SMART CART

Author:
Victor Barrena Cardenas

Faculty Mentors:
Ralph Barrera

Sponsor/Customer:
Joe Huang

Contributors:
Prasanna Venkatesh Muralidharan
Tyler Hart

NOTICE

This report represents the results of a student project at the University of Dayton. Because of the nature of this project, the faculty and student team members do not warrant or guarantee the accuracy of the results nor that they are suitable for any particular purpose. The sponsor agrees that if ideas, concepts or designs from this project are implemented, the sponsor is solely responsible for the reliability, performance, and safety of the concepts and designs, and shall indemnify and hold harmless the University of Dayton, employees, students, and any other representatives from and against all claims, losses, or damages arising out of scope of this agreement.

May 5, 2016

To whom it may concern:

This letter is to verify Victor Barrena Cardenas' participation in the ECE 432L Spring 2016 course at the University of Dayton.

The ECE 432L course is multidiscipline class involving Computer, Electrical, and Mechanical Engineering students working on projects submitted by local industrial clients. The project that Victor worked on involved developing a GPS controlled locking mechanism to control the movement carts.

Victor participated in the proposal, research, development, design, and prototype build of a device that performed the function requested by the client. He attended class and performed additional out of class activities with the team to make this a successful effort.

I was the professor of record for the ECE students in this class. Victor contribution to this project was very valuable.

Ralph Barrera

Dr. Ralph Barrera
Adjunct Professor
Electrical and Computer Department

Department of Electrical and Computer Engineering
300 College Park Dayton, OH 45469-0232
Tel: (937) 229-3611 Fax: (937) 229-4529
http://engineering.udayton.edu

Propósito

El robo de carros de supermercado se ha convertido en una gran pérdida para los supermercados que los usan. Un gran número de carros de la compra son robados por personas para varios propósitos, con un coste de entre 75 y 150 dólares por cada carro dependiendo del modelo. Como resultado, hay una necesidad urgente de desarrollar una tecnología que pueda evitar la continua pérdida de los carros. La idea del "Carro Inteligente" es una de esas tecnologías avanzadas y es una investigación con gran valor para estos establecimientos.

Descripción del proyecto

En la Universidad de Dayton se nos ha pedido trabajar en un innovador Proyecto llamado "Carro Inteligente" que consiste en un carro de la compra que Evita ser robado de unos límites específicos. El cliente propuse una lista de los requisitos del proyecto. El carro trabaja en dos zonas (Zona "casa", Zona "roja") controlado por una señal GPS que no dejará al carro salir a la zona de bloqueo o zona "roja" por lo que nadie puede llevarse el carro del parking del supermercado. El carro es también capaz de trabajar dentro de la tienda incluso aunque la señal GPS se pierda y pueda ser recuperada cuando salga al parking. Conceptos de electrónica y mecánica se mezclan para lograr esta idea innovadora.

Resultado

El diseño del "Carro Inteligente" fue finalmente probado. El control del carro estaba proporcionada por un microcontrolador Arduino para seguir las coordenadas GPS de dónde se encontraba el carro en cada momento y comprobar si está dentro de las coordenadas del parking. Una vez fuera de los límites fijados, el Arduino manda una señal a un servomotor en una de las ruedas conectadas a una barra de material ABS S30 construida en una impresora 3D para bloquear el carro. Además el servo era capaz de bloquear la rueda cuando fuera necesario. El "Carro Inteligente" también tenía un sensor de efecto Hall de cara a empezar a recibir una señal GPS cuando el carro empezara a moverse.

Conclusión

El diseño ha sido capaz de programar el Arduino para definir los límites de un supuesto parking usando un pulsador y guardando las coordenadas GPS en una tarjeta SD. La tarjeta SD se carga en el Arduino por lo que éste sabe cuáles son los límites. Se ha verificado que el GPS sigue al carro correctamente determinando si está o no dentro de los límites. Por otra parte, el servo ha sido capaz de mover una barra dentro y fuera de un diseño para para la rueda con el consecuente bloqueo del carro. La meta era tener una unidad funcional del "Carro Inteligente" y ésta se ha conseguido.

Table of Contents

1. Background

1.1 Project Description

In the University of Dayton we have been asked to work in an innovative product called the Smart Cart that it is a shopping cart that avoids being taken away from a specified limit. The client provided a list of design requirements. The cart works in two zones (Home, Red) controlled by a GPS signal that won't allow the cart leave from a "lock zone" so no one can take the cart away from the store parking lot. The cart is also able to work within the store even though the GPS signal will be lost, and pick up the GPS signal when the car exits the store. Electronics and Mechanics have been mixed in order to achieve this innovative idea.

2. Executive Summary

2.1 Purpose

The theft of shopping carts has become a huge loss to stores that use them. Huge numbers of the shopping carts are removed by people for various purposes, which typically cost between $75 and $150 each with some models even more expensive. As a result, there is an urgent need for a technology that can prevent the continuing loss of shopping carts. The idea of 'Smart Cart' is one of those superior technologies and is of great research value.

2.2 Results

Smart Cart design was finally tested. The Smart Cart controls were all run through an Arduino board to track the GPS coordinates of where the cart is at and check to see if the cart remains within the boundary of the parking lot. Once outside those boundaries, the Arduino board would send a signal to the servo motor on the wheel connected to an ABS S30 rod to lock the wheel. The Arduino board was able to track where the cart was and determine if it was inside the desired boundary. Also the servo motor was able to lock up the wheel when told to. The Smart Cart also had a Hall Effect Sensor in order to start receiving GPS coordinates when the cart moves.

2.3 Conclusion

The design was able to get the Arduino board to define a given boundary using a pushbutton and save the GPS coordinates to a SD card. The SD card can then be loaded onto the Arduino so that the Arduino knows what its boundaries are. It was verified that the GPS tracks the cart correctly by determining whether or not the cart is within the defined boundary. Separately, the servo motor was able to move a pin in and out when told to stop the wheel from moving any further, thus stopping the cart. The goal was to have a complete functioning Smart Cart and it was achieved.

2.4 Recommendations

A better way to attach the pin to the servo motor would also be preferred so that there is not any slipping of the arm attached to the pin. Also, more permanent wiring and covering would also need to be added in order to make the cart functional in all weather conditions.

3. Research

3.1. GPS Research

The team did research and found that the hardware from *Adafruit* called Arduino UNO with a SD card slot on itself to log data is more suitable for this project. Moreover, by adding an Adafruit Ultimate GPS Logger Shield on the Arduino UNO, the team can get coordinates and store those coordinates onto the SD card setting up a data library. It was supposed to be a low cost project so all the research was done trying to reduce the prize at the minimum.

3.2. Servomotor Research

For using an automatic brake system controlled by an Arduino the team decided to use a servomotor. Towerpro MG92B 360º Mini Digital Robotic Servo was used for the project and it is explained on section 6.2.2. The wiring and coding was easy and it could move the brake properly.

3.3. Brake Research

Brake system was designed on SolidWorks as Appendix E shows.. It is an ABS S30 rod 3D printed moved by the servo and slots were made into the cart wheel in order to lock the cart when the rod went into them. Testing worked well. The problem that the maximum output voltage that the Arduino could give was 5V so other designs were discarded like a 12V actuator or a 12V servo. This wasn't a problem because of the success of the 5V servo in locking the cart when everything was mounted in the system.

Section 3.3.1 to Section 3.3.4 show different brake systems that could have fit the design but that had some relevant disadvantages. Section 3.3.5 shows the final brake used to build the Smart Cart.

3.3.1. Brake calipers

Figure 1. Brake calipers

The team came with a conceptual idea of using brake calipers to stop the motion of the moving wheels. It was operated by pressing the brake lever attached to the cart which was controlled by the push-pull rod attached to a DC motor for up and down moment of the brake lever. The provided action brought the contraction and expansion of the brake calipers.

The negative aspect of the given design was complexity in designing. The braking force depends completely upon the brake calipers which gets worn on continuous usage and that would have been an added cost to the design.

3.3.2. Foot brakes

Figure 2. Foot Brakes

The foot brake design was a conceptual idea adapted in many industrial carts for halting the motion of the cart. It is the easiest and the simplest way to brake the motion of wheel. It works when a certain amount of force is exerted on the foot lever projected from the caster of the wheel. The idea was to set up a spring mechanism to pull the brake lever up and down for braking the motion of the wheel.

The problem with this design was a huge amount of force was required to be pressed on the brake lever breaking it.

3.3.3. Drum brakes

Figure 3. Drum brake

The idea for drum brakes was generated after “BLICKLE DEAD MAN CASTER” was seen where a drum brake was installed within a swivel caster wheel. On pushing down the hand lever it attached to the cart wheel comes to complete rest.

The negative concern in this idea was that the drum brakes required a huge amount of pressing force which cannot be control by any electrical unit controlled by the Arduino used. Its cost of installation is also high.

3.3.4. Brake collar locking mechanism

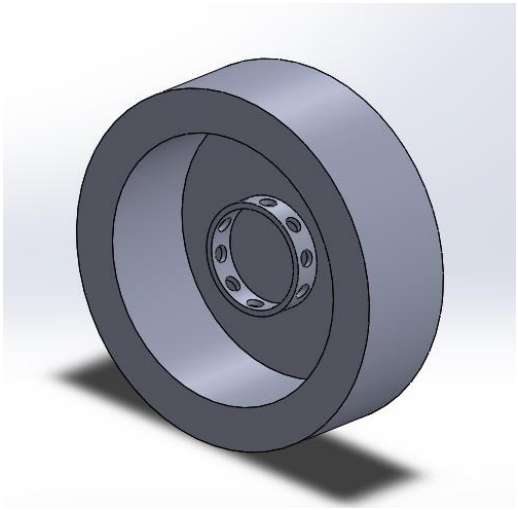


Figure 4. Brake collar locking mechanism

The brake collar mechanism was a much better idea generated during the research period. In this design the brake collar acts as a camshaft. This concept works when the metal rod of same diameter as holes made enters the hole by using spring loaded mechanism to stop the motion acting as a braking system

The negative concern in this design is that it created hindrance to the installation of servo motor as it was difficult to pull all the electrical wires connected to servo motor within the wheel causing a problem for the wheel’s circular movement.

3.3.5. Rod and slotted wheel mechanism

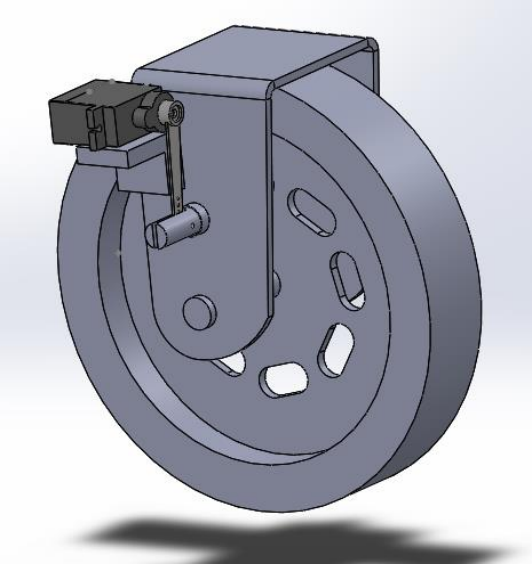


Figure 5. Rod and slotted wheel mechanism

The rod and slotted wheel mechanism was the final design used for this project, The conceptual idea consists of a slotted rod which is connected by a elongated servo horn using connecting pin makes a linear motion through the bearing provided via slots to stop the wheel from moving. The to and fro action is maintained by the servomotor placed on a L-shaped bracket at a specified angle of (27.43º).

4. Procedure

4.1 Product Realization Process

The following flowchart shows how this project has been done during the Spring 2016 semester:

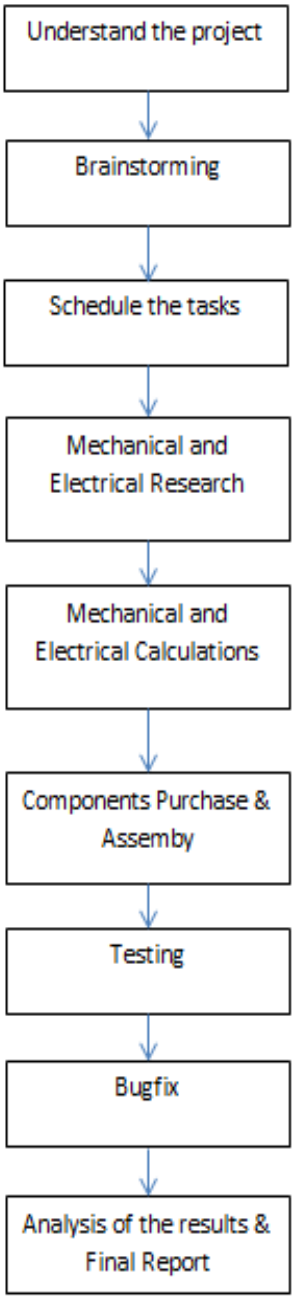


Figure 6. Product realization process

4.1.1 Identify the need

After understanding the purpose of the project, all the engineering knowledge was put on the idea of stopping the carts stealing. It was necessary to use a GPS system and a motor connected each other and communicating as the customer rides the cart through the parking lot. A whole mechanical system had to be designed as well.

