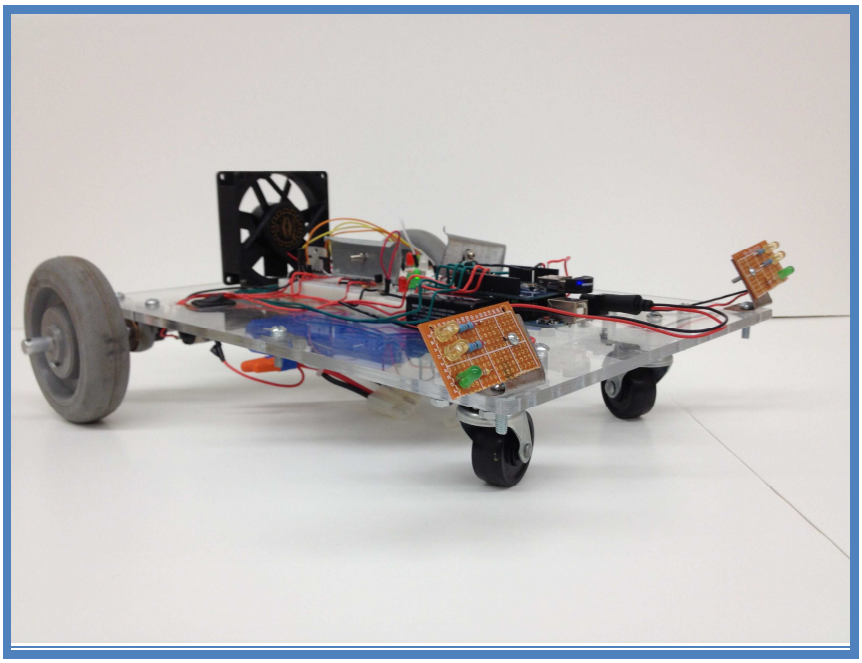
Follower Cart



Department of Mechanical and Aerospace Engineering
ME 106Mechatronics
Professor Furman

Designers:
David Coyle
Allan Glover

Table of Contents

SUMMARY

For ME106 Term Project for Fall 2012, our team designed a prototype for a cart that could follow a wheel chaired individual around a store without any input from the individual. Other initial design concepts included RFID locking or door opening systems, automated drip irrigation systems, hands-free sink faucet, etc. The cart follower was finally decided upon as it seemed the most useful in everyday life for a person with limited facilities (see Figure 1).

Once the project idea was finalized, the system design began to take shape. The first solution involved transforming a small radio flyer using a motor for the drive and a servo to control the front axle with a single wheel. This was followed by a design with a front axle converted to a single pivoting column-to-wheel for steering. It was then decided that a custom design chassis would be better than using a radio flyer for later manipulations. For the final design an acrylic sheet was cut, and two motors were used in the rear to control both the drive and the direction. This eliminated the need for a servo and provides the ability for sharp turns and zero-radius turning.

In parallel to the mechanical system being designed, the programming process was underway; the key to success was designing the system to function with no input from the user. One of the early designs called for the use of an accelerometer to provide linear acceleration combined with infrared LED (IR) light(s) being mounted on the moving object to track distance. In this design, it was presumed the accelerometer would provide left-right motion data (x-direction) as the object *accelerated* to the left or right. The IR lighting would be used to read distance by reading intensities, thereby giving distance (y-direction) data. At this point it was not known how the data would be projected to the Arduino.

Nintendo's Wii remote provided a platform that had both accelerometers built in and used IR data sensor which it was assumed would provide distance data. However the accelerometer did not provide accurate linear acceleration. It was found that a gyroscope and compass were needed in addition to the accelerometers to obtain linear acceleration. While trying to find a new solution, a previously unknown feature of the Wii remote, the IR camera, was discovered which provided the solution for getting both direction and distance data in a steady stream. The camera in the Wii remote reads the two brightest IR objects it sees and provides location and intensity of two pairs of front mounted IR light emitting diodes (LEDs). Additionally it operates using Bluetooth communication. It was found that Bluetooth shields with Wii remote libraries could be purchased to attach directly to the Arduino Uno. The solutions to both design and programming were now in hand, the next step was the integration process.

The construction of the vehicle involved laser cutting the chassis using the CAD laser cutter in the ME 106 Mechatronics lab at San Jose State University (SJSU). We also laser cut custom gears using CAD drawings downloaded from sdp-si.com, to minimize cost because the gear bore size and diametrical pitch specified in our drawings were expensive. Calculations were performed to determine the torque, power, and RPM requirements for the motor. From there the power interface power interface and electrical system was assembled. There was in incident with a short

circuit and we had to replace our batteries. We learned first-hand the importance of a fuse to protect from overload. The code was being written simultaneously to assembly, and each system was tested during construction. Final testing and program adjustment was made during the last week before project presentations with the system operating correctly almost immediately.

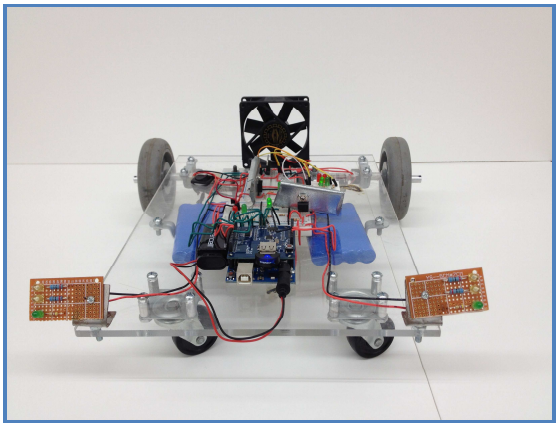


Figure 1. Front View of Cart

INTRODUCTION

The basic requirements of this project were to program an Arduino Uno microprocessor to received input from at least one sensor (type undetermined) and provide outputs to at least one motor or actuator to solve a particular problem.

The Follower Cart project combined and added to the required elements. It was a proof-of-concept that can be expanded in scale to be used, for example, as a grocery cart to follow a person in wheelchair within a certain range. As currently designed, an IR camera similar to the one employed in the Wii remote would be mounted to the rear of a wheelchair or scooter and would follow. This process could not require any input from the person in the chair. As yet this sort of cart is not available in any commercial outlet, so if the proof-of-concept could be shown to be operational, the scaled up model could be commercially produced.

When the idea was initially conceived it was designed with user inputs through a Wii nun chuck (See Figure 2). It was thought the small wireless joystick would be easily manipulated by the user. Further brainstorming allowed the team to conceive of the idea of a fully independent of user input, it could simply follow the user.

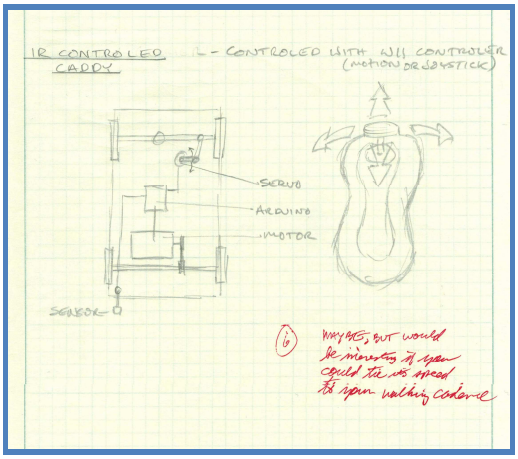


Figure 2. Initial Idea Submitted to ME106

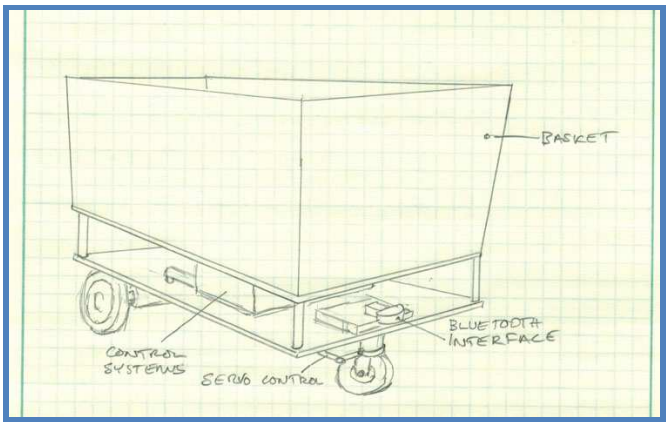


Figure 3. Early Design with Cart Basket

Once it was decided that the system should be independent, research into available equipment lead the lead to decide on a system that would use the accelerometer of the Wii remote, a IR range finder, and a USB shield with an attached dongle that would communicate position of the user to the Arduino which would then process the inputs and output signal to the control system. The first design was to use a single motor for drive and a servo mounted on a caster to control direction (see Figure 3). The team finally decided on controlling and driving the cart using two motors driving two independently driven axles at the rear of the cart, with a pair of rotating casters in the front of the cart (see Figure 4).

Upon experimentation with the Wii remote, it was determined accelerometer data would not work for a true hands-off control. The data it provided depended on the remote being tilted along it's x-,y-, and z-axis's which therefore required user input. Linear acceleration is not possible without the use of a gyroscope and a compass.

Therefore, instead of using the accelerometer in the Wii remote, the system would use the IR camera and input data from the Wii remote data transmitted via Bluetooth to the Arduino Uno. This IR camera system allowed IR lights mounted on the front of the cart to be tracked by the IR camera software, not only providing x-direction data, but by reading the intensity of the lights, provided y-direction data. In the end, intensity was not able to be extracted for the Wii remote data stream, and so we used the y-direction data to calculate the distance. In order to do this the Wii remote must be mounted at an angle.

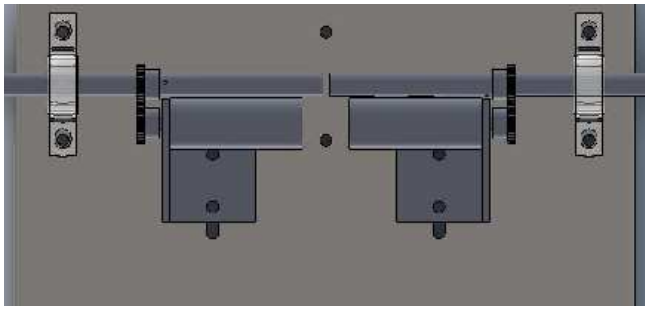


Figure 4. Design of Independent Axles on Acrylic

SPECIFICATIONS

It was decided that for the proof-of-concept, a basket would not be required, so the team focused on designing on board a flat acrylic sheet which could be laser cut to exact specifications (see Appendix 6) however the system was designed to carry approximately 10-12 lbs outside of its own weight, which if scaled up 6-7 times, would allow it to carry approximately 70 lbs.

The speed of the concept cart was designed to mimic a person's walking speed of approximately 2-3 mph. Though a person in a wheel chair could potentially move faster than this, it was decided the scaled up version could incorporate higher speeds if required. This concept cart also only operates in forward direction. It was decided that concept of the project would be proven if it operated correctly in the forward direction. Reversing operations could be added later.

