnson
Mount Box To Enclose Hardware	9/14/12	9/18/12	{Josh Chumbler},Matt Liner,John Johnson
Research/Purchase New Battery	9/19/12	9/20/12	{Matt Liner},Josh Chumbler,John Johnson
Build Control Shelf	9/21/12	9/26/12	{Matt Liner},Josh Chumbler
Cycle 1 Wrap Up	9/27/12	10/3/12	{Brandon Poole},Geoff Bennet,Matt Liner,Josh Chumbler,John Johnson,Harshit Goyal

Figure 1. Project Timeline Proposal Phase and Cycle 1



Name	Begin ...	End date	
Senior Design	8/16/12	11/14/12	
Team Management/Admin	8/16/12	11/14/12	{Brandon Poole},Geoff Bennet,Josh Chumbler
Proposal Phase	8/27/12	9/4/12	
Cycle 1	9/5/12	10/3/12	
Cycle 2	10/4/12	11/14/12	
Software Development Voice/Gesture Commands	10/4/12	10/31/12	{Geoff Bennet},Harshit Goyal
Software Development Error Based Control	10/4/12	10/31/12	{Brandon Poole},Harshit Goyal
Troubleshooting	10/4/12	11/7/12	{Brandon Poole},Geoff Bennet,Matt Liner,Josh Chumbler,John Johnson,Harshit Goyal
Mount Fan	10/4/12	10/10/12	{Matt Liner},Josh Chumbler
Mount Shelf for Kinect	10/4/12	10/10/12	{Matt Liner},Josh Chumbler,John Johnson
Add Additional Support for Motor Controller/Micro ...	10/11/12	10/16/12	{Josh Chumbler},Matt Liner,John Johnson
Paint Hardware Box	10/17/12	10/31/12	{Matt Liner}
Clean Up Wiring	10/17/12	10/31/12	{Josh Chumbler}
Mount Ultrasonic Sensors	11/1/12	11/7/12	{Josh Chumbler},Matt Liner,John Johnson
Research Capacitor Bank Design for Kinect	10/17/12	10/24/12	{John Johnson}
Research Ultrasonic Sensors	10/25/12	10/31/12	{John Johnson}
Paint/Clean Up Shelf for Kinect	11/1/12	11/7/12	{Matt Liner},Josh Chumbler,John Johnson
Implement software for ultrasonic sensors	11/1/12	11/7/12	Brandon Poole,Geoff Bennet,Harshit Goyal
Cycle 2 Wrap Up	11/8/12	11/14/12	{Brandon Poole},Geoff Bennet,Matt Liner,Josh Chumbler,John Johnson,Harshit Goyal

Figure 2. Project Timeline Cycle 2

The total cost of building the cart is $983.00 with $450.00 of that total being donated to the group by outside sources. For a breakdown of the costs of the project see Table 1 below.

Table 1. Cost Breakdown

Parts	**Cost**	**Parts**	**Cost**
Paint	$15.00	**Wood**	$30.00
Shelf	$16.00	**Kinect**	$100.00 – Donated
2-Swithces	$10.00	**2-Motors**	$60.00 - Donated
Cart	$40.00	**New Axel**	$20.00
Tablet PC	$200.00 – Donated	**SaberTooth Controller**	$150.00 - Donated
Arduino Microcontroller	$40.00	**Battery #1**	$40.00
Battery #2	$25.00	**Small Fan**	$10.00 - Donated
3-Ping Sensors	$75.00 – Donated	**Emergency Stop Switch**	$5.00 - Donated
Wire	$25.00	**AC-DC Converter**	$3.00
Capacitor Bank	$1.00	**Laptop Mount**	$50.00
Shrink Wrap	$12.00	**Zip Ties**	$5.00
Shelf Brace	$8.00	**Hardware (bolts, nuts, etc…)**	$8.00
Box Brackets	$20.00	**Breadboards**	$15.00

In addition to the contribution by each team member the Electrical Engineering Department at Auburn University graciously allowed the team access to its wood and machine shops. This gave the team the ability to design and build some of the hardware aspects of the cart with the help of Ms. Linda Barresi and her student workers. In order to have plenty of space to work and store the project and tools the team was given access to a special lab in the basement of Broun Hall by Dr. Robert Nelms. Finally Dr. Thaddeus Roppel gave the team a great deal of assistance by offering his knowledge on certain things the team faced while working on the project. His assistance along with everyone else that chipped in with small ideas over the semester gave the team a great advantage in completing the project.

Obstacles and Solutions

Over the course of this project the team did encounter a few minor obstacles that should be noted for future reference. One of the biggest things the team came across when working on the project was the microcontroller overheating and then shutting down the entire cart to the point that it wouldn’t run at all. The team had initially been using an Arduino Motor Shield to communicate to the motors but after researching it and doing some tests the team discovered that it was insufficient for the requirements we had. The Arduino was actually rated for around four amps and the motors were pulling close to eight. An easy fix to this problem was to change out the motor shield and replace it with a SaberTooth Motor Driver. After a bit of reworking the software the change to the new motor driver completely fixed the issues the team was having. Another problem the team had pop up was traction for the back tires that were being used to propel the cart with two differential drive motors. To fix this issue the team merely trimmed the axle down some that was stabilizing the back wheels and thereby tightened up the small amount of flex that was occurring on the cart at the rear axle. This small change greatly increased the amount of traction to the back wheels giving the cart better control.

<u>Hardware Overview</u> – James Liner

The first thing the team had to consider when building our AU Smart Cart was what cart to use in terms of size and functionality. Initially the team thought of using a standard shopping cart such as the ones seen at your local grocery store. However, the team decided that a standard shopping cart would be difficult to move around effectively with small electric motors. A larger cart would also present a challenge for our team in terms of moving it from place to place. Ultimately, our team decided to go with a cart a little smaller than normal. The cart dimensions are 23.5 inches tall, 15.5 inches wide, and 18.5 inches long. The team also had to make several modifications to the cart to make it useable for our project. The back axle had to be elongated to make room to mount motors and the hardware team had to drill holes in the framing to hide wiring just to name a few of the said modifications. The back axel was modified by buying a piece of metal rod the same diameter as the original axel and cutting it to meet our needs. We also had to have a control shelf to mount the eyes of our robot as well as any switches needed to turn the robot or any other components on/off. The control shelf is made from sheet metal and has a 10 inch platform that is just long enough for our Xbox Kinect to easily calibrate and keep track of the user from its final mounted position. The team also had to come up with a way to protect the electrical components that are needed to control and power the cart. This was accomplished by building a wooden enclosure and painting it black to match the cart.

As mentioned earlier in the hardware overview the team wanted to use small electric motors to power our robot. The reason for this was because the team had them readily available. The team was lucky enough to have a couple of Power Wheels Jeeps donated to us and the motors that we removed from the Power Wheels seemed to be very strong and reliable. The team also had to discuss whether or not we wanted to make our robot a differential drive robot. This was an easy choice for the group. In terms of maneuverability, differential drive was our easiest option. The team also had a total of four motors and we only needed two to accomplish a differential drive for our cart. This gave us a complete backup in terms of our motors. This is very comforting considering we have used motors and we have no idea how much use they have had. Once the team knew the plan for the motors we had to mount them and make them functional. In order to do this we used zip ties to fasten the motors and gear assembly to the cart. Then we took the sleeve that was once attached to the wheels of the Jeep and drilled holes in it and the wheels of our cart to essentially attach our wheels to our motors.

Another huge piece of hardware is the Xbox Kinect. The Kinect serves as the eyes of our robot. The Kinect sends information about the location of the user to our PC, then the PC tells the Arduino microcontroller that information, then the Arduino communicates it to the Sabertooth motor shield. There are several reasons the team decided to use the Kinect. One of the main reasons we chose the Kinect was availability.   The group has two members with Kinects and both were willing to donate them to the project. This again gave us security in terms of a back up. If one of our Kinects broke we had another to take its place. Another key reason the group chose the Kinect is it is easy to modify its functionality to meet the needs of the group. Since all of the coding for the Kinect is pretty easy to get your hands on the software team didn’t have a lot of trouble integrating the Kinect into our control system.

