5-2016

Campus Cruiser

Mikey Robison
Trinity University, mrobison@trinity.edu

Todd Edwards
Trinity University, tedward1@trinity.edu

CAMPUS CRUISER

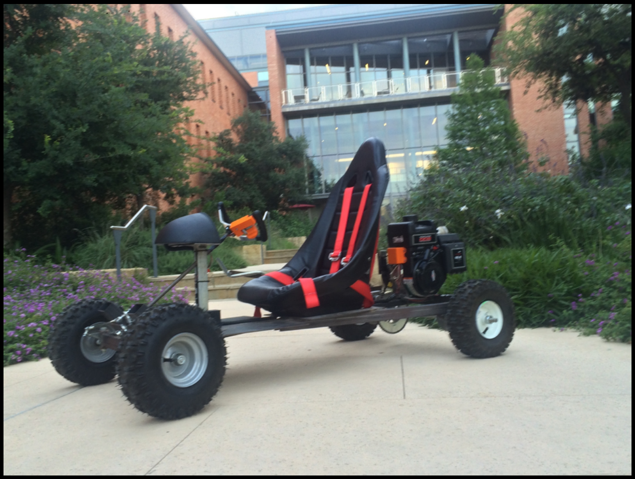
Figure 1:The Campus Cruiser

Mikey Robison and Todd Edwards

GROUP F

ENGR 4367

5/2/2016

Dr. Nickels

Dr. Nickels

Table of Contents

Design Summary

Throughout the course of this semester we designed and built an electronic control unit for a gas-powered go-kart. The goal for our project was to design and install an electronic control unit that would incorporate: an electronic throttle, an electronic kill switch, two tachometers – one to measure vehicle speed and one to measure engine shaft speed – with LCDs to display the speed and RPM values. The control unit, displays, and controls were mounted on steering wheel and dashboard housings that included two LEDs to display the status of the engine kill switch and a speaker which beeps to alert the user that the engine is ready to be started once the control system has been activated and initialized. The finished go-kart can be seen in Figure 1.

System Details

The control unit was designed to use three PIC16F88 microcontrollers mounted on a printed circuit board. The first PIC was designed to handle the throttle system, kill switching, diagnostic LEDs, and diagnostic buzzer. The remaining two PICs were designed to decode, process and display on the LCDs the inputs from the speed tachometer and engine shaft tachometers respectively. The overall control unit functionality is illustrated in the functional block diagram seen in Figure 2.

The control unit was powered using a 12V 2Ah battery. The battery mounted next to the steering column in a 3D printed box and connected to the circuit through a key-lock switch mounted on the steering column seen in Figure 3 so that the vehicle could not be operated without the key, for added security. The battery was connected to the circuit board through an LM-317 linear voltage regulator to step the battery down to 5V to protect the PICs and other circuitry rated for 5V. The software flowchart and wiring diagram for the control system can be seen in Figure 4, 5 respectively.

The printed circuit board, wiring, and displays were all mounted inside a 3D printed dashboard mounted on top of the steering column. At the top are red and green LEDs that serve as a visual indication to the user whether or not the engine can be started. The display panel also houses the speedometer and RPM displays as well as a piezo buzzer that beeps to indicate that the system is ready for the engine to be started. The dashboard and wiring can be seen in Figure 9, 10.

Throttle System

The throttle system for our go-kart was designed to use a potentiometer mounted on the steering wheel to vary a voltage read by a PIC16F88 microcontroller. The microcontroller was programmed to turn a small hobby servo motor in a 130˚ arc corresponding to approximately a 40˚ turn of the potentiometer. The servo was connected to a spring underneath the gas tank that is directly connected to the engine's built in mechanical control system where more tension on the spring corresponds to more power output from the engine.

The servo mounting and assembly can be seen in Figure 6. A throttle paddle was developed using a small door hinge with the potentiometer mounted inside the hinge pin so that moving the hinge towards the operator would turn the potentiometer the control surface and mounted potentiometer were covered with 3D printed covers for user comfort and protection of the components. The throttle assembly and covers can be seen in Figure 7, 10.

