Emergency Braking System for Autonomous Golf Cart

Final Report

Written by:

Andrew Dockery

Joshua Whitman

Alex Wild

Fay Yang

Team Leader:

Joshua Whitman

Sponsored by:

Dr. Girish Chowdhary

1. Introduction

1.1. Background

The overall purpose of Oklahoma State University's Golf Cart project is to instrument and automate a fleet of autonomous Golf Cart systems for on-demand mobility on OSU campus, while providing engineering students with a hands-on application of the concepts learned in their curricula. Students learn about sensor selection and integration, autonomous navigation in cluttered environments, localization and mapping, dynamics and control of surface vehicles, as well as mechanical and electrical drive system optimization through hands-on engineering projects. The project was created in Fall 2014 for Dr. Chowdhary's Automatic Control Systems (MAE 4053/ECEN4413).

1.2. Problem Statement

As of August 2015, the golf cart had a very inadequate braking system. With the current system, it can take up to 8 seconds for the cart to come to a complete start. The final cart should have human-or-better braking. Furthermore, the golf cart does not know when to hit the brakes in an emergency, for example if a pedestrian, bike or other vehicle cuts in front of the cart.

1.3. Deliverables

At the beginning of the semester, this team promised a number of products and results to our sponsor Dr. Chowdhary. We have reproduced this list of deliverables below.

- Final Product
    - A system that is the prime controller of accelerator and brakes. The software will receive acceleration and brake requests from Stabilis (the navigation system already developed for the golf cart).
    - This system will be able to detect emergency brake conditions. An emergency brake condition is whenever braking will either avoid or mitigate the impact of a collision, under the assumption that external objects and/or people maintain their current velocity.
    - The system will be able to apply the brakes and bring the golf cart to a complete stop in human-or-better time
- CAD Models for Analysis – the team will model the mechanics of the braking system in Solidworks and perform structural analysis on it.
- Documentation of all analysis
- CAD or Construction – the team will provide instructions for building and maintaining the mechanics of the braking system

The documentation of each of our deliverables is contained in the present report.

1.4. A Glimpse at the Final Design

The final design includes both mechanical, electrical, and software systems working together in unison. In order to provide a glimpse of what the whole final product looks like before delving into details, we have provided images below of the mechanical system, the electronic sensors, and the microcontroller code.

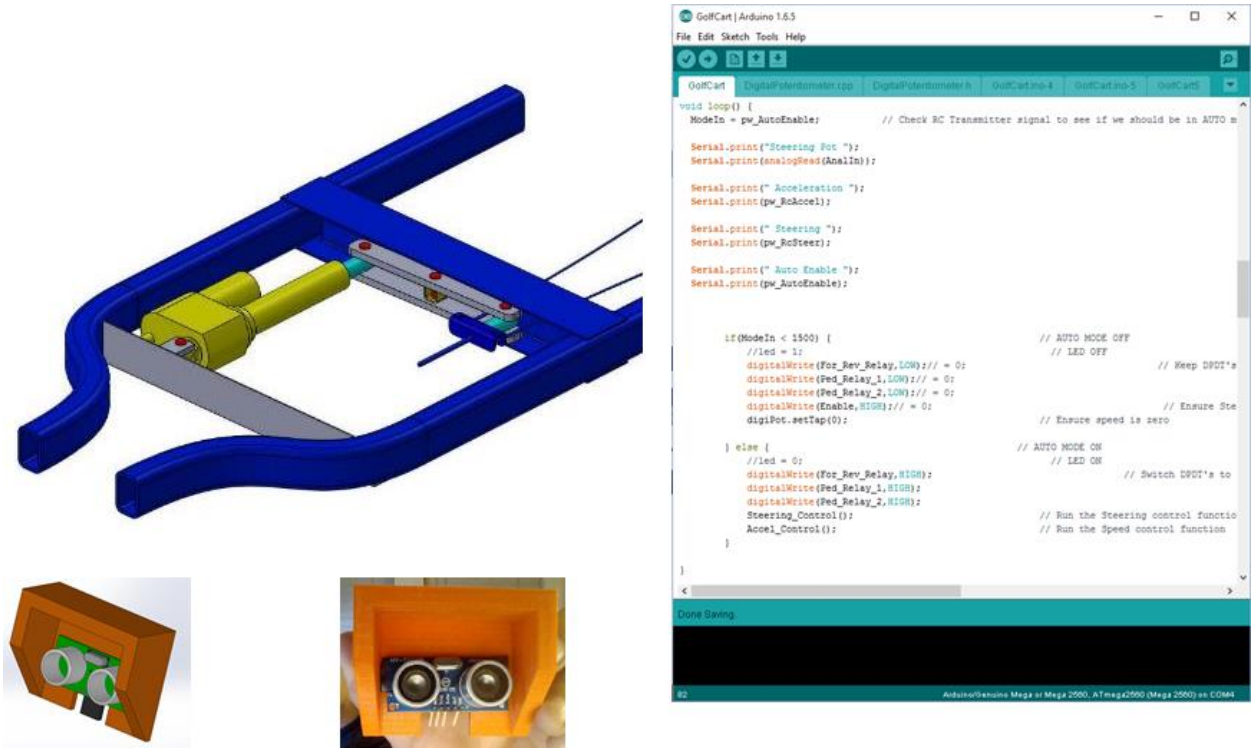

Figure 1.1.1 Final Design Reveal

2. Work Done

2.1 Mechanical System

Throughout the semester one of our primary goals was to replace the braking actuator that had been installed on the cart by a previous capstone design team. Their actuator was underpowered and could not apply the golf cart brakes as rapidly as was necessary to achieve a quick and safe stop from normal operating speed. Our first step in the replacement of the existing actuator was to set a goal for the performance of the new system as a whole, including the amount of force that needed to be applied to the brake mechanism, the deflection of the brake system under that force, and the rate at which the load needed to be applied.

Force was measured using a load cell from an ordinary bathroom scale that was amplified and subsequently read with an Arduino Uno. For our tests this load cell was taped to the sole of one team member's (member A) shoe as he pushed the brake pedal while another team member (member B) read the deflection of the pedal from a ruler stabilized on a common datum point. As member A deflected the pedal slowly downward member B announced the crossing of each 0.25" increment and deflection and load were recorded to create the graph shown below. (Figure 2.1.1) After we were satisfied that our model represented the true relation between force and pedal displacement we carefully measured the lever arm lengths of the brake pedal and determined the pedal ratio. This pedal ratio allowed us to convert our model to represent the deflection and force in the brake cable rather than at the pedal, this new model is shown below (Figure 2.1.2). As we were interested in applying a force to the cable this model was of particular interest to us, specifically the roughly 900-1000lb maximum load and the 0.5" maximum deflection allowed us to determine our actuator requirements.

We examined the possibility of using pneumatic, hydraulic, or electromagnetic actuators as they could be made to apply extremely quickly, however practical reasons, including space, power required, and cost/availability, led us to the decision to use a standard screw-type linear electric actuator. The fastest readily available actuator we could find that could generate 1000lbs of force was a ServoCity SDA4-67 and after tentatively choosing that actuator we began design of our mounting brackets and associated hardware. Figures 2.1.3 and 2.1.4 below show Solidworks models of our two major revisions. The first version (Figure 2.1.3) was a self-contained steel frame structure that hung on the main aluminum cross beam of the golf cart, requiring no modifications to the original golf cart frame and distributing the 1000lb load over a large area so as to minimize stress within the beam. Because the system hung below the cart it significantly reduced ground clearance, ultimately it was decided that this could cause problems if the cart was driven off of curbs, etc. The system could easily be damaged if ground contact was made and thus the system was redesigned to protect the actuator from any damage. Figure 2.1.4 shows the revised system, with the actuator protected by the existing frame rails. This mounting method requires more modification to the frame of the cart, as well as higher loads within the main aluminum beam. Accounting for the higher loads we determined that this mounting was still safe and would not damage the golf cart frame through hand calculations (modeling the beam as simply supported with a distributed load near the center) and with Solidworks Finite Element Analysis (modeling the applied load over the area of the pivot point base). The results are shown in Figure 2.1.5 below.