As with any project there are some improvements that can be made to the hardware of our cart. One of the main things the hardware team could improve on is the motors. The motors have been fine so far but it is a big concern that we have no idea when they might fail. If we had brand new motors it would give us much more confidence in terms of our cart reliability. Another thing that would make the cart better is to use something other than wood to make the component box. When it comes to making modifications on the fly the wood was our best choice but if the team had time to get something fabricated Plexiglass or sheet metal may be better options because of their durability. All in all, our cart functions exactly the way the team intended for it to and everyone is happy with both how it looks and how it functions.

Electrical Overview– Harshit Goyal

The Electrical hardware selection for AU Smart Cart provided the foundation for the scheme and design of the project. Each selection played a vital role in the specifics of construction and the ultimate operation of the design. Specific selected pieces created a center that all other hardware selections were based from. The team worked efficiently within the constraints for the hardware selection for AU Smart Cart.

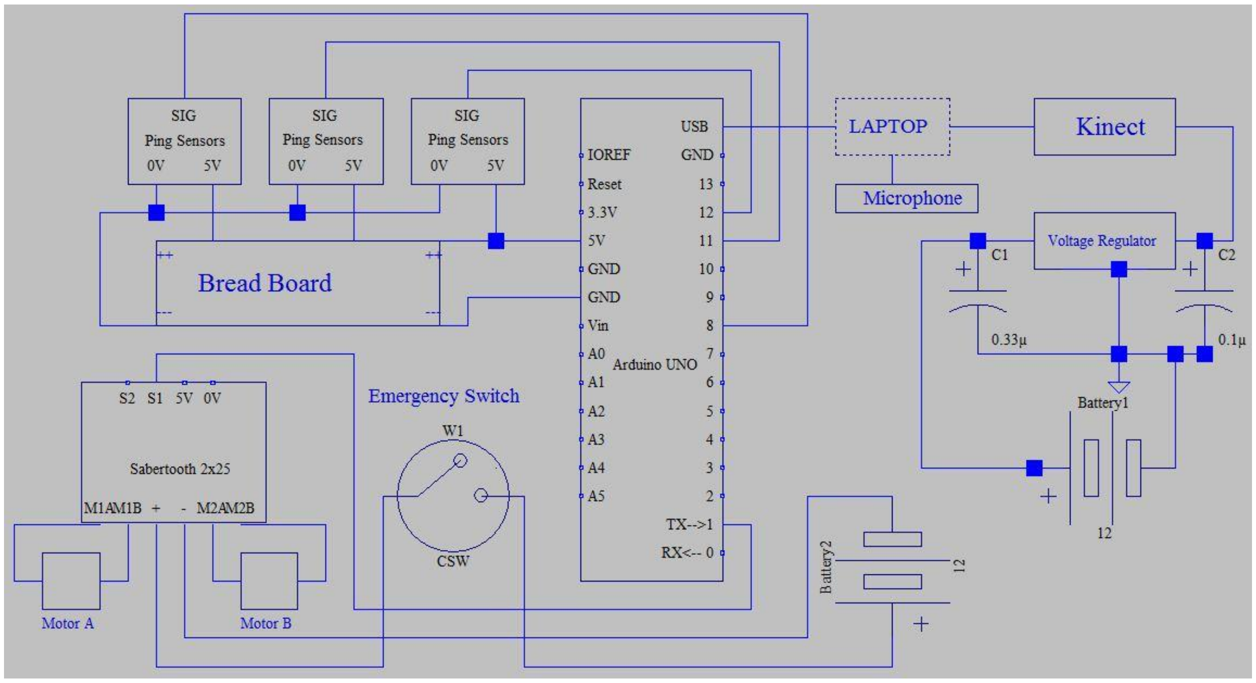


Figure3. Schematic of AU Smart Cart Showing Electrical Hardware and Design

To successfully build a shopping cart that can move automatically, a micro-controller with the proper programming is essential. The team decided to buy 2 separate micro-controllers for the design project. With various options available the team decided to implement an Arduino UNO REV.3 and a Sabertooth 2x25 motor controller into the design of AU Smart Cart.

Arduino Uno Rev.3

The Arduino Uno is a microcontroller board based on the ATmega328. With14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button, The Arduino microcontroller can readily handle all of the operation aspects of the project plus various add-on features that might be presented in the future. The main function of Uno is to communicate with the motor controller and laptop. The Uno is connected to laptop via USB cable. Two primary considerations were focused upon in the selection of the Arduino Uno microcontroller which was familiarization and ease of use. The Arduino microcontroller readily satisfied the needed aspects of the design. For an Arduino Uno Controller image see figure4 below and for additional Uno specifications see table 2 on the following page.

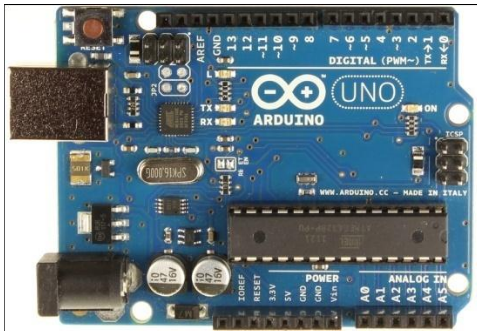


Figure 4. Ardunio Uno Microcontroller

Table 2. Arduino Uno Specifications

Model:	Arduino UNO REV.3
Specifications:	Operating Voltage- 5V Input voltage- 7 V- 12 V Digital I/O pins- 14 (6 for PWM Outputs) Analog Input Pins- 6 DC Current per I/O Pin- 40mA

Sabertooth 2x25

One of the main necessities was that the team implemented a process by which every motor on the AU Smart cart can be controlled with regards to speed and timing using an autonomous process. The most reasonable means to do this was to use a motor controller. On this basis the team chose to employ a Sabertooth 2x25 motor controller. The Sabertooth 2X25 is one of the most versatile, efficient and easy to use dual motor drivers used in general purpose robotics. Sabertooth allows you to control two motors with: analog voltage, radio control, serial and packetized serial. The operating mode is set with the onboard DIP switches so there are no jumpers to lose. Sabertooth features screw terminal connectors - making it possible for you to build a robot without even soldering. While working in cycle-1, Arduino motor shield was used in cart design but because it was limited to 4A of current, and the motors drew approximately 8A it resulted in overheating of the motor shield. So in order to resolve that problem the Sabertooth motor controller was used to supply two DC brushed motors with up to 25A each with peak currents of 50A per channel achievable for a few seconds. Also one of the most important reasons to choose this over Arduino was its very fast stops and reverses - giving our robot a quick and nimble edge. See figure 5 for an image of the Sabertooth 2x25 and table 3 for additional specifications.

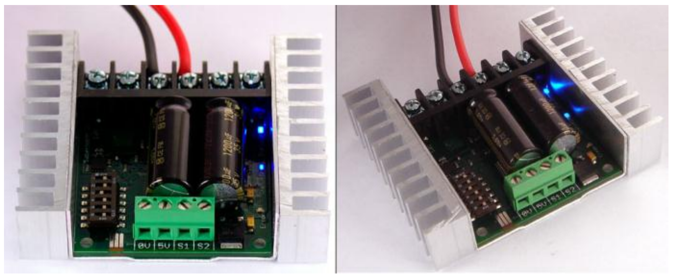


Figure 5. Sabertooth 2x25 Motor Controller

Table 3. Sabertooth 2x25 Specifications

Model:	Sabertooth 2X25
Specifications:	25A continuous, 50A peak per channel. Synchronous regenerative drive

	Ultra-sonic switching frequency Thermal and over current protection Lithium protection mode Input modes: Analog, R/C, simplified serial, packetized serial Size: 2.6” x 3.2” x .8” 65 x 80 x 20 mm Weight: 96g / 3.5oz

Voltage Regulator to Power Kinect

In order to power the Kinect a 12 V regulator is added so that it can be connected to a 12 V DC battery. Using the Kinect with a power supply splitter, a voltage regulator, 0.33μF and 0.1μF capacitors, and a small prototyping board we constructed a circuit to power the Kinect from a battery. See figure 6 in below for a circuit diagram of the voltage regulator setup.