Kill Switch

The kill switch for the go-kart was designed to use a solenoid relay to control the sparkplug connection. The original kill switch was disassembled and re-wired to the solenoid relay so that the sparkplug would be grounded, thus disabling the engine, when the solenoid was not powered. When the solenoid was energized the spark plug is connected to the starter coil and thus the engine is able to be started. The solenoid was mounted on the front of the engine in a 3D printed box for protection from outside interference. This assembly can be seen in Figure 8. In order to protect the PIC from the high current that the solenoid sinks when active a N-channel MOSFET was used as a power switch to control the solenoid current using the low current PIC output.

The solenoid was controlled through the same PIC16F88 used to control the throttle. The button on the upper right of the steering wheel seen in Figure 7 was read as an input by the PIC and when pressed would set the solenoid low so that the engine would die. Using this configuration also meant that if power was lost, the circuit was damaged, or no battery was connected the engine would not start or continue running, making the vehicle safer to operate.

Tachometers

In order to measure the speed of the go-kart as well as the speed of the engine output shaft we used two separate tachometers, one mounted on the axle and one mounted on the output shaft. The tachometers were built in two parts. The first part was a U shaped bracket with a small Hall effect sensor mounted on one inside and a permanent magnet mounted on the other inside. The second part was an aluminum disk approximately six inches in diameter. Each disk was positioned so that it would rotate between the magnet and the sensor. Each disk had a small strip of steel screwed onto the face so that the strip would block the magnetic field from the magnet and the sensor would produce a five volt pulse each time the strip passed through the sensor. The engine shaft tachometer had a single strip to measure the speed, the axel tachometer had two strips to allow for a more precise speed measurement.

These hall effect sensors were connected to each of their respective PICs. These PICs were programmed to count the pulses received in a given period of time and, according to the tachometer in question, perform some arithmetic to produce the desired value. For the speed tachometer the speed in MPH was displayed. For the engine shaft tachometer, the revolutions per minute were measured and then displayed in progress bar form ranging between 0 and 3600, the limits of the engine performance. The speedometer and RPM tachometer assemblies can be seen in Figure 11, 12 respectively.

Design Evaluations

Output Display

All of our Output displays performed their function exactly as anticipated. The two status LEDs were similar to what we used in class and were relatively simple to use. The two LCDs that display the speed and engine RPM were quite difficult to implement and required extensive research that was not found anywhere in the course material. Since we used PICs for all of our functionality we had to learn how to use LCDs completely from scratch. Creating the iterating progress bar for the RPM tachometer was especially difficult and required strenuous effort to pull off. Using two LCDs with different display modes also added an extra degree of difficulty. We believe our project deserves a rating of 20 in this category.

Audio Output Display

Besides the sweet tones from the engine our audio output display consisted of a Piezo buzzer designed to indicate when the system was ready for the engine to be started. This topic was not discussed in class or lab and took some significant research in the book to develop the software to achieve the desired response. The buzzer works exactly as desired and works every time. We believe our project deserves a rating of 15 in this category.

Manual User Input

Our manual user inputs consisted of the throttle potentiometer and the kill switch button. The button is simple enough and is the same as every other button we used all year. The potentiometer is similar to what we used in the labs throughout the semester. However, significant software modifications had to be made since the potentiometer was restricted to approximately 40˚ of rotation. This was very difficult to achieve but the solution works exactly as desired every time. Besides the software modifications the method of mounting and interacting with the potentiometer was very difficult to design and implement. We believe that our project deserves a rating of 15 in this category.

Automatic Sensor

Our project had two versions of the same automatic sensor. Both tachometers took extensive and strenuous independent research to develop. The construction of two tachometers from scratch alone was very difficult but developing the software to process the outputs from the sensors and give the correct value was incredibly difficult. We had to do some significant research to learn how to use the PICs to count pulses and produce a usable value. The speedometer was especially difficult due to the fact that the changes in speed of the go-kart are not very significant, a change of 1 Hz of the tach frequency correlates to a change of about 2.5 MPH. Since the compiler we were using does not support floating point decimal calculation we had to find a solution to give us better resolution for the speed display. We ended up increasing the sampling frequency to once a second and doubling the number of steel strips on the tachometer wheel to two to ensure we had enough resolution. The result was very good and gives a very accurate speed reading. We believe that our project definitely deserves a rating of 20 in this category.