Finally, it was decided that the system would only begin to operate when the user was 3+ ft from the cart. The distance was a very arbitrary decision however having a "begin running" distance was important because the user could be moving very short distances back and forth while browsing and the cart could jerk violently while mimicking these movements. It would allow very purposeful control of the cart. Furthermore it was decided that the cart should not constantly twist back in forth in search of the IR beacon. It must operate in a smooth manner with smooth radius turns.

Further additions that could or would have to be added to an actual operating car would include:

- Multiple speeds
- Obstacles avoidance
- Dead-man switch

DESIGN

Figure 5 shows the block diagram for the car. In table 1, the components of the cart are documented.

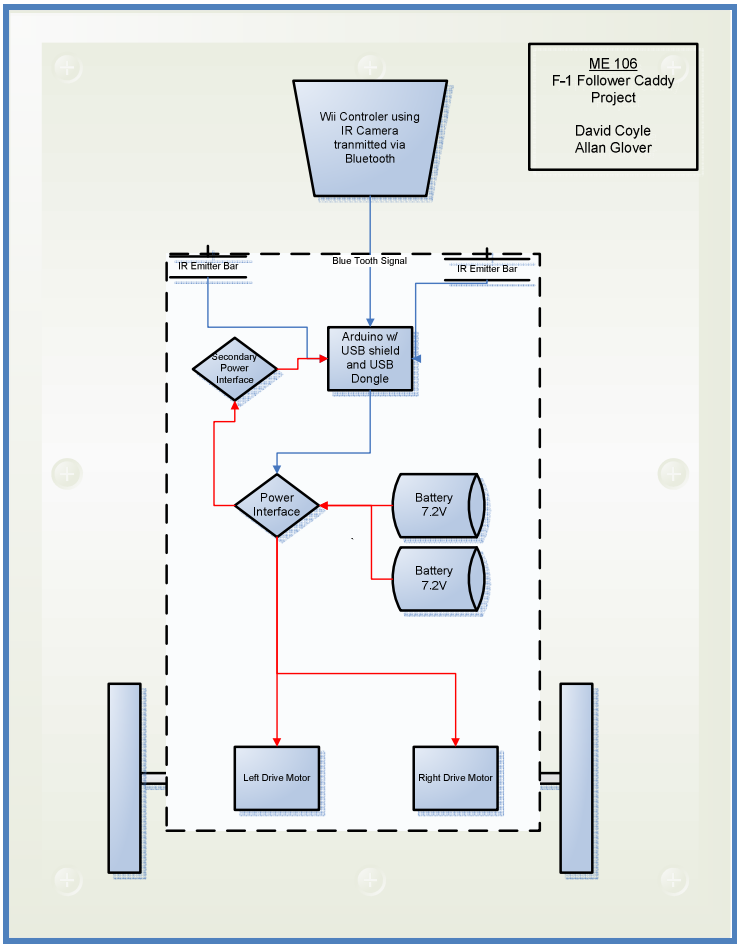


Figure 5. Block Diagram Schematic of Cart

Table 1. Cart Components

COMPONENTS	PURPOSE
POWER 2 x Rechargable Batter packs	Provide more than 14.5v and more than 1400mAh
POWER REGULATOR 1 x LM350 (3 Amp rating) 1 x LM317 (1 Amp rating)	 LM350: Receive >14.5v from batteries and provide constant 12.5v for drive system and supply voltage for LM317 LM317: Provide 7.5v for IR LED's and Arduino
DRIVE 2 x Hennkwell IND CO DC Gear Motors	Provide approx. 67 oz-in T_{rms} at 40-60rpm at max voltage input of 12.5 v
CONTROLER Arudino UNO USB 2.0 Shield w/ USB Bluetooth dongle	Connect to Wii remote via Bluetooth shield for Arduino. Interpret the data from Wii remote and provide overall control of cart user free
MOTOR DRIVER L298 Full-Bridge Driver Multiwatt 15 Package	Take input from Arduino input and provide power to each motor
DEVICES Wii remote IR Light Mounts	Use IR camera functionality to relay to cart position and direction of movement by tracking the intensity of the IR lights mounted on the cart Provide IR lights

Underside

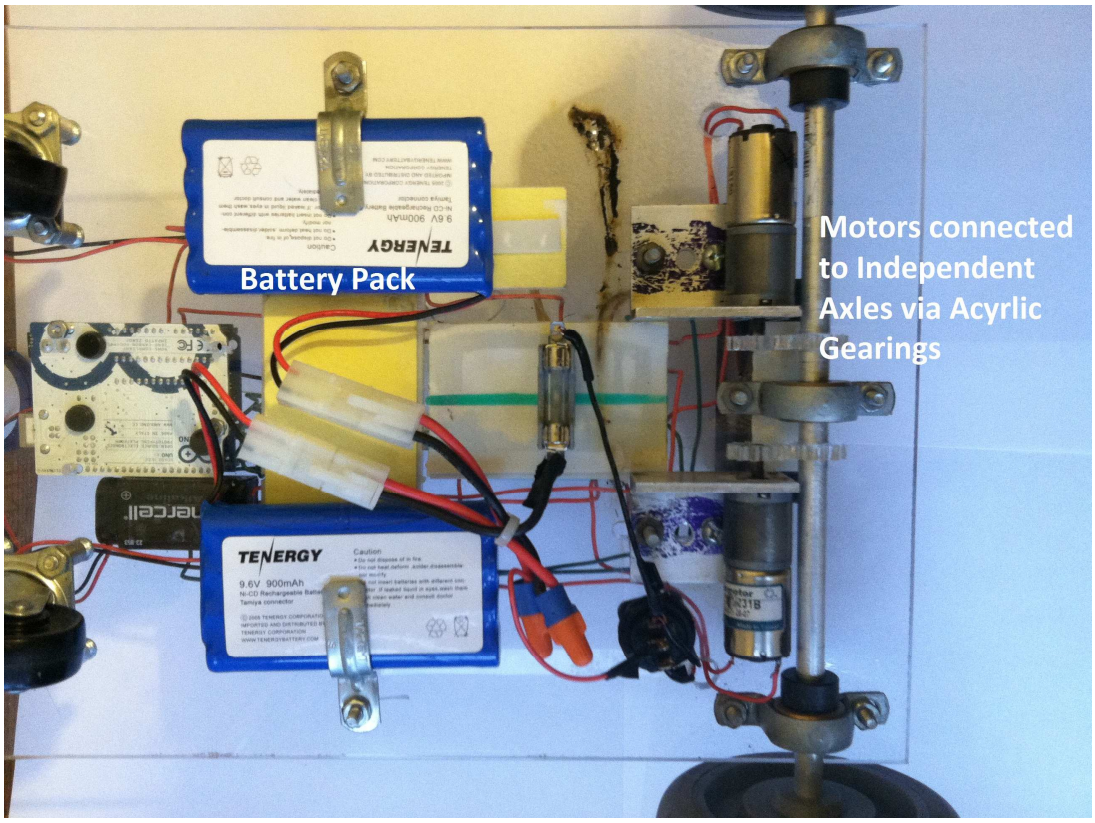


Figure 6. Underside of Cart (note Batteries, Motors, Gearing, Independent Axles

Chassis

Chassis was designed to be rectangular in a similar ratio to a shopping cart, 10" x 14". It is constructed of ¼" clear acrylic and cut using the CAD laser cutter in the ME106 lab (see Figure 7 and Appendix 6 for drawings). Additional holes were drilled as needed to run wiring.

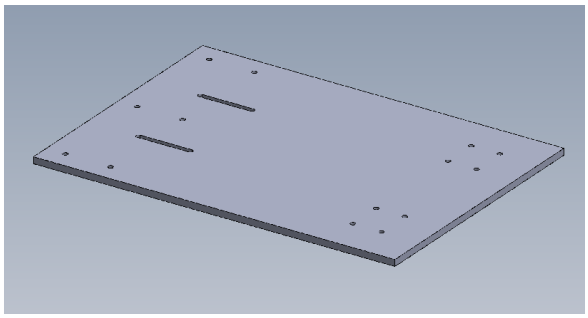


Figure 7. CAD Drawing of Acrylic Chassis

The chassis is supported by three pillow block bearings and 4" Colson wheels purchased from McMaster-Carr.com. The axle is 3/8" aluminum stock. See the Bill Of Materials (BOM) in Appendix 5.

Motor mounts were custom machined out of aluminum L-bracket (See Figures 8, 9, and Appendix 6). These rigid motor mounts are capable of securing the motor under hi-torque. The motor mounts were attached by bolts passing through the motor mount and motor positioning slots in the acrylic chassis. The slots allowed for precise positioning of the motor and room for adjustment if different size gears were used.

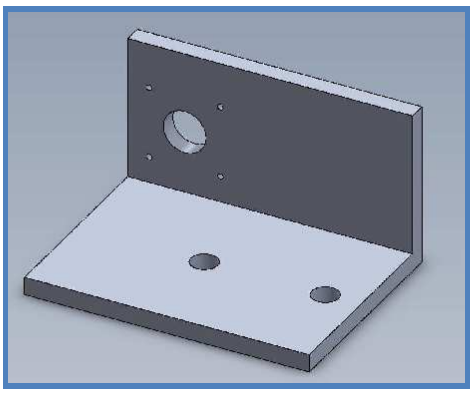


Figure 8. CAD Drawing of Motor Mounts

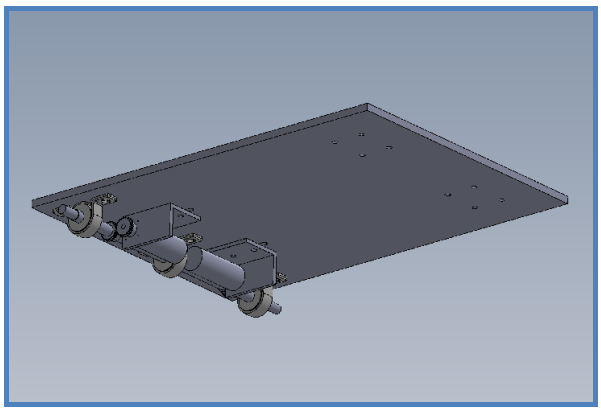


Figure 9. Motor Assembly

Power System

Battery Packs: Two standard rechargeable battery packs supplied power to the motor via power regulator. They were wired in series and provided 19.6V in total. A fuse of 1.5A is connected in series to protect against over draw from the motors or any failure resulting in high current draw

Drive

Motors: Mechanical power was supplied by two Hennkwell IND CO DC Gear Motors which were powered independently. The motors supply a stall torque of 343 oz-in and a continuous torque at 50 rpm of approximately 130 oz-in (see cut sheet in Appendix 1). The required rms torque is approximately 50 oz-in, and the max torque required is approximately 100 oz-in. This was determined to be sufficient for the needs of the cart carrying a load of approximately 10-12 lbs. (see calculations in Appendix 2). Two 1N4001 Diodes were added across the motor terminals to handle any back EMF from the motors stopping (see Figure 12 and 13 Wiring Diagram) Acrylic gears were also cut on the CAD laser cutter to transfer power from the motors to the axles (see Figure 10).