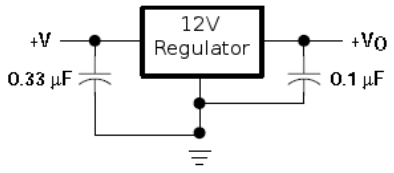


Figure 6. Schematic of a 12 V Regulator

Ping Sensors

Parallax's PING))) ultrasonic sensor provides a very low-cost and easy method of distance measurement. The Ping sensor measures distance using sonar; an ultrasonic (well above human hearing) pulse is transmitted from the unit and distance-to-target is determined by measuring the time required for the echo return. These sensors provide precise, non-contact distance measurements within a 2 cm to 3 m range. In order to provide the AU Smart cart with a sense of vision the team decided to use PING))) ultrasonic sensors to judge distance. These sensors were mounted on each side of the AU Smart Cart except the front side to detect objects and the environment around it. The two primary considerations that lead our team to choose ping sensor were cost and ease of use. Its 3-pin header makes it easy to connect using a servo extension cable, no soldering required. And as they were donated, the costs of these sensors were $0. Figure 7 below shows an image of the PING))) sensor and a wiring guide of it.

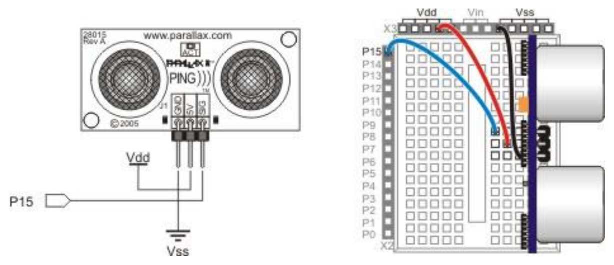


Figure7. Ping))) Sensor Schematic and Wiring Diagram

Batteries

Batteries are used in our project to power theSabertooth2x25 motor controller board and a 12 V regulator which will be used in order to power the Kinect. The 12 V batteries provided power to Sabertooth motor controller through the pins Vin and GND which in turn powers the 2 brushed DC motors. The reason to employ 2 batteries was that the Kinect was shutting off because of large current surge from the motor controllers; giving the Kinect its own power source solves that problem.

Possible Improvements on the Electrical Hardware Design

- Multi sensors pose estimation.
- Laser to Kinect calibration.
- Reduce number of batteries from two to one.

Processing / Kinect Data-Geoff Bennett

The Microsoft Kinect announced a major improvement in motion sensing input device technology. It is equipped with an RGB camera, a depth sensor consisting of an IR laser projector combined with a monochrome CMOS sensor, and multi-array microphone; in turn providing full-body motion capture video data in 3D under any ambient light conditions, facial recognition, and voice recognition capabilities. Processing is an open source programming language and environment for people and beginner programmers who want to create images, animations, and interactions. Initially the IDE was developed as a software sketchbook and to teach fundamentals of computer programming with a visual context to beginner programmers. Processing creates programs using 2D, 3D, or PDF output and is integrated with OpenGL for accelerated 3D. There are over 100 libraries that extend the software for numerous applications. Using Microsoft Kinect and Processing libraries specifically made for the Kinect our choice for programming was the Processing IDE.

The Kinect uses an RGB and depth camera. Since a computer can understand depth images easier than a conventional color image, we were mainly focused on using the depth camera. The depth image returned to the computer uses color pixels that tell you how far apart the image is from the camera and in turn the pixel data directly correspond to where the object begins, where it ends and determining if people are in the image. The Kinect has the ability to capture the same depth image in a bright room or in a pitch black one which makes the depth camera more reliable and even easier for a computer program to understand. The depth camera shines a grid of infrared dots and then recognizes that grid from the IR projector. Each Kinect was calibrated to know exactly where each dot from its projector appears when projected against a flat object at a known distance. Any object that is closer than the Kinect's calibration distance will push the dots out of position in only one direction and vice versa pushing an object the opposite direction if it is far away. Since the Kinect is calibrated as such it recognizes the displacement of these dots to determine the distance the object is within the image frame. The only drawback to using a depth camera is that it cannot be used outside because sunlight makes it difficult for the program to understand the depth of objects.

When Microsoft released the software and drivers behind the Kinect, it made developing programs that uses the depth camera easier to track people and detect individual body parts provided by that user. We didn't have to analyze the depth image ourselves, since it was already determined, but made accessing the data for that image very easy. The data for a user is gathered based on a Cartesian coordinate system. Once a user has been tracked via user calibration, the program now is capable of tracking data from every joint and limb of the user. This calibration process is completed within a series of stages within the program. In the calibration process the program initiates various callbacks triggered by *OpenNI*, a preset library for Kinect. The calibration process starts in the *onNewUser* function and is sent into *startPoseDetection* once a user has entered the scene. They are then detected by the *onStartPose* function and sent to *requestCalibrationSkeleton* once a user has assumed a *"psi"* pose. The *"psi"* pose is a default pose from the *SimpleOpenNI* library and requires that a user raise both hands above both shoulders respectively. If everything is successful thus far the calibration process is ended with the *onEndCalibration* function and once a user has completed a successful calibration, the program then starts the skeletal tracking with the *startTrackingSkeleton* function. At this point every joint and limb from the calibrated user is now available in either 2D or 3D coordinates. We chose 3D using the Cartesian coordinates because our robot needed to determine how far away the user is from the Kinect and that is determined from the z – axis data from the torso. The x – axis data from the torso represents how far from the left and right the user is from the center point of the Kinect which is the origin. ***See appendix A - Processing Main.**

We incorporated speech recognition into our project using a library called *STT*. *STT* stands for speech – to – text and became very useful in determine specific voice commands our robot will comprehend. Currently, the most basic voice commands that are implemented are "start" and "stop." The program is capable of interpreting many different voice commands and also translates different languages for voice recognition but for our purposes the basic commands were only used. The library itself listens to a microphone input, via XBOX headset and XBOX gaming receiver for computers, and then sends recordings of your voice to Google for processing. If the transcription was successful the *transcribe* method is called and we initialized *"start"* and *"stop"* to essentially start and stop our robot. We chose the internet connected transcription method versus conventional transcribe methods because Google is a free service and most places have free accessible Wi-Fi hot spots. ***See appendix A - Processing Main.**

We also needed to communicate to the cart with hand gestures for users who did not want to communicate via voice. We created a few different hand gestures specifically made to be robot/user friendly. In order to halt the robot from all movements, momentarily, a user must raise their left hand above their left shoulder. This gesture sends a specific value to the motor controllers that represent stop or a zero PWM. The robot will return normal operation when the user lowers their left hand. This gesture was implemented to prevent the user from having to restart the program or calibration process to initiate cart movement. When a user raises their right hand above their right shoulder, and their left hand remains below the torso, the cart is now controlled by depth data interpreted from the right hand instead of the torso. The right hand control was implemented to make it easier for the user to position the cart without having to move their actual position, but just movements of their right hand. Finally, we needed a toggle start and stop gesture to halt the cart all together until needed at a later time. We calculated data using the user's left hand, left elbow, and left shoulder that if a user has extended their arm 75% or greater towards the cart, the cart will then stop all movements until the user has extended their arm again. Both the voice commands "start" and "stop" and toggle start/stop with the left arm represent the same commands within the program, but must be done independently from one another. ***See appendix A - Processing – Processing Main & Skeletal Poser.**

Motor Control Method – Brandon Poole

In order to have the cart follow a user around, there needed a way to convert the user's position into independent motor speeds. The team determined that the best option was to use an error based control method. The error based controller used the difference that existed between the measured state and a desired state to control the output to the motors. The AU Smart Cart used a specific type of error based controller called a PID controller. The PID controller combined the current error of the user's position with the accumulated previous error and the rate of change in the error. The team chose a PID controller because it allowed for the cart to respond more quickly and allow for the cart to respond to loading. If someone had loaded the cart with items that slowed it down, then the integral portion of the control would accumulate error to compensate and drive the cart faster. The derivative portion allowed the cart to respond more quickly to changes in user's position.