Finding that this actuator would indeed fit and work without any problems, we ordered it, fabricated the necessary mounting components, and installed the actuator as detailed in the Solidworks model.

After finishing the installation of the actuator we tested it under RC control, connecting the Pololu JRK motor controller to an RC receiver and setting our actuator endpoints as necessary. We slowly applied the brakes using a remote control and observed, looking for any unexpected deflection that could indicate that our FEA model was inaccurate. At full load/full application of the brakes the central aluminum beam had deflected as we expected (approximately 2mm, as predicted by our models) and we determined the static load application test a success. we then applied the brakes (from 0-100%) in roughly a step input to observe the application. It behaved as expected, with the system behavior resembling a critically damped system with settling time of approximately 1 second. We deemed this dynamic test a success as well.

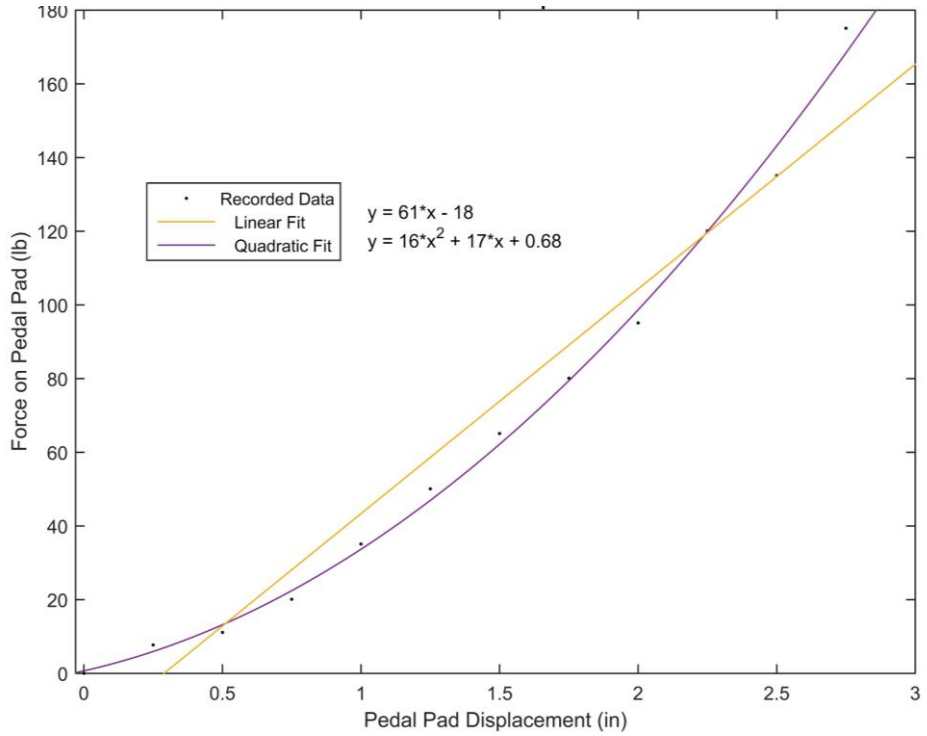

Figure 2.1.1 Brake Force vs. Displacement

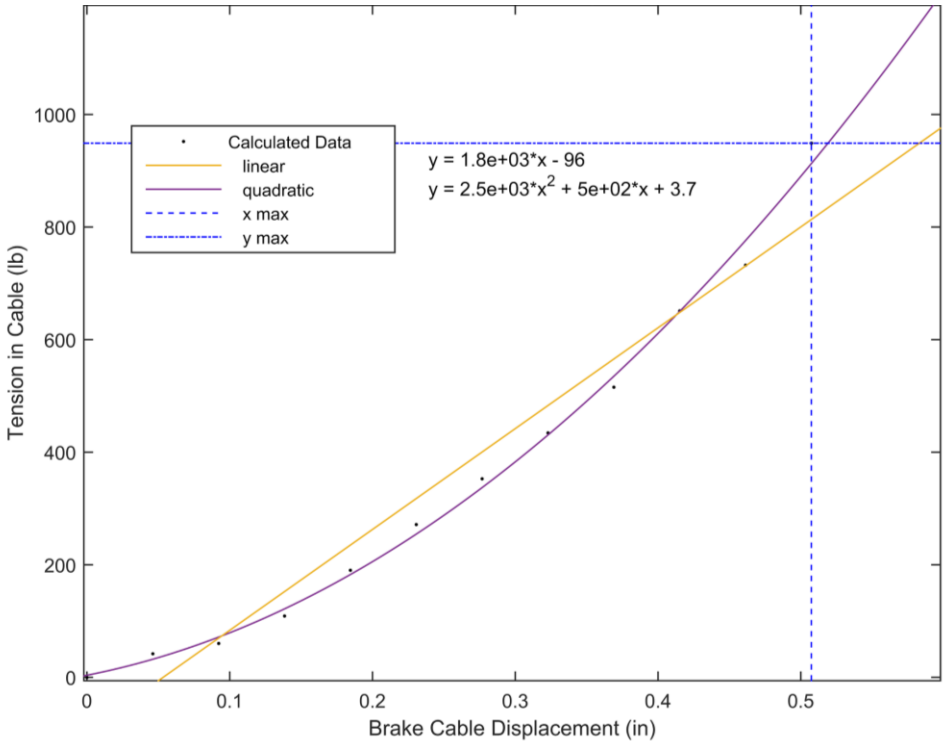

Figure 2.1.2 Brake Cable Force vs. Displacement

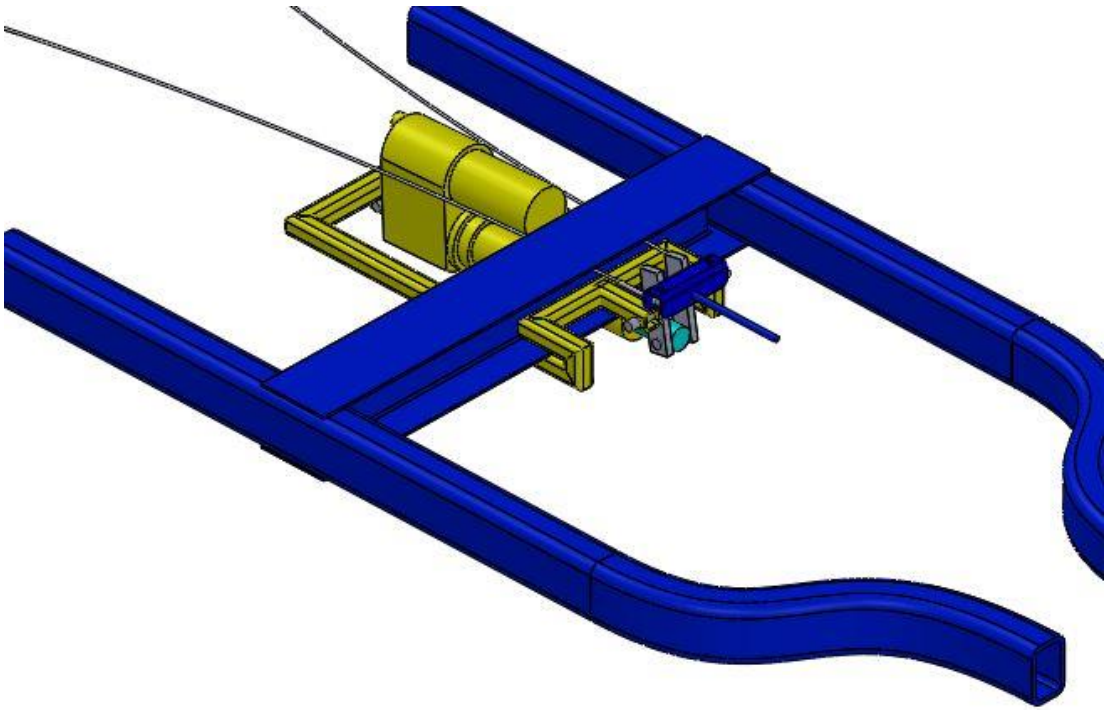

Figure 2.1.3 Actuator Mounting, Initial Design

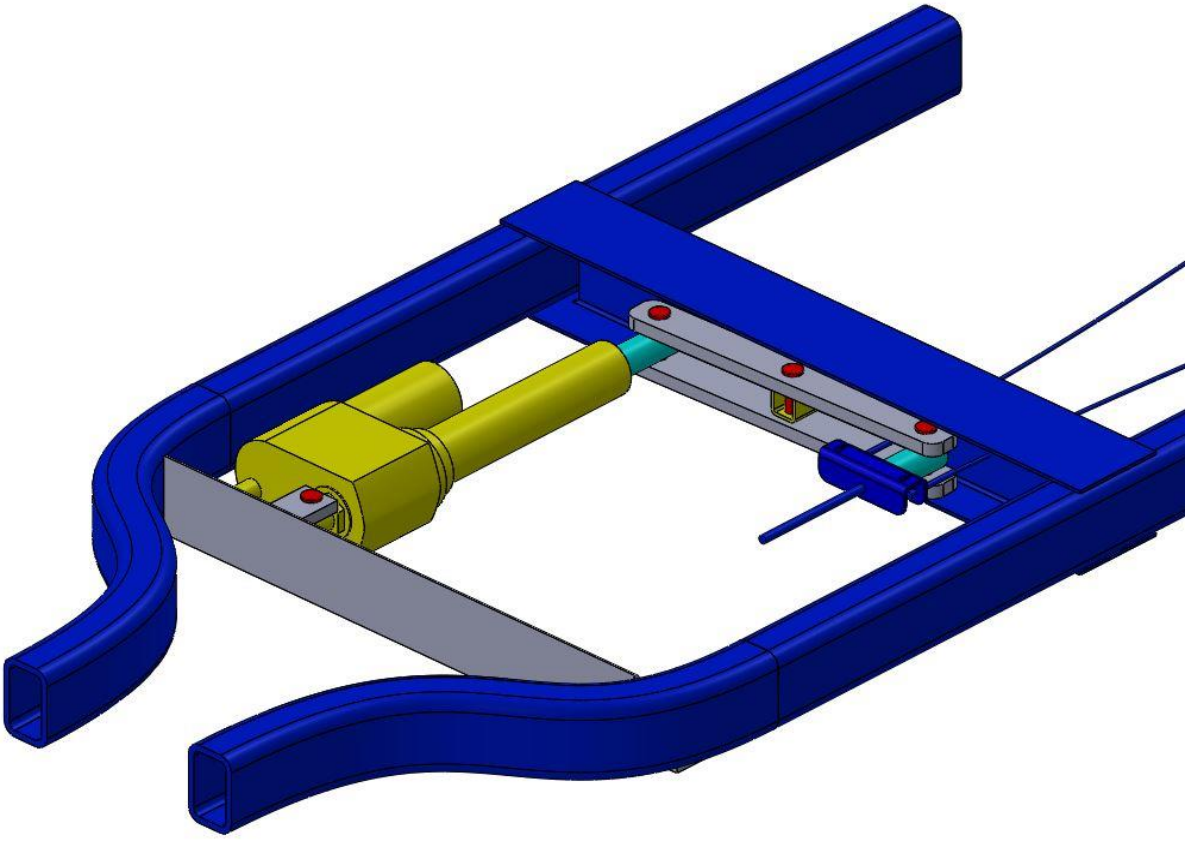

Figure 2.1.4 Actuator Mounting, Final Design

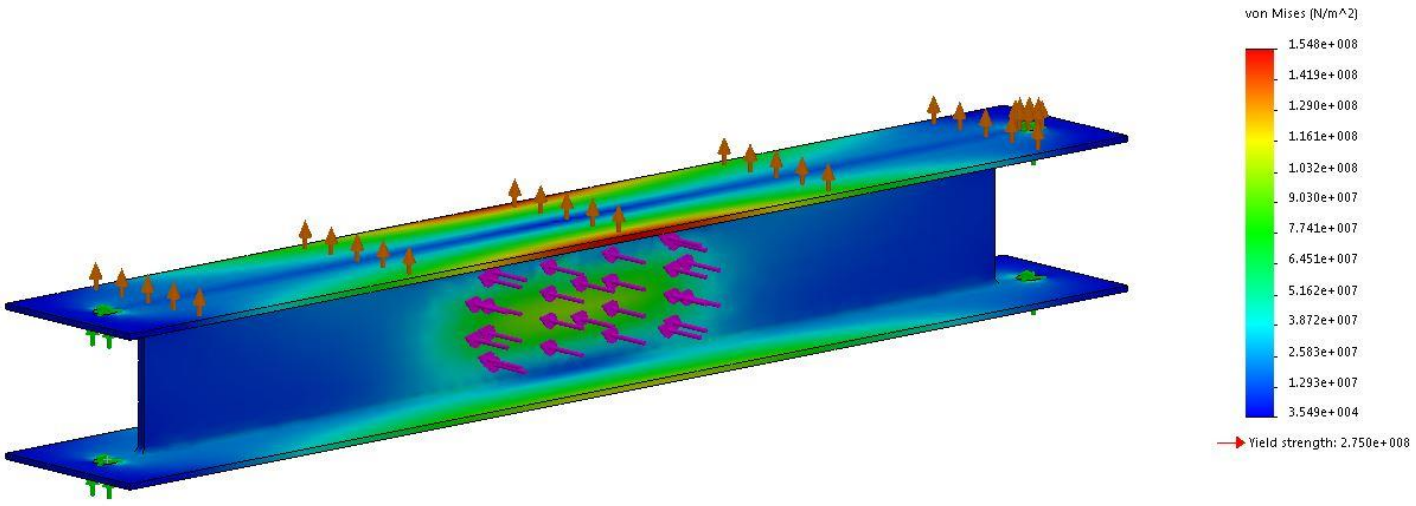

Figure 2.1.5 Solidworks Finite Element Analysis

2.2. Emergency Braking Condition Detection

Detecting an Emergency Braking Condition (EBC) is a critical required safety feature for any autonomous vehicle. With the DAS Cart transitioning from radio control to autonomous control the ability to brake safely is critical. Three scanning systems were selected over the course of the semester: Stationary Lidars, scanning lidar, and stationary ultrasonic.

The initial system selected, Stationary Lidars, were 4 cone lidars mounted on each corner of the golf cart with the designed capability to have both reverse and forward autonomous braking. Code was developed for this system in collaboration with the Autonomous Controls course. In meetings with Dr. Chowdhary the team came to the conclusion that the navigational sensors and autonomous braking sensors needed to be separate. This would allow the new braking sensors to operate independent from any navigational errors and become an additional line of defense.