4.1.2 Schedule the tasks

Spring 2016 semester was time enough to complete the task. Figure 6 describes the procedure that took from January 20th until May 6th.

4.1.3 Electrical Research

The electrical design was started with a GPS board from *Navspark* that was independent from an Arduino. It was much easier to have an Arduino board and a GPS device connected so a new board from *Adafruit* was found to be perfect meeting all the requirements. And more detailed electrical research followed up.

4.1.4 Mechanical and Electrical Calculations

The calculations were mostly based on the Mechanical system. The forces the system can create and withstand were of great importance for the team as criteria to pick out the best design. The material for the different components that this design would use was also put into great consideration. For the electrical part, the only simple calculation was about setting up the boundary. The team needed to define the area depending on the coordinates. The boundary can finally be expressed as intervals in the code.

4.1.5 Components Purchase and Assembly

For the electrical part, a device was needed to spell out the GPS boundaries and save coordinates somehow, which is why a GPS shield with a SD card was used to be attached to an existing Arduino board. A Hall Effect Sensor was thought of for the use of tracking the revolutions of the wheel so that the Arduino was not continuously checking for a GPS signal. Mechanical assembly was done after finishing the drawings and use the University of Dayton Kettering Labs 3D Printer.

4.1.6 Testing

The GPS shield was tested to verify that it was receiving GPS coordinates correctly. Next, the SD card needed to be able to save the coordinates with the push of a pushbutton. These coordinates were then had to be able to be read by the Arduino code in order for the

cart to know whether or not it was inside the desired boundary. All the components in the code with tracking and reading GPS coordinates ran into small troubles, but eventually the some changes were able to get them to work properly. The servo was also successfully tested. For keeping track the motion of the wheel, the Hall Effect Sensor was also tested separately and proved to work correctly.

4.1.7 Bugfix

After finishing the testing, it is inevitable to confront with bugs. For the electrical part, the bug in the GPS Arduino code had to be fixed and the code needed to be retested and the process of getting coordinates needed to be repeated.

4.1.8 Analysis of the results

The whole system testing was done on May 5th and the design was working perfect.

5. Design

5.1 Mechanical Design

5.1.1 Mechanical Design Description

The mechanical design in this project was making a braking system to lock the wheel when the cart goes to the red zone. The design was made of five parts which are caster, bearing, wheel, rod, and servo motor. The caster is holding the wheel and it welded into the cart. The wheel has eight slots on them to apply the brake on it. The servo motor mounted on to the wheel caster using a L-shaped bracket to push the rod inside the wheel to brake it where the bearing which was welded into the caster is holding the pin and making it move easily.

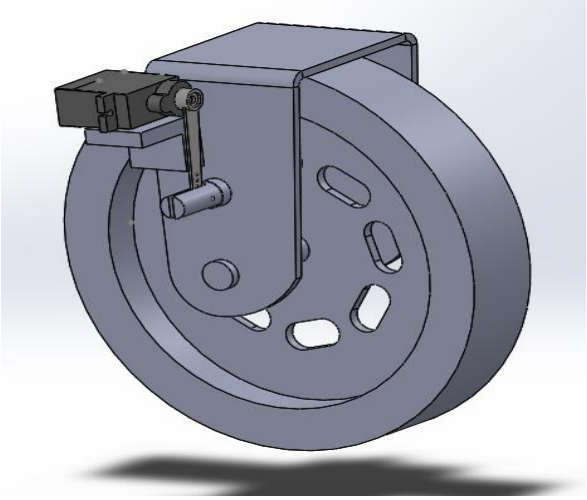


Figure 7. Final design of the brake

5.1.1.1 Caster

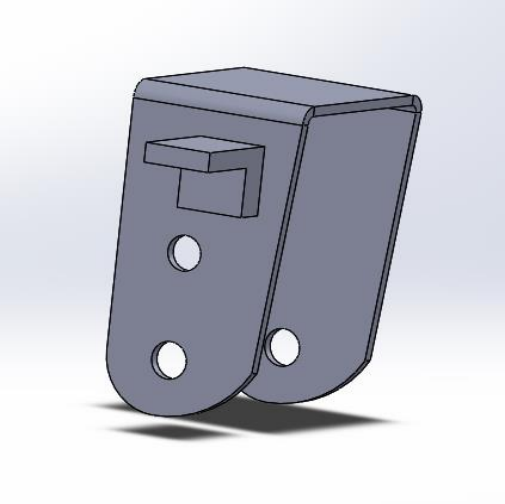


Figure 8. Caster

The caster that is shown in Figure 8 was made by stainless steel material. The top of the caster was attached to the cart to hold the wheel that the brake system is going to apply on it. As shown in Figure 8 above, the caster has two holes and one L-shaped bracket. The topmost hole is a place for the bearing to hold the rod which is going to go inside one of the slots on the wheel to lock it. The bracket is to provide alignment and support to the servomotor for the linear motion of the rod through the bearing hole. The bottom holes are used to bolt the wheel and caster for the movement of cart.

5.1.1.2 Wheel

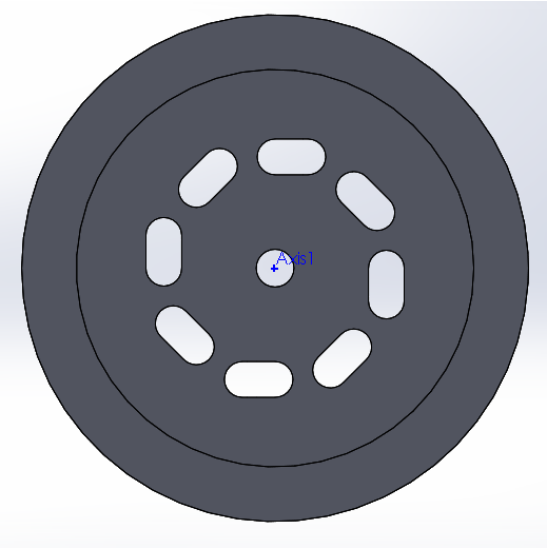


Figure 9. Wheel

In this project, a five inch rubber wheel was used to apply the braking system on it. As shown in Figure 10 above the wheel has eight slots. The dimensions of these slots are 0.33 inch length and 0.18 heights.

5.1.1.3 Servomotor

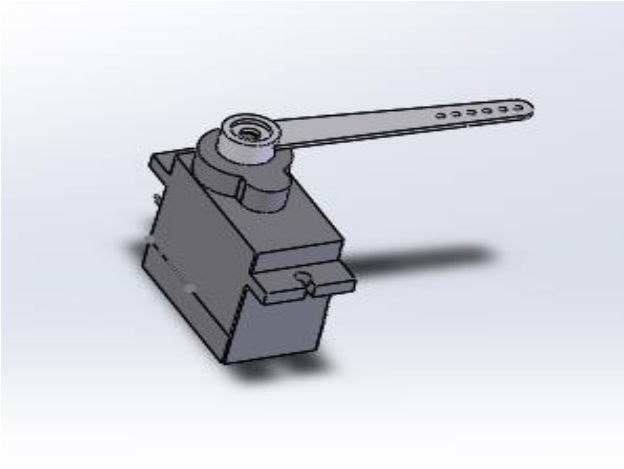


Figure 10. Servomotor

The servomotor was stuck to the caster with a mix of epoxy and concrete and then attached to rod that was made by the 3D printer. When the cart reaches the red zone a 5V signal will be sent to a servomotor that will stop the cart by acting on the brake. It is a very simple but effective circuit useful for many different electro mechanic designs. Section 5.2.2. shows the servomotor features and how to connect and code everything in order to send the signal that makes the cart stop.

5.1.1.4 Rod

Figure 11. Rod

The 1.6 inches length with 0.3 inches diameter ABS S30 rod was used in this project as shown in the figure above. The rod was attached to a servo arm that attached to the servomotor. The rod has a small slot as is shown in Figure 13 above to put the arm inside it and then pin it with small screw. Basically, the servomotor rotates to push the rod inside the wheel to lock it.

5.1.1.5 Bearing

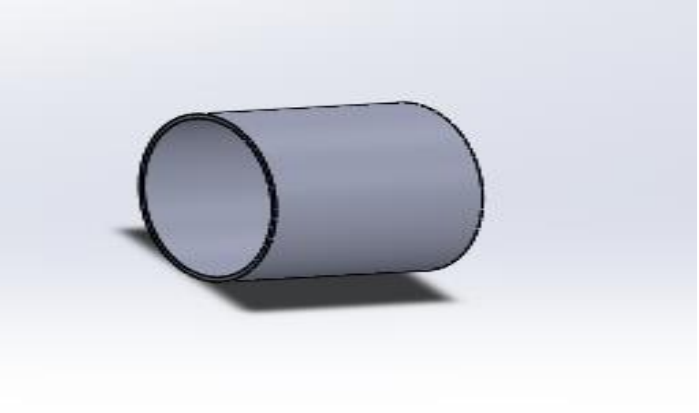


Figure 12. Bearing

The 0.32 inches bearing outside diameter and 0.31 inches inside diameter was used to hold the rod and make it move easily. Also the advantage of using bearing is taking the force that goes to the rod instead of having all the forces goes through the rod which it may break if it has a lot of forces acting on it. The material that the bearing was made of is steel. The implemented tolerance was +0.020/-0.02 inches. The upper hole in the caster seen in Figure 8 is the one that holds this bearing.

5.1.2 Mechanical Calculations

The first assumption was neglected the pushing force (applied force) because of the big distance between it and the wheel that has the brake system on it. As result, the only force that acts on the wheel on the X-axis is a friction force as shown in Figure 13 below. To calculate the friction force, the max friction coefficient rubber to asphalt was used which is 0.9 friction coefficient rubber to dry asphalt.

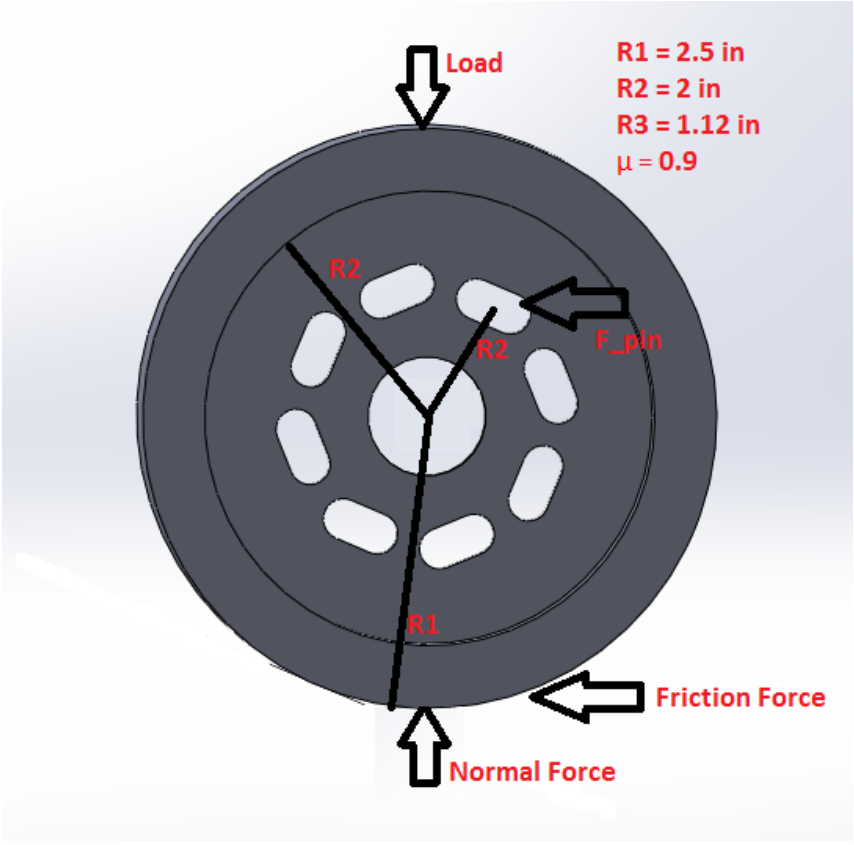


Figure 13. Free body diagram

Load=W=250/4=62.5 lb (weight of the cart used divided by the four wheels)

$$\Sigma Fy=0$$

$$N=W$$

$$N=W = (250/4)\ lb = 62.5\ lb$$

$$F_{friction}=\mu\ N \qquad (1)$$

$$\mu= 0.9 \text{ (for rubber)}$$

$$F_{friction}=0.9(62.5) = 56.25$$

$$\Sigma M=0$$

$$T=F_{friction} *(2.5) = F_{pin} *(1.12)$$

$$F_{pin}= \frac{(2.5 * F_{friction})}{1.12} = 125.55\ lb$$

As shown in the calculation above, the force on the rod and the rotational force (torque) for the braking system were calculated. The force on the rod is equal to 125.55 lb. The material used for the rod that stops the wheel is stained steel. Figure 14 and Figure 15 show the shear and the bending moment applied to the rod when the brake stops the wheel. The bearing can be modeled as a fixed support when analyzing these mechanical effects. The bearing holds the rod preventing the movement in one of the ends of it. The wheel is applying a 125.55 lb force over the pin at 1.18 inches from the fixed support.