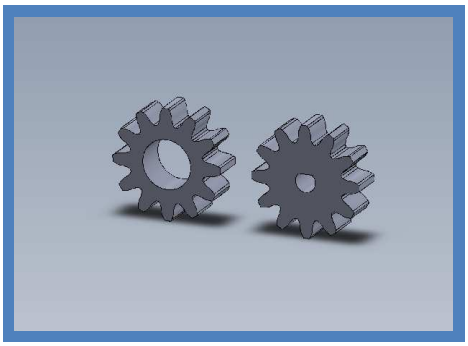


Figure 10: CAD model of laser cut Acrylic gears

Top Side

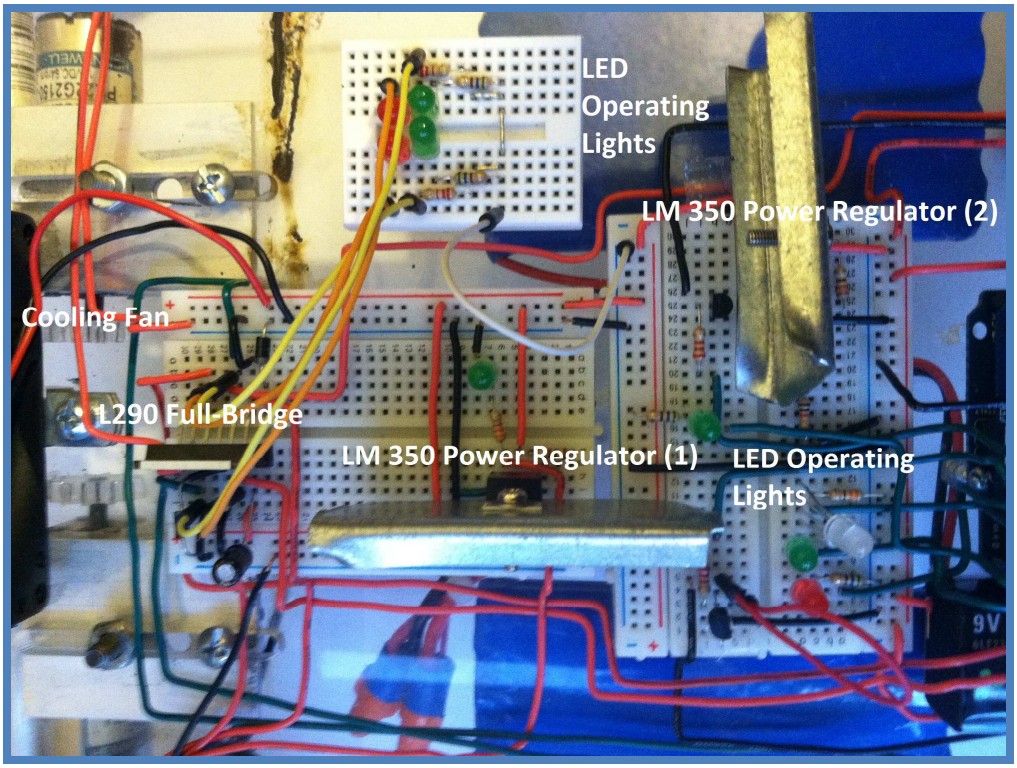


Figure 11. Overhead Shot of Cart

Power Regulators: Initially a single LM350 power regulator was used to reduce the voltage to approximately 12.5V. Later a second LM317 regulator was added to further reduce the voltage to approximately 7.5V to supply power to the IR system and the Arduino microcontroller (see Appendix 4 and Figures 11). When batteries were at full charge the LM350 dissipates about 8 Watts and so a crude sheet metal cooling fin was added. The LM317 also dissipates a substantial amount of heat due to its lower amperage rating; therefore a cooling fin was also attached to it.

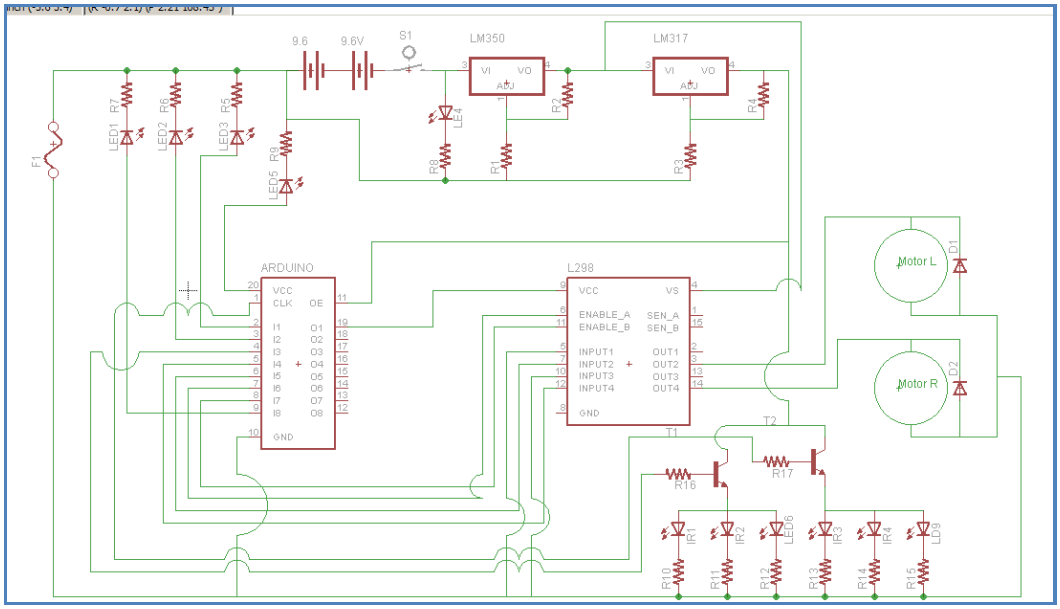


Figure 12. Wiring Diagram

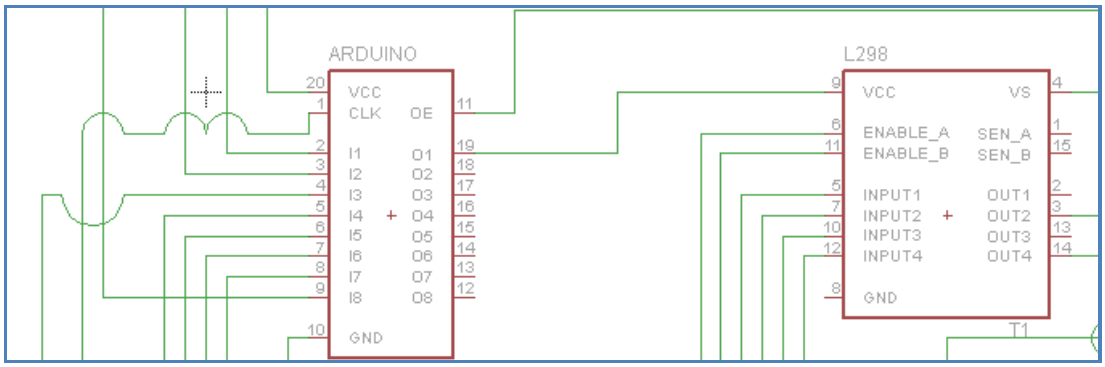


Figure 13. Wiring Diagram Close up (pin 19 is +5V out and pin 11 is +7.5V in. Pin 20 should be pin 0)

Table 2. Wiring Diagram

Schematic Name	Function	Part # / Model	Value
R1	LM 350 adj.		730 Ω
R2	LM 350 reference		120 Ω
R3	LM 317 adj.		890 Ω
R4	LM 317 reference		220 Ω

R5	GREEN status LED		560 Ω
R6	BLUE status LED		320 Ω
R7	RED status LED		560 Ω
R8	PWR GREEN status LED		3.3 kΩ
R9	Wii.Connect status LED		560 Ω
R10	IR LED 1 Resistor		68 Ω ½ W
R11	IR LED 2 Resistor		68 Ω ½ W
R12	IR status LED 1 Resistor		410 Ω
R13	IR LED 3 Resistor		68 Ω
R14	IR LED 4 Resistor		68 Ω
R15	IR status LED 2 Resistor		410 Ω
R16 – R17	Transistor base Resistors		2.2 kΩ
LED1		Generic	Red LED
LED2		Generic	RGB LED
LED3 – LED7		Generic	Green LED
IR1-IR4	Infrared Emitter	OP 290A	890nm High intensity IR LED's (150 mA continuous)
D1 – D2	Back EMF protection Diodes	1N4001	1A
T1 – T2	IR LED transistors	P2N2222A	300 mA
L298	Motor Driver	L298	
F1	Fuse		1.5 A

Control System

Microcontroller Hardware: An Arduino Uno R3 microcontroller was used to control the Cart Follower. A USB shield built and sold by Circuits@Home.com acted as the interface between the Arduino and the Wii remote (see Figure 14). The dongle received Bluetooth communication from the Wii remote, and the USB shield passed this information to the Arduino. For the Arduino Uno and USB shield to function, it utilized over ten (10) libraries supplied by Circuits@Home. They can be downloaded from github.com. Due to the length of the libraries, the library names only with web links are listed in Appendix 7. The USB shield uses pins 8 through 13, however we were still able to use pins 8 and 9 for status LED's.

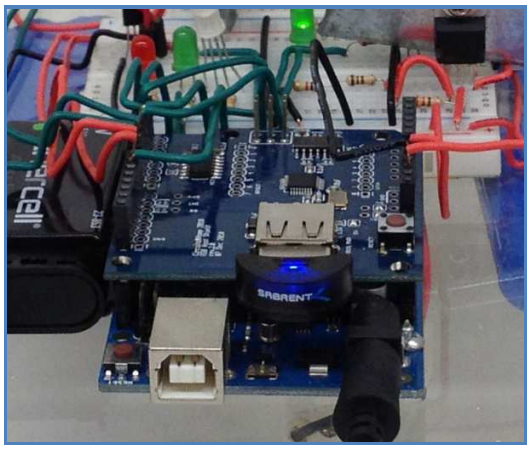


Figure 14. Close-up of USB Shield Mounted on Arduino (note LED Light Cluster Behind: Steady Green LED indicates Wii remote connected)

Several more libraries are needed for Bluetooth connectivity and access to the Wii remote also listed in Appendix 7.

Wii remote Library: A Wii remote library (hereafter referred to as the "Wii" library) was written by Kristian Sloth Lauszus of TKJ Electronics in Denmark. His code is available for free on the popular website github.com. However this Wii library did not include any code for interfacing the IR camera and the Arduino. The code was written by our team to initialize, connect, and read data from the IR camera. Mr. Lauszus was consulted during the code writing process and was instrumental in allowing our team to succeed in the interfacing process (code writing). Appendix 8 documents the code that was written for the Wii IR library, parsed together. The library(s) web link is in Appendix 7.