Actuators, Mechanisms & Hardware

Our project had two actuators. The kill switch solenoid relay was relatively easy to use as it only required a high or a low to function and it was easy to extrapolate what we learned in class to implement it. Our other actuator was the servo motor used to control the throttle. This was much more difficult to do as it was not covered in any of the course material. Since we only used PICs there was no easy command to control a servo. Once we learned how to control servos through extensive research we decided to use the pulse out command modified by the potentiometer input to control the position of the servo. As far as hardware goes we had to extensively modify a pre-existing go-kart. We re-built the rear axle, adding a larger sprocket, and open differential. We re-built the steering assembly, including making modifications to the frame. We also completely re-built the brakes. We believe that our project deserves a rating of 20 in this category.

Logic Processing and Control

For our LCC category we included open-loop control of a servo motor using a potentiometer that was very difficult to implement due to the fact that the PWM command we learned in class is not suitable for servo control. We inevitably learned how to use a slightly modified version of the pulse out command to control the servo. We also included counting and arithmetic with our two tachometers and displays that took a lot of independent research as well as construction of the tachometers. We believe our project deserves a rating of 20 in this category.

Partial Parts List

Solenoid Relay	-	The electronics shop	-
Small Hobby Servo	From Arduino starter kit	The electronics shop	-
hall effect sensor	US5881LUA	Adafruit.com	$2.00
High strength ‘rare-earth’ magnet	Product ID: 9	Adafruit.com	$2.00
12V 2Ah battery	-	The electronics shop	-
PIC16F88 X3	-	The electronics shop	$3.74
Key-Lock switch	CKC8039-ND	Digi-key.com	$11.73
Piezo buzzer	From Arduino starter kit	The electronics shop	-
16mm Panel Mount Momentary Pushbutton - Black	Product ID: 1505	Adafruit.com	$0.95

Lessons Learned

The biggest difficulty we faced with this project was dealing with the fact that we were strapping a bunch of electronics to a motor vehicle. To try and solve this problem we used a printed circuit board, designed in eagle to solder all of our connections to. **DO NOT TRUST THE AUTOROUTER.** The auto router makes lots of mistakes, though it may seem daunting, route everything by hand it will guarantee a better board. I ended up scraping copper off my board because one of my signals was accidentally connected through the 5V bus, an easy fix had I caught it in the schematic. Another piece of advice about eagle: double check the pinouts in the schematic, they may be labeled in a different order than your wiring diagram. In our case the voltage regulator schematic I used to design the circuit had pins in a different order than the eagle schematic which resulted in the wrong pin being connected to everything that needed 5V.

Try and think ahead as much as possible when making design decisions. Think about what things will look like and how they will work. We faced a lot of difficulty when connecting our wires to the board because of the way we connected them. It was very difficult to make the connections in the small space of the dashboard. They also weren’t as solid as we would have liked. If I could go back and re-do the wire connectors so that they would be easier to connect and more robust connections.

Appendix

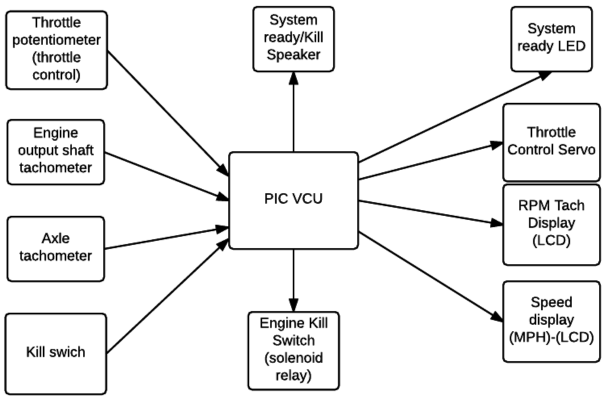
Throttle potentiometer (throttle control)
Engine output shaft tachometer
Axle tachometer
Kill swich
System ready/Kill Speaker
PIC VCU
Engine Kill Switch (solenoid relay)
System ready LED
Throttle Control Servo
RPM Tach Display (LCD)
Speed display (MPH)-(LCD)

Figure 2: Functional block diagram outlining overall functionality of go-kart controller