Shear Force diagram and Bending Moment diagram were calculated for knowing the forces applied to the rod when the break is acting.

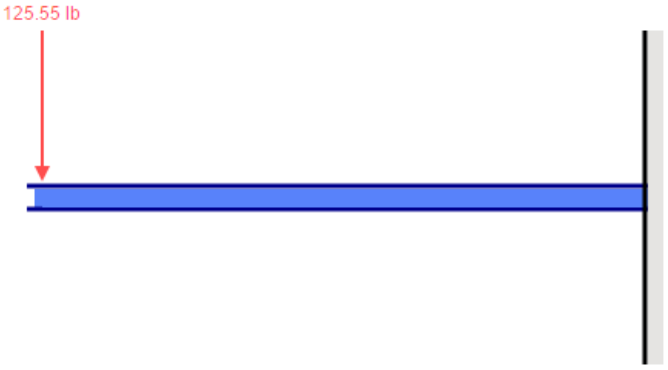


Figure 14. Rod stress state

Figure 15. Shear force diagram

Figure 16. Bending moment diagram

To determine the resistance of the ABS S30 rod, the Equation 2 was used:

$$\sigma = \frac{Fp}{A} \tag{2}$$

A = Area of the ABS S30 cylinder (in^2)

$$A = 2\pi rh + 2\pi r^2 \tag{3}$$

Where:
r =0.15 in
h=1.6in
F_{pin} = Pin force (force applied to the rod)
σ = Yield strength for ABS S30 = 2900 psi

By using Equation 2 and Equation 3, we ended up with:

$$2900(\text{psi}) = \frac{Fp}{1.65\ in\text{^}2}$$

Therefore, the resistance of the ABS S30 rod was calculated to be F_{pin} = 4785 lb which meets the required force 125.55lb.

The travel distance of the rod for locking the cart can be seen as follows:

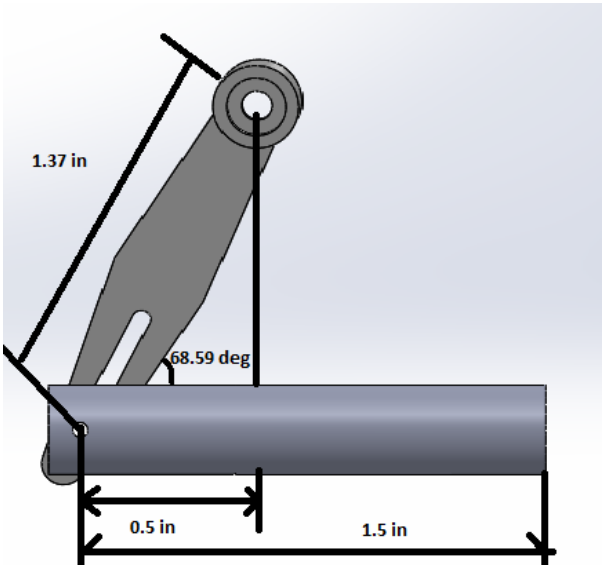


Figure 17. Rod and servomotor arm paths

Length of the rod from the centerline of pin till the end =1.5 inches

Length of the servo arm from centerline of the servomotor gear till the end= 1.37 inches

The hypotenuse for the rod is 1.37 in

Therefore,

Cosine inverse (0.5/1.37)= 68.59 degrees

This the angle at which the servo arm moves to provide linear motion to the rod for breaking the wheel movement.

Thus, the servomotor should be set to 68.59 degrees for achieving the linear motion of rod.

The travel distance is 0.5 inches to get the wheel locked.

5.1.3. Mechanical Drawings

The templates show what the team designed for the mechanical system. All the components explained can be seen all the measures in US units. The program used for doing these templates was SolidWorks.

"Work load assembly" template shows the 5.1" wheel that has the braking system with the bearing, the servomotor and the caster on it. Holes were made in the wheel to introduce a ABS S30 rod in it in order to stop it. Those holes can be seen on this template. An overall 3D drawing of this wheel is shown as well on this template.

"Magnet assembly" template shows how the team implemented a magnet onto the left back wheel of the cart in order to use a Hall Effect Sensor to detect when the cart is moving.

"Servo motor horn" template describes the component that the servo is moving in order to move the rod. This is like a blade printed by the 3D printer available in University of Dayton Kettering Labs.

"Bearing" template shows the bearing that holds the rod when it is going through the caster. This bearing was purchased after doing some research on how could implement a support that only allows the linear movement for locking the wheel and that also holds the rod when the wheel is locked.

"Slotted Rod" template shows the ABS S30 rod used for locking the cart.

These templates can be seen on Appendix F.

5.2 Electrical Design

The electrical design is based on some different circuits that have to be exposed separately before explaining the system overall design. A GPS Shield welded on the Arduino Uno with an SD card in it communicates with the Arduino and after defining the boundaries with a push button circuit the microcontroller is able to know if it is inside the boundaries. If the cart reaches the red zone a 5V signal is sent to a servomotor to lock the cart. A Hall Effect sensor is used to start receiving GPS coordinates. A push button circuit code for saving the coordinates that was used for define the boundaries of the store parking lot.

5.2.1 Hall Effect Sensor Circuit.

Hall Effect Sensors are devices which are activated by an external magnetic field. A magnetic field has two important characteristics flux density (B) and polarity (North and South Poles). The output signal from a Hall Effect Sensor is the function of magnetic field density around the device. When the magnetic flux density around the sensor exceeds a certain pre-set threshold, the sensor detects it and generates an output voltage called the Hall Voltage, V_H. Consider the diagram in Figure 18.

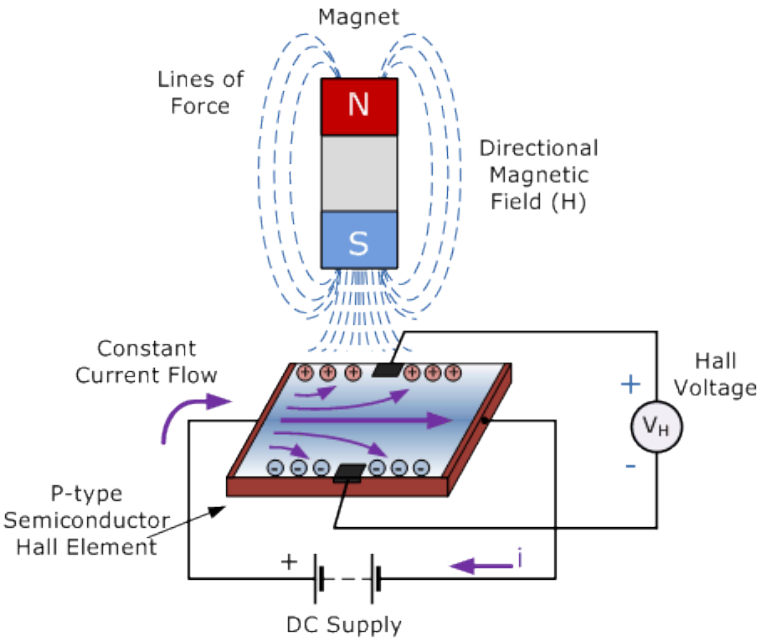


Figure 18. Hall Effect Sensor theory diagram

The output signal for linear (analogue) sensors is taken directly from the output of the operational amplifier with the output voltage being directly proportional to the magnetic field passing through the Hall sensor. This output Hall voltage is given as:

$$V_H = R_H * (\frac{I}{t} * B) \qquad (4)$$

- Where:
- V_H is the Hall Voltage in volts
- R_H is the Hall Effect coefficient
- I is the current flow through the sensor in amps
- t is the thickness of the sensor in mm
- B is the Magnetic Flux density in Tesla

The design works with 5V coming from the Arduino. Two magnets had been incorporated to one of the sides of the wheel so the Hall Effect sensor detects the magnetic field that they create each time that the wheel turns around. This can be seen on Appendix E in the template "Magnet Assembly".

This design uses a Hall Effect switch to turn the Arduino UNO's built-in led on and off with a magnet.

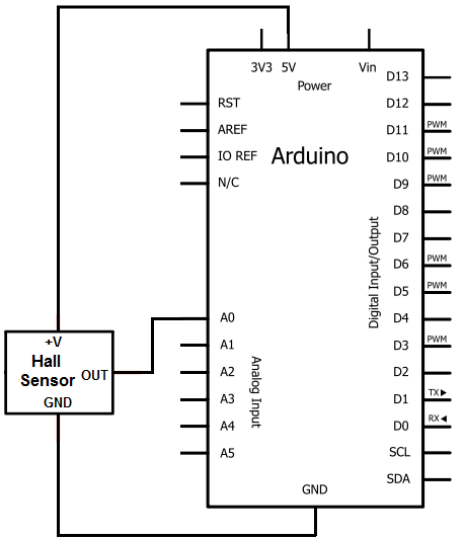


Figure 19. Hall Effect Sensor connections

US1881 Hall Effect Sensor was used

- Connect pin 1 of the switch to the Arduino +5V supply (red wire)
- Connect pin 2 to 0V (black wire)
- Connect pin 3 to Arduino input pin 12 (orange wire)

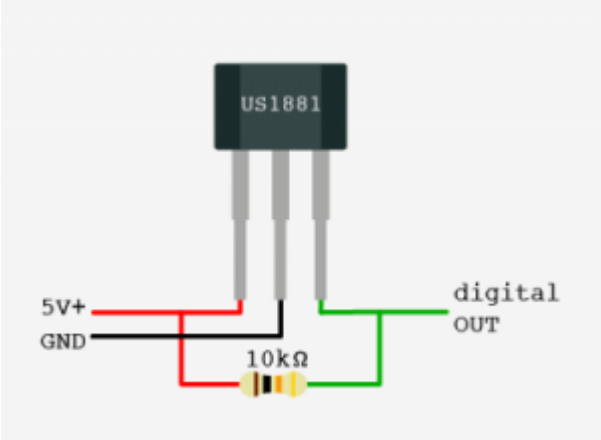


Figure 20. Hall Effect Sensor pins

A pull-up resistor was required between pin 1 and pin 3 to pull-up the switch's output to +5V. The resistor chosen is a 1K resistor.

When the switch detects a magnet, it will pull its output pin low, which can be easily detected on the Arduino board.

Figure 23 shows how a LED turns on when the Hall Effect Sensor is detecting the magnetic field.

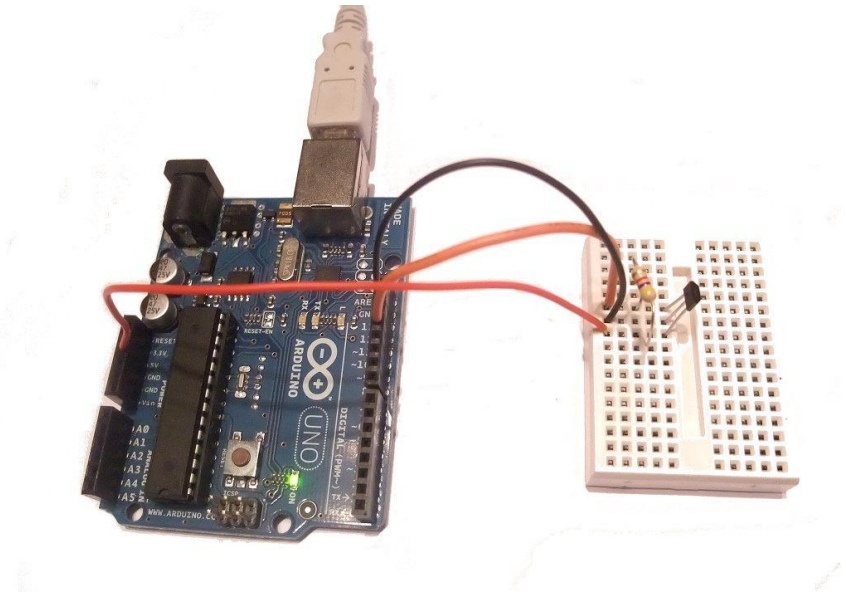


Figure 21. Circuit image

The code used for controlling the Hall Effect sensor with the Arduino is attached in the Appendix A of this report as well as its explanation.