Motor Controller: A L298 Full H-bridge Driver was used to take signals from the Arduino Uno and provide power to the each of the motors independently and simultaneously. The L298 was selected because it can handle up to 4A total current and can drive two motors simultaneously. Also, it has the ability to drive motors forward and reverse, however we did not use that function on the current version. A data sheet link is documented in Appendix 4.

LED Lighting: LED operating lights were added to the system in order for the designer/user to know how the system was functioning (see Figure 15). The status lights (bottom right of Figure 11) consist of the following:

- A blue LED is illuminated when the Wii remote is within 3' (remote-to-cart mounted IR LEDs distance),
- A green LED is illuminated when it is outside the 3' range and the cart is activated
- A red LED is illuminated when no IR signal is being received by the Wii Remote

In addition, a steady green LED is illuminated when the Wii Remote is connected to the Arduino, and a steady green LED is illuminated when the power switch is in the on position (see Figure 9).

An additional LED panel can be connected for debugging and is located top-center of Figure 9. The red LED's are connected to left and right Enable pins of the motor driver, and the green LED's are connected to the Input Pins for the left and right motors on the motor driver.

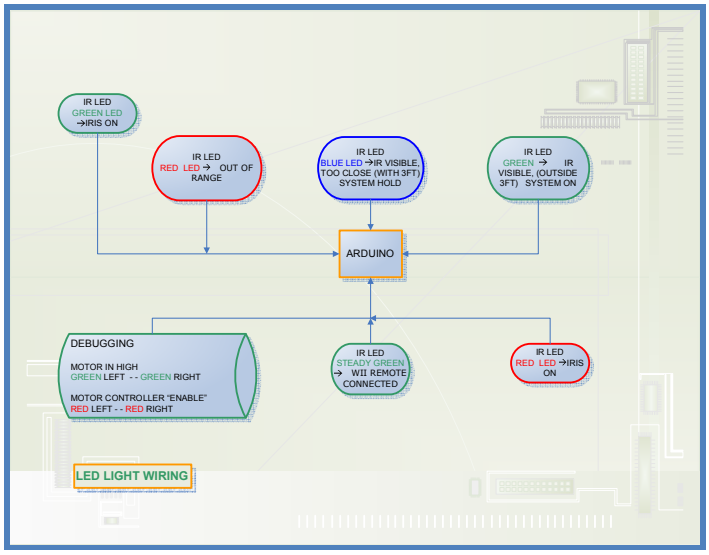


Figure 15. LED Functionality Diagram

Wii Remote: Provided by the good people of Nintendo, Japan. The technical specifications are available at http://wiibrew.org/wiki/Wiimote#IR_Camera. This site has a complete run-down of all features. Key features employed in this project were the IR camera and the Bluetooth communication features. The IR camera has on board image processing that tracks the four brightest images that it sees and outputs x and y axis coordinates, an intensity value, a size value, and a pixel bounding box for each point. The Wii remote has a full camera, and actually uses a passive lens filter so that it sees infrared light. There are 3 reporting formats that the Wii remote can report in. The report format is set in the Wii IR library we wrote. We used the Extended Reporting mode, which outputs x and y axis coordinates and a size value for each of the 4 points. The Full reporting mode would have been useful, as it provides intensity data. This mode could not be made operational for unknown reasons.

There are numerous sensitivity levels of the IR camera, set by an 11 byte block programmed in the Wii library. We used a known working "high" sensitivity setting.

The Wii remote has one bug. When the IR camera is initialized it turns on in one of 3 states: IR camera on but not taking data, IR camera on and taking data and half sensitivity, IR camera on and taking data at full sensitivity (http://wiibrew.org/wiki/Wiimote#IR_Camera). The state it ends up in is random which may have been the cause of some irregularities in the operation of the cart. Therefore if the cart is not responding very well, it is likely that it is not in full sensitivity mode (different from the sensitivity mode that is set within the code).

Arduino Code: Over 400 lines of Arduino code was written to control the Cart Follower. It is well commented and can be seen in Appendix 8. The Wii remote buttons were all programmed to provide user and designer functionality and debugging:

Buttons used by user

- Button (+): initializes the cart, turns on IR, and begins reading data
- Button (1): Initializes IR camera sensitivity (if needed as stated in Wii Remote section above)
- Button (-) : turns all functionality off, but Wii remote remains connected

Buttons used for debugging

- Button (1): Initializes IR camera. Press more than once if it doesn't read IR the first time
- Button (2): Gets status request to see if IR is initialized (only used with Serial Monitor)
- Button (A): Toggle IR LED's
- Button (B): Enables/Disables motor controller

- Controller Direction (DOWN): Is used for debugging, outputs raw IR data (only used with Serial Monitor)
- Button (HOME): Turns everything off and disconnects
- Button (+): Initializes the cart, turns on IR camera, and begins reading data
- Button (-): Turns all functionality off, but Wii remote remains connected

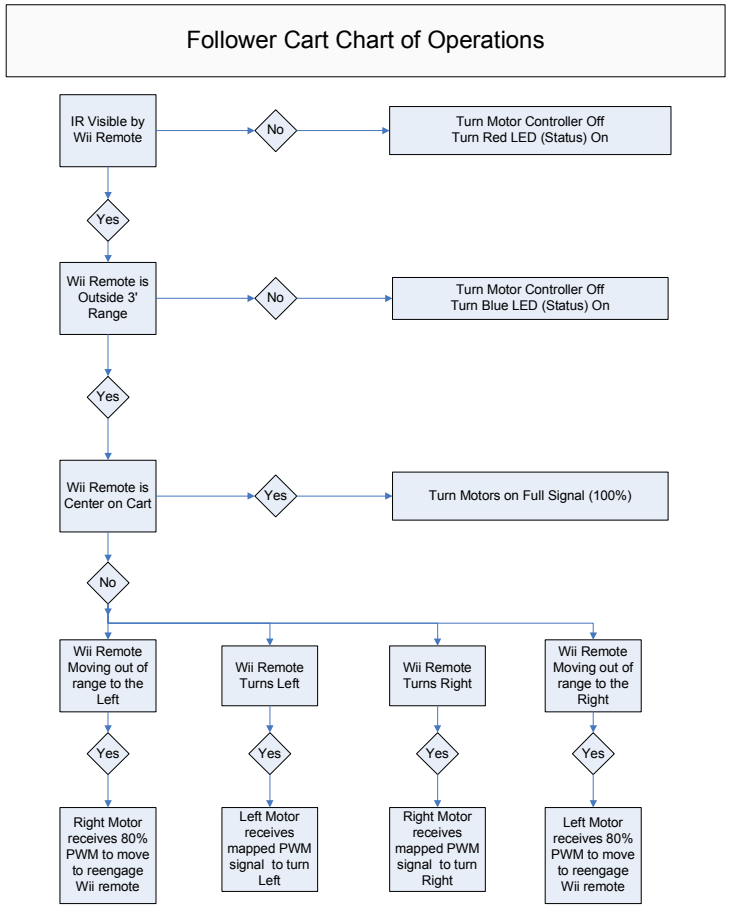


Figure 16. Programming Flow Chart

To control when the cart should move, the program uses the y axis coordinates from the IR camera to test for the distance between the Wii remote and the cart (see Figure 16). The threshold values are set so that the cart activates when the person is farther than 3 feet from the cart. Hysteresis was used so that the cart would not jump in and out of the 'if' statement causing intense jerking of the cart. Once the user is beyond 3 feet, both motors are activated at 100% power.

To control the direction of travel, the program uses the x axis coordinates from the IR camera to track how sharp of a turn the user is making. The x axis values range from 0 to 1023. For example, if a sharp right turn is made by the user, the program will read a value of 1023 for the right IR LED and a low value for the left IR LED. Based on these two values the program will enter an "if" statement corresponding to which direction the user is turning. The analogWrite Arduino function was used to write a Pulse Width Modulation (PWM) signal proportional to the low value read by the right IR LED. This slows down the appropriate motor so that the cart makes the correct turn. Because the PWM signal is proportional to the x-axis values read by the IR camera, the cart is capable of making turns at different speeds which allows it to accurately follow a person. Because of the data the Wii remote outputs when NO IR is in range, a correction factor was need for certain instances. For example, if too sharp a turn was made, the cart would interpret it as a turn in the opposite direction. To account for this, the program would flow into another 'if' statement if the x axis values reached a critical value (see Figure 16). In that case the cart would turn in the opposite direction until the x-values were back in range. This effectively reduced the amount of direction errors the cart has on sharp turns.

OUTCOME AND CONCLUSION

Project Performance

This project can be considered an incredible success. This project is the first of its kind: all other follower type vehicles found while researching this project were dependent on GPS tracking, or if IR lights were used, range finding equipment and programming were utilized. This is the first known use of the combination of the Wii remote IR camera reading the movement of cart mounted IR lights which was transmitted via Bluetooth, received via a dongle attached to a USB shield and then passed on to the Arduino Uno which was programming to take this data and output signals correctly through the driver to the motor to drive the cart correctly and follow the movements of the Wii remote.

Conclusion

The team found it hard to balance the requirements of the project with possibilities and potential of the cart project. The first and foremost goal of the project was to have the cart correctly follow the Wii remote which was largely successful. Addition debugging would be required to have the cart follow with greater accuracy as the system would sometime misinterpret directional inputs and go the opposite direction. When contact was completely lost, reengaging the IR system required that the IR lights blink, but sometime even that didn't completely get the system functional again. Also it was noted that the cart system had problems operating correctly highly reflective surfaces which would have to be addressed on the full scale model.

As previous mentioned, further additions that could be added in the future to an actual operating car might include:

- Multiple speeds
- Obstacles avoidance
- Dead-man switch

REFERENCES

Carryer, J. Edwards, Ohline, R. Matthew & Kenny, Thomas W. (2011). *Introduction to Mechatronic Design.* Upper Saddle River, NJ: Pearson Higher Education

Lauszus, Kristian Sloth (2012). Personal Emails and website. http://www.tkjelectronics.dk/

Scherz, Paul. (2007). *Practical Electronics For Inventors (2nd ed.*) New York, NY: McGraw-Hill.

USB_HOST_SHIELD_2.0 (edited 2012). Contains Arduino USB shield libraries. Retrieved October-November 2012, from https://github.com/felis/USB_Host_Shield_2.0

Wiimote (n.d.). Wiimote Wiki. Retrieved October-November 2012, from http://wiibrew.org/wiki/Wiimote

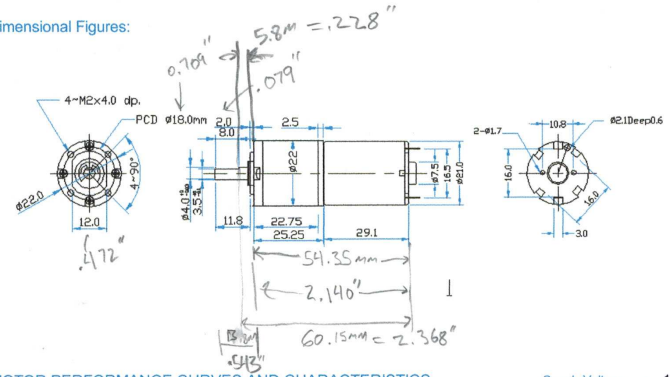
APPENDIX 1: Motor Report

HENNKWELL IND. CO., LTD.