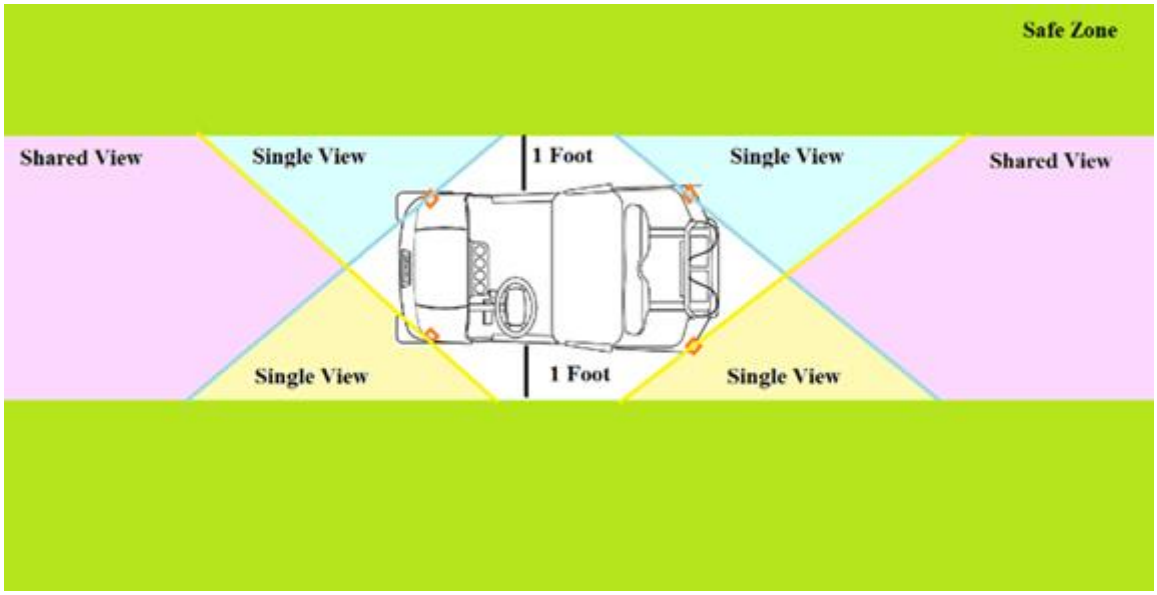

Figure 2.2.1 Corner Lidar Configuration

Programming for the this setup was based on overlapping views being used to rule out noise in the system and not create unnecessary braking.

The next system was a scanning Lidar system with a single linear Lidar mounted on a rotating servo. The code was created for an Arduino and when the team discussed the new system with Dr. Chowdhary he wanted a simpler system. His goal was to have a system that would generate few false positives. We discussed our concern with other sensors not being able to detect at ranges suitable for an appropriate stopping distance. The team chose to settle for emergency braking at lower, more realistic speeds, to prevent the need of additional scanning Lidars.

The final system designed and manufactured is multiple ultrasonic sensors mounted on the front of the vehicle to detect last minute emergency situations. The team chose to do a system independent of Stabilus navigation to minimize error. The final design uses ultrasonic sensors along with a 3D printed shield to reduce signal noise. These will be wired through the cart to the same system as the actuators, leaving no middle men between emergency brakes and the sensors involved.

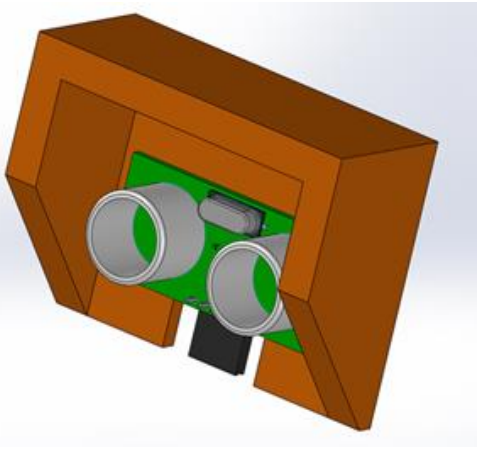

Figure 2.2.2 Ultrasonic Sensor Mounting & Shield

The shield is designed to prevent signal noise and protect the sensors from damage.

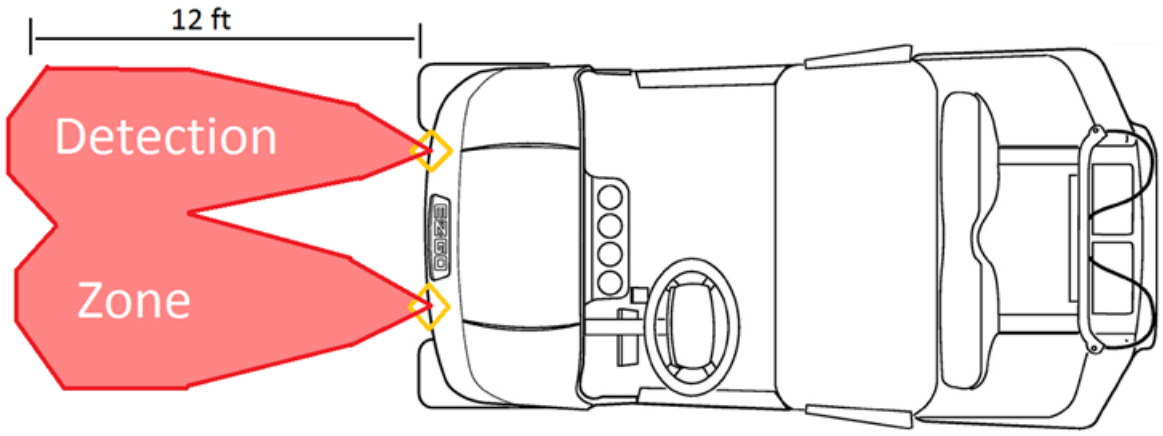

Figure 2.2.3 Detection Zone of Ultrasonic Sensors

In the future this system can be improved by adding more ultrasonic sensors to reduce chance of false positives. Additional ideas for future improvement include finding ways to scan lower to the ground without triggering braking conditions on curbs or small shrubs.

Figure 2.2.4 Final Mounted Design

2.3. Electronics System

The first step in designing the electronics side of the emergency braking system was to define how the various electronic systems would connect and interact. There many different electronic systems on the autonomous golf cart, composing a complex interacting network. An overview of these systems is presented in Figure 2.3.1.

We decided that the Emergency Braking System (EBS) should function like the “nervous system” of the vehicle. The “brain” would be Stabilis. Stabilis (the “brain”) normally controls the actuators (“muscles”) through the EBS (“nervous system”); however, under emergency conditions, the EBS can act on its own (“reflexes”) to avoid disaster.

Stabilis is a navigational computer which runs on a Beagle Bone board. Stabilis makes actuator requests to the EBS using standard servo interface. This also allows us to connect an RC receiver in place of Stabilis, temporarily making the golf cart’s brain an actual human being. In order to perform an emergency braking procedure, the EBS needed have direct control over the acceleration and braking. We also decided to let the EBS have control of the steering, since Stabilis is currently not competent to do so.The EBS should return sensor data regarding the position of the steering wheel, the position of the brakes, etc. to Stabilis.

