RC Control

CSS 427: Final Report

Brad Minogue, Carl Martinez, Amit Moolani, Alexander Yelle, Enoch Fu

Implementation of a remote control subsystem for an Elcano tricycle.

RC Control

CSS 427: Final Report

Introduction

This project led to the implementation of a remote control subsystem for use on an Elcano tricycle. Using the knowledge and experience that we gained in previous labs and lectures we were able to fully integrate a remote control into the steering, accelerating, and braking of an Elcano tricycle. An Arduino with a DAC connected acted as the gateway from the RC receiver to the servo-motor and throttle. The process took 3 weeks to complete, each week accomplishing a different goal:

1. Adding throttle connectivity to the trike.
2. Writing and testing code to translate RC input data to correct PWM signals.
3. Hooking up the Arduino to the trike, testing functionality.

Problems Addressed

- ### Adding Throttle Capability
  The project started with troubleshooting the throttle connection on the trike. Everything appeared to be wired correctly but we would get no response from the throttle. We had to test every connection and power source to find the root of the problem. After testing all existing connections and finding nothing wrong we had to turn to the manual to find out why the throttle was not working. The end result was that the control power supply pin was not connected. After making the appropriate connection the trike gained throttle capabilities.

- ### RC Receiver Connection
  We used the left joystick from the RC receiver to receive input for acceleration, steering and braking. Right and left for steering, forward and backward for acceleration and braking. Attaching the RC receiver to an Arduino allowed us to translate the inputs into a PWM signal to control the servo-motor and throttle. We wrote code to make a linear translation from RC input to the PWM signal. A DAC was attached to the Arduino to allow for correct throttle output.

- ### Calibration of Input and Output
  Using existing code we already had the range of numbers for the output values. The RC input required testing to find the maximum and minimum values for steering, braking, accelerating, and how large the dead zone should be.

- ### Bike Connections
  We then were able to connect our programmed Arduino to the shield connected to the yellow trike. The trike had battery packs to provide the Arduino and motors with power. Connections existed from the motors to a breadboard that sat on the trike. After reading through schematics and checking connections with the previous setup, we connected our RC receiver to get power from the Arduino’s 5V output. We also connected the Channels 1 and 2 from the receiver to be read into pins 2 and 3 of the Arduino. These gave the RC transmitted

the ability to control the steering and braking of the trike respectively. We then connected the motor's steering and braking input pins to the Arduino's steering and braking output pins. These were pins 6 and 7 on the Arduino respectively. Then the throttle wires had to be snipped and connected to the 5V and GND pins of the Arduino. The input for the throttle was connected to the Arduino via pin 12 on the 15 pin DAC. Once those connections were made and the trike was fastened and powered up, we were able to confirm the working of our trike.

- Brakes
  The braking system of the trike had been manipulated in the past, restricting its functioning. We were able to find the loose connection in the brake that did not allow the brake to ever fully compress over the wheel to give it adequate braking. We attempted to fix this by tightening the brake wires and moving the location of the pivot holding the brakes to the brake system motor. After an hour and a half of debugging we were able to mildly improve the system, but still needs much more work to be able to work to the full of its capability.