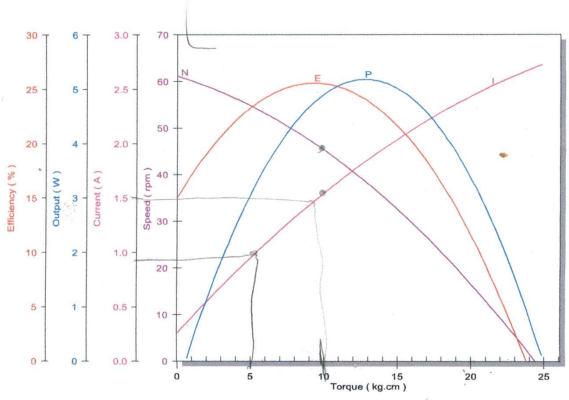
Dynamic Test Report

Product:	DC Gear Motor	Rated Voltage:	12	VDC
Customer:	PK22-950306-04	No Load Speed:	64	RPM
Mode NO.:	PK22G2150-231B	No Load Current:	0.26	A
Test Date:	2007/01/12	Gear Ratio:	1:231	

(A) Dimensional Figures:



(B) MOTOR PERFORMANCE CURVES AND CHARACTERISTICS:



Supply Voltage = 12 VDC

AT NO LOAD
- SPEED = 61.1 RPM
- CURRENT = 0.26 AMP

AT STALL EXTRAPOLATION
- TORQUE = 24.61 kg-cm
- CURRENT = 2.71 AMP

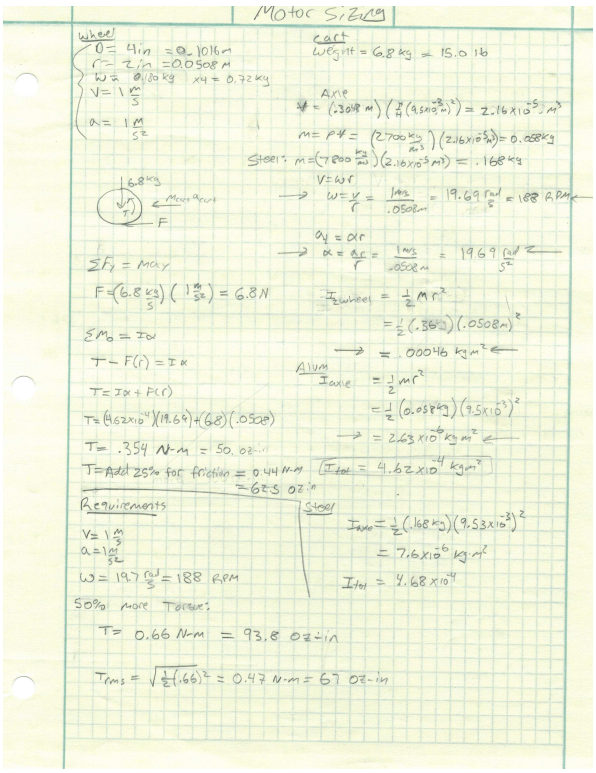
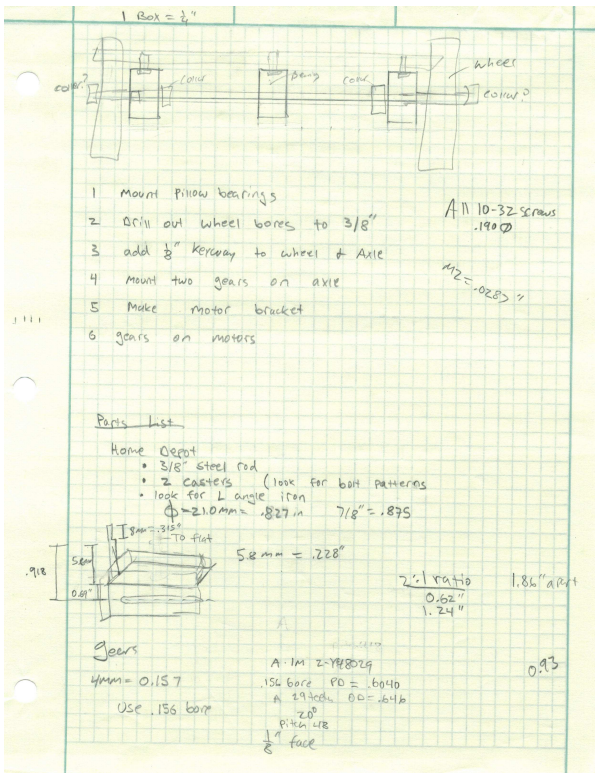
AT MAXIMUM EFFICIENCY
- EFFICIENCY = 25.57 %
- SPEED = 46.8 RPM
- TORQUE = 9.28 kg-cm
- CURRENT = 1.47 AMP
- OUTPUT = 4.76 WATTS

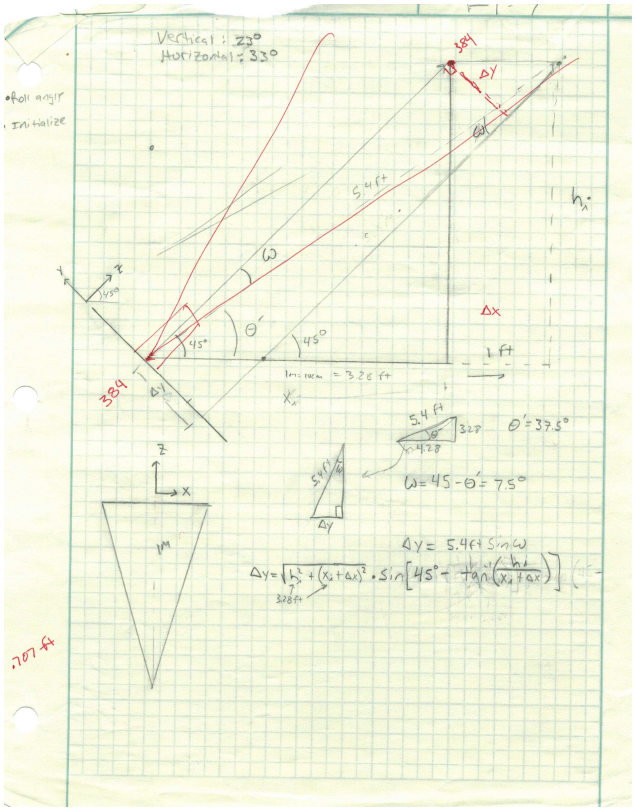
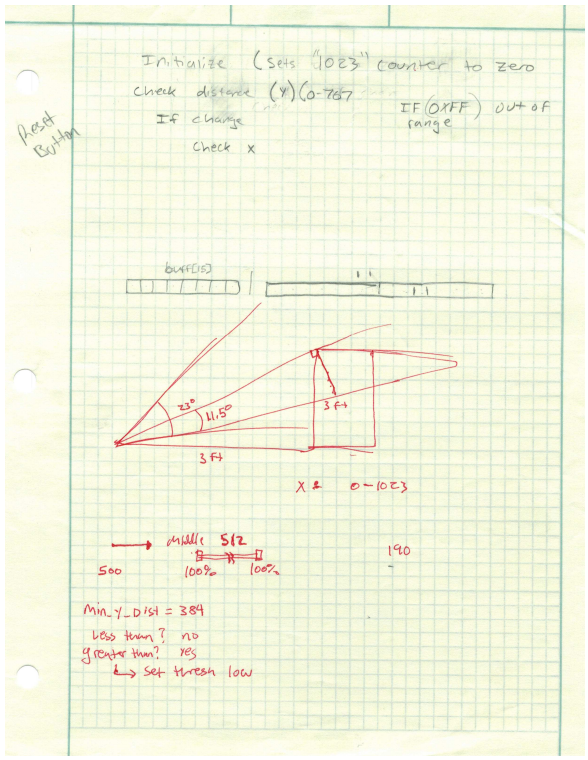
AT MAXIMUM OUTPUT
- SPEED = 38.7 RPM
- TORQUE = 12.73 kg-cm
- CURRENT = 1.83 AMP
- OUTPUT = 5.18 WATTS
- EFFICIENCY = 24.11 %

Torque Conversions:
1.0 kgf-cm = 0.098 Nm = 98 mNm
= 13.887 oz-in = 0.867 lb-in

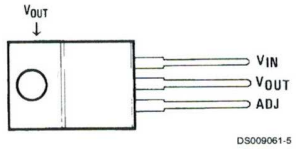
PREPARED BY: _____ APPROVED BY: _____

APPENDIX 2 - Calculations

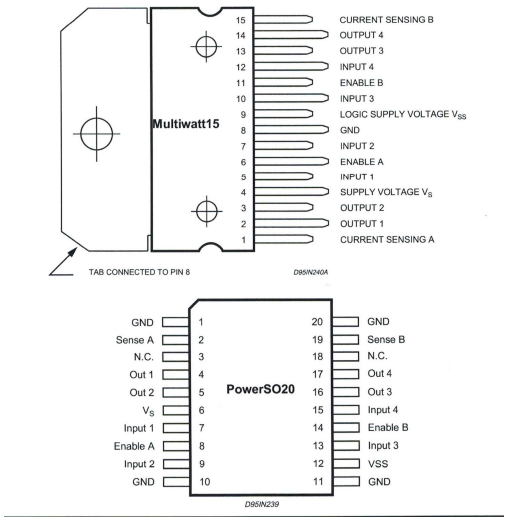
	
Torque and Velocity Calcs	Rear Axle Design

	
Wii Remote Angle Determination	Cart-Wii Remote Angles

APPENDIX 3 – LM350 Datasheet Web Address

(TO-220) Plastic Package  Front View Order Number LM350AT or LM350T See NS Package Number T03B	www.jameco.com/Jameco/Products/ProdDS/23931.pdf LM350.pdf Double-Click Icon for Data Sheet