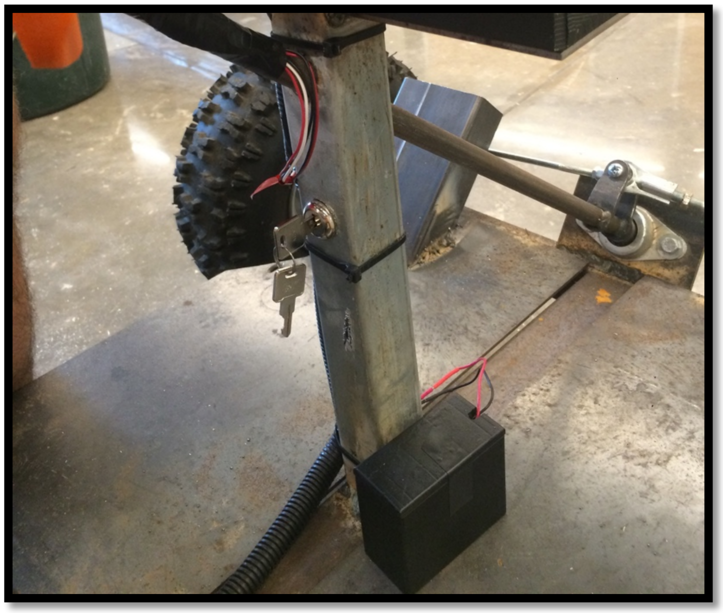

Figure 3: Key-lock power switch and battery housing

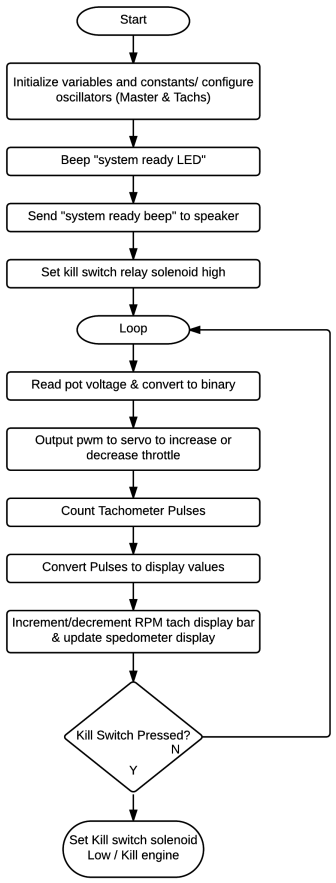
Start
Initialize variables and constants/ configure oscillators (Master & Tachs)
Beep "system ready LED"
Send "system ready beep" to speaker
Set kill switch relay solenoid high
Loop
Read pot voltage & convert to binary
Output pwm to servo to increase or decrease throttle
Count Tachometer Pulses
Convert Pulses to display values
Increment/decrement RPM tach display bar & update spedometer display
Kill Switch Pressed?
N
Y
Set Kill switch solenoid Low / Kill engine