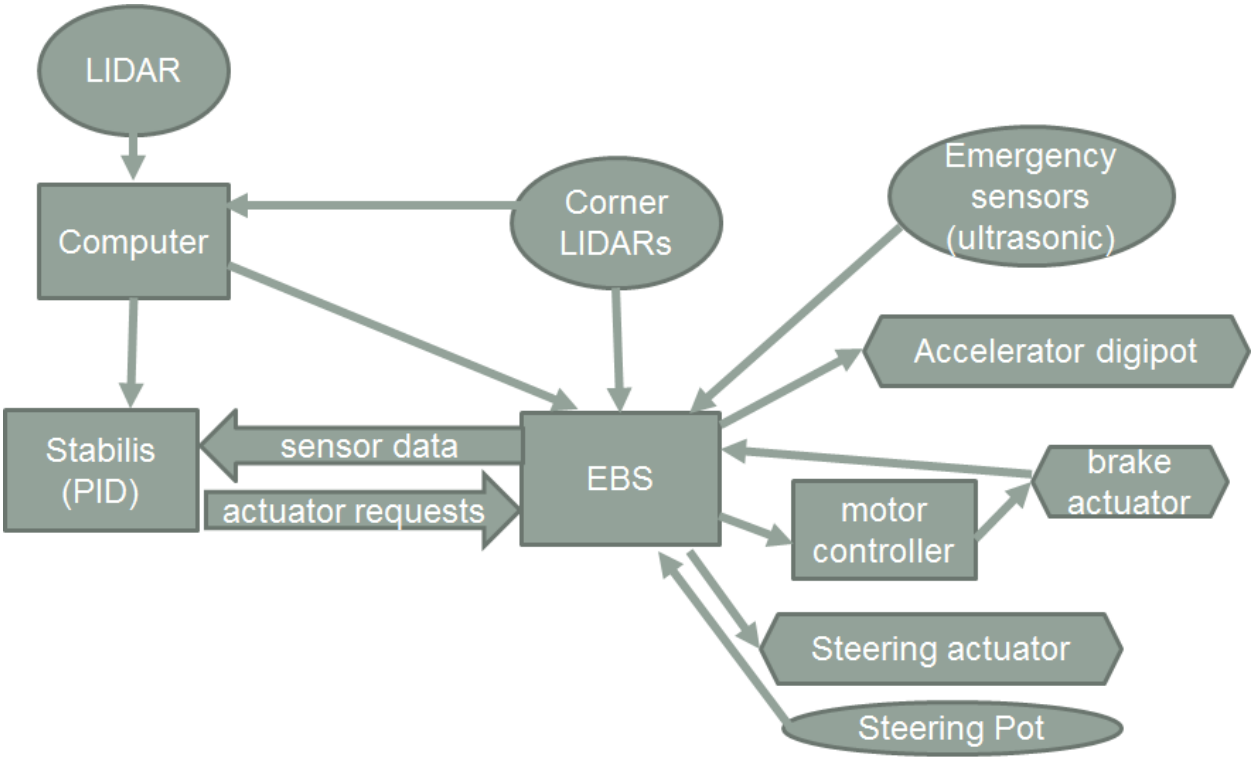

Figure 2.3.1 Electronics System Overview

The EBS connects to a variety of different actuators and sensors. The EBS receives data from a potentiometer indicating the position of the steering wheel. The system controls the steering actuator via a PWM output. The system controls the brake actuator by sending servo signals to Pololu JRK motor controller. The brake actuator should return position data directly to the EBS. The EBS controls the accelerator via a digital potentiometer and a set of relays. The relays control whether the power is enabled and the direction the wheels turn. The EBS communicates via SPI interface with the digital potentiometer.

There are a number of different ways for the EBS to detect when an Emergency Braking Condition (EBC) exists. As described in Section 2.2, we began by considering using LIDAR sensors. The overhead LIDAR is very complex and needs to be interpreted by a computer in order to generate useful data for either Stabilis or the EBS. Because the EBS is a safety-critical system, we decided that there should be as few ways for it to fail as possible. This means that the EBS should have as direct access as possible to the emergency braking sensors and to the braking actuators. Middle-men are to be avoided. Therefore, we began to consider relying on the Corner LIDARS. These are much more simple and could be interpreted directly by the EBS. However, as detailed in Section 2.2, we found that these were not suitable. We ended up using ultrasonic sensors. These are extremely reliable and independent of Stabilis.

Figure 2.3.2 Stabilis

The next major design decision for the electronics system was to decide what platform on which to build the EBS. We had a number of different options (See also Figure 2.3.3:

1. Freedom Board. Graduate student Hunter Young was already using a microcontroller called a “Freedom Board” as a middle-man between Stabilis and the acceleration and steering actuators. This platform would be a good candidate for the EBS since the EBS has to perform many of the functions already being performed by Freedom Board. However, the Freedom Board is also very obscure. Few people besides Mr. Young know how to program with it.
2. Arduino. Arduino is a ubiquitous hobbyist-level microcontroller. There are extensive libraries, documentation, and help available for this platform. It is capable of driving SPI, ICSP, analog, servo, and many other kinds of interfaces. This platform is the primary platform used in MAE 4733 Mechatronics, which several of our team members have taken.
3. Beagle Bone. Beagle Bone is a full Linux computer mounted on a single board. It is the platform Stabilis runs on. Thus, an advantage of using Beagle Bone is that we would have a uniform computer platform for the whole autonomous golf cart. A large disadvantage of Beagle Bone is that it does not have native support for PWM output.

After long consideration of our options, we chose to use an Arduino Mega 2560. The Freedom Board was a bad choice from the beginning, and needed to be phased out. Dr. Chowdhary advised us that the Beagle Bone was overkill for what we wanted to accomplish and that there was no reason to use it. Therefore we went with the Arduino which we all know and love. We chose the Arduino Mega 2560 rather than the Arduino Uno because the latter did not have enough PWM outputs and interrupts to meet our requirements.

Figure 2.3.3 From left to right: Freedom Board, Arduino Mega 2560, Beagle Bone

The next major task to accomplish was to port the code used in the Freedom Board over to Arduino. This turned out to be a much more tedious and difficult process than might be expected. The printed circuit board containing the relays and digital potentiometer is, in the words of an electrical engineering student we had look at it, “a mess”. It took multiple weeks of troubleshooting before we were able to achieve full RC control of the steering and acceleration.

One of the major improvements we made to the control of the different actuators on the cart was the implementation of Proportional-Integral-Derivative (PID) control. Hunter’s code used bang-bang control. We implemented PID using a standard Arduino library. This had several important improvements. We significantly reduced the chatter and noise in the steering system. We were able to accelerate and brake smoothly.

In its current state, the EBS detects an emergency braking condition whenever an object appears within a certain trigger range of the utlrasonic sensors (we defined this to be 40 inches). In stationary tests, the cart performs well, stopping within seconds of an object entering its danger zone. More extensive field tests should be performed in the future.

3. Final Design

3.1. Detailed Description

The emergency braking team has put together a three part system consisting of brake actuation, emergency detection, and new platform implementation. Overall the deliverables were met creating a safer autonomous vehicle.

The brake actuation received a drastic overhaul from the old system which would pull on the brake pedal to a new system which directly applies tension to the brake cable. This system consists of a linear actuator mounted under the passenger side of the golf cart. A lever is then used to transfer the energy from the actuator across a horizontal lever below the floor board, directly applying force to the brake cable. The new actuator operates ten times faster than the previous actuator, leading to a dramatically safer vehicle

Secondly, a new emergency detection system was designed and implemented. Two ultrasonic sensors, mounted on the front of the cart, are pinging 10 times a second to detect obstacles in the direct path of the cart. If an object is detected, the brakes apply immediately.

By creating sensors that operate independent of the navigation system, a reliable safety fallback was created for the golf cart.

The third project the team oversaw was the implementation of a new computer platform for the different actuators on the golf cart: steering, acceleration, and braking. Switching from Beaglebone to Arduino allows the actuators of the cart to operate as servos to be used by the stabilus navigation system. The new platform has PID control implemented for more efficient control as well as a more accessible programming library for future students.

3.2. Evaluation

The following deliverables were assessed in their degree of completion

1. A system that is the prime controller of accelerator and brakes. The software will receive acceleration and brake requests from Stabilis (the navigation system already developed for the golf cart).
    a. The system includes fully ported code from Beaglebone to Arduino. Requests are received manually or via radio control as the Actuators are operating as Servos to be used by future design teams. This deliverable was met to Dr. Chowdhary and the Autonomous control team's design.
2. This system will be able to detect emergency brake conditions. An emergency brake condition is whenever braking will either avoid or mitigate the impact of a collision, under the assumption that external objects and/or people maintain their current velocity.
    a. The system is completely manufactured but requires some future testing in order to perfect execution.
3. The system will be able to apply the brakes and bring the golf cart to a complete stop in human-or-better time
    a. The designed and manufactured brakes which operate as intended in human or better time. Time from detection to full actuation is under 2 seconds, which was discussed in our opening project discussion.