The assembly of the Hall Effect sensor onto the cart was done after coding it. First a 1.37x0.78 inch portion of breadboard was cut in order to have the least material as possible in the project. Wires, resistors and the sensor were placed on the breadboard as described before in this section. A 0.6 inch diameter magnet was placed on one of the sides of the wheel. The breadboard was placed on the caster so that each rotation of the wheel the Hall Effect sensor was detecting the magnet. This assembly can be seen on Figure 22.

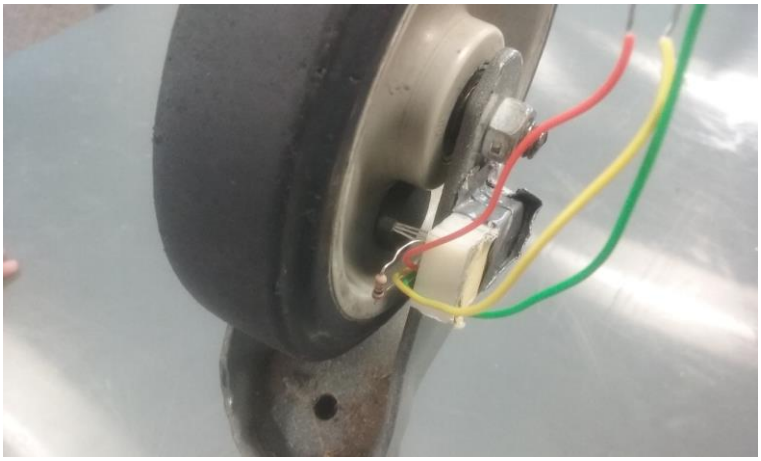


Figure 22. Hall Effect Sensor assembly

When the final Smart Cart was built a Hall Effect sensor wasn't tested within the GPS code. It was a time problem. Even though the Hall Effect sensor wasn't tested with the GPS it was assembled and tested separately and it was perfectly working.

We can see the following in Appendix E:

```
   //delay(500);
   // Check to see if wheel went around twice
//   while(rev_count < 2) {
//     // read the state of the hall effect sensor:
//     hallState = digitalRead(hallPin);
//
//     if (hallState == LOW) {
//       rev_count++;
//     }
//     delay(100); //Delay so that doesn't count same hallState more than once
//   }
```

This is a way to detect two revolutions of the wheel so the Hall Effect sensor would detect the beginning of the cart motion. This system was not finally developed in the final design because of there an unexpected GPS problem that was fixed on the last day of the semester.

5.2.2 Servomotor Circuit.

When the cart reaches the red zone a 5V signal is sent to a servomotor that stops the cart by acting on the brake. It is a very simple but effective circuit used for many different electro mechanic designs. In this section we explain the servomotor features and how to connect and code everything in order to send that signal that will make the cart stop.

The servo used for this project is a Towerpro MG92B 360º Mini Digital Robotic Servo.

Figure 23. Towerpro servomotor

It can achieve a great speed and has the ability to rotate continuously through 360° at full travel. This 360° feature is perfect for robotic use.

Features:
• High resolution
• Accurate positioning
• Fast control response
• Constant torque throughout the servo travel range
• Excellent holding power
• Ability to rotate continuously through 360° in both directions

Specs:
Input Voltage: 5~6.6V
Operating Force: 6.83lb@5V, 7.49lb@6.6V (at an arm length = 0.98 inches)
Operating Speed: 0.78sec/60°@5V, 0.48sec/60°@6V
Normal Servo Travel: 90° (45° each way) 360° when full travel is reached
Case: Top & bottom plastic, middle is alloy
Bearing: Double ball bearing
Gears: 6061-T6 aluminum
Temperature Range: 0~55°C
Dimensions: 0.897 x 0.472 x 1.22 inches
Weight: 0.03 lb

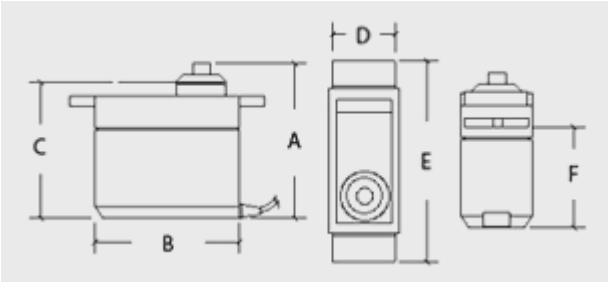


Figure 24. Servomotor dimensions

A	1.377 inches
B	0.905 inches
C	1.22 inches
D	0.472 inches
E	1.22 inches
F	0.826 inches

Table 1. Servomotor dimensions

The angle described by the servo movement was from 30º to 70º. This can be seen on Appendix B.

Arduino connections were the following:

-Brown wire to the Arduino Ground

-Red wire to the Arduino 5V pin (VCC)

-Orange wire to an Arduino Pin 4

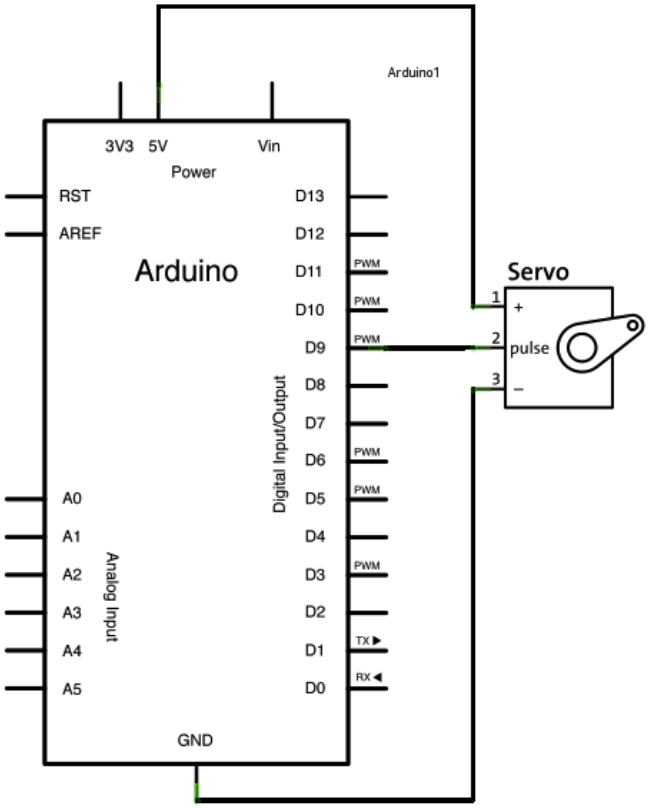


Figure 25. Servomotor circuit

.

After wiring the circuit and coding a single servo code, the desired movement from 30º to 70º was achieved so the servo motor code could be included in the final code on Appendix E.

In Figure 26 and Figure 27 we can see the servo assembled onto the caster and doing both locking and unlocking positions. The rod wasn't ready at that moment so that is why there is nothing attached to the servo horn for locking the wheel in both figures.

Figure 26. Servo in 30 degrees position (unlocked)

Figure 27. Servo in 70 degrees position (locked)

5.2.3 Push Button Circuit

This design needed a way to save the coordinates received from the GPS as the corners of the parking lot boundaries a push button circuit was connected to the Arduino board by pressing it each time the GPS shield was on a corner point so the coordinate was saved into the SD card of the GPS shield. This is a simple circuit that involves a switch. Pushbuttons or switches connect two points in a circuit when you press them. For this circuit we needed the following:

-Arduino Board.
-Momentary Button or Switch.
-10kΩ resistor.
-Hook-up wires.
-Breadboard.

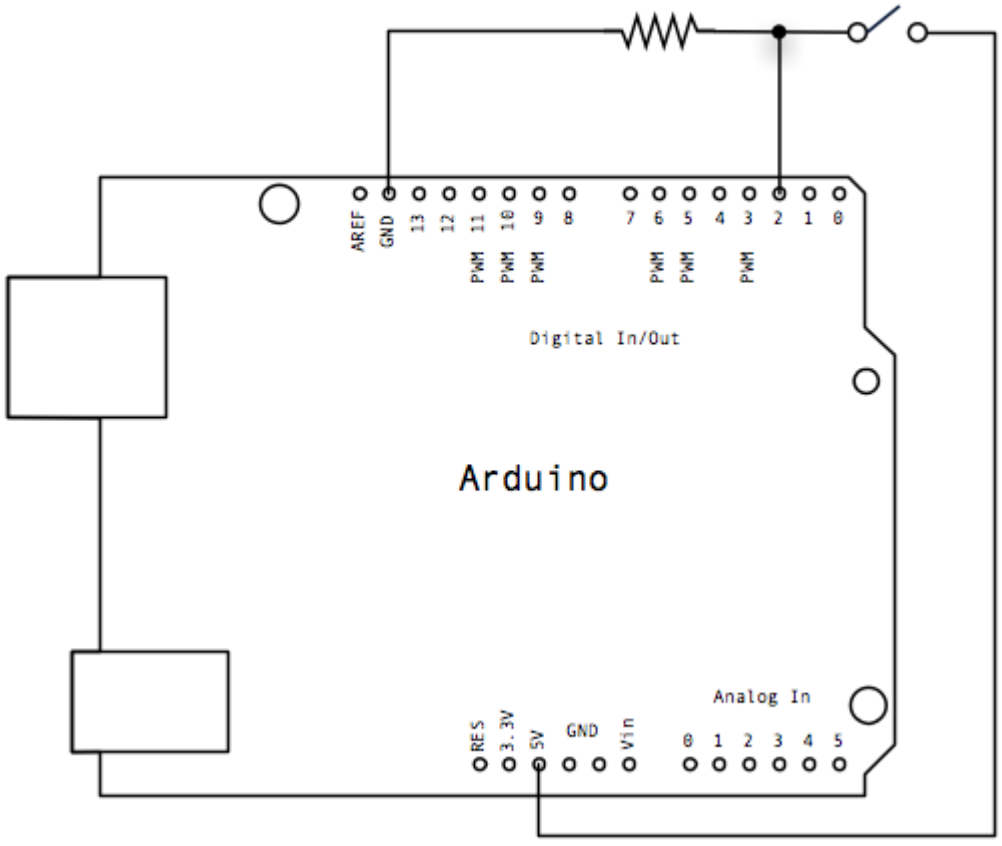


Figure 28. Push Button Circuit

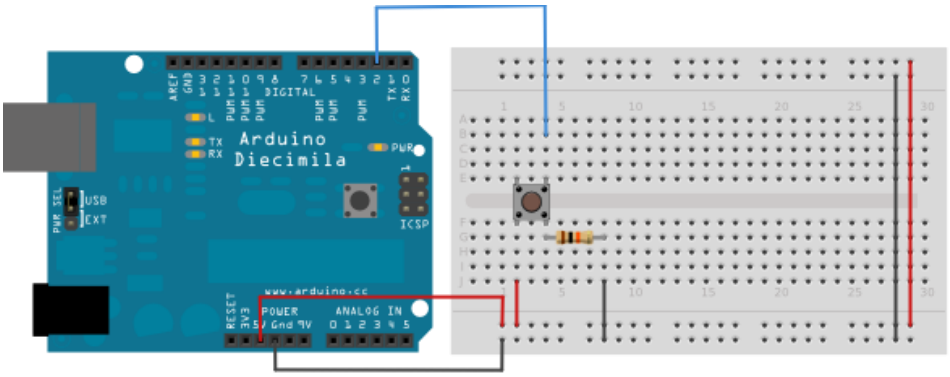


Figure 29. Push Button circuit image

There were three wires connected from the Arduino to the breadboard. The first two, red and black, were connected to the two long vertical rows on the side of the breadboard to provide access to the 5V supply and ground. The third wire went from digital pin 2 to one leg of the pushbutton. That same leg of the button was connected through a pull-down resistor (here 10Kk ohm) to ground. The other leg of the button was connected to the 5V supply.

When the push button is open there is no connection between the two legs of the pushbutton, so the pin is connected to ground (through the pull-down resistor) and we read a LOW. When the button is closed (pressed), it makes a connection between its two legs, connecting the pin to 5V, so that a HIGH is read.

Appendix C shows the code used for testing this circuit with LEDs before the implementation of it to the GPS code.

This circuit was implemented to the GPS code so that each time the Push Button circuit gave a HIGH we saved the coordinate that the GPS was receiving into the SD and after that we could define that point as a corner of the boundary so that we could save those coordinates in a text file that would be read in the final code. The following code has been extracted from Appendix E and shows how the code read the save GPS coordinates from the .TXT file:

```
file = SD.open("GPSLOG44.TXT", FILE_READ); //File that has the GPS boundaries
if (!file) {
  Serial.println("open error");
  return;
```

There were 4 corner points saved in the SD card text file for the final design.