Figure 4: Software flowchart for the go-kart control unit

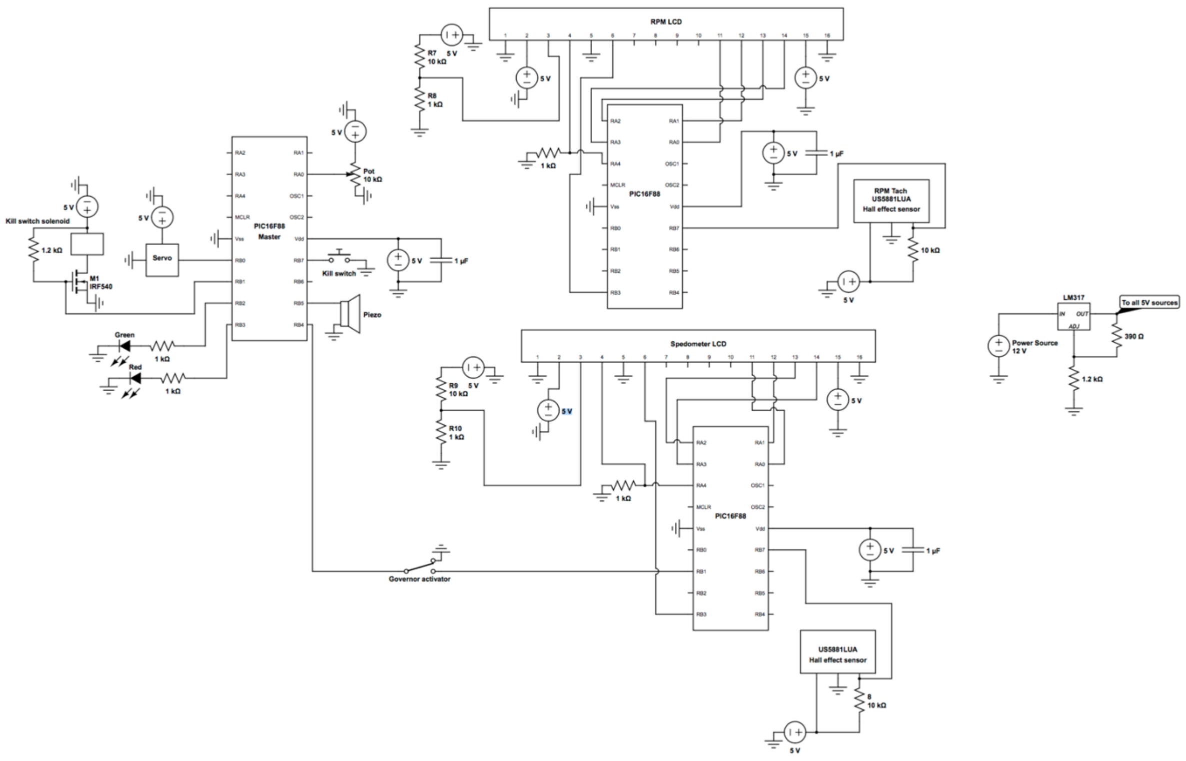

Figure 5: Wiring diagram for go-kart control unit

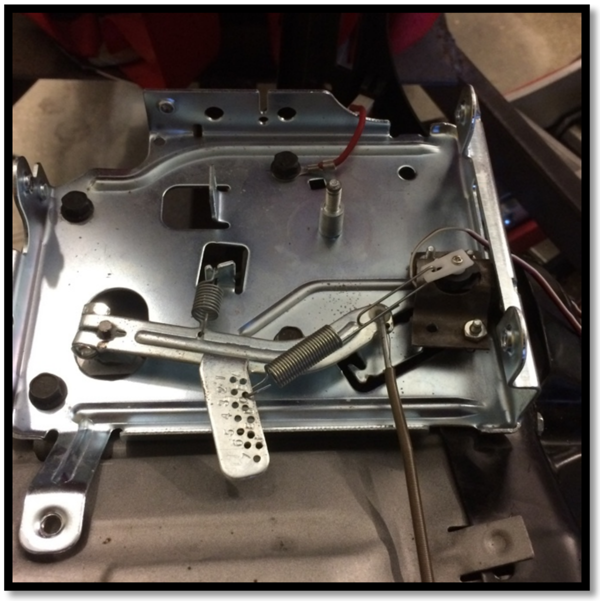

Figure 6: Throttle servo mounting and assembly

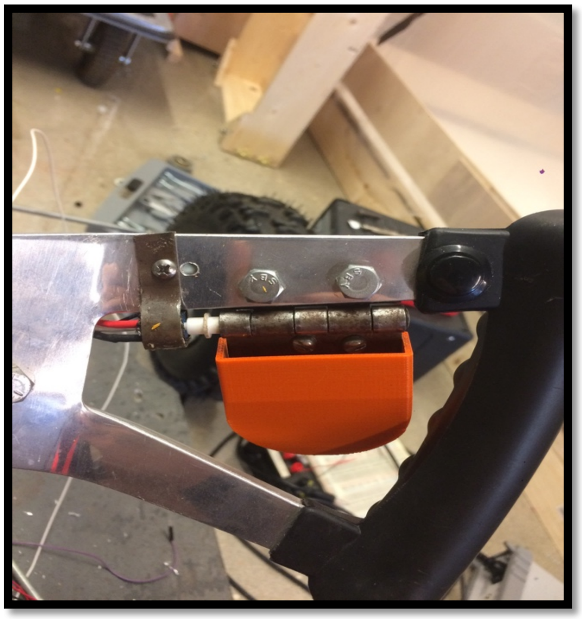

Figure 7: Throttle potentiometer assembly with throttle cover

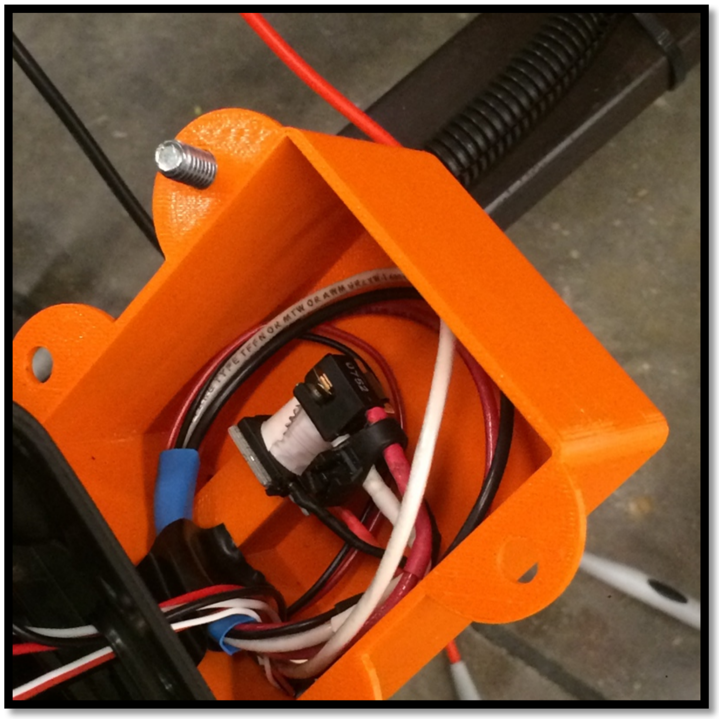

Figure 8: Kill switch solenoid wiring and housing

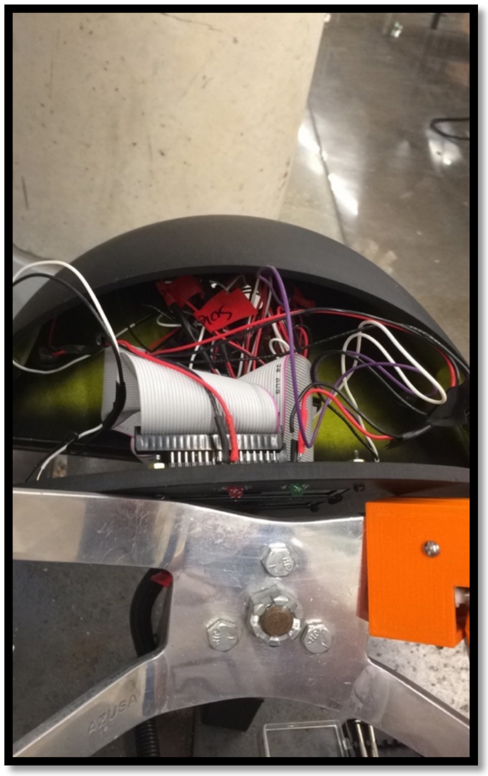

Figure 9: Wiring inside dashboard

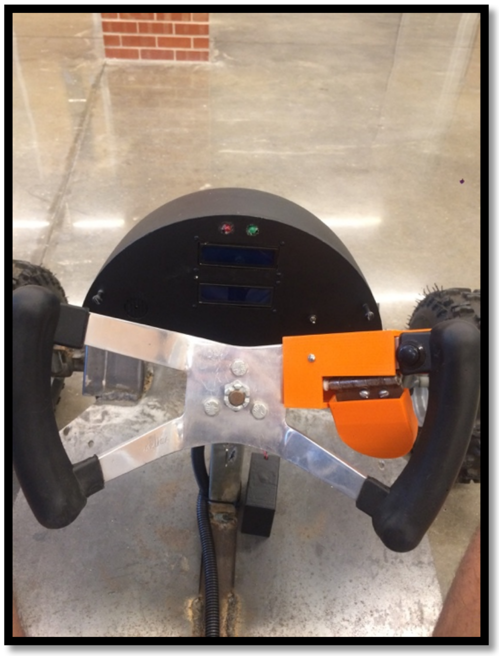

Figure 10: Dashboard and steering wheel assembly

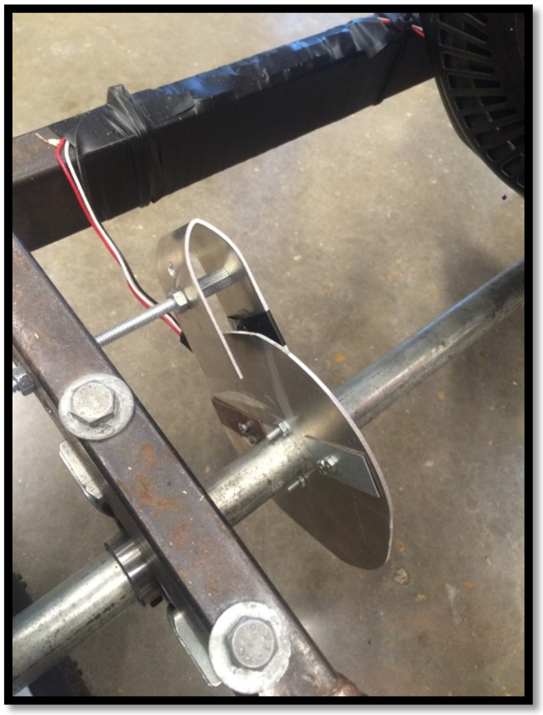
Figure 11: Speedometer tachometer

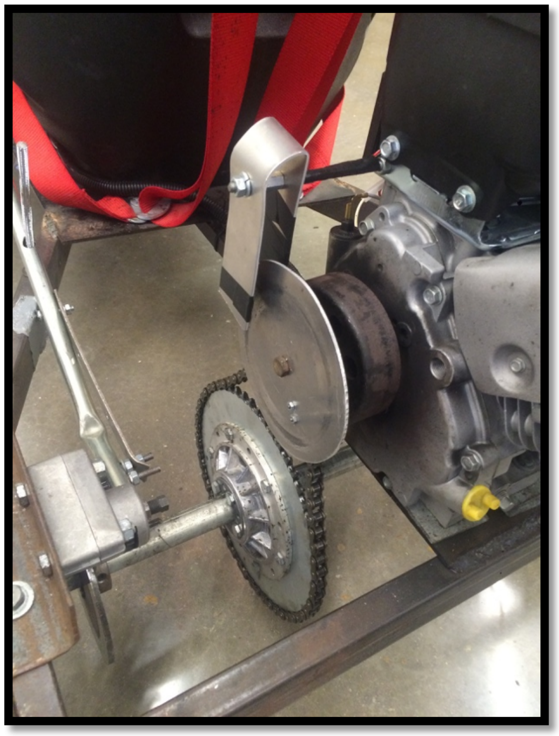
Figure 12: RPM tachometer

```
'****************************************************************
'*  Name    : Go kart servo control                                    *
'*  Author  : [Mikey Robison and Todd Edwards]                   *
'*  Notice  : Copyright (c) 2016 [select VIEW...EDITOR OPTIONS] *
'*          : All Rights Reserved                               *
'*  Date    : 3/27/2016                                         *
'*  Version : 1.0                                               *
'*  Notes   :                                                   *
'*          :                                                   *
'****************************************************************

' The following configuration bits and register settings
' enable the internal oscillator, set it to 8MHz,
' disables master clear, and turn off A/D conversion

' Configuration Bit Settings:
' Oscillator                    INTRC (INT102) (RA6 for I/O)
' Watchdog Timer                Enabled
' Power-up Timer                Enabled