The cart needed to maintain a desired distance between itself and the user, as well as maintain its heading so that it traveled to the user. Therefore, two PID controllers were used in parallel. The first PID controller that will be discussed is the controller that was used to maintain the cart's distance to the user. The microcontroller read in the distance from the Kinect of the user in cm and subtracted off the predetermined desired distance of the user. This left it with some amount of positive or negative error. It saved that value and added it to the accumulated error, and subtracted the error from the error that existed from the last z position data received. It then scaled each value by its appropriate tuning value and added those values together. The AU Smart Cart team found the tuning values experimentally.

A similar method of maintaining the cart's heading was used. Instead of considering the error that existed between the measured user position and desired user position the controller considered the error that existed in how far offset the user was from the centerline of the cart in the x dimension.

The two controllers were combined to drive the wheels independently by adding the heading value to the distance value for the left motor, and subtracting the two values for the output to the right motor. This provided them with the PID control they desired to drive the cart. Figure 8 below is a graphic representation of cart control.

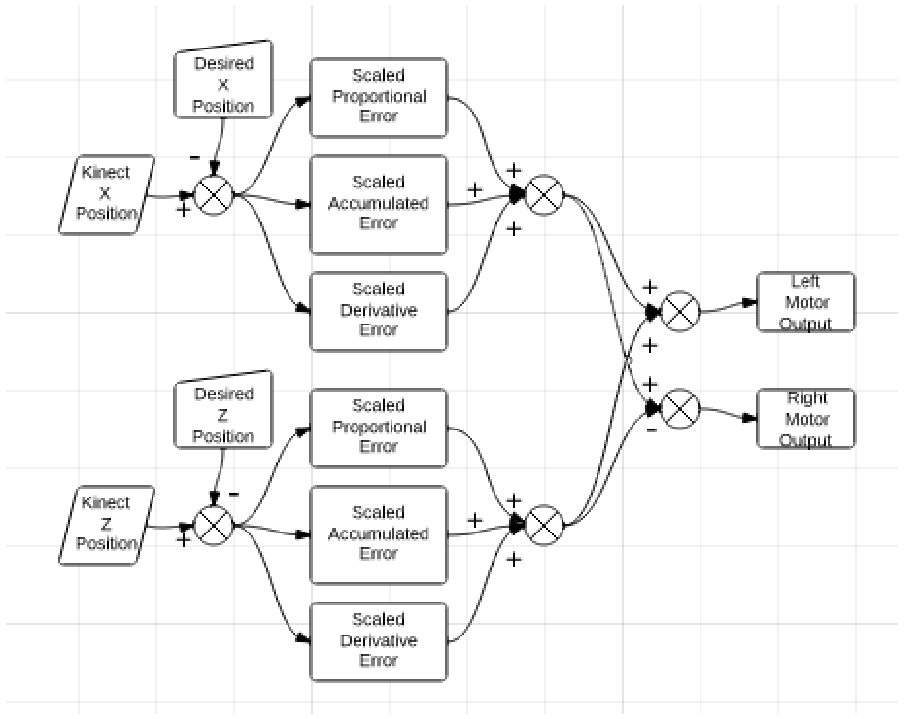


Figure 8. AU Smart Cart Control Flow Chart

The complete Arduino Uno code for the PID controls can be found in Appendix B.

The desired distance of the user from the Kinect was 140 cm. This was chosen because it allowed the cart to remain at a reasonably close distance while not following too far away to make maneuvering more difficult (if following too far away turning corners can become very difficult because the Kinect loses line of sight of the user in the turn). The maximum measured range of the user to the Kinect to was limited to 255 cm. That was because the serial communication method that was used would cause the microcontroller to read in erroneous values if a value greater than 255 was sent through the serial connection. It is important to note that the cart's vision was not limited to 255 cm; it was just the maximum position value the Uno would read in.

In order to communicate data to the Sabertooth from the Uno a serial connection was established between the transmit pin of the Uno and the S1 pin of the Sabertooth. The Sabertooth read in a range of values between -128 and 128 for each motor. In the case of 128 that motor would be at full speed forward, and -128 it would be in full reverse.

In addition to driving the cart, the Uno also had hand gesture and voice commands to start and stop the cart. Whenever the cart was stopped by the user, Processing would write the desired user position values for z and x to the serial port. Whenever the Uno read the exact desired user position values for the cart position, it would overwrite all of the other cart controls and send stop commands to the motors as well as erasing any accumulated error in the PID controllers. This was so that when the PID controller restarted it wasn't basing its output on old accumulated error. This allowed the team to simplify serial communication between Processing and the Uno.

A series of acoustic sonar sensors on the sides and rear of the cart were used in order to limit the cart from bumping into things. The sensors would determine if an obstacle was nearby and halt the cart in one of two ways. In the event that an obstacle was detected by the rear sonar, each motor would only be allowed to drive forward, and not backwards. The wheels and gearboxes stick out from the sides of the cart by 20 cm, therefore the side sonars kept the wheels and gearboxes from bumping into things. When the side sonars detected an obstacle within 20cm, the cart would halt. This prevented the wheels and gearboxes from colliding with any obstacles the cart may be near. In the case the sonar detected an object, the stop commands overrode all other commands received by the cart.

The PING))) sensors did encounter enough noise error to cause the cart to shudder and stop occasionally. To counter this issue team implemented an average value filter for the sensors. Each sensor maintained three sensor readings at any given time. Once per program cycle one of those three values was updated. The three readings were added together and divided by three to create an average value.

That value determined whether or not an obstacle was present. The Arduino code for the sonar bumper as well as the filter can be found in Appendix B.

Standards

In accordance with, OSHA Technical Manual (OTM), Section IV: Chapter 4, Subsection 5: Control and Safeguarding Personnel, the cart is equipped with safeguarding devices. The standard requires that one or more of the following safeguard devise be equipped: mechanical limiting devices, non-mechanical limiting devices, presence-sensing safeguarding devices, fixed barriers (which prevent contact with moving parts), interlocked barrier guards. The AU Smart Cart is equipped with two of those devices; the non-mechanical limiting device in the emergency stop switch, and the presence sensing safeguarding devices in the sonar sensors.

Contemporary Issues

One additional consideration of the AU Smart Cart is the ethical impact the cart will have. The AU Smart Cart should have no negative ethical shortcomings. The primary ethical concern of a service providing product would be that it is taking jobs away from individuals. The cart does not do that since there is no job that exists to push someone's shopping cart around. Another consideration is the environmental impact the cart has. The cart is powered by two 12 V batteries, which if not properly disposed of can cause significant environmental concerns. It will be important for the owners of the cart to properly dispose of used batteries once the batteries are no longer serviceable.

Conclusion

In conclusion the AU Smart Cart is a revolution in shopping. It successfully follows and performs well as a personal shopping assistant making it the perfect product for everyone from young lazy youths to the elderly. The development of the cart has been a tremendous learning experience for the students that developed it as well. The teamwork, project development, time management, prototyping, writing, and orating skills will be invaluable to the team members as they enter into their careers. The AU Smart Cart is an exciting development in robotics and shopping, but it is only the beginning. For future projects with the cart the team would like to make a few changes/additions. One thing the team would like to do is clean up the motors and gear assembly that drive the cart. The current assembly is somewhat bulky and the team feels that by changing it out with a different system the overall performance of the cart could be enhanced. Another addition the team would like to make is to add a scanner to the cart that will recognize items when you place them into the cart and log the carts content. This would make keeping up with the users current shopping cost less challenging as well as give them a better method of keeping up with their shopping list. Overall the AU Smart Cart was a success. In the realm of robots it proves to be an affordable and effective personal shopper.