3.3. Recommendations for Future Work

1. Brake Actuation
    a. The upgrade to an 1,000 pound actuator is putting a significant, yet below yield, load on a beam beneath the golf cart. Routine inspection on the aluminum beam for any signs of fatigue will help the cart remain safe.
    b. Tweaking the lever point can modify the braking to be more forceful or quicker based on demonstrated need over the next few months.
2. Autonomous Braking
    a. using more ultrasonic sensors can broaden the range and reduce the chance of noise in the signal
    b. The sensors could be better protected to deal with outdoor conditions
3. Arduino Platform
    a. We could have a more sophisticated means of detecting emergency braking conditions
    b. Physically, the wiring of the electronic systems is just passable for performing tests. Connections should be better secured and organized in the future.

3.4. Budget Summary

Table 3.4.1 Budget Summary

Component	*Requirements*	*Details*	*Quantity*	*Cost*
Actuator	Heavy duty around 1000lb linear actuator	ServoCity, SDA12-67	1	$399.99
Motor Controller	Handling 20 amps at 12V dc	Jrk 12v12	1	$99.99
Microcontroller	See Section 2.3	Arduino Mega 2560	1	$37.76
Parts fabrication	-	Physics and chemistry instrument shop	-	$46.29
SUM				$583.97

The budget for this project was dominated by the price of the linear actuator. This cost about 70% of our total budget, coming in at $400 dollars for the 1010-lb linear actuator. In sum, we spend $583.97 for our Golf Cart project.