' MCLR Pin Function             Input Pin (RA5 for I/O)
' Brown-out Reset               Enabled
' Low Voltage Programming           Disabled
' Flash Program Memory Write            Enabled
' CCP Multiplexed With              RB0
' Code                      Not Protected
' Data EEPROM                   Not Protected
' Fail-safe Clock Monitor           Enabled
' Internal External Switch Over         Enabled

' Define configuration settings (different from defaults)
#CONFIG
    __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#endconfig

' Set the internal oscillator frequency to 8 MHz
define OSC 8
OSCCON = %01111000

define  ADC_BITS        10      ' Set number of bits in result
define  ADC_CLOCK       3       ' Set clock source (3=rc)
define  ADC_SAMPLEUS    15      ' Set sampling time in uS

'Set register I/O
TRISA = %11111111
TRISB = %11000001

'Define I/O pin names and initialze
servo var portB.0
low servo
kill var portB.7
relay var portB.1
green_status var portB.2
red_status var portB.3
buzz_status var portB.5
governor_in var portb.4
i var byte
' Set up ADCON1
ADCON1 = %10000000 ' Right-justify results (lowest 10 bits)
ansel.6=0 'turn off AN6 to allow digital I/O
' Enable PORTB pull-ups
OPTION_REG = $7f
```

```
'Declare adc variables
ad_word var word '10 bit word from adcin
ad_byte var byte 'pot position byte

inititial: 'power on, wait for LCDs, give audible & visual signal that go-kart is
ready
    low red_status
    low green_status
    toggle green_status
    pause 300
    toggle green_status
    pause 300
    toggle green_status
    pause 300
    toggle green_status
    pause 300
    toggle green_status
    sound buzz_status,[ 100,600]


main:
while(1)
    adcin 0, ad_word
    ad_byte = ad_word