Appendix A: Processing Code and Skeletal Poser

Processing Main

```
import SimpleOpenNI.*;  //import SimpleOpenNI lib

import processing.serial.*;  //import serial lib

import ddf.minim.*;  //import LoadSnippet lib

import com.getflourish.stt.*;  //import stt lib

Serial port;  //declare serial object

SimpleOpenNI kinect;  //declare SimpleOpenNI object

Minim minim;  //declare minim object

AudioSnippet snip;  //declare AudioSnippet

STT stt;  //declare STT

SkeletonPoser poseRight;  //check skeleton poser class

SkeletonPoser poseLeft;  //check skeleton poser class

SkeletonPoser poseNormal;  //check skeleton poser class

int zposition, xposition, zrightHand, xrightHand, holdStop, toggleStop, toggleReturn, voiceStop, xprint, zprint, i, holdx, holdz;

int j = 0;

String result;

String start = "start";

String stop = "stop";

void setup() {

  size(640, 480);  //double the width to display two images

  stt = new STT(this);  //initiate stt for code

  stt.enableDebug();  //enable console output with relevant information

  stt.setLanguage("en"); //set the output with relevant information

  stt.enableAutoRecord();  //automatically record if given volume threshold is reached

  stt.disableAutoThreshold();  //disables analysis of the enviornmental volume after STT

  stt.setThreshold(90.0);  //sets threshold that is used for speech recognition

  minim = new Minim(this);  //initiate minim for code

  snip = minim.loadSnippet("TruckBackingUpBeep.mp3");  //access the .mp3 being used

  kinect = new SimpleOpenNI(this); //use SimpleOpenNI for code

  kinect.enableDepth();  //initialize kinect depth camera

  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);  //enable user tracking
  kinect.setMirror(true);
  port = new Serial(this, "COM5", 9600);
  poseRight = new SkeletonPoser(kinect);
  poseLeft = new SkeletonPoser(kinect);
  poseNormal = new SkeletonPoser(kinect);
 //rules for right arm/hand
  poseRight.addRule(SimpleOpenNI.SKEL_RIGHT_HAND, PoseRule.ABOVE,SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  poseRight.addRule(SimpleOpenNI.SKEL_LEFT_HAND,PoseRule.BELOW, SimpleOpenNI.SKEL_TORSO);
  //rules for left arm/hand
  poseLeft.addRule(SimpleOpenNI.SKEL_LEFT_HAND,PoseRule.ABOVE,SimpleOpenNI.SKEL_LEFT_ELBOW);
  poseLeft.addRule(SimpleOpenNI.SKEL_LEFT_HAND,PoseRule.ABOVE,SimpleOpenNI.SKEL_LEFT_SHOULDER);
  poseLeft.addRule(SimpleOpenNI.SKEL_LEFT_ELBOW,PoseRule.LEFT_OF,SimpleOpenNI.SKEL_LEFT_SHOULDER);
  poseLeft.addRule(SimpleOpenNI.SKEL_RIGHT_HAND,PoseRule.BELOW,SimpleOpenNI.SKEL_LEFT_HIP);
  strokeWeight(5);
}
void draw() {
  kinect.update(); //update process with new kinect data
  PImage depth = kinect.depthImage(); //save depthImage to variable PImage
  image(depth, 0, 0); //draw the depthimage in the space 0,0
  IntVector userList = new IntVector(); //make a vector of ints to store list of users
  kinect.getUsers(userList);  //write the list of detected users into vector
  //if users are found
  if (userList.size() > 0) {
    //get the first user
   int userId = userList.get(0);
   drawSkeleton(userId); //draw the skeleton on the depth image
   if (kinect.isTrackingSkeleton(userId)) {
    //check to see if user is in pose for left hand and stop cart
    if(poseLeft.check(userId)){
      holdStop = 1;
     stroke(255,0,0);
       PVector rightHand = new PVector();
       PVector leftHand = new PVector();
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, rightHand);
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND, leftHand);
       PVector convertedRightHand = new PVector();
       kinect.convertRealWorldToProjective(rightHand, convertedRightHand);
       ellipse(convertedRightHand.x, convertedRightHand.y, 30, 30);
       xrightHand = int(leftHand.x);
       zrightHand = int(leftHand.z);
       xprint = xrightHand/10;
       zprint = zrightHand/10;
    } else{
      holdStop = 0;
  }

  if(poseRight.check(userId)){
    //drawSkeleton(userId);
    stroke(0,0,255);
       PVector rightHand = new PVector();
       PVector leftHand = new PVector();
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, rightHand);
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND, leftHand);
       PVector convertedLeftHand = new PVector();
       kinect.convertRealWorldToProjective(leftHand, convertedLeftHand);
       ellipse(convertedLeftHand.x, convertedLeftHand.y, 30, 30);
       xrightHand = int(leftHand.x);
       zrightHand = int(leftHand.z);
       xprint = xrightHand/10;
       zprint = zrightHand/10;
i = 1;
      }else{
  i = 0;
}
    //check to see if user is in the pose for right hand and allow right hand to control cart
    //if user is successfully calibrated
       stroke(0,255,0);
       PVector torso = new PVector();  //make a vector to store torso
       PVector leftHand = new PVector();
       PVector leftElbow = new PVector();
       PVector leftShoulder = new PVector();
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_HAND, leftHand);
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_ELBOW, leftElbow);
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER, leftShoulder);
       kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_TORSO, torso);  //put the position of torso into vector

       zposition = int(torso.z);  //z-coordinate corresponds to how far it is from kinect, convert to int and make the value be motorValue
       xposition = int(torso.x);  //x-coordinate cooresponds to how far it is from the origin being center of kinect
if(i == 0){
      xprint = xposition/10;
      zprint = zposition/10;
}
if (zprint > 255){
  zprint = 255;
}
xprint = xprint + 100;

if(xprint < 0){
  xprint = 0;
}
if(xprint > 200){
  xprint = 200;
}
//Arm Measurement
int leftShoulderZ = int(leftShoulder.z);
int leftHandZ = int(leftHand.z);
float foreArmLength = leftHand.dist(leftElbow);
foreArmLength = int(foreArmLength);
float humerusLength = leftElbow.dist(leftShoulder);
humerusLength = int(humerusLength=humerusLength);
int armLength = int(foreArmLength + humerusLength);
int distance = leftHandZ - leftShoulderZ;
```

```
distance = abs(distance);
if(distance > .75*armLength){
 if(toggleReturn == 0){
 toggleReturn = 1;
 if(toggleStop == 0){
  toggleStop = 1;
 }else{
  toggleStop = 0;
     }
   }
}
if(distance < .65*armLength){
  toggleReturn = 0;
}
if(start.equals(result) == true){
  voiceStop = 0;
}
if(stop.equals(result) == true){
  voiceStop = 1;
}
int stop = voiceStop+holdStop+toggleStop;
if(stop == 0){
  port.clear();
  port.write(xprint);
  port.write(zprint);
  //println("start");
    if(zprint < 120){
  snip.play();  //play snip for backwards
}
if(zprint > 120){
  snip.rewind();  //rewind
  snip.pause();  //pause
}
}
```

```
if(stop != 0){


 port.clear();
 port.write(100);
 port.write(140);
 //println("stop");
}
    }else{
      port.clear();
      port.write(100);
      port.write(140);
  }
  }
}
//draw limbs and joints for all skeleton objects
void drawSkeleton(int userId) {
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD, SimpleOpenNI.SKEL_NECK);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK, SimpleOpenNI.SKEL_LEFT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,SimpleOpenNI.SKEL_LEFT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW, SimpleOpenNI.SKEL_LEFT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK, SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER, SimpleOpenNI.SKEL_RIGHT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW, SimpleOpenNI.SKEL_RIGHT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER, SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER, SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO, SimpleOpenNI.SKEL_LEFT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP, SimpleOpenNI.SKEL_LEFT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE, SimpleOpenNI.SKEL_LEFT_FOOT);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO, SimpleOpenNI.SKEL_RIGHT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP, SimpleOpenNI.SKEL_RIGHT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE, SimpleOpenNI.SKEL_RIGHT_FOOT);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP, SimpleOpenNI.SKEL_LEFT_HIP);
}
//if skeleton is calibrated draw the joints for the user
```

```
void drawJoint(int userId, int jointID) {
  PVector joint = new PVector();

  float confidence = kinect.getJointPositionSkeleton(userId, jointID, joint);
  if(confidence < 0.2){
  return;
}
  PVector convertedJoint = new PVector();
  kinect.convertRealWorldToProjective(joint, convertedJoint);
  ellipse(convertedJoint.x, convertedJoint.y, 5, 5);
}
void drawLimb(int userId, int jointType1, int jointType2)
{
  PVector jointPos1 = new PVector();
  PVector jointPos2 = new PVector();
  float confidence;
  // draw the joint position
  confidence = kinect.getJointPositionSkeleton(userId, jointType1, jointPos1);
  confidence = kinect.getJointPositionSkeleton(userId, jointType2, jointPos2);
  line(jointPos1.x, jointPos1.y, jointPos1.z,jointPos2.x, jointPos2.y, jointPos2.z);
}
// user-tracking callbacks!
void onNewUser(int userId1) {
  println("start pose detection");  //prints line to console to report whats happening
  kinect.startPoseDetection("Psi", userId1);  //check to see if user is in start position
}

//check current user successful, if successful skeleton data is available
void onEndCalibration(int userId, boolean successful) {
  if (successful) {
    println(" User calibrated !!!");  //report user calibrated
    kinect.startTrackingSkeleton(userId);  //start the skeleton tracking
  }
  else {
    println(" Failed to calibrate user !!!");  //if calibration failed report to user in console
    kinect.startPoseDetection("Psi", userId);  //restart calibration to detect start pose
  }
}
//user has been detected, check for user position
void onStartPose(String pose, int userId) {
  println("Started pose for user");  //print out results on sketch
  kinect.stopPoseDetection(userId);  //after user has assumed detection pose stop tracking
  kinect.requestCalibrationSkeleton(userId, true);  //find all user's joint positions
}
//method is called if transcription was successfull
void stop()
{
  // always close Minim audio classes when you are done with them
  snip.close();
  // always stop Minim before exiting
  minim.stop();
  super.stop();
}
//method is called if transcription was successfull
void transcribe (String utterance, float confidence)
{
  println(utterance);
  result = utterance;
}
```