4. Appendices

4.1. Arduino Code

4.1.1 “GolfCart.ino”

```
/**    Created by: Joshua Whitman
      Date Created: Fall 2015
      Platform Used: Arduino MEGA
      Description: Program designed to be used to control the speed, and
steering for an autonomous golf cart based on code originally developed by
Hunter Young on FreedomBoard*/

#include <Arduino.h>
#include <SPI.h>
#include <Servo.h>
#include <PID_v1>
#include <TimerOne.h>
#include "DigitalPotentiometer.h" // dR = .072558 kOhm

DigitalPotentiometer digiPot(5); // 5=PTC12. Digital Potentiometer used to
control Acceleration (MCP4131-103)

//=============================================================================
=
//                                          CONTROL INPUTS
//=============================================================================
=
byte Auto_Enable = 19; //PwmIn Auto_Enable(PTA13);           // PWM Signal
for Switching from Automatic control to Manual control  (Channel 6)
byte RcAccel = 20; //PwmIn RcAccel(PTD5);                   // PWM Signal
for Acceleration Control (Channel 3)
byte RcSteer = 21; //PwmIn RcSteer(PTD0);                   // PWM Signal
for Steering Control (Channel 1)
byte AnalIn = A0; //AnalogIn AnalIn(PTB0);                 // Analog signal
input from the linear potentiometer for steering control

// ULTRASONIC SENSORS
byte SONAR_ECHO_PIN1        =30;
byte SONAR_TRIGGER_PIN1     =32;
byte SONAR_ECHO_PIN2        =34;
byte SONAR_TRIGGER_PIN2     =36;
volatile int sensorcycle;
volatile double leftdist;
volatile double rightdist;
int trigger_value = 40; //anything within 30 inches of vehicle should trigger
brakes

//=============================================================================
=
//                                              OUTPUTS
//=============================================================================
=
// STEERING
```

```
byte Enable = 2; //DigitalOut Enable(PTD4);                 // Steering I/O
Signal for Enable (Move Increment Mode: Pulse Enable line low to initiate
movement)

byte A_Out = 3; /*DigitalOut A_Out(PTA12);/** MODE:  PIN FUNCTION
*   Velocity Control(Bi-PWM):Inhibit(Optional)
*   Move Incremental Distance:Increment Select */
byte B_Out = 4; /*PwmOut B_Out(PTA4);   /** MODE:  PIN FUNCTION
*   Velocity Control(Bi-PWM):PWM Pulsewidth Signal
*   Move Incremental Distance:Home Switch (Optional) */

//  RELAY CONTROL (ACCELERATION)
byte pedalSwitch = 8; //DigitalOut pedalSwitch(PTC7);           // On/Off
Signal sent to Autonomy-Mode Pedal Switch Control SPDT relay
byte reverse = 9; //DigitalOut reverse(PTC0);               // On/Off Signal
sent to Autonomy-Mode Reverse Control SPDT relay
byte forward = 10; //DigitalOut forward(PTC3);               // On/Off Signal
sent to Autonomy-Mode Forward Control SPDT relay
byte For_Rev_Relay = 11; //DigitalOut For_Rev_Relay(PTC4);         // On/Off
Signal used to switch between Autonomy-Mode and Manual-Mode through the DPDT
relay
byte Ped_Relay_1 = 12; //DigitalOut Ped_Relay_1(PTC5);           // On/Off
Signal used to switch between Autonomy-Mode and Manual-Mode through the DPDT
relay
byte Ped_Relay_2 = 13; //DigitalOut Ped_Relay_2(PTC6);           // On/Off
Signal used to switch between Autonomy-Mode and Manual-Mode through the DPDT
relay

//BRAKE ACTUATOR
Servo brake; //6


//=============================================================================
=
//                                          CODE SECTION
//=============================================================================
=

// Initialize Global Variables
float AccelIn;                          // Variable to store the
Accelerometer RC signal
float SteerIn;                          // Variable to store the Steering RC
signal
float ModeIn;                           // Variable to store the Auto/Manual
Mode RC signal
float PotIn;                            // Variable to store the Analog value
from the potentiometer

volatile int prev_t_RcAccel = micros();
volatile int pw_RcAccel = 0.0;
volatile int prev_t_RcSteer = micros();
volatile int pw_RcSteer = 0.0;
volatile int prev_t_AutoEnable = micros();
volatile int pw_AutoEnable = 0.0;

void setup() {
  Serial.begin(9600);
```

```
  attachInterrupt(digitalPinToInterrupt(RcAccel), RcAccelRise, RISING);
  attachInterrupt(digitalPinToInterrupt(RcSteer), RcSteerRise, RISING);
  attachInterrupt(digitalPinToInterrupt(Auto_Enable), AutoEnableRise,
RISING);

  pinMode(Auto_Enable,INPUT);
  pinMode(RcAccel,INPUT);
  pinMode(RcSteer,INPUT);

  pinMode(Enable,OUTPUT);
  pinMode(A_Out,OUTPUT);
  pinMode(B_Out,OUTPUT);
  pinMode(For_Rev_Relay,OUTPUT);
  pinMode(Ped_Relay_1,OUTPUT);
  pinMode(Ped_Relay_2,OUTPUT);
  pinMode(forward,OUTPUT);
  pinMode(reverse,OUTPUT);
  pinMode(pedalSwitch,OUTPUT);

  digitalWrite(forward,LOW);//forward = 1;
  digitalWrite(pedalSwitch,HIGH);//
  digitalWrite(reverse,HIGH);//                                  // Change the
relays to go Forward
  digiPot.setTap(0);                                             // Set the Golf
cart speed (Start off at 0 ohms; No speed for safety)

  pinMode(SONAR_TRIGGER_PIN1, OUTPUT);
  pinMode(SONAR_ECHO_PIN1, INPUT);
  pinMode(SONAR_TRIGGER_PIN2, OUTPUT);
  pinMode(SONAR_ECHO_PIN2, INPUT);
  sensorcycle = 1;
  Timer1.initialize(500000);         // initialize timer1, and set a 500 ms
period
  Timer1.attachInterrupt(pingsensor);  // attaches callback() as a timer
overflow interrupt

  brake.attach(6);

  delay(1000);                                                   // Used for
Steering control to make sure all hardware is fully initialized (Prevents
software issues with the variable calculations)
}

void loop() {
  ModeIn = pw_AutoEnable;           // Check RC Transmitter signal to see if
we should be in AUTO mode or MANUAL mode

  Serial.print("Steering Pot ");
  Serial.print(analogRead(AnalIn));

  Serial.print(" Acceleration ");
  Serial.print(pw_RcAccel);

  Serial.print(" Steering ");
  Serial.print(pw_RcSteer);
```

```
  Serial.print(" Auto Enable ");
  Serial.print(pw_AutoEnable);

  Serial.print(" Ultrasonic ");
  Serial.print(leftdist);
  Serial.print(":");
  Serial.print(rightdist);


  if (leftdist < trigger_value || rightdist < trigger_value) {
    Serial.println("EMERGENCY");
    digiPot.setTap(0);
    brake.write(20);
    delay(1000);
  } else {
    brake.write(160);

    if(ModeIn < 1500) {                                    // AUTO MODE OFF
      //led = 1;                                           // LED OFF
      digitalWrite(For_Rev_Relay,LOW);// = 0;
// Keep DPDT's in MANUAL MODE
      digitalWrite(Ped_Relay_1,LOW);// = 0;
      digitalWrite(Ped_Relay_2,LOW);// = 0;
      digitalWrite(Enable,HIGH);// = 0;
// Ensure Steering servo can't be controlled
      digiPot.setTap(0);                                   // Ensure speed is
zero

    } else {                                              // AUTO MODE ON