5.2.4 Ultimate GPS Logger shield for Arduino

This shield came with a GPS unit and a microSD socket on it so it was perfect for what the we needed for this project. We needed to save some coordinates into somewhere to define the boundaries and know whether they were in or outside of the defined free-run zone so this was the reason why we needed an SD card in the cart control system. The GPS unit was used to receive the coordinates from 3-4 satellites at the same time so that it could determinate by mathematical triangulation where the GPS was.

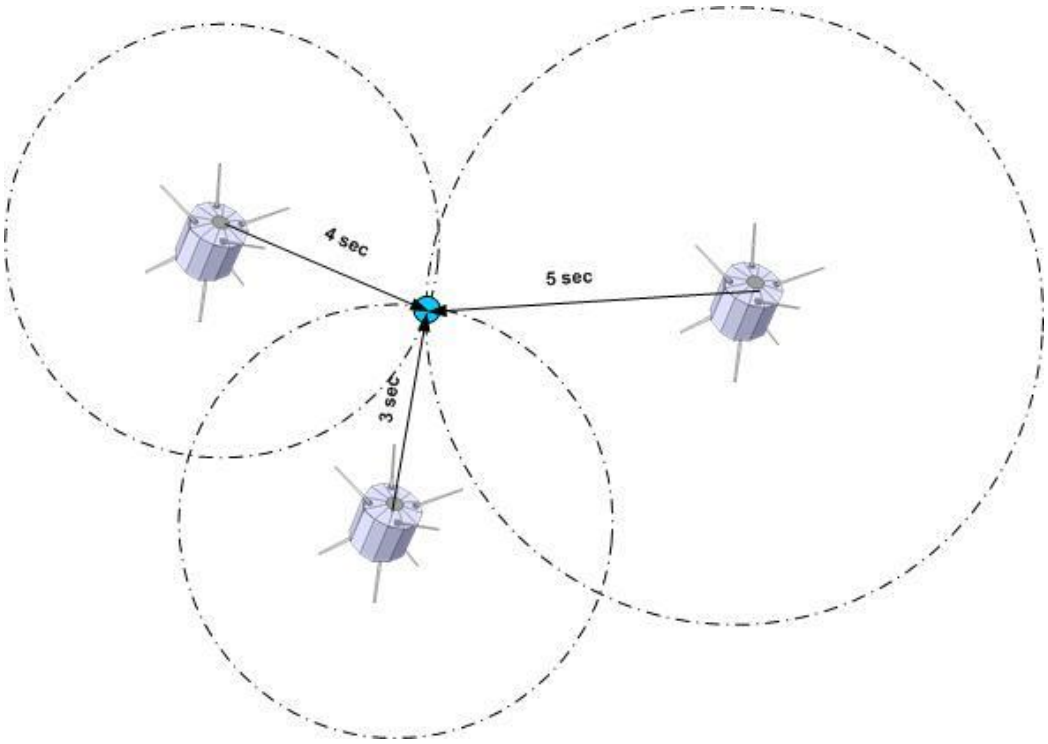


Figure 30. GPS explanation three satellites

This design was developed in University of Dayton Kettering Labs so that the we were working with a latitude and longitude with similar first decimal numbers as the following:

Latitude: 39.737

Longitude: -84.1764

The free-run zone corner points were defined with the push button circuit as we have said on Section 5.2.3.

These points were being checked using Google Maps like the following figure shows:

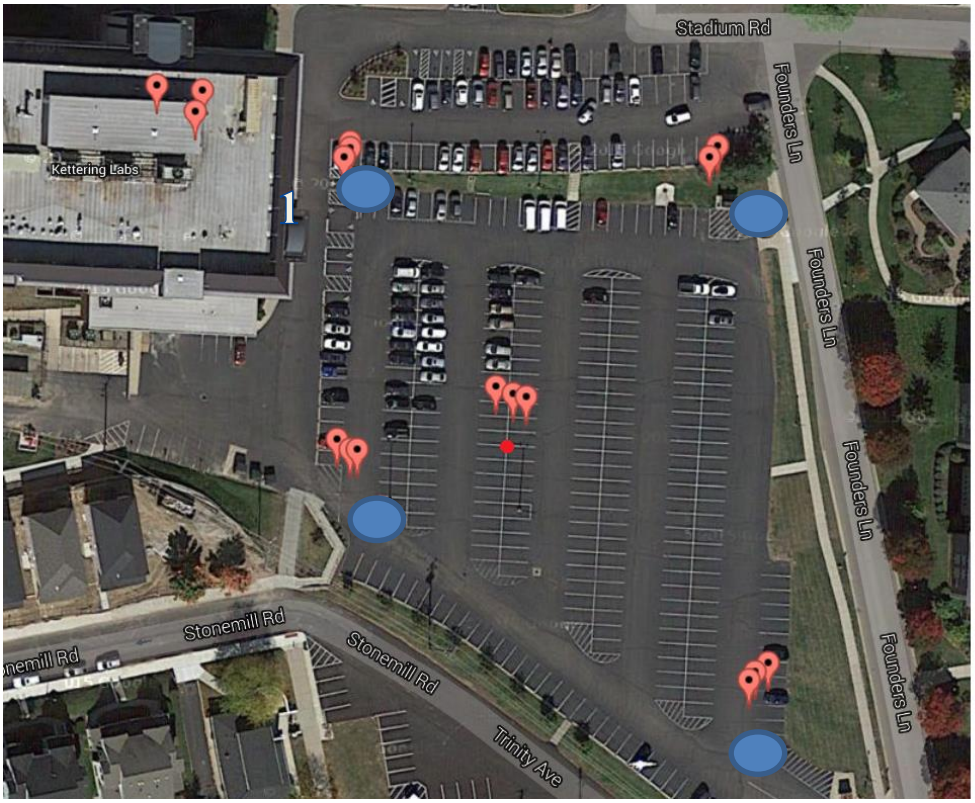


Figure 31. Boundaries used for the project

Figure 31 shows the parking lot used for the project boundaries. Those points that were closer to the University of Dayton Kettering Labs building for engineering were less accurate due to the building height hindering the signal from one or two of the satellites. We can see four corners. The most accurate points were saved in the .txt file and the cart locked when outside the boundaries with a tolerance of +/-1 meter.

This GPS shield was bought in *Adafruit.com.* It came assembled with a GPS unit and a microSD socket already on it, but the we still needed to put headers on so we could plug it into the Arduino board. The nice thing about using these headers is they don't add anything to the height of the board, and they make a nice solid connection. This assembly process can be seen on Figure 35.

Figure 32. GPS shield

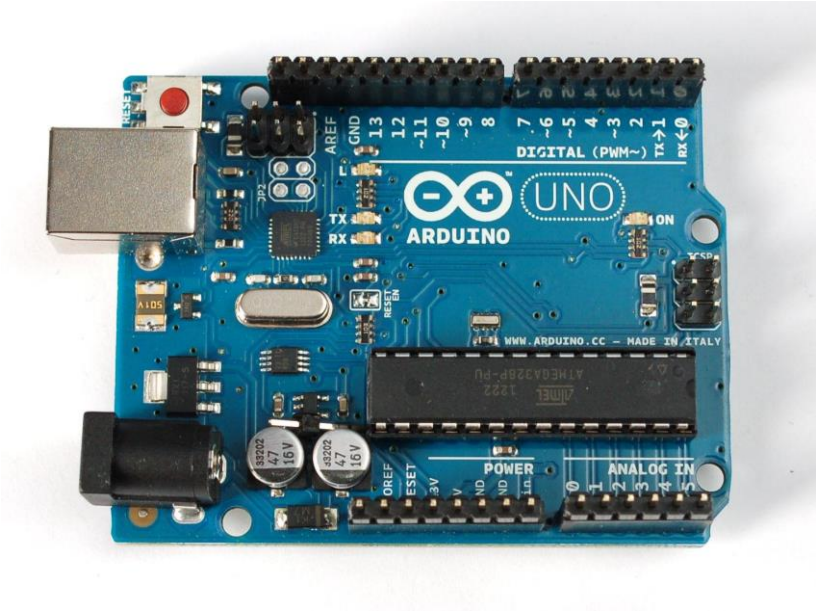


Figure 33. Arduino Uno

We placed the shield on the top and solder all the pins.

Figure 34. GPS shield onto the Arduino Uno

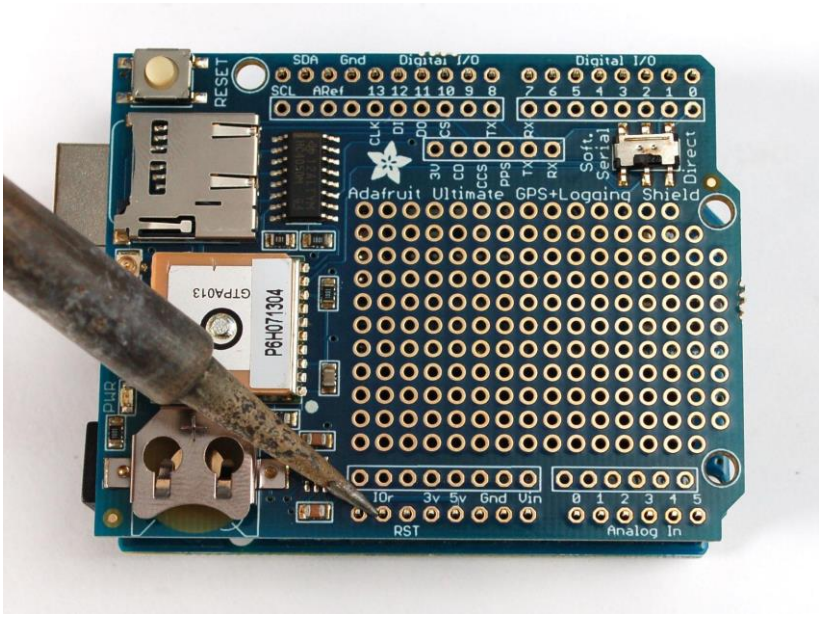


Figure 35. Welding

The GPS shield is capable to record coordinates and display them in the Serial Monitor based on example codes that came with the shield. For its use with the Smart Cart, the GPS coordinates just needed to be saved onto the SD card, which is also on the same shield. The coordinates are saved into a .txt file on the SD card. Once the coordinates are saved onto the SD card, the Arduino can then parse the coordinates and use them to define the boundaries of the parking lot. By using a simple algorithm the GPS shield can determine whether or not the current GPS coordinate is inside or outside the defined parameter of the parking lot and tell the brake to lock up to prevent the cart from leaving the parking lot.

The steps for setting up the shield are shown on Appendix D.

5.2.5 Power supply

An Arduino UNO can be supplied by two ways for this project:

-USB connection (5V).

-Battery pack (AA).

Due to we were looking to develop a low-cost design we picked a portable USB battery assembled to the cart to power the whole electric system of it. We had to find something able to power the cart for more than 6 hours so that a cart can work during all the morning through the store. An USB power station has to be built within the store to charge these USB portable batteries.

Generally the battery is calculated on the current classification of mAh. The battery life or capacity can be calculated from the entrance of the current rating of the battery and the load current circuit. The battery will be high when the load current is less and vice versa. The calculation to determine the battery capacity can be divided mathematically in equation 5.

$$Battery\ life\ = \frac{Battery\ capacity\ in\ mAh}{Load\ current\ in\ mAh} * 0.7 \qquad (5)$$

The factor 0.7 allows tolerances to external factors which may affect the life of the battery.

The average Smart Cart electrical system consumption is about 220-250mAh.

The external battery used to power the project is Anker PowerCore mini 3350mAh Lipstick-Sized Portable Charger.

Figure 36. Anker portable battery

Specifications:

-Size: 3.5 x 0.9 inches.

-Battery capacity: 3350mAh

-Weight: 3 ounces.

Equation 5 was used to determine the approximate hours of life that the Smart Cart can have with this power supply.

$$Battery\ life\ = \frac{335\ mAh}{250\ mAh} = 9.38\ hours$$

Due to this is a low-cost project the team thought that this would be a good idea in order to decrease the cost of using AA, which could provide more power but the cost is much higher in the long-term.

6. Cost

This project was started to be a low-cost project so the team tried to reduce the cost to the minimum possible. Costs for electrical and mechanical part are shown in sections 7.1 and 7.2. Total cost of the project was **$136.23**

6.1 Electrical Cost

Table 2. Electrical cost

Arduino Uno	Adafruit	$24.95
Ultimate GPS Logger shield	Adafruit	$44.95
Anker PowerCore+ mini	Amazon	$9.95
Hall Effect Sensor – US5881LUA	Adafruit	$2.00
Magnets (100 u)	Amazon	$8.47
Servomotor	Amazon	$15.99
microSD Card 16GB	Amazon	$10.07
TOTAL		**$116.38**

6.2 Mechanical Cost

Table 3. Mechanical Cost

Components	Website	Cost
Bearing	McMASTER-CARR	$5.05
Aluminum Servo Arm	Amazon	$8.36
Grand Approve Caster Brake	Amazon	$8.37
5” Wheel	Amazon	$7.99
Loctite Metal and Concrete Epoxy Syringe (Glue)	Amazon	$6.81
TOTAL		**$19.85**

7. Overall system

Once the electrical and mechanical parts were designed, the team came up with an overall assembly of the circuits and the mechanical components. The servomotor circuit and the brake system were both assembled onto the right back wheel while the Hall Effect Sensor circuit described in section 5.2.1 was connected from the Arduino to the left back wheel. The Arduino with GPS shield and a 5V battery remained inside a polymer box assembled in the center of the cart. A magnet was assembled onto the left back wheel as shown in the *Magnet Assembly* template in Appendix F where mechanical assembly is described as well.