Skeleton Poser

```
class PoseRule {
  int fromJoint;
  int toJoint;
  PVector fromJointVector;
  PVector toJointVector;
  SimpleOpenNI context;
  int jointRelation; // one of:
  static final int ABOVE = 1;
  static final int BELOW = 2;
  static final int LEFT_OF = 3;
  static final int RIGHT_OF = 4;
  PoseRule(SimpleOpenNI tempContext,
  int tempFromJoint,
  int tempJointRelation,
  int tempToJoint)
{
  context = tempContext;
  fromJoint = tempFromJoint;
  toJoint = tempToJoint;
  jointRelation = tempJointRelation;
  fromJointVector = new PVector();
  toJointVector = new PVector();
}
boolean check(int userID){
  // populate the joint vectors for the user we're checking
  context.getJointPositionSkeleton(userID, fromJoint, fromJointVector);
  context.getJointPositionSkeleton(userID, toJoint, toJointVector);
  boolean result= false;
  switch(jointRelation){
  case ABOVE:result = (fromJointVector.y > toJointVector.y);
  break;
  case BELOW:result = (fromJointVector.y < toJointVector.y);
  break;
  case LEFT_OF:result = (fromJointVector.x < toJointVector.x);
  break;
  case RIGHT_OF:result = (fromJointVector.x > toJointVector.x);
  break;
}
return result;
  }
}
class SkeletonPoser {
  SimpleOpenNI context;
  ArrayList rules;
SkeletonPoser(SimpleOpenNI context){
  this.context = context;
  rules = new ArrayList();
}
void addRule(int fromJoint, int jointRelation, int toJoint){
  PoseRule rule = new PoseRule(context, fromJoint, jointRelation, toJoint);
  rules.add(rule);
}
boolean check(int userID){
  boolean result = true;
  for(int i = 0; i < rules.size(); i++){
    PoseRule rule = (PoseRule)rules.get(i);
    result = result && rule.check(userID);
    }
return result;
  }
}
```

Appendix B: Arduino Uno Code

```
#include <PID_v1.h>

#include <NewPing.h>

#include <SabertoothSimplified.h>

SabertoothSimplified ST;


unsigned int frontLeft = 7;

unsigned int backLeft = 8;

unsigned int frontRight = 12;

unsigned int backRight = 11;


int A = 1;

int B = 2;


//Sensor Noise Filter

int FR1, FR2, FR3;

int FL1, FL2, FL3;

int BR1, BR2, BR3;

int frCount, flCount, brCount, flRangeAvg, frRangeAvg, brRangeAvg;


//X PID Variables

float Xi, Xp, Xd, XE, XEr, XPrev_Error, XI, XInt, XD, XDer, ROut, LOut, XOutput;


//Z PID Variables

float Zi, Zp, Zd, ZE, ZEr, ZPrev_Error, ZI, ZInt, ZD, ZDer, ZOutput;

//Sonar Setup

NewPing sonar1(frontLeft,frontLeft,60);

NewPing sonar3(frontRight,frontRight,60);

NewPing sonar4(backRight,backRight,60);


unsigned int in1, in2, in3, in4;


void setup() {
  Serial.begin(9600); // Start serial communication at 9600 bps
  ST.motor(A, 0); //Initialize motors at a stop
  ST.motor(B, 0);
  Zp = .8;
  Zd = 1;
  Zi = .001;
  Xp = .7;
  Xd = .7;
  Xi = .001;
  ZPrev_Error = 0;
  ZI = 0;
  ZE = 0;
  ZD = 0;
  XPrev_Error = 0;
  XI = 0;
  XE = 0;
  XD = 0;
  frCount = 0;
  flCount = 0;
  brCount = 0;
}
void loop() {
  unsigned int in1 = sonar1.ping();
  unsigned int in3 = sonar3.ping();
  unsigned int in4 = sonar4.ping();




  if (Serial.available() > 1) { // If data is available to read
    signed int valx = Serial.read(); //Read in X position value
    signed int valz = Serial.read(); //Read in Z position value
    unsigned int fL = in1 / US_ROUNDTRIP_CM;
    unsigned int fR = in3 / US_ROUNDTRIP_CM;
    unsigned int bR = in4 / US_ROUNDTRIP_CM;
```

```
 if(valx == 100 && valz == 140){
   ROut = 0;
   LOut = 0;
   ZI = 0;
   XI = 0;
 }
 else{
   ZE = valz - 140;
   if(ZE > 0){
     ZE = map(ZE, 0, 115, 0, 128);
   }
   else{
     ZE = map(ZE, -50, 0, -75, 0);
   }
```

//PID Controller

```
   //Proportional Error
   ZEr = ZE*Zp;
   //Integral Error
   ZI = (ZE + ZI);
   ZInt = ZI*Zi;
   //Derivitive Error
   ZD = ZE-ZPrev_Error;
   ZDer = ZD*Zd;
   ZPrev_Error = ZE;
   ZOutput = ZDer+ZInt+ZEr;
   if(ZOutput > 120){
     ZOutput = 120;
   }
   if(ZOutput < -120){
     ZOutput = -120;
   }
   XE= valx - 100;
   XEr = XE*Xp;
   //Integral Error
```

```
XI = (XE + XI);
XInt = XI*Xi;
//Derivitive Error
XD = XE-XPrev_Error;
XDer = XD*Xd;
XPrev_Error = XE;
XOutput = XDer+XInt+XEr;
ROut =  ZOutput + XOutput;
LOut =  ZOutput - XOutput;
```