      //led = 0;                                           // LED ON
      digitalWrite(For_Rev_Relay,HIGH);                                //
Switch DPDT's to AUTO MODE
      digitalWrite(Ped_Relay_1,HIGH);
      digitalWrite(Ped_Relay_2,HIGH);
      Steering_Control();                                  // Run the Steering
control function
      Accel_Control();                                     // Run the Speed
control function
    }
  }

}

//=========================================================
//                      INTERRUPTS
//=========================================================
void RcAccelRise() {
  attachInterrupt(digitalPinToInterrupt(RcAccel), RcAccelFall, FALLING);
  prev_t_RcAccel = micros();
}
void RcAccelFall() {
  attachInterrupt(digitalPinToInterrupt(RcAccel), RcAccelRise, RISING);
  pw_RcAccel = micros()-prev_t_RcAccel;
  //Serial.println(pwm_value);
}

void RcSteerRise() {
```

```
  attachInterrupt(digitalPinToInterrupt(RcSteer), RcSteerFall, FALLING);
  prev_t_RcSteer = micros();
}
void RcSteerFall() {
  attachInterrupt(digitalPinToInterrupt(RcSteer), RcSteerRise, RISING);
  pw_RcSteer = micros()-prev_t_RcSteer;
  //Serial.println(pwm_value);
}

void AutoEnableRise() {
  attachInterrupt(digitalPinToInterrupt(Auto_Enable), AutoEnableFall,
FALLING);
  prev_t_AutoEnable = micros();
}
void AutoEnableFall() {
  attachInterrupt(digitalPinToInterrupt(Auto_Enable), AutoEnableRise,
RISING);
  pw_AutoEnable = micros()-prev_t_AutoEnable;
  //Serial.println(pwm_value);
}

//=========================================================
//                       ACCELERATION
//=========================================================
void Accel_Control()
{

    AccelIn = pw_RcAccel;// * 1000000;      // Receive RC PWM from the
acceleration channel

    int RCtoDigi = digiPot.RcToDigiPot(AccelIn);   // Convert RC to Digi Pot
acceptable value

    if(RCtoDigi > 9) {                             // Positive values mean go
Forward
        digitalWrite(forward,HIGH);//forward = 1;
        digitalWrite(pedalSwitch,HIGH);//
        digitalWrite(reverse,LOW);//               // Change the relays to go
Forward
        digiPot.setTap(RCtoDigi);                  // Send the command to the
Digi Pot
    } else if(RCtoDigi < -9) {                     // Negative values mean
Reverse
        digitalWrite(forward,LOW);
        digitalWrite(pedalSwitch,HIGH);
        digitalWrite(reverse,HIGH);                 // Change the relays to
go Reverse
        digiPot.setTap(-1 * RCtoDigi);              // Send the command to
the Digi Pot
    } else {                                        // Neutral if in PWM
signal is in the deadzone
        digitalWrite(forward,LOW);
        digitalWrite(pedalSwitch,LOW);
        digitalWrite(reverse,LOW);
    }                                               // Turn off all relays
}
```

```
//=========================================================
//                       STEERING
//=========================================================

// PWM Parameters for steering control
#define STEER_MAX   1915.0
#define STEER_MID   1517.0
#define STEER_MIN   1128.0

#define MAX_POSITION 90.0
#define MID_POSITION 43.0
#define MIN_POSITION 5.0

// Initialize Global Variables for Steering
int current_position = 0;
int goal = 0;
int position_error = 0;
float out = 0;
#define NAVG 17

PID myPID(&current_position,&duty,&goal,2,5,1,DIRECT);

// Function used to correlate RC pulsewidth to a corrected Duty Cycle
(Equations were derived in Excel using keys positions and PWM signals to
model trendlines)
float RC_calc(float current_pulse)
{
    float ans = 0;

    if(current_pulse >= 1858.0) //
        ans = 0.1754 * current_pulse - 245.96;
    else if(current_pulse <= 1210.0)
        ans = 0.0556 * current_pulse - 57.222;
    else
        ans = 0.108 * current_pulse - 120.84;
    return ans;
}

void Steering_Control()
{
    // Take NAVG readings of pot and average them
    current_position = 0;
    int i;
    for (i=0; i<NAVG; i++)
      current_position += map(analogRead(AnalIn),585,111,0,100);// Read in
the Potentiometer's value
    current_position = current_position/((float)NAVG);

    digitalWrite(Enable,HIGH);      // Enable servo to allow for proper motor
control
    digitalWrite(A_Out,LOW);        // Disable Inhibit (Allows for servo
control)

    SteerIn = pw_RcSteer;                                          //
Receive RC signal pulsewidth from the steering channel
    float goal_calc = RC_calc(SteerIn);                            //
```

```
Correct the Duty Cycle to be used to control the velocity
    goal = constrain(map(goal_calc,6,90,0,100),0,100);              //
Calculate the Goal position based off of encoder parameters
    position_error = goal - current_position;                       //
Calculate the error
    Serial.print(" curr_pos "); Serial.print(current_position);
    Serial.print(" goal_steer "); Serial.println(goal);

    // If we are within a safe operating range then command the velocity
    if(current_position > 100){
        digitalWrite(A_Out,LOW);// = 0;
        analogWrite(B_Out,0.99*255);//.write(0.99);
    } else if(current_position < 0) {
        digitalWrite(A_Out,LOW);// = 0;
        analogWrite(B_Out,0.01*255);//.write(0.01);
    }

    else {
            if(position_error >= 3){
                digitalWrite(A_Out,LOW);// = 0;
                analogWrite(B_Out,0.01*255);//.write(0.01);
// Command the velocity full speed in Direction 1
            }
            else if(position_error <= -3){
                digitalWrite(A_Out,LOW);// = 0;
                analogWrite(B_Out,0.99*255);//.write(0.99);
// Command the velocity full speed in Direction 2
            }
            else{
                digitalWrite(A_Out,HIGH);// = 1;
                analogWrite(B_Out,0.5*255);//.write(0.5);
// Command the velocity to 0
            }
    }
}


//ULTRASONIC SENSOR CODE
void pingsensor()
{
  //switch (sensorcycle){
    //case 1: {
      digitalWrite(SONAR_TRIGGER_PIN1, HIGH);
      delayMicroseconds(10);
      digitalWrite(SONAR_TRIGGER_PIN1, LOW);

      // Wait for Echo Pulse
      unsigned long pulse_length = pulseIn(SONAR_ECHO_PIN1, HIGH);

      // Convert Pulse to Distance (inches)
      // pulse_length/58 = cm or pulse_length/148 = inches
      leftdist = pulse_length / 148;

      //shift sensorcycle forward
      sensorcycle = 2;
      //Serial.println("LEFT"); Serial.println(leftdist);
    }
```

```
    case 2: {
      // Trigger the SRF05:
      digitalWrite(SONAR_TRIGGER_PIN2, HIGH);
      delayMicroseconds(10);
      digitalWrite(SONAR_TRIGGER_PIN2, LOW);

      // Wait for Echo Pulse
      unsigned long pulse_length = pulseIn(SONAR_ECHO_PIN2, HIGH);

      // Convert Pulse to Distance (inches)
      // pulse_length/58 = cm or pulse_length/148 = inches
      rightdist = pulse_length / 148;

      //shift sensorcycle forward
      sensorcycle = 1;
    }
  }
}
```