Code

```
//RC_Control.ino
#include <SPI.h>
//RC input values
const int DEAD_ZONE = 75;
const int ACC_START = 1322;
const int TURN_START= 1322;
//max brake
const int MIN_ACC = 911;
//full throttle
const int MAX_ACC = 1730;
//max turn left
const int MIN_TURN = 911;
//max turn right
const int MAX_TURN = 1733;

//OUTPUT values
const int MIN_ACC_OUT = 40;
const int MAX_ACC_OUT = 227;

const int MIN_BRAKE_OUT = 167;
const int MAX_BRAKE_OUT = 207;

const int LEFT_TURN_OUT = 126;
const int RIGHT_TURN_OUT = 250;
const int STRAIGHT_TURN_OUT = 187;

//ARDUINO PIN SELECTORS
const int TURN_PIN = 2;
const int ACC_PIN =3;
const int ACC_OUT_PIN = 5;
const int STEER_OUT_PIN = 6;
const int BRAKE_OUT_PIN = 7;
const int SelectCD     = 49;  // Select IC 3 DAC (channels C and D)
const int SelectAB     = 53;  // Select IC 2 DAC (channels A and B)

void setup()
{   //Set up pins
   pinMode(ACC_OUT_PIN, OUTPUT);
   pinMode(BRAKE_OUT_PIN, OUTPUT);
   pinMode(STEER_OUT_PIN, OUTPUT);
   // SPI: set the slaveSelectPin as an output:
   pinMode (SelectAB, OUTPUT);
   pinMode (SelectCD, OUTPUT);
   pinMode (10, OUTPUT);
   SPI.setDataMode( SPI_MODE0);
   SPI.setBitOrder( MSBFIRST);
   // initialize SPI:
   // The following line should not be neccessary. It uses a system library.
   PRR0 &= ~4;  // turn off PRR0.PRSPI bit so power isn't off
   SPI.begin();
   for (int channel = 0; channel < 4; channel++)
       DAC_Write (channel, 0);   // reset did not clear previous states

   Serial.begin(9600);
}
//Checks for each input: accelerating, turning, braking
void loop() {
 //Steering
 int test = pulseIn(TURN_PIN, HIGH,25000);
 if(testTurn(test))
 {
   //Serial.print("turn: ");
   //Serial.println(convertTurn(test));
   steer(convertTurn(test));
 }
 else
 {
   steer(STRAIGHT_TURN_OUT);
 }
 //Braking
 test = pulseIn(ACC_PIN, HIGH,25000);
 if(testBrake(test))
 {
   //Serial.print("braking: ");
   //Serial.println(convertBrake(test));
   brake(convertTurn(test));
 }
 else
 {
   brake(MIN_BRAKE_OUT);
 }
 //Accelerating
 if(testAcc(test))
 {
   //Serial.print("Acc: ");
   //Serial.println(convertThrottle(test));
   moveVehicle(test);
 }
 else
 {
   moveVehicle(MIN_ACC_OUT);
 }
 delay(50);
}
//Converts RC values to corresponding values for the DAC
int convertTurn(int input)
{
 //full turn right = 250, full turn left = 126
 int dacRange = RIGHT_TURN_OUT - LEFT_TURN_OUT;
 int rcRange = MAX_TURN - MIN_TURN;
 input-= MIN_TURN;
 double output = (double)input /(double)rcRange;
 //swap left and right
 output = 1 - output;
 output *= dacRange;
 output += LEFT_TURN_OUT;
 //set max and min values if out of range
 int trueOut = (int)output;
 if(trueOut > RIGHT_TURN_OUT)
   trueOut = RIGHT_TURN_OUT;
 else if(trueOut < LEFT_TURN_OUT)
   trueOut = LEFT_TURN_OUT;
 return trueOut;
```

```
}
int convertBrake(int input)
{
 //full brake = 207, min = 167
 const int dacRange = MAX_BRAKE_OUT - MIN_BRAKE_OUT;
 const int rcRange = (ACC_START - DEAD_ZONE) - MIN_ACC;
 input -= MIN_ACC;
 double output = (double)input /(double)rcRange;
 output *= dacRange;
 output += MIN_BRAKE_OUT;
 //set min values if out of range
 int trueOut = (int)output;
 if(trueOut < MIN_BRAKE_OUT)
   trueOut = MIN_BRAKE_OUT;
 return trueOut;
}
int convertThrottle(int input)
{
 //full throttle = 227, min = 40
 const int dacRange = MAX_ACC_OUT - MIN_ACC_OUT;
 const int rcRange = MAX_ACC - (ACC_START + DEAD_ZONE);
 input -= (ACC_START + DEAD_ZONE);
 double output = (double)input /(double)rcRange;
 output *= dacRange;
 output += MIN_ACC_OUT;
 //set max values if out of range
 int trueOut = (int)output;
 if(trueOut > MAX_ACC_OUT)
   trueOut = MAX_ACC_OUT;
 return trueOut;
}

//Tests for inputs
boolean testTurn(int turn)
{
 return (turn > ACC_START + DEAD_ZONE || turn < ACC_START - DEAD_ZONE);
}
boolean testAcc(int acc)
{
 return (acc > ACC_START + DEAD_ZONE);
}
boolean testBrake(int brake)
{
 return (brake < ACC_START - DEAD_ZONE);
}
//Send values to output pin
void steer(int pos)
{
analogWrite(STEER_OUT_PIN, pos);
}
void brake(int amount)
{
analogWrite(BRAKE_OUT_PIN, amount);
}
void moveVehicle(int acc)
{
   /* Observed behavior on ElCano #1 E-bike no load (May 10, 2013, TCF)
     0.831 V at rest       52 counts
     1.20 V: nothing       75
     1.27 V: just starting 79
     1.40 V: slow, steady  87
     1.50 V: brisker       94
     3.63 V: max          227 counts
     255 counts = 4.08 V
     */
  DAC_Write(0, acc);
}
/*------------------------------------------------------------------------------------*/
/* DAC_Write applies value to address, producing an analog voltage.
```

```
// address: 0 for chan A; 1 for chan B; 2 for chan C; 3 for chan D
// value: digital value converted to analog voltage
// Output goes to mcp 4802 Digital-Analog Converter Chip via SPI
// There is no input back from the chip.
*/
void DAC_Write(int address, int value)

/*
REGISTER 5-3: WRITE COMMAND REGISTER FOR MCP4802 (8-BIT DAC)
A/B  —  GA  SHDN  D7 D6 D5 D4 D3 D2 D1 D0 x x x x
bit 15                                      bit 0

bit 15   A/B: DACA or DACB Selection bit
        1 = Write to DACB
        0 = Write to DACA
bit 14   — Don’t Care
bit 13   GA: Output Gain Selection bit
        1 = 1x (VOUT = VREF * D/4096)
        0 = 2x (VOUT = 2 * VREF * D/4096), where internal VREF = 2.048V.
bit 12   SHDN: Output Shutdown Control bit
        1 = Active mode operation. VOUT is available.
        0 = Shutdown the selected DAC channel. Analog output is not available at the channel that was shut
down.
        VOUT pin is connected to 500 k (typical)
bit 11-0 D11:D0: DAC Input Data bits. Bit x is ignored.

With 4.95 V on Vcc, observed output for 255 is 4.08V.
This is as documented; with gain of 2, maximum output is 2 * Vref

*/

{
 int byte1 = ((value & 0xF0)>>4) | 0x10; // acitve mode, bits D7-D4
 int byte2 = (value & 0x0F)<<4;           // D3-D0
 if (address < 2)
 {
     // take the SS pin low to select the chip:
     digitalWrite(SelectAB,LOW);
     if (address >= 0)
     {
       if (address == 1)
         byte1 |= 0x80;  // second channnel
       SPI.transfer(byte1);
       SPI.transfer(byte2);
      }
     // take the SS pin high to de-select the chip:
     digitalWrite(SelectAB,HIGH);
 }
 else
 {
     // take the SS pin low to select the chip:
     digitalWrite(SelectCD,LOW);
     if (address <= 3)
     {
       if (address == 3)
         byte1 |= 0x80;  // second channnel
       SPI.transfer(byte1);
       SPI.transfer(byte2);
     }
      // take the SS pin high to de-select the chip:
     digitalWrite(SelectCD,HIGH);
 }
}
```