//Ping Sensor Filter

```
if(fL > 0){
  if(flCount == 0){
    FL1 = fL;
    flCount = 1;
  }
  if(flCount == 1){
    FL2 = fL;
    flCount = 2;
  }
  if(flCount == 2){
    FL3 = fL;
    flCount = 0;
  }
  flRangeAvg = (FL1 + FL2 + FL3)/3;
  if(flRangeAvg < 30 && flRangeAvg > 25){
    if(ROut > 0){
      ROut = 0;
    }
    if(LOut > 0){
      LOut = 0;
    }
  }
  if(flRangeAvg >= 30){
    flCount = 0;
  }
}
if(bR > 0){
  if(brCount == 0){
    BR1 = bR;
    brCount = 1;
  }
  if(brCount == 1){
    BR2 = bR;
    brCount = 2;
  }
  if(brCount == 2){
    BR3 = bR;
    brCount = 0;
  }
  brRangeAvg = (BR1 + BR2 + BR3)/3;

  if(brRangeAvg < 40 && brRangeAvg > 25){
    if(ROut < 0){
      ROut = 0;
    }
    if(LOut < 0){
      LOut = 0;
    }
  }
  if(brRangeAvg >= 40){
    brCount = 0;
  }
}
if(fR > 0){
  if(frCount == 0){
    FR1 = fR;
    frCount = 1;
  }
```

```
    if(frCount == 1){
     FR2 = fR;
     frCount = 2;
    }
    if(frCount == 2){
     FR3 = fR;
     frCount = 0;
    }
    frRangeAvg = (FR1 + FR2 + FR3)/3;
    if(frRangeAvg < 30 && frRangeAvg > 25){
     if(ROut > 0){
      ROut = 0;
     }
     if(LOut > 0){
      LOut = 0;
     }
    }
    if(brRangeAvg >= 40){
     brCount = 0;
    }
   }
  }
  ST.motor(A, LOut);
  ST.motor(B, ROut);
 }

}
```

Appendix C: User Guide – John Johnson

Introduction

Congratulations on your purchase of the Au Smart Cart. We are confident that you will be pleased and highly impressed with user friendliness commands of this cart. We want for you to experience all the benefits and operate the cart safely so please read this manual carefully.

As you read this manual certain keys words in Bold print ex: "**Important and Remember**" are used as caution indicators to follow directions carefully.

User Guide Table of Contents

Cart Safety

IMPORTANT SAFETY INFORMATION

Most accidents with this autonomous shopping cart can be prevented if you follow all instructions in this manual. The most common hazards are discussed below, along with the best way to protect yourself and others.

Avoid Throwing Items in the Cart

Throwing items inside the cart can cause serious damages to the Kinect and to the computer screen. Carefully place items in the cart. Before placing items in the cart make sure you do one of the following commands: left hand above left shoulder, right hand extended out to toggle commands or say "Stop" to halt cart. Failure to do so, the cart will move backwards and could cause damage anything behind it.

Clear Rear Area

Even though the cart has a beep signal to indicate backing up, **NEVER** walk towards the cart without making sure the rear area is clear. This could cause serious damage to a small child or anyone who is unaware the coming towards them.

Avoid Unnecessary Hand Movements

Unnecessary hand movements can endanger other people or cause damages to cart. The cart responds to both left and right hands so if wrong movements can make the cart hit someone or bump into other objects. Only raise either hand above waist when needed to ensure safety of others and the cart.

Keep Wheels free from any Debris

Do not stick fingers or any items in between wheels. The wheels are constantly turning and this could harm you or damage the cart.

Emergency Shut Down

If at any time you feel your safety is at risk, or the cart is functioning improperly, press the red estop in the front of the cart for immediate shut down.

Turn Cart Off When Not In Use

After you are done using the cart, it is you're responsibly to the turn the cart off. Never leave the cart unattended or running. If cart is not turned off properly this could cause damages to the internal parts of the cart or drain the battery.

IMPORTANT MESSAGE TO PARENTS

YOUR CHILDREN’S SAFETY IS VERY IMPORTANT to us at Auburn University.That’s why we urge you to read this message before letting your child operate this shopping cart. This shopping cart is a privilege, not a toy. As with any equipment, bad judgments can result in serious injuries. You can help prevent accidents by making good decisions about if,when, and how your youngster operates this equipment.

The first question you’ll need to ask is whether your youngster is capable of operating this cart safely. Remember, young people vary widely, and AGE IS NOT THE ONLY FACTOR.

Physically, a youngster must be LARGE ENOUGH AND STRONG ENOUGH to easily start the cart and control its direction. The youngster also needs enough size, strength, and coordination to comfortably reach and operate the controls.

Another, tougher question you need to ask is if your youngster has enough MATURITY AND RESPONSIBILITY to safely operate this cart SUPERVISION is also very important. Walk with your youngster during the first few minutes of using the cart. Even after the youngster hasbecome confident with the cart, do not let the young person use the chart without good adult supervision. It’s up to parents to make sure that the cart is properly maintained and kept in safeoperating condition.

By always placing safety first, your youngster will acquire useful skills and a sense of accomplishment. And you’ll both get the best results from the cart.

Controls

COMPONENT IDENTIFICATION

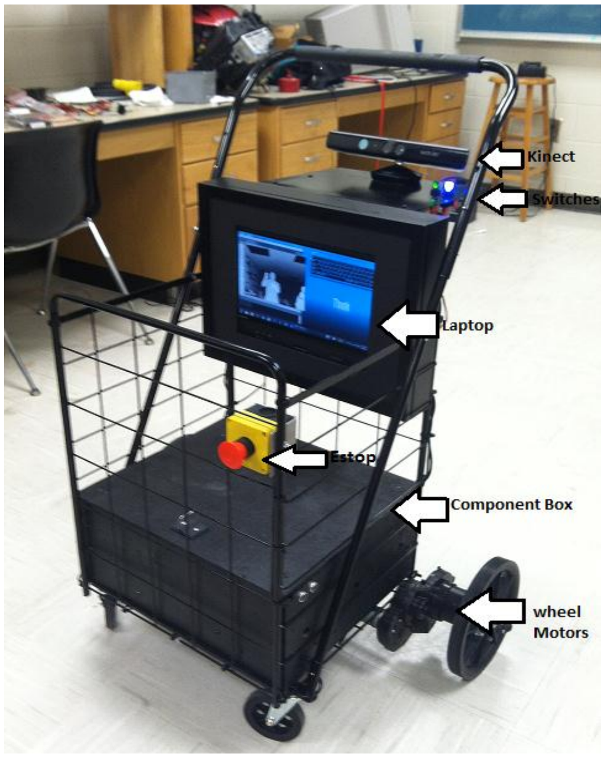


Figure 9. AU Smart Cart Component Layout

DESCRIPTION OF CONTROLS

Motor Battery and Kinect Battery

The motor battery provides power to the motor controller and fan.

Figure 10. The Motor Battery

The Kinect battery provides power to the Kinect.

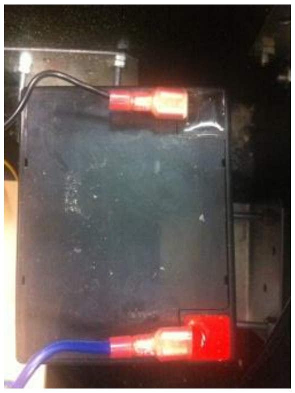


Figure 11. The Kinect Battery

Switches

The switches give power to the laptop and Kinect. Both switches the must be ON to start and operate the cart. Both switches should be kept OFF when cart is not in use.

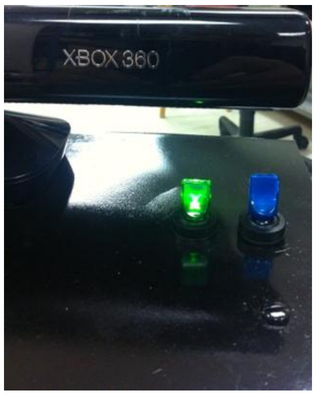


Figure 12. The AU Smart Cart Power Switches

Kinect

The Kinect is use to detect: your skeleton (body) ,measure the distance between you and the car, and listen to voice commands. It can see people standing between 0.8 meters (2.6 feet) and 4.0 meters (13.1 feet) away.

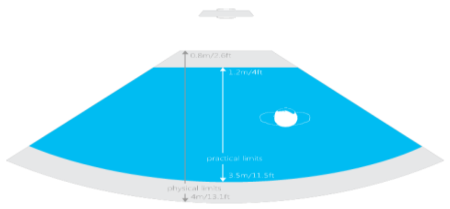


Figure 13. Kinect Field of View **Figure 14. The Microsoft Kinect**

Laptop

The Laptop is used to show the user what the Kinect recognizes and also to better help the user align him or herself for initial start position.