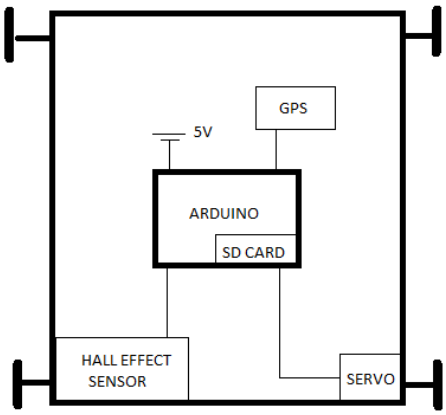


Figure 37. Cart assembly

Figure 38. Final design

8. PWM

Once all the code and electric wiring was finished it was the right moment to go out and test the design in the parking lot. When the cart went out the boundaries the servo received the signal and tried to lock the cart but there was an unexpected movement from the servo. When it tried to go through the bearing and lock the wheel it seemed like the signal was very noisy and the servo couldn't complete the movement as desired.

After doing some research we found that the *Servo.h* library that was being used in the Appendix E code interrupts the movement causing the servo motor to not behave the way it was supposed to. To solve this, another way to control the servo was found: PWM.

A new servo library used PWM signals instead of interrupts so the old *Servo.h* library was swapped out and a new *PWMServo.h* library was put within the code.

The PWM signal is used as a technique for controlling analog circuits. Servos are controlled by sending them a pulse of variable width. This means that the output is a pulse modulated signal, formed by a series of pulsed (value 5V) spread over a given time so that the average value of the output signal is matched with the analog signal pursued to imitate.

In conclusion, the PWM is a modulation that we are doing to the signal in order to control the amount of energy that we are sending to the servo. So the servo is finally controlled by three wires: one for the 5V voltage, another one for the ground connection and the last one for the signal that reaches the servo modulated by PWM.

Appendix A: Hall Effect Sensor code

```
/*
 Hall Effect Switch

 Turns on and off a light emitting diode(LED) connected to digital
 pin 13, when Hall Effect Sensor attached to pin 2 is triggered by a magnet

 Hall effect sensor used is the A1120EUA from Allegro Microsystems

.*/

// constants won't change. They're used here to set pin numbers:
const int hallPin = 12;     // the number of the hall effect sensor pin
const int ledPin =  13;     // the number of the LED pin
// variables will change:
int hallState = 0;          // variable for reading the hall sensor status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the hall effect sensor pin as an input:
  pinMode(hallPin, INPUT);
}

void loop(){
  // read the state of the hall effect sensor:
  hallState = digitalRead(hallPin);

  if (hallState == LOW) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);  // when the Hall Effect  Sensor detects the magnet
  }
  else {
    // turn LED off:
```

```
    digitalWrite(ledPin, LOW); //if there isn't a magnetic field the LED remains off.
  }
}
```

Appendix B: Servomotor code

This code easily explains how to move the servo chosen for this project.

```
#include <Servo.h>
Servo servo;
int angle = 10;
void setup() {
  servo.attach(8);
  servo.write(angle);
}
void loop()
{
 // scan from 30 to 70 degrees
  for(angle = 30; angle < 70; angle++)
  {
    servo.write(angle);
    delay(15);
  }
  // now scan back from 70 to 30 degrees
  for(angle = 70; angle > 30; angle--)
  {
    servo.write(angle);
    delay(15);
  }
}
```

Appendix C: Push Button code

The following code is for saving the desired coordinates into the SD card. It was obtained from Arduino help support.

```
#include <SPI.h>
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
#include <SD.h>
#include <avr/sleep.h>

const int buttonPin = 2;     // the number of the pushbutton pin
int buttonState = 0;

//
// Tested and works great with the Adafruit Ultimate GPS Shield

boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy

// Set the pins used
#define chipSelect 10
#define ledPin 13

File logfile;

// read a Hex value and return the decimal equivalent
uint8_t parseHex(char c) {
  if (c < '0')
    return 0;
  if (c <= '9')
    return c - '0';
  if (c < 'A')
    return 0;
  if (c <= 'F')
```

```
    return (c - 'A')+10;
}

// blink out an error code
void error(uint8_t errno) {
  /*
  if (SD.errorCode()) {
   putstring("SD error: ");
   Serial.print(card.errorCode(), HEX);
   Serial.print(',');
   Serial.println(card.errorData(), HEX);
   }
   */
  while(1) {
    uint8_t i;
    for (i=0; i<errno; i++) {
      digitalWrite(ledPin, HIGH);
      delay(100);
      digitalWrite(ledPin, LOW);
      delay(100);
    }
    for (i=errno; i<10; i++) {
      delay(200);
    }
  }
}


void setup() {

  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);

  // for Leonardos, if you want to debug SD issues, uncomment this line
```

```
  // to see serial output
  //while (!Serial);

  // connect at 115200 so we can read the GPS fast enough and echo without dropping
chars
  // also spit it out
  Serial.begin(115200);
  Serial.println("\r\nUltimate GPSlogger Shield");
  pinMode(ledPin, OUTPUT);

  // make sure that the default chip select pin is set to
  // output, even if you don't use it:
  pinMode(10, OUTPUT);

  // see if the card is present and can be initialized:
  //if (!SD.begin(chipSelect, 11, 12, 13)) {
    if (!SD.begin(chipSelect)) {      // if you're using an UNO, you can use this line instead
    Serial.println("Card init. failed!");
    error(2);
  }
  char filename[15];
  strcpy(filename, "GPSLOG00.TXT");
  for (uint8_t i = 0; i < 100; i++) {
    filename[6] = '0' + i/10;
    filename[7] = '0' + i%10;
    // create if does not exist, do not open existing, write, sync after write
    if (! SD.exists(filename)) {
      break;
    }
  }

  logfile = SD.open(filename, FILE_WRITE);
  if( ! logfile ) {
    Serial.print("Couldnt create ");
    Serial.println(filename);
```

```
  error(3);
 }
 Serial.print("Writing to ");
 Serial.println(filename);

 // connect to the GPS at the desired rate
 GPS.begin(9600);

 // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitude
 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
 // uncomment this line to turn on only the "minimum recommended" data
 //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
 // For logging data, we don't suggest using anything but either RMC only or RMC+GGA
 // to keep the log files at a reasonable size
 // Set the update rate
 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 100 millihertz (once every 10 seconds), 1Hz or 5Hz update rate

 // Turn off updates on antenna status, if the firmware permits it
 GPS.sendCommand(PGCMD_NOANTENNA);

 // the nice thing about this code is you can have a timer0 interrupt go off
 // every 1 millisecond, and read data from the GPS for you. that makes the
 // loop code a heck of a lot easier!
 //useInterrupt(true);

 Serial.println("Ready!");
}


void loop() {
 // read the state of the pushbutton value:
 buttonState = digitalRead(buttonPin);
```

```
if (! usingInterrupt) {
  // read data from the GPS in the 'main loop'
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
  if (GPSECHO)
    if (c) Serial.print(c);
}

// if a sentence is received, we can check the checksum, parse it...
if (GPS.newNMEAreceived()) {
  // a tricky thing here is if we print the NMEA sentence, or data
  // we end up not listening and catching other sentences!
  // so be very wary if using OUTPUT_ALLDATA and trying to print out data

  // Don't call lastNMEA more than once between parse calls!  Calling lastNMEA
  // will clear the received flag and can cause very subtle race conditions if
  // new data comes in before parse is called again.
  char *stringptr = GPS.lastNMEA();

  if (!GPS.parse(stringptr))   // this also sets the newNMEAreceived() flag to false
    return;  // we can fail to parse a sentence in which case we should just wait for another

  // Sentence parsed!
  Serial.println("OK");
  if (LOG_FIXONLY && !GPS.fix) {
    Serial.print("No Fix");
    return;
  }

  // Rad. lets log it!
  Serial.println("Log");
  if (buttonState == HIGH) {
  Serial.println("Button Pressed!!!!");
  delay(4000);
```

```
    logfile.print(GPS.latitudeDegrees, 6);
    logfile.print(",");
    logfile.println(GPS.longitudeDegrees, 6);
    //logfile.println(";");
    }
    if (strstr(stringptr, "RMC") || strstr(stringptr, "GGA"))   logfile.flush();
  }
}


/* End code */
```

Appendix D: Setting up the GPS shield

The GPS shield has been set following the instructions in:
https://www.adafruit.com/products/1272

The necessary steps are explained here:

First, load a 'blank' sketch into the Arduino:

```
// this sketch will allow you to bypass the Atmega chip
// and connect the GPS directly to the USB/Serial
// chip converter.
// Connect VIN to +5V
// Connect GND to Ground
// Connect GPS RX (data into GPS) to Digital 0
// Connect GPS TX (data out from GPS) to Digital 1
void setup() {}
void loop() {}
```

This will free up the converter so it ease to directly wire and bypass the Arduino chip. Once this sketch has been uploaded, the switch is flipped on the shield to Direct.

Figure 39. GPS shield pins

Now the USB cable, and open up the serial monitor from the Arduino IDE and be sure to select 9600 baud in the drop down. A text like the following appears

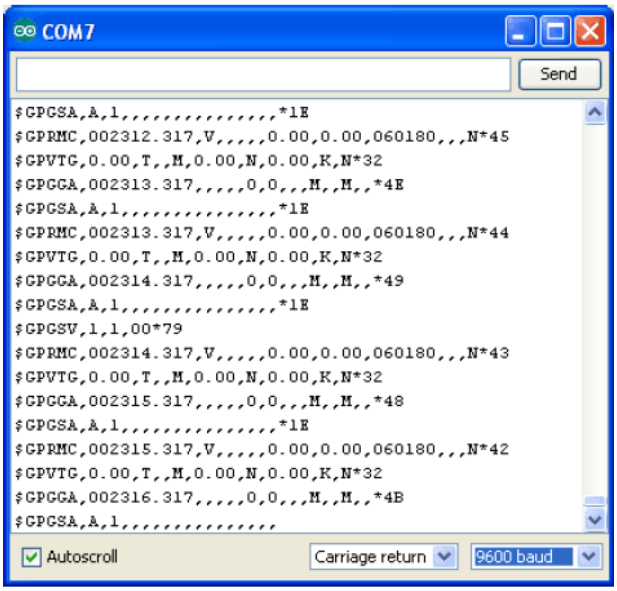


Figure 40. Settings text

:

Look for the line that says:
$GPRMC,194509.000,A,4042.6142,N,07400.4168,W,2.03,221.11,160412,,,A*77
This line is called the RMC (Recommended Minimum) sentence and has pretty much all of the most useful data. Each chunk of data is separated by a comma.
The first part 194509.000 is the current time GMT (Greenwich Mean Time). The first two numbers
19 indicate the hour (1900h, otherwise known as 7pm) the next two are the minute, the next two are
the seconds and finally the milliseconds. So the time when this screenshot was taken is 7:45 pm
and 9 seconds. The GPS does not know what time zone you are in, or about "daylight savings" so you will have to do the calculation to turn GMT into your timezone.
The second part is the 'status code', if it is a V that means the data is Void (invalid). If it is an A that
means its Active (the GPS could get a lock/fix)

Once you get a fix using your GPS module, verify your location with google maps (or some other mapping software). Remember that GPS is often only accurate to 5-10 meters and worse if you're indoors or surrounded by tall buildings.