APPENDIX 4 – L298 Full-Bridge Driver



http://www.alldatasheet.com/datasheet-pdf/pdf/22437/STMICROELECTRONICS/L298.html

L298 Full H-bridge.pdf

Double-Click Icon for Data Sheet

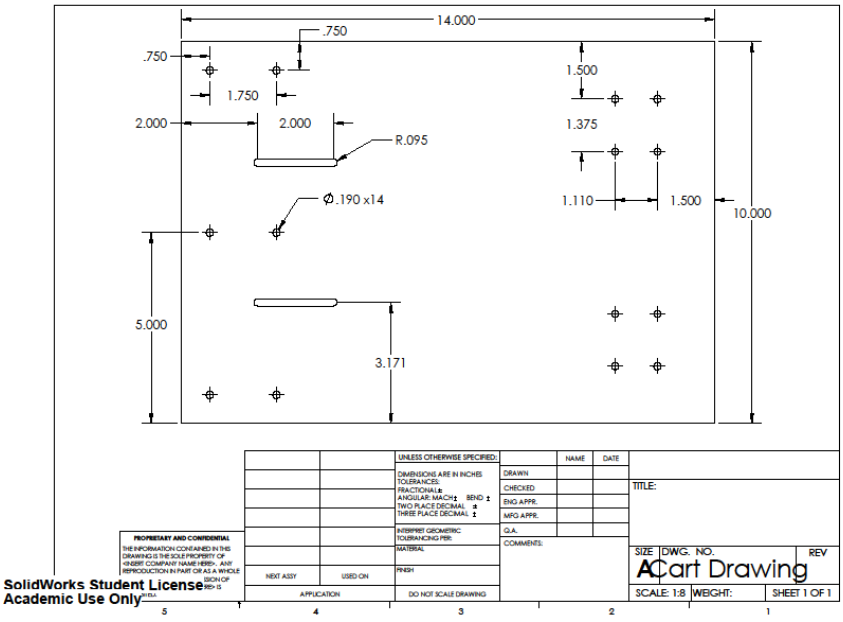
APPENDIX 5: BILL OF MATERIALS

<table>
<tr><th colspan="3">Mechanical System</th></tr>
<tr><td>Qty.</td><td>Item</td><td>Cross Reference</td></tr>
<tr><td>1</td><td>10x14x¼” Acrylic sheet</td><td></td></tr>
<tr><td>2</td><td>Small casters ~ two inches in height</td><td></td></tr>
<tr><td>3</td><td>3/8” bore pillow block bearings</td><td></td></tr>
<tr><td>1</td><td>3/8” aluminum rod ~18”</td><td></td></tr>
<tr><td>2</td><td>4” Colson wheels with soft tread</td><td></td></tr>
<tr><td>2</td><td>12V Hennkwell IND CO DC Gear Motors</td><td></td></tr>
<tr><td>1</td><td>1.5x1.5” aluminum angle bracket ~12”</td><td></td></tr>
<tr><td>1</td><td>¼” Acrylic for gears</td><td></td></tr>
<tr><th colspan="3">Electrical System</th></tr>
<tr><td>Qty.</td><td>Item</td><td>Cross Reference</td></tr>
<tr><td>2</td><td>9.6V Batteries</td><td></td></tr>
<tr><td>1</td><td>IR Emitter System</td><td>Figure 12 (schematic)</td></tr>
<tr><td></td><td>
<table>
<tr><td>Qty.</td><td>Item</td><td>Cross Reference</td></tr>
<tr><td>4</td><td>IR 890nm LED’s</td><td>IR1 – IR4</td></tr>
<tr><td>4</td><td>68 Ω Resistors (1/2 W)</td><td>R10, R11,R13,R14</td></tr>
<tr><td>2</td><td>Green LED’s</td><td>LED6 – LED7</td></tr>
<tr><td>2</td><td>460 Ω Resistors</td><td>R12, R15</td></tr>
<tr><td>2</td><td>P2N2222A Transistors</td><td>T1, T2</td></tr>
<tr><td>2</td><td>3.3 k Ω Resistors</td><td>R16, R17</td></tr>
</table>
</td><td></td></tr>
<tr><td>1</td><td>Wii Remote</td><td></td></tr>
</table>

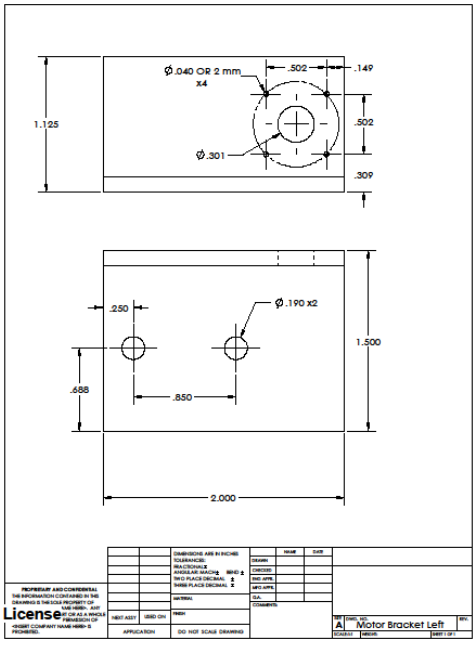
<table>
<tr><td>1</td><td colspan="3">Power Interface & Motor Control</td><td>Figure 12 (schematic)</td></tr>
<tr><td></td><td colspan="3"><table>
<tr><th>Qty.</th><th>Item</th><th>Cross Reference</th></tr>
<tr><td>1</td><td>LM350 Voltage Reg.</td><td>IR1 – IR4</td></tr>
<tr><td>1</td><td>LM317T Voltage Reg.</td><td>R10, R11,R13,R14</td></tr>
<tr><td>1</td><td>730 Ω Resistor</td><td>R1</td></tr>
<tr><td>1</td><td>120 Ω Resistor</td><td>R2</td></tr>
<tr><td>1</td><td>890 Ω Resistor</td><td>R3</td></tr>
<tr><td>1</td><td>220 Ω Resistor</td><td>R4</td></tr>
<tr><td>2</td><td>1N4001 1A Diode</td><td>D1, D2</td></tr>
<tr><td>1</td><td>L298 Motor Driver</td><td></td></tr>
</table></td><td></td></tr>
<tr><td>1</td><td colspan="3">Arduino Uno R3</td><td></td></tr>
<tr><td>1</td><td colspan="3">USB 2.0 Host Shield</td><td></td></tr>
<tr><td>1</td><td colspan="3">Bluetooth Dongle</td><td></td></tr>
<tr><td>1</td><td colspan="3">Small 12V fan</td><td></td></tr>
<tr><td>1</td><td colspan="3">Switch rated above 5 A</td><td></td></tr>
<tr><td>1</td><td colspan="3">Fuse and fuse holder: 1.5A</td><td></td></tr>
</table>

APPENDIX 6: CAD Drawings

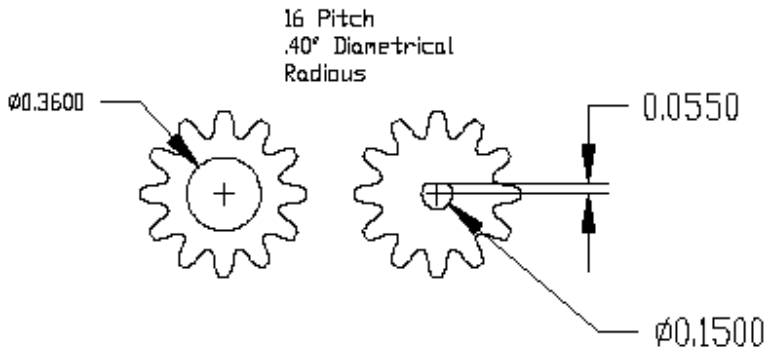
Cart Chassis Drawing



Motor Mount Drawing



Gears



APPENDIX 7: Library List

Retrieved from https://github.com/felis/USB_Host_Shield_2.0:

address.h
adk.h
avrpins.h
BTD.h
Hexdump.h
Hid.h
Hidboot.h
Hiduniversal.h
Hidusagestr.h
Hidusagetitlearrays.h
Max3421.h
Message.h
Usb.h
Usbhub.h
Usbhost.h
Usb_ch9.h

APPENDIX 8: Wii.cpp Code Written for IR Camera

```
/**Wii.h code**/

uint8_t statusRequestPublic(); //public version of statusRequest()
                void IRinitialize();
                void EnableIRCamera1(); //Sets bit 2 of output report 13
                void EnableIRCamera2(); //Sets bit 2 of output report 1A
                void WriteSensitivityBlock1();
    void WriteSensitivityBlock2();
                void write0x08Value();
                void setWIIModeNumber(uint8_t* mode_number);


                int8_t IR_state; //stores the value in 12capinbuf[12] (0x08 means IR enabled)
                int16_t IR_object_x1; // IR x position data 10 bits
                int16_t IR_object_y1;  //IR y position data 10 bits
                int8_t IR_object_s1; // IR size value
                int16_t IR_object_x2;
                int16_t IR_object_y2;
                int8_t IR_object_s2;

                double getIRx1() { return IR_object_x1; }; //IR object 1 x position (0-1023)
                double getIRy1() { return IR_object_y1; }; //IR object 1 y position (0-767)
                double getIRs1() { return IR_object_s1; }; //IR ojjcet 1 size (0-15)

                double getIRx2() { return IR_object_x2; };
                double getIRy2() { return IR_object_y2; };
                double getIRs2() { return IR_object_s2; };

/**Wii.cpp code**/

if(l2capinbuf[9] == 0x33){ //Read the IR data

                IR_object_x1 = (l2capinbuf[15] | ((uint16_t)(l2capinbuf[17] & 0x30) << 4));
                IR_object_y1 = (l2capinbuf[16] | ((uint16_t)(l2capinbuf[17] & 0xC0) << 2));
                IR_object_s1 = (l2capinbuf[17] & 0x0F);

                IR_object_x2 = (l2capinbuf[18] | ((uint16_t)(l2capinbuf[20] & 0x30) << 4));
                IR_object_y2 = (l2capinbuf[19] | ((uint16_t)(l2capinbuf[20] & 0xC0) << 2));
                IR_object_s2 = (l2capinbuf[20] & 0x0F);
}

//parsed

/**************************************************/
/*The following functions are for the IR camera */
/**************************************************/

void WII::IRinitialize(){ //Turns on and initializes the IR camera

                                EnableIRCamera1();
#ifdef DEBUG
                                Notify(PSTR("\r\nEnable IR Camera1 Complete"));
#endif
                                delay(80);

                                EnableIRCamera2();
#ifdef DEBUG
                                Notify(PSTR("\r\nEnable IR Camera2 Complete"));
#endif
                                delay(80);

                                write0x08Value();
#ifdef DEBUG
                                Notify(PSTR("\r\nWrote hex number 0x08"));
#endif
                                delay(80);

                                WriteSensitivityBlock1();
#ifdef DEBUG
                                Notify(PSTR("\r\nWrote Sensitivity Block 1"));
#endif
                                delay(80);

                                WriteSensitivityBlock2();
#ifdef DEBUG
                                Notify(PSTR("\r\nWrote Sensitivity Block 2"));
#endif
                                delay(80);

                                uint8_t mode_num[] = {0x03};
                                setWIIModeNumber(mode_num); //change input for whatever mode you want i.e. 0x01, 0x03, or 0x05
#ifdef DEBUG
                                Notify(PSTR("\r\nSet Wii Mode Number To 0x"));
                                PrintHex<uint8_t>(mode_num[0]);
#endif
                                delay(80);

                                write0x08Value();
#ifdef DEBUG
                                Notify(PSTR("\r\nWrote Hex Number 0x08"));
#endif
                                delay(80);

                                setReportMode(false, 0x33); //note wiiMotePitch won't return values anymore because it uses output report 0x31 or 0x35
                                //setReportMode(false, 0x3f);
#ifdef DEBUG
                                Notify(PSTR("\r\nSet Report Mode to 0x33"));
#endif

                                Notify(PSTR("\r\nIR enabled and Initialized"));
}

uint8_t WII::statusRequestPublic() { //This is a public version of the functio so it can be called from Arduino
    uint8_t cmd_buf[3];
    cmd_buf[0] = 0xA2; // HID BT DATA_request (0xA0) | Report Type (Output 0x02)
    cmd_buf[1] = 0x15;
    cmd_buf[2] = (HIDBuffer[2] & 0x01); // Keep the rumble bit
    HID_Command(cmd_buf, 3);

                return 0x11;
}

void WII::EnableIRCamera1(){
                uint8_t cmd_buf[3];
    cmd_buf[0] = 0xA2; // HID BT DATA_request (0xA0) | Report Type (Output 0x02)
    cmd_buf[1] = 0x13; //output report 13
    cmd_buf[2] = 0x04;  // Keep the rumble bit and sets bit 2
    HID_Command(cmd_buf, 3);
}

void WII::EnableIRCamera2(){
```

```
                uint8_t cmd_buf[3];
    cmd_buf[0] = 0xA2; // HID BT DATA_request (0xA0) | Report Type (Output 0x02)
    cmd_buf[1] = 0x1A; //output report 13
    cmd_buf[2] = 0x04 | (HIDBuffer[2] & 0x01); // Keep the rumble bit and sets bit 2
    HID_Command(cmd_buf, 3);
}

void WII::WriteSensitivityBlock1(){
                uint8_t buf[9];
                buf[0] = 0x00;
                buf[1] = 0x00;
                buf[2] = 0x00;
                buf[3] = 0x00;
                buf[4] = 0x00;
                buf[5] = 0x00;
                buf[6] = 0x90;
                buf[7] = 0x00;
                buf[8] = 0x41;

                writeData(0xB00000, 9, buf);
}

void WII::WriteSensitivityBlock2(){
                uint8_t buf[2];
                buf[0] = 0x40;
                buf[1] = 0x00;

                writeData(0xB0001A, 2, buf);
}

void WII::write0x08Value(){
                uint8_t Value[]={0x08};
                writeData(0xb00030, 1, Value);
}

void WII::setWIIModeNumber(uint8_t* mode_number){ //mode_number in hex i.e. 0x03 for mode extended mode
                writeData(0xb00033,1,mode_number);
}
```

