ME 4370-001 Mechatronics

Vehicle Subsystems Simulation Project

05/08/2020

Oussama Amara, Said Mohamed

Dr. Canfield

Table of Contents

Introduction

The goal of this project was to simulate different subsystems in a vehicle using ATmega 2560 processors as electronic control units. We simulated three different subsystems: side mirror adjustment, automatic headlights, and security/car alarm. There are multiple processors communicating with each other using the TTUCAN protocol software to achieve this. The CAN (Controller Area Network) enables a vehicle's controllers to operate independently without a need for a central controller. All nodes are able to transmit messages to the bus, which consists of two data lines. Nodes use filters to ensure that they only process messages that are addressed to them. This system consists of four nodes: Node 0 has the actuators for controlling mirror position, Node 1 has the user interface to control the mirrors and sends data to Node 0, Node 2 has the LEDs for the headlights and the buzzer for the car alarm, Node 3 has the user interface to control the headlights and security; it sends data to Node 2.

Procedure for Design

Side Mirror Adjustment: We use a combination of stepper motors and servo motors on Node 0 to adjust the x and y position of the mirrors. The control interface on Node 1 consists of an analog joystick that is used to control position, and two buttons to choose which mirror to adjust.

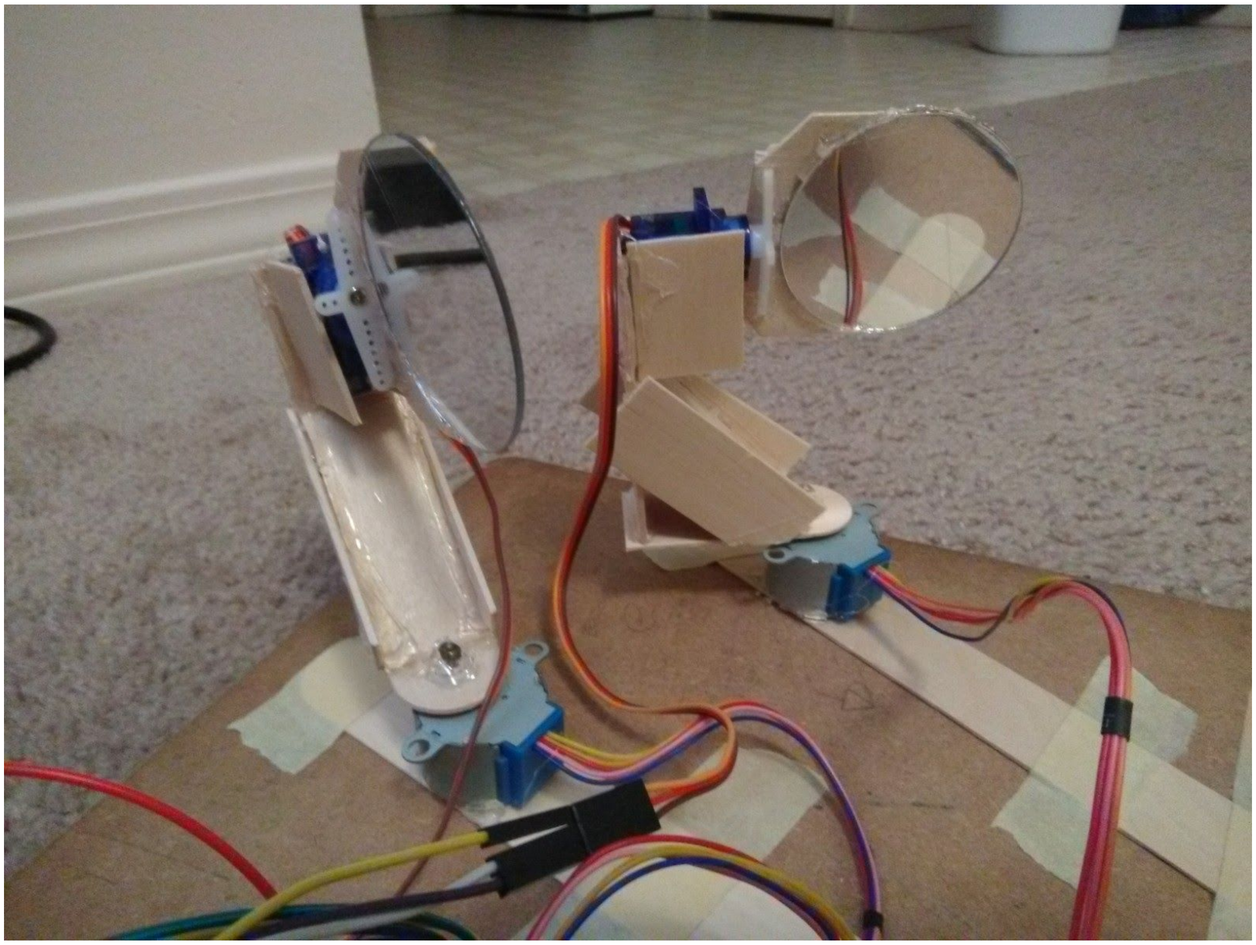
The servos on Node 0 are controlled using Pulse Width Modulation. The position of the joystick on Node 1 is measured using Analog to Digital Conversion.