Appendix E: GPS code

```
#include <SPI.h>
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
#include <SD.h>
#include <PWMServo.h> // this is the library that we have mentioned in Section 8

// For Hall Effect Sensor
const int hallPin = 12;     // the number of the hall effect sensor pin
// variables will change:
int hallState = 0;          // variable for reading the hall sensor status
int rev_count = 0;

//// For Servo Motor
PWMServo myservo;  // create servo object to control a servo
                   // a maximum of eight servo objects can be created

int pos = 0;    // variable to store the servo position

// For reading in .txt file
File file;
// Set GPSECHO to 'false' to turn off echoing the GPS data to the Serial console
// Set to 'true' if you want to debug and listen to the raw GPS sentences
#define GPSECHO  true
/* set to true to only log to SD when GPS has a fix, for debugging, keep it false */
#define LOG_FIXONLY false

// this keeps track of whether we're using the interrupt
// off by default!
boolean usingInterrupt = false;
void useInterrupt(boolean); // Func prototype keeps Arduino 0023 happy

double lat1;
double lon1;

double lat3;
double lon3;

int Locked_State;
/* set to true to only log to SD when GPS has a fix, for debugging, keep it false */
#define LOG_FIXONLY false

SoftwareSerial mySerial(8, 7);
Adafruit_GPS GPS(&mySerial);
```

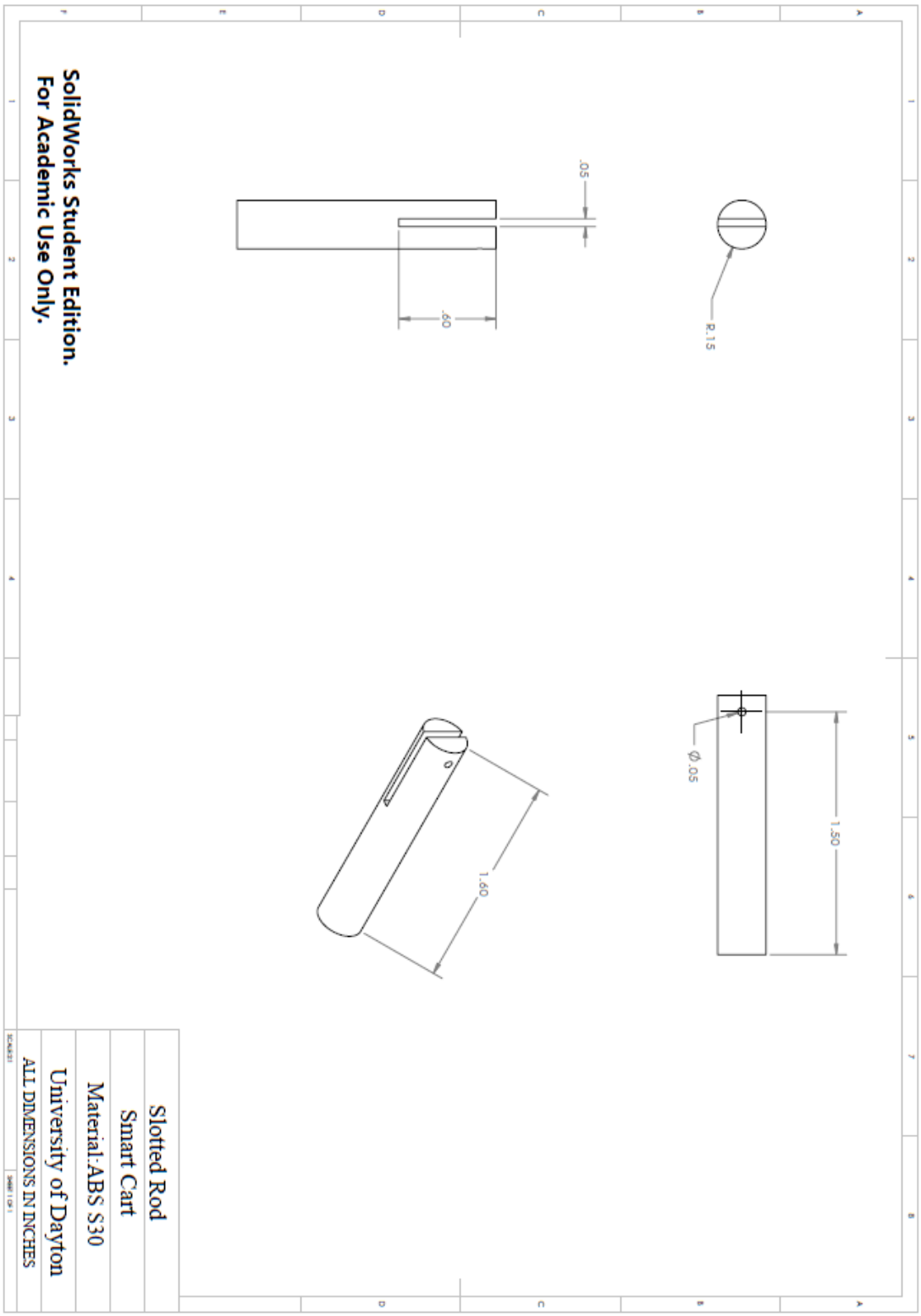
```
bool readLine(File &f, char* line, size_t maxLen) {
  for (size_t n = 0; n < maxLen; n++) {
    double c = f.read();
    if ( c < 0 && n == 0) return false;  // EOF
    if (c < 0 || c == '\n') {
      line[n] = 0;
      return true;
    }
    line[n] = c;
  }
  return false; // line too long
}

bool readVals(double* v1, double* v2) {
  char line[40], *ptr, *str;
  if (!readLine(file, line, sizeof(line))) {
    return false;  // EOF or too long
  }
  *v1 = strtod(line, &ptr);
  if (ptr == line) return false;  // bad number if equal
  while (*ptr) {
    if (*ptr++ == ',') break;
  }
  *v2 = strtod(ptr, &str);
  return str != ptr;  // true if number found
}

void setup(){
  //initialize the servo motor
  myservo.attach(SERVO_PIN_A);  // attaches the servo on pin 9 to the servo object



  // initialize the hall effect sensor pin as an input:
  pinMode(hallPin, INPUT); //Hall Effect Senor to PIN 13



  double x, y;
  Serial.begin(115200);
  if (!SD.begin(SS)) {
    Serial.println("begin error");
    return;
  }
  file = SD.open("GPSLOG44.TXT", FILE_READ); //File that has the GPS bounaries
  if (!file) {
```

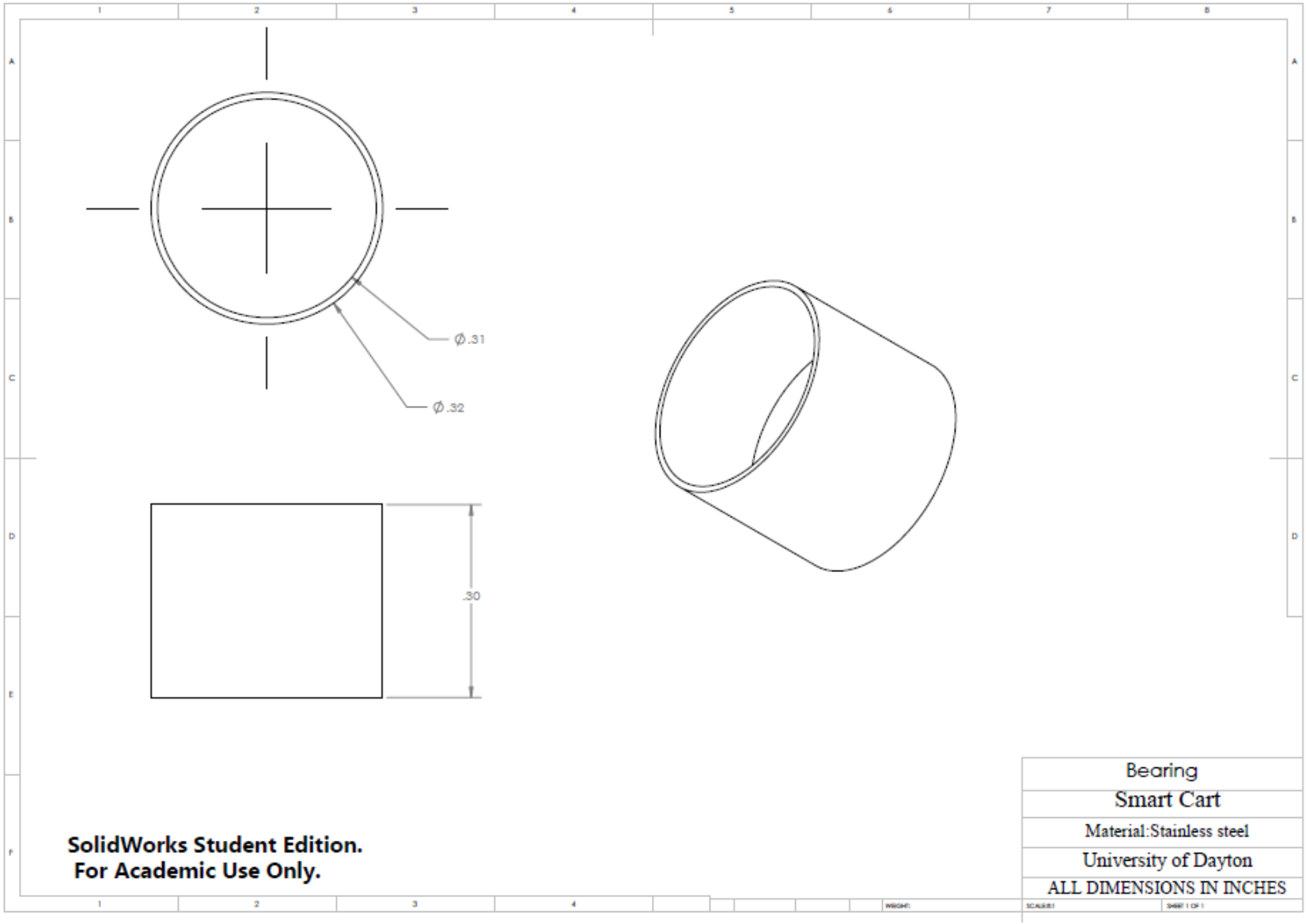
```
   Serial.println("open error");
   return;
  }
   readVals(&x,&y);
   lat1=x;
   lon1=y;
   Serial.println(lat1,6);
   Serial.println(lon1,6);
   readVals(&x,&y);
   double lat2=x;
   double lon2=y;
   Serial.println(lat2,6);
   Serial.println(lon2,6);
   readVals(&x,&y);
   lat3=x;
   lon3=y;
   Serial.println(lat3,6);
   Serial.println(lon3,6);
   readVals(&x,&y);
   double lat4=x;
   double lon4=y;
   Serial.println(lat4,6);
   Serial.println(lon4,6);
  // connect to the GPS at the desired rate
  GPS.begin(9600);

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  // uncomment this line to turn on only the "minimum recommended" data
  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
  // For logging data, we don't suggest using anything but either RMC only or RMC+GGA
  // to keep the log files at a reasonable size
  // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 100 millihertz (once every 10 seconds), 1Hz or 5Hz update rate

  // Turn off updates on antenna status, if the firmware permits it
  GPS.sendCommand(PGCMD_NOANTENNA);

}
byte servo_angle = 0;

void loop() {


   //delay(500);
```