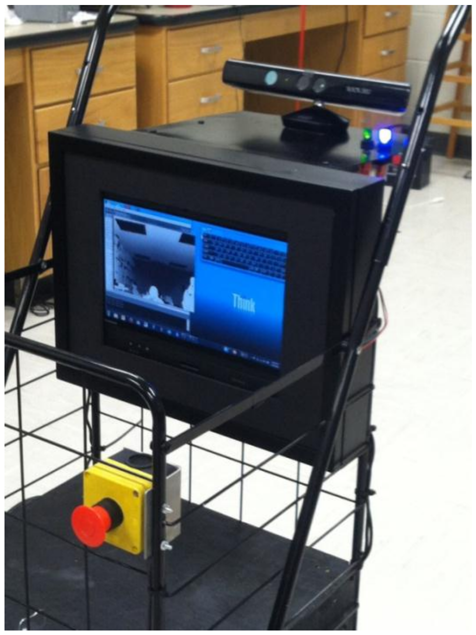


Figure 15. The Tablet Laptop

<u>Before Operation</u>

ARE YOU READY TO SHOP?

Be sure you are not holding anything in your hand. Cellular devices should not be used while operating this cart. For proper usage, your undivided attention is needed at times.

CHECK YOUR SURROUNDING AREA

For your safety and for the safety of others, always inspect the areabefore using the cart.

Objects

Anything that is in your path that might problems should be moved so that the cart will not have any issues tracking its user.

People and Pets

People and animals that can move into your path or into a position could be hit by the cart. Clear the area of people and pets, especially children. Their safety is your responsibility.

<u>Operation</u>

STARTING THE CART

The cart is ready for usage when you flip ON both switches.

1. Turn both switches to the ON position. The **GREEN** switch powers the Kinect and **BLUE** switch powers the fan and microcontroller. The switches should now light up indicating the cart is on and has power as shown in Figure 16.

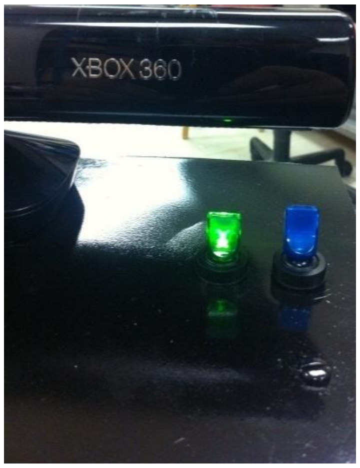


Figure 16. Shows both switches in the ON position for cart to begin commands.

2. Move directly in front of the cart. The cart will only begin to track the user skeleton after Figure 17 position is assumed.

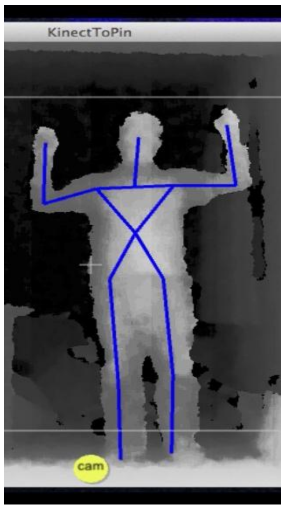


Figure 17. Displays position user must take for Kinect to track and start the cart.

3. Once Kinect has locked onto the user, you can now drop arms and began to shop as shown in Figure 18.

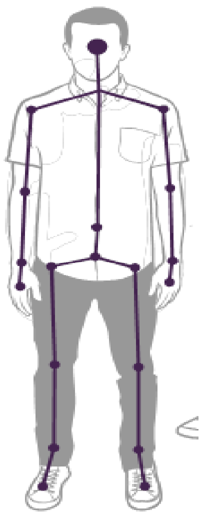


Figure 18. Showing relaxed user position for Kinect correct tracking.

4. If you want the cart to **STOP** for any reason, raise your left hand above left shoulder. The cart will remain stopped as long as your left hand is raised as shown in blue circle Figure 19.

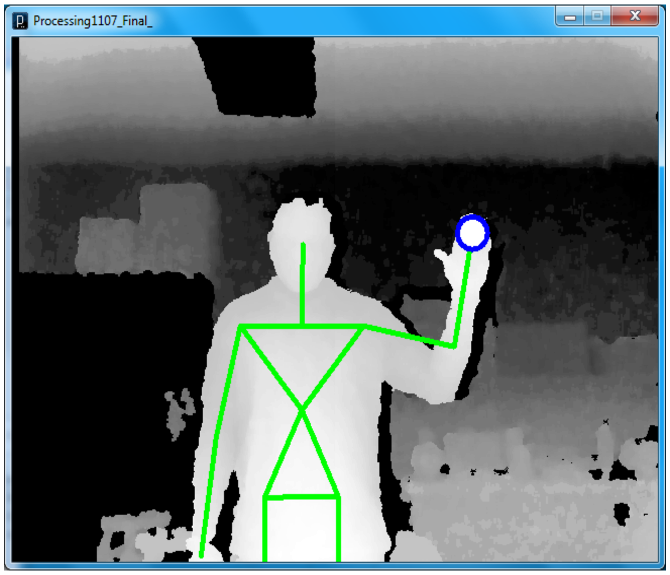


Figure 19. Displays left hand above left shoulder displaying how to stop the cart.

5. If you want to temporarily disable or **STOP** the cart for any reason, raise your left hand out fully extended towards cart. This will toggle the cart (pause). The cart will remain in this position until your left hand is extended again for the cart to resume commands.
6. If you want the cart to move left or right without moving your entire body, just raise your right hand about your right shoulder as shown in red circle Figure 20.

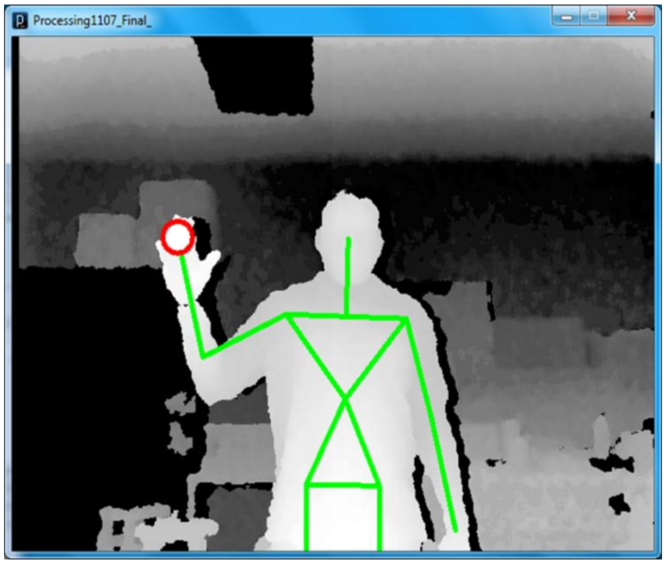


Figure 20. Displays the cart following in any direction when right hand is raised.

7. The cart will respond to voice commands such as **START** and **STOP.** The cart will follow only those two commands. Do not give multiple commands at once. The cart will only recognize one command at time.
8. The cart also has an E stop button. If for any reason you want to disable the cart press the red button as shown in Figure 21.

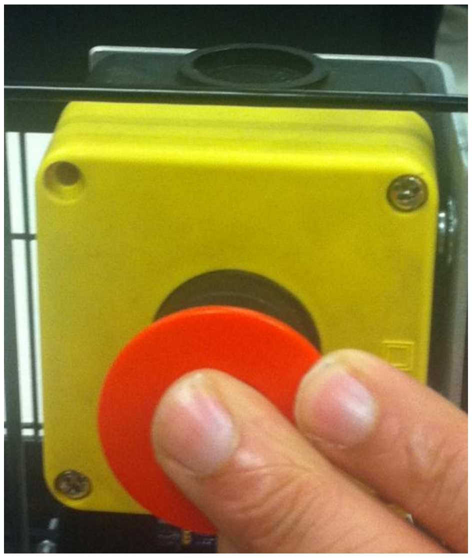


Figure 21. Displays E-stop pressed to disable cart.

Troubleshooting:

1. If the cart is not responding to any commands, make sure both switches are on and are lit. If not you have a power issue and the cart needs to be returned Auburn University for repairs.
2. If the cart loses your skeleton, press the E-stop to shut don't the cart and turn off switches and repeat instructions on page 12.

Storage:

The cart should be stored in dry and cool temperature room.