Appendix 9: Arduino Program

```
/****Wii Remote Status Lights and Button Functionality****/

/** Wii Remote LED 1 on means Disconnected
/** Wii Remote LED 4 on means Wii remote connected
/** Button 1 initializes IR and turns it ON. Press more than once if it doesn't read IR the first time
/** Button 2 gets status request to see if IR is initialized (only used with Serial Monitor)
/** Button A turns toggles IR LED's ON/OFF
/** Button B Enables/Disables motor controller
/** Button + initializes the cart, turns on IR, and begins reading data
/** Button - turns all functionality off, but remians connected
/** DOWN is used for debugging, outputs raw IR data (only used wit hSerial Monitor)
/** HOME turns everything off and disconnects

/**********************************************************************************/

/** RED LED -- IR out of range
/** BLUE LED -- Person too close
/** Green LED -- Person furthur than 3 feet

/**********************************************************************************/

/**X and Y in the program refers to the x and y coordinates of the points the IR camera /**is tracking
/**X ranges from 0 - 1023, and Y ranges from 0 - 767

/**********************************************************************************/

#include <Wii.h>
USB Usb;
BTD Btd(&Usb); // You have to create the Bluetooth Dongle instance like so
/* You can create the instance of the class in two ways */
//WII Wii(&Btd,PAIR); // This will start an inquiry and then pair with your Wiimote - you only have to do this once
WII Wii(&Btd); // After that you can simply create the instance like so and then press any button on the Wiimote

//#define DEBUG
//#define DEBUG_Y
//#define EXTRADEBUG_X_Y
//#define RAWVALUES


/**Y threshhold and miscellaneous values. Thess values are used to change the linear distance away from the cart at which it activates**/
#define Y_THRESH_CONSTANT 30
#define CENTER_Y_DIST 300
int MIN_Y_DIST = CENTER_Y_DIST;
#define MIN_Y_DIST_THRESH_LOW (CENTER_Y_DIST + Y_THRESH_CONSTANT)
#define MIN_Y_DIST_THRESH_HIGH (CENTER_Y_DIST - Y_THRESH_CONSTANT)

//** X threshhold and miscellaneous values. These values are used for tracking the lateral movements of the Wii remote**/
//**LEFT or _1 refers to the left IR emiiters WHEN FACING THE CART
//**RIGHT or _2 refers to the right IR emiiters WHEN FACING THE CART
#define X_THRESH_CONSTANT 5
#define X_1_CENTER 300   //360
#define X_2_CENTER 720
#define X_1_CENTER_THRESH 65
#define X_2_CENTER_THRESH 65

#define X_1_THRESH_LOW (X_1_CENTER - X_1_CENTER_THRESH)
#define X_1_THRESH_HIGH (X_1_CENTER + X_1_CENTER_THRESH)
#define X_2_THRESH_LOW (X_2_CENTER - X_2_CENTER_THRESH)
#define X_2_THRESH_HIGH (X_2_CENTER + X_2_CENTER_THRESH)

#define X_1_OVERTURN 900
#define X_2_OVERTURN 123

#define IR_outofrange_thresh 50

//**BLUE LED
#define LED_pin_close 3 //pin used for BLUE LED indicating too close
#define DIM 20 //pwm light intensity for BLUE LED indicating cart is too close

//**GREEN LED steady when moving
#define LED_pin_moving 2 //pin used for LED indicating moving forward

//**RED LED blink when no IR in range
#define LED_pin_outofrange 9 //pin used for LED indicating out of range

//**GREEN if WiiMote connected
#define LED_pin_WiiMoteConnected 0


/****Motor Pins****/
//**Right refers to cart right
//**Left refers to cart left
#define MOTOR_RIGHT_ENABLE 7
#define MOTOR_RIGHT 6 //PWM pin!!!
#define MOTOR_LEFT_ENABLE 8
#define MOTOR_LEFT 5 //PWM pin!!!

#define IR_TRANSISTOR_R 1
#define IR_TRANSISTOR_L 4

#define TYPICAL 1 //State variable. not used in this rev
#define X1_LAST_VISIBLE 2  //State variable. not used in this rev
#define X2_LAST_VISIBLE 3  //State variable. not used in this rev

/* x-axis hysteresis threshhold values */
int x1_thresh_low;
int x1_thresh_high;
int x2_thresh_low;
int x2_thresh_high;

int initialize; //initialization variable used to inialize program once
int state; //state variable. not used in this rev

bool printAngle;
bool readIR;
```

```
bool debug_1;

/**The following are used to keep track of which side of the cart the Wii remote goes out of range**/
/**Only some of them are used in the current rev**/
bool IR_x2_outofrange;
bool x2_last_visible;
bool IR_outofrange;
bool y_last_visible = false;
bool out_in = false;
bool IR_x2_interrupt;
bool x1_left_last_visible;
bool x2_right_last_visible;
bool IR_x_outofrange;

void setup() {
  //Serial.begin(115200); //pins 0 and 1 control important functions, they will not work if Serial.begin is called
  pinMode(LED_pin_outofrange, OUTPUT);
  pinMode(IR_TRANSISTOR_R, OUTPUT);
  pinMode(IR_TRANSISTOR_L, OUTPUT);
  pinMode(LED_pin_close, OUTPUT);
  pinMode(LED_pin_moving, OUTPUT);
  pinMode(LED_pin_WiiMoteConnected, OUTPUT);
  pinMode(MOTOR_RIGHT_ENABLE, OUTPUT);
  pinMode(MOTOR_RIGHT, OUTPUT);
  pinMode(MOTOR_LEFT_ENABLE, OUTPUT);
  pinMode(MOTOR_LEFT, OUTPUT);

  PORTD = 0x00; //set pins 0 - 7 low
  PORTB &= ~0x03; //set pin 8 and 9 low

  if (Usb.Init() == -1) {
   // Serial.print(F("\r\nOSC did not start"));
    while(1); //halt
  }
  Serial.print(F("\r\nWiimote Bluetooth Library Started"));
}
void loop() {
  Usb.Task();

  if(Wii.wiimoteConnected) { //function that returns whether Wii remote connects with bluetooth shield
    digitalWrite(LED_pin_WiiMoteConnected, HIGH); //status LED

    if(Wii.getButtonClick(HOME)){ // Turns all cart funcionality off and disconnects
      PORTD = 0x00; //set pins 0 - 7 low
      PORTB &= ~0x03; //set pin 8 and 9 low
      digitalWrite(LED_pin_close, LOW); //this won't turn off without this for some reason
      disableIR(); //function that runs IR on/off
      Wii.disconnect(); // Disconnect the Wiimote - it will establish the connection again since the Wiimote automatically reconnects
      }

    else {

      if(Wii.getButtonClick(ONE)) //initialize IR camera
        Wii.IRinitialize();

      if(Wii.getButtonClick(TWO)) //check status request. Returns if IR is intialized or not (Serial Monitor only)
       Wii.statusRequestPublic();

      if(Wii.getButtonClick(A)){ //toggles IR emitters
        digitalWrite(IR_TRANSISTOR_R, (digitalRead(IR_TRANSISTOR_R))^1);
        delay(100);
        digitalWrite(IR_TRANSISTOR_L, (digitalRead(IR_TRANSISTOR_L))^1);
      }

      if(Wii.getButtonClick(B)) { //toggles motor enables, useful for debugging and acts as a quick kill switch
       digitalWrite(MOTOR_RIGHT_ENABLE, (digitalRead(MOTOR_RIGHT_ENABLE))^1);
       digitalWrite(MOTOR_LEFT_ENABLE, (digitalRead(MOTOR_LEFT_ENABLE))^1);
       Serial.println("MOTOR CONTROLLER toggled");
      }

      if(Wii.getButtonClick(PLUS)){ //Initializes different things and overall activates the cart
        Wii.setAllOff();
        Wii.setLedOn(LED4);
        readIR = true; //All cart functionality code only runs if readIR = true
        disableIR();
        enableIR();
        digitalWrite(MOTOR_RIGHT_ENABLE, HIGH);
        digitalWrite(MOTOR_LEFT_ENABLE, HIGH);
        initialize = 0; //used in IR processing below to simply run a section of code once
        state = TYPICAL; //state varialbe not used in current rev
        }

      if(Wii.getButtonClick(MINUS)){ //turns all cart functionality off, but leaves Wii remote connected to Arduino
        Wii.setAllOff();
        Wii.setLedOn(LED1);
        readIR = false;
        PORTD = 0x00; //set pins 0 - 7 low
        PORTB &= ~0x03; //set pin 8 low
        digitalWrite(LED_pin_close, LOW); //For some reason PORTD and PORT B won't turn this one off
        disableIR(); //Port style won't work
      }

#ifdef RAWVALUES   //Used for debugging, returns raw values from IR camera (two brightest points: x and y coordinates and size)
      if(Wii.getButtonClick(DOWN)){
        debug_1 = !debug_1;
      }
         if(debug_1){
              initialize = 0;
              Serial.print(F("\r\n y1: "));
              Serial.print(Wii.getIRy1());
              Serial.print(F("\t y2: "));
              Serial.print(Wii.getIRy2());
              Serial.print(F("\t x1: "));
              Serial.print(Wii.getIRx1());
              Serial.print(F("\t x2: "));
              Serial.print(Wii.getIRx2());
              Serial.print(F("\t s1:"));
              Serial.print(Wii.getIRs1());
```

```
              Serial.print(F("\t s2:"));
              Serial.print(Wii.getIRs2());
            }
#endif

/*****************************************************************/
/****************** IR processing program below ******************/
/*****************************************************************/

        /**enter/exit IR loop if PLUS or MINUS button is pressed**/
        /**All code below this (except for functions at the bottom) is inside this if statement)**/

        if(readIR){

             if(initialize == 0){//set the thresh hold to the current wii position the first time through
                 x1_thresh_low = (Wii.getIRx1() - X_THRESH_CONSTANT);
                 x1_thresh_high = (Wii.getIRx1() + X_THRESH_CONSTANT);
                 x2_thresh_low = (Wii.getIRx2() - X_THRESH_CONSTANT);
                 x2_thresh_high = (Wii.getIRx2() + X_THRESH_CONSTANT);
#ifdef DEBUG