Automatic Headlights: To create automatic headlights, we used a photoresistor sensor on Node 3 to detect ambient light and sent the measured voltage to Node 2. Based on the amount of light detected, Node 2 turns the lights on or off. The voltage from the photoresistor was measured using Analog to Digital Conversion. There are three buttons connected to Node 3 that are used to turn on the headlights, select the automatic setting, or turn on bright headlights.

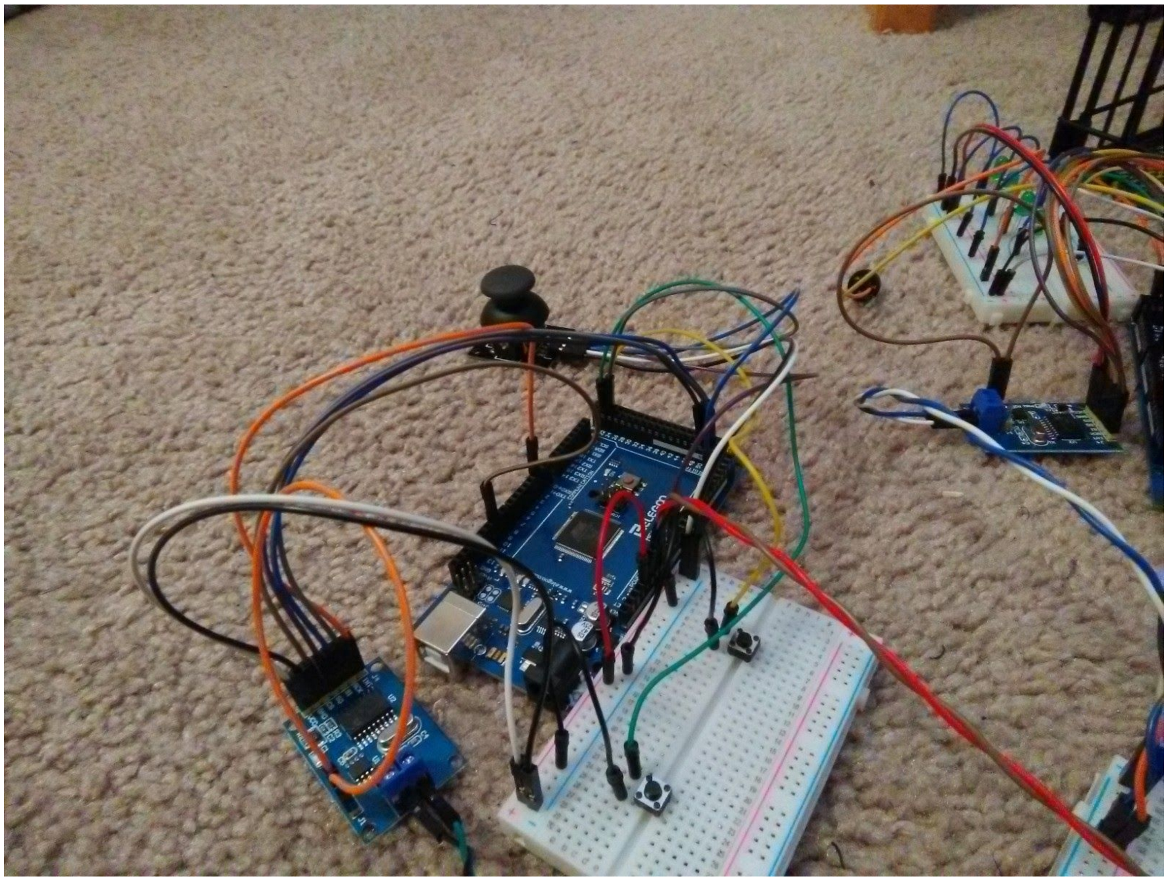
Car alarm: We used a PIR motion sensor on Node 3 to trigger an external interrupt on a rising edge and turn on a car alarm created by a passive buzzer on Node 2. We created a real-time interrupt using output compare matching on Node 2 to control the pitch of the car alarm. There is a button on Node 3 to arm and disarm the security system.

Procedure for Development