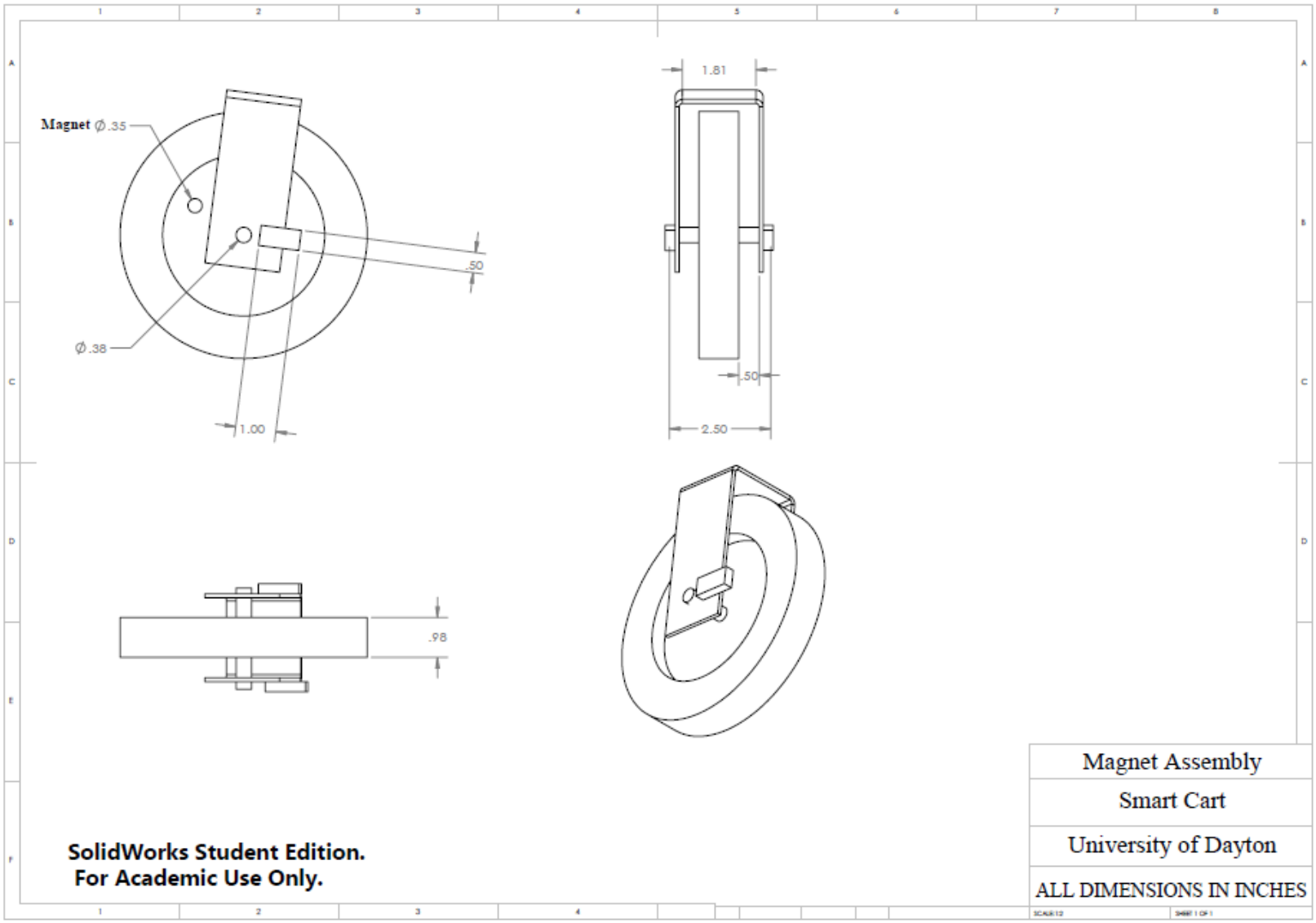
```
   // Check to see if wheel went around twice
//    while(rev_count < 2) {
//      // read the state of the hall effect sensor:
//      hallState = digitalRead(hallPin);
//
//      if (hallState == LOW) {
//        rev_count++;
//      }
//      delay(100); //Delay so that doesn't count same hallState more than once
//    }


   if (! usingInterrupt) {
   // read data from the GPS in the 'main loop'
   char c = GPS.read();
   // if you want to debug, this is a good time to do it!
   if (GPSECHO)
     if (c) Serial.print(c);
   }
     // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
   // a tricky thing here is if we print the NMEA sentence, or data
   // we end up not listening and catching other sentences!
   // so be very wary if using OUTPUT_ALLDATA and trying to print out data

   // Don't call lastNMEA more than once between parse calls!  Calling lastNMEA
   // will clear the received flag and can cause very subtle race conditions if
   // new data comes in before parse is called again.
   char *stringptr = GPS.lastNMEA();

   if (!GPS.parse(stringptr))   // this also sets the newNMEAreceived() flag to false
     return;  // we can fail to parse a sentence in which case we should just wait for another

   // Sentence parsed!
   Serial.println("OK");
   if (LOG_FIXONLY && !GPS.fix) {
     Serial.print("No Fix");
     return;
   }
   double Current_Lat = GPS.latitudeDegrees;
   double Current_Lon = GPS.longitudeDegrees;

   Serial.print("Current Latitude: ");
   Serial.println(Current_Lat, 6);
   Serial.print("Current Longitude: ");
   Serial.println(Current_Lon, 6);
   Serial.println(lat1,6);
```

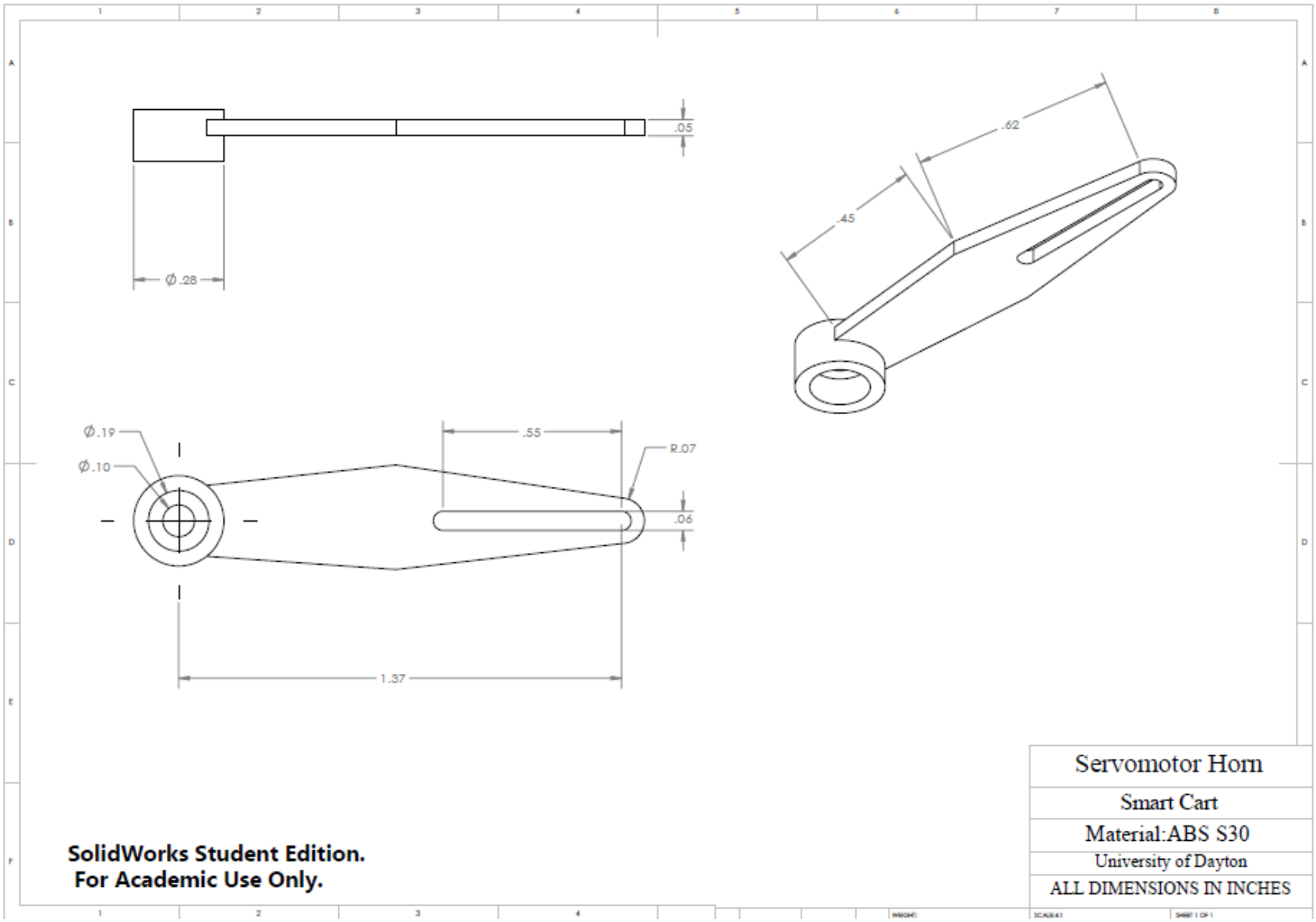
```
   Serial.println(lat3,6);
   Serial.println(lon1,6);
   Serial.println(lon3,6);

    if(lat3 < Current_Lat && Current_Lat < lat1 && lon1 < Current_Lon && Current_Lon < lon3) {
       Serial.println("INSIDE BOUNDARY!!!! GOOD!!!!");
       Serial.println("WHEEL UNLOCKED");
//       for(pos = 30; pos < 70; pos += 1)  // goes from 0 degrees to 180 degrees
//       {                                  // in steps of 1 degree
//         myservo.write(pos);              // tell servo to go to position in variable 'pos'
//         delay(15);                       // waits 15ms for the servo to reach the position
//       }
       myservo.write(30);
    }
     else {
       Serial.println("WHEEL LOCKED!!");
       Serial.println("OUTSIDE!");
//       for(pos = 70; pos>=30; pos-=1)     // goes from 180 degrees to 0 degrees
//       {
//         myservo.write(pos);              // tell servo to go to position in variable 'pos'
//         delay(15);                       // waits 15ms for the servo to reach the position
//       }
         myservo.write(70);
     }

  }

}
```

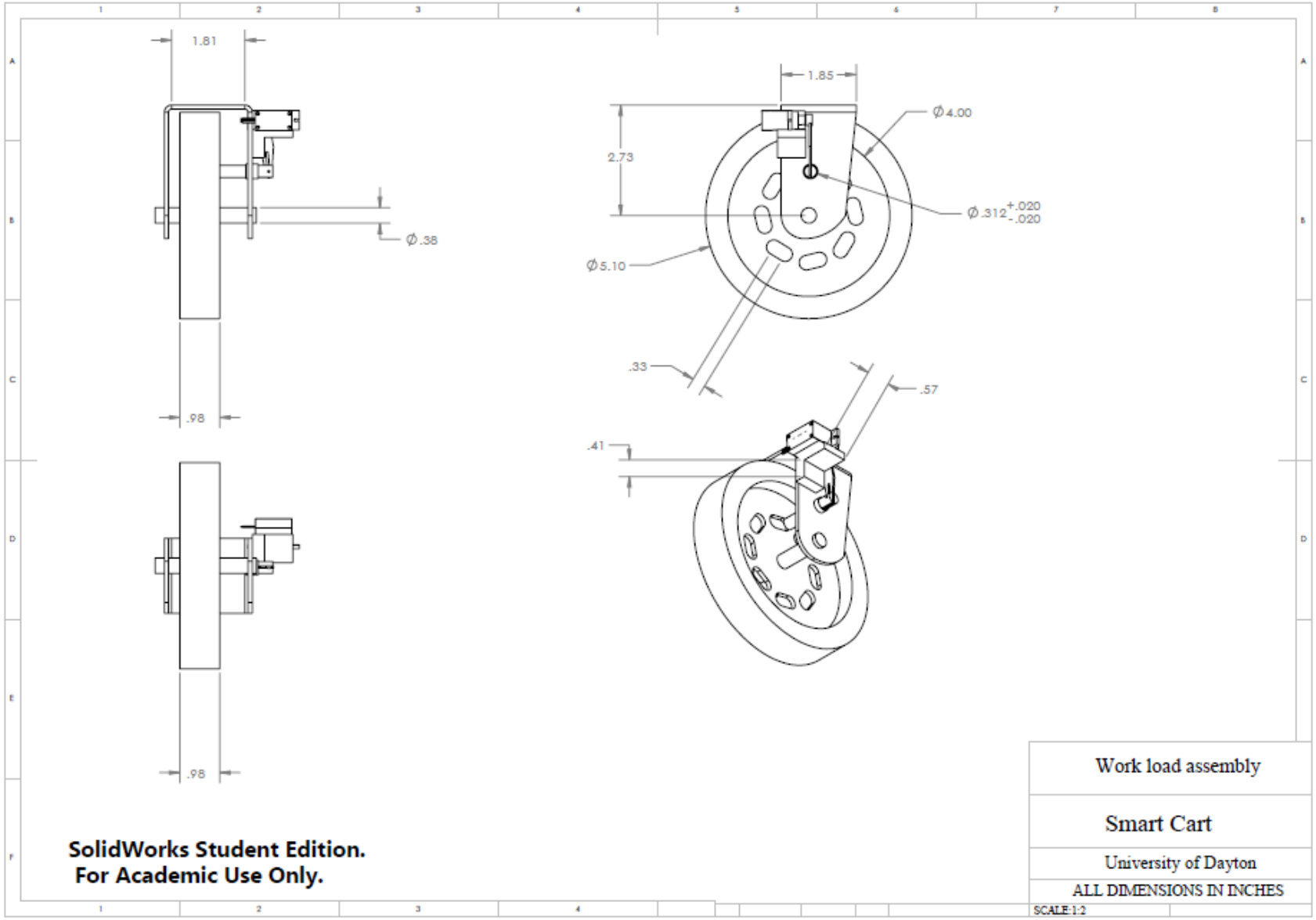
Appendix F: Mechanical Drawings




Ø.31
Ø.32
.30
SolidWorks Student Edition.
For Academic Use Only.
Bearing
Smart Cart
Material:Stainless steel
University of Dayton
ALL DIMENSIONS IN INCHES


Magnet Ø.35
Ø.38
.50
1.00
1.81
.50
2.50
.98
SolidWorks Student Edition.
For Academic Use Only.
Magnet Assembly
Smart Cart
University of Dayton
ALL DIMENSIONS IN INCHES


.05
Ø.28
.62
.45
Ø.19
Ø.10
.55
R.07
.06
1.37
SolidWorks Student Edition.
For Academic Use Only.
Servomotor Horn
Smart Cart
Material:ABS S30
University of Dayton
ALL DIMENSIONS IN INCHES


1.81
Ø.38
.98
.98
1.85
Ø4.00
2.73
Ø.312 +.020 -.020
Ø5.10
.33
.57
.41
SolidWorks Student Edition.
For Academic Use Only.
Work load assembly
Smart Cart
University of Dayton
ALL DIMENSIONS IN INCHES
SCALE:1:2