4.1.2. “DigitalPotentiometer.cpp”

```
#include "DigitalPotentiometer.h"

DigitalPotentiometer::DigitalPotentiometer(byte csPin)
{
  _cs = csPin;
  pinMode(_cs,OUTPUT);
  digitalWrite(_cs,LOW);

  SPI.begin();
}


int DigitalPotentiometer::setTap(int value) {
  Serial.print(" value");Serial.print(value);
  char ret;
  SPI.transfer(0);
  ret = SPI.transfer(value);
  Serial.print("transferred ");

  return ret;//(ret1 << 8) | ret2;
}

int DigitalPotentiometer::RcToDigiPot(float PwmSignal) {
    PwmSignal = PwmSignal;
    int DigiPotVal = (PwmSignal - RC_MID) * DigitalPotentiometer_MAX /
RC_AVG;
      return DigiPotVal;
}
```

4.1.3 “DigitalPotentiometer.h”

```
#ifndef DigitalPotentiometer_h
#define DigitalPotentiometer_h
#include <Arduino.h>
#include <SPI.h>
```

```
//#define MOSI    PTD2
//#define MISO    PTD3
//#define SCK           PTD1

#define DigitalPotentiometer_MIN 0
#define DigitalPotentiometer_MAX 128

#define RC_MID 1520//1513 //1534
#define RC_AVG 408 //399 // 429

class DigitalPotentiometer
{
public:
    DigitalPotentiometer(byte csPin);
      int initTCON();
      int readTCON();
      int readStatus();
      int increment();
      int decrement();
      int setTap(int value);

      /** Converts the incoming PWM signal and converts it to a value to be
sent to the digital potentiometer     *     * @param PwmSignal is the RC
controlled PWM signal sent from the transmitter through the receiver, Range
found to be 1095 us to 1935 us with 1515 us being the middle point     *
* @returns the value that is sent to the Digi Pot to change the Pot's value,
represented between -1 to 1     */
      int RcToDigiPot(float PwmSignal);

private:
    byte _cs;
    void enable();
    void disable();
};

#endif
```

4.2. Codes & Standards

1. Dr. Chowdhary listed some codes and standards (http://www.nhtsa.gov/cars/rules/import/FMVSS/), but those were meant for road worthy vehicles so we discussed using them a inspiration from these regulations and others to create a functionally safe vehicle without necessarily being able to meet them.
2. Motor Vehicles. 47-11-1116. Self-propelled or motor-driven and operated vehicles – Golf carts, all-terrain, and utility vehicles – operation on streets, highways, and roadways within unincorporated areas.
    a. The self-propelled or motor-driven and operated vehicles described in this section shall be prohibited from operating or shall be limited in operation on the streets and highways of this state.
    b. Golf carts and utility vehicles, as defined by Section 1102 of this title, shall not be operated on the streets and highways of this state except:
3. Golf carts or utility vehicles owned by the Oklahoma Tourism and Recreation Department, and operated by employees or agents of the Department or employees of independent management companies working on behalf of the Department, may be

operated on the streets and highways of this state during daylight hours or under rules developed by the Oklahoma Tourism and Recreation Commission, when the streets and highways are located within the boundaries of a state park. The Department shall have warning signs placed at the entrance and other locations at those state parks allowing golf carts or utility vehicles to be operated on the streets and highways of this state located within the boundaries of those state parks. The warning signs shall state that golf carts and utility vehicles may be operating on streets and highways and that motor vehicle operators shall take special precautions to be alert for the presence of golf carts or utility vehicles on the streets and highways;

4. The municipal governing body has adopted an ordinance governing the operation of golf carts and/or utility vehicles on city streets; provided, such ordinances shall include necessary vehicle lighting and safety requirements;
5. Golf carts or utility vehicles may operate on state highways only if making a perpendicular crossing of a state highway located within the boundaries of a municipality which has adopted an ordinance governing the operation of golf carts and/or utility vehicles; or
6. The board of county commissioners of a county has approved the operation of golf cart and/or utility vehicle traffic on roadways within the county, and: a. the roadway has a posted speed limit of twenty-five (25) miles per hour or less, b. the roadway is located in an unincorporated area, and c. appropriate signage, cautioning motorists of the possibility of golf cart or utility vehicle traffic, is erected by the board of county commissioners.83
7. Oklahoma (HB 3007)
    a. Introduced: January 2012.
    b. Status: Committee.
    c. Key Features: Defines “autonomous vehicle,” “artificial intelligence,” and “sensors,” and directs state Department of Public Safety to adopt rules for license endorsement and for operation, including insurance, safety standards, and testing of AVs (Oklahoma Legislature, 2012).

4.3. Safety Precautions

8. We discussed with Dr. Chowdhary the importance of safety for our project. The braking requirements as quantified by braking distance, desired deceleration, and anti-wheel lock/skid considerations are the most important. Reliability of the braking system is paramount too. There should be little or no delay between issuance of the braking command and the actuation.
9. Operating the Cart
    a. Do not operate the golf cart unless accompanied by another individual.
    b. Never drive recklessly or jokingly.
    c. Avoid Distractions while operating the golf cart.
    d. Never operate the vehicle under the influence of any drug or alcohol.
    e. Only carry the number of passengers as there are seats (2).
    f. Do not allow anyone to ride standing in the vehicle, and do not put the vehicle in motion until all passengers are sitting.
    g. Always use hand signals to indicate turns.
    h. Avoid sharp turns at a high rate of speed.
    i. Drive at a maximum speed relative to the outside conditions (rain or snow).
    j. Always yield to pedestrians and be aware of other vehicles.
    k. Working on the Cart
    l. Use proper lifting technique when moving heavy objects.
    m. Wear safety glasses at all times.

10. If in a machine shop area, follow machine shop rules including but not limited to:
    a. Take extreme care when dealing with high voltage, wear proper protection if necessary (gloves).
    b. No long hair or sleeves.
    c. Closed toe shoes or boots.
    d. Do not use equipment without proper training
    e. Ask assistance before attempting something you're unsure of.
    f. Report any safety incidents to the team lead and then to Dr. Delahoussaye.
    g. No horseplay or reckless use of equipment.
    h. Dispose of waste according to the machine shop's standards
    i. Obey all posted signs, warnings, and special instructions.
    j. If food is allowed where you are working, keep it away from all equipment.
    k. Follow Shop and University guidelines for dealing with emergencies

4.4. Free Body Diagram of Lever

The following is a free body diagram of the lever which connects the actuator to the brake cable. As can be seen, a force of 2000 lbs is exerted on the pin.