    pulsout servo,1125-ad_word 'scale pot and offset for use with throttle pot
    pause 10 - (ad_byte/100)

    if kill = 1 then
        high relay
        pause 10
    else
        low relay
        pause 10
    endif

    if relay =1 then
        high green_status
        low red_status
    else
        high red_status
        low green_status
    endif

wend
```

```
'****************************************************************
'*  Name    :Go-kart spedometer                                 *
'*  Author  : Mikey Robison & Todd edwards                      *
'*  Notice  : Copyright (c) 2016 [select VIEW...EDITOR OPTIONS] *
'*          : All Rights Reserved                               *
'*  Date    : 4/1/2016                                          *
'*  Version : 1.0                                               *
'*  Notes   : counts shaft rotations, translates to mph, then   *
'*          : displays on an LCD                                *
'****************************************************************
' Define configuration settings (different from defaults)
#CONFIG
    __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#endconfig

' Set the internal oscillator frequency to 8 MHz
define OSC 8
OSCCON = %01111000
ansel=0 'deactivate adc


'Declare variables
rpm_word var word '16 bit value for counts every 1/10 sec on pin 7
rpm_byte var word
speed var word
governor  var portb.1
    pause 500 'let LCD initialize
    main:
        lcdout $fe,1
        lcdout $fe,$80+5,"Speed"
        lcdout $fe,$c0+11, "MPH"

while (1)

    count portb.7, 1000, rpm_word 'counts pulses every 1/10 sec
    rpm_byte = rpm_word*125 'scale to pulses per second
    speed=rpm_byte/100  'conversion from rotations per second to MPH

    if speed>10 then
        lcdout $fe, $c0+7,dec speed