                 Serial.print(F("\r\n x1 initial high threshold: "));
                 Serial.print(x1_thresh_high);
                 Serial.print(F("\t x2 initial low threshold: "));
                 Serial.print(x1_thresh_low);
#endif
                 initialize = 1;
                 /**set out-of-range state variables to false. See variable declaration for explination**/
                 IR_x2_outofrange = false;
                 x2_last_visible = false;
                 IR_outofrange = false;
                 IR_x_outofrange = false;
                 x1_left_last_visible = false;
                 x2_right_last_visible = false;
             }
/**he following two if statements are used to blink the IR LED's when the IR comes back /**in range

             if(Wii.getIRx2() < 100){ // This tracks if the Wii remote goes out of range to the right of the cart. 100 is the x coordinate
               x2_last_visible = true;
               IR_x2_outofrange = false;
             }

             if(out_in){//true if going out of range, false if coming into range
               if((Wii.getIRy1() < 100) || (Wii.getIRy2() < 100)){
                 y_last_visible = true;
                 IR_outofrange = false;
               }
             }

/*************************************************************************************/

             /*Turn motor off if IR is out of range*/
             if((Wii.getIRy1() == 1023) && (Wii.getIRy2() == 1023)){
#ifdef DEBUG
               Serial.println("IR out of range - motor controller turned off");
#endif
               /*Turn on RED LED if no IR is detected*/
               digitalWrite(LED_pin_moving, LOW);
               digitalWrite(LED_pin_close, LOW);
               digitalWrite(LED_pin_outofrange, HIGH);

               /* Turn off motors */
               digitalWrite(MOTOR_LEFT, LOW);
               digitalWrite(MOTOR_RIGHT,LOW);

               /*Get new threshhold values for when wii comes back into frame*/
               x1_thresh_low = (Wii.getIRx1() - X_THRESH_CONSTANT);
               x1_thresh_high = (Wii.getIRx1() + X_THRESH_CONSTANT);
               x2_thresh_low = (Wii.getIRx2() - X_THRESH_CONSTANT);
               x2_thresh_high = (Wii.getIRx2() + X_THRESH_CONSTANT);

               IR_x2_outofrange = true;
               IR_outofrange = true;
               out_in = false;
               IR_x_outofrange = true;
             }


       /*******If IR in range, proceed********/
             else{
               digitalWrite(LED_pin_outofrange, LOW);

               if(IR_outofrange && y_last_visible){ //true if IR is out of range and it went out of range in the y direction
                 disableIR();
                 enableIR();
                 IR_outofrange = false;
                 y_last_visible = false;
               }

       /******Person is within 3 feet, do not move*******/
              if((Wii.getIRy1() < MIN_Y_DIST) || (Wii.getIRy2() < MIN_Y_DIST)){

#ifdef DEBUG_Y
                 Serial.print(F("\r\n Person is within 3 feet: y1, y2: "));
                 Serial.print(Wii.getIRy1());
                 Serial.print(F("\t, "));
                 Serial.print(Wii.getIRy2());
#endif
                 /*Turn on BLUE LED if within 3 feet*/
                 digitalWrite(LED_pin_outofrange, LOW);
                 digitalWrite(LED_pin_moving, LOW);
                 analogWrite(LED_pin_close, DIM);

                 /* Turn off motors */
                 digitalWrite(MOTOR_LEFT, LOW);
                 digitalWrite(MOTOR_RIGHT, LOW);

                 /*Get new threshhold values*/
                 MIN_Y_DIST = MIN_Y_DIST_THRESH_LOW;
```

```
                x1_thresh_low = (Wii.getIRx1() - X_THRESH_CONSTANT);
                x1_thresh_high = (Wii.getIRx1() + X_THRESH_CONSTANT);
                x2_thresh_low = (Wii.getIRx2() - X_THRESH_CONSTANT);
                x2_thresh_high = (Wii.getIRx2() + X_THRESH_CONSTANT);
              }

        /*****If person is farther than 3 feet, move*******/
              else if((Wii.getIRy1() > MIN_Y_DIST) || (Wii.getIRy2() > MIN_Y_DIST)){

                out_in = true; //keeps track of if the wii remote is going out of range or in range

/** Blinks IR LED's if x2 goes out and into range, this is an attempt to establish x_1 IR emitter first in the IR camera*/
                if(x2_last_visible && IR_x2_outofrange){
                  disableIR();
                  enableIR();
                  x2_last_visible = false;
                  IR_x2_outofrange = false;
                  IR_x2_interrupt = true;
                }
#ifdef DEBUG_Y
                Serial.print(F("\r\n Person is furthur than 3 feet: y1, y2: "));
                Serial.print(Wii.getIRy1());
                Serial.print(F("\t, "));
                Serial.print(Wii.getIRy2());
#endif
                /*Turn on GREEN LED if within 3 feet*/
                digitalWrite(LED_pin_outofrange, LOW);
                digitalWrite(LED_pin_close, LOW);
                digitalWrite(LED_pin_moving, HIGH);

                MIN_Y_DIST = MIN_Y_DIST_THRESH_HIGH; //get new y threshhold value

           /***if position moves above or below the x thresh hold: proceed***/
                if(((Wii.getIRx1() > x1_thresh_high) || (Wii.getIRx1() < x1_thresh_low)) || ((Wii.getIRx2() > x2_thresh_high) || (Wii.getIRx2() < x2_thresh_low))){
#ifdef EXTRADEBUG_X_Y
                  Serial.print(F("\r\n y1: "));
                  Serial.print(Wii.getIRy1());
                  Serial.print(F("\t y2: "));
                  Serial.print(Wii.getIRy2());
                  Serial.print(F("\t x1: "));
                  Serial.print(Wii.getIRx1());
                  Serial.print(F("\t x2: "));
                  Serial.print(Wii.getIRx2());
                  Serial.print(F("\t s1: "));
                  Serial.print(Wii.getIRs1());
                  Serial.print(F("\t s2: "));
                  Serial.print(Wii.getIRs2());
#endif

                    /*if x1 and x2 is centered turn both motors on at 100% speed*/
                    if((Wii.getIRx1() < X_1_THRESH_HIGH) && (Wii.getIRx2() > X_2_THRESH_LOW)){
                      analogWrite(MOTOR_LEFT, 255);
                      analogWrite(MOTOR_RIGHT, 255);
                      Serial.println("CENTERED");
                    }
                    /*if turning too far right, start turning left to correct it*/
                    else if(Wii.getIRx2() < X_2_OVERTURN){
                        analogWrite(MOTOR_RIGHT, 255);
                        analogWrite(MOTOR_LEFT, map(Wii.getIRx2(), 0, X_2_OVERTURN, 0, 115)); //maps the pwm signal from 0 to 115
                        Serial.println("correcting left");
                        Serial.println(map(Wii.getIRx2(), 0, X_2_OVERTURN, 0, 115));
                    }
                    /*if turning too far left, start turning right to correct it. Note: only this case has to check two values because they range 0-1023 right to left*/
                    else if((Wii.getIRx1() > X_1_OVERTURN) && (Wii.getIRx1() != 1023)){
                        analogWrite(MOTOR_LEFT, 255);
                        analogWrite(MOTOR_RIGHT, map(Wii.getIRx1(), X_1_OVERTURN, 1023, 108, 0)); //maps the pwm signal from 0 to 115
                        Serial.println("correcting right");
                        Serial.println(map(Wii.getIRx1(), X_1_OVERTURN, 1023, 115, 0));
                    }
                    /*turn right*/
                    else if(Wii.getIRx2() < (X_2_THRESH_LOW)){
                      analogWrite(MOTOR_LEFT, 255);
                      analogWrite(MOTOR_RIGHT, map(wii.getIRx2(), 0, X_2_THRESH_LOW, 0, 150)); //maps the pwm signal and slows down the appropriate motor
                      Serial.println("turning RIGHT");
                      Serial.println(map(Wii.getIRx2(), 0, X_2_THRESH_LOW , 0, 150));

                      x2_right_last_visible = true;
                    }
                    /*turn left*/
                    else if(Wii.getIRx1() > (X_1_THRESH_HIGH)){
                      analogWrite(MOTOR_RIGHT, 255);
                      analogWrite(MOTOR_LEFT, map(Wii.getIRx1(), X_1_THRESH_HIGH, 1023, 150, 0)); //maps the pwm signal and slows down the appropriate motor
                      Serial.println("turning LEFT");
                      Serial.println(map(Wii.getIRx1(), X_1_THRESH_HIGH, 1023, 150, 0));

                      x1_left_last_visible = true;
                    }


                    x1_thresh_low = (Wii.getIRx1() - X_THRESH_CONSTANT);
                    x1_thresh_high = (Wii.getIRx1() + X_THRESH_CONSTANT);
                    x2_thresh_low = (Wii.getIRx2() - X_THRESH_CONSTANT);
                    x2_thresh_high = (Wii.getIRx2() + X_THRESH_CONSTANT);

                  }
                }
              }

        }//closes if(IR) statement

      }//else
    }//if wii.connected
}//loop

void enableIR()
{
  delay(200);
  digitalWrite(IR_TRANSISTOR_R, HIGH);
```

```
  delay(200);
  digitalWrite(IR_TRANSISTOR_L, HIGH);
}

void disableIR()
{
  digitalWrite(IR_TRANSISTOR_R, LOW);
  digitalWrite(IR_TRANSISTOR_L, LOW);
}
```