    else
        lcdout $fe, $c0+7, dec speed, "  " 'clear extra 0 if speed less than 10
    endif

    if speed > 20 then
        high governor
    else
        low governor
    endif
wend
```

```
'****************************************************************
'*  Name    : Go-Kart Rpm Code                                  *
'*  Author  : [Mikey Robison & Todd Edwards]                    *
'*  Notice  : Copyright (c) 2016 [select VIEW...EDITOR OPTIONS] *
'*          : All Rights Reserved                               *
'*  Date    : 3/30/2016                                         *
'*  Version : 1.0                                               *
'*  Notes   :                                                   *
'*          :                                                   *
'****************************************************************
' Define configuration settings (different from defaults)
#CONFIG
    __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#endconfig

' Set the internal oscillator frequency to 8 MHz
define OSC 8
OSCCON = %01111000
'turn off adc
ansel=0


'Declare variables
rpm_word var word '10 bit word from adcin (tachometer)
rpm_byte var byte 'RPM byte
i var byte
prev_rpm var byte
    pause 500 'let LCD initialize
    main:
        lcdout $fe,1
        lcdout $fe,$80,"0     RPM   3600"
 prev_rpm = 0

while (1)
    count portb.7, 100, rpm_word
    rpm_byte = rpm_word*10/4 'scale input to 0-16
    if (prev_rpm<>rpm_byte) then
    'set i to follow tach voltage

     for i=  16 to rpm_byte step -1  'clear progress bar'
    lcdout $fe,$C0+i," "
    next i

    for i= 0 to rpm_byte 'iterate up progress bar'
    lcdout $fe,$C0+i,$ff
    next i
    prev_rpm = rpm_byte
else
 prev_rpm=rpm_byte
endif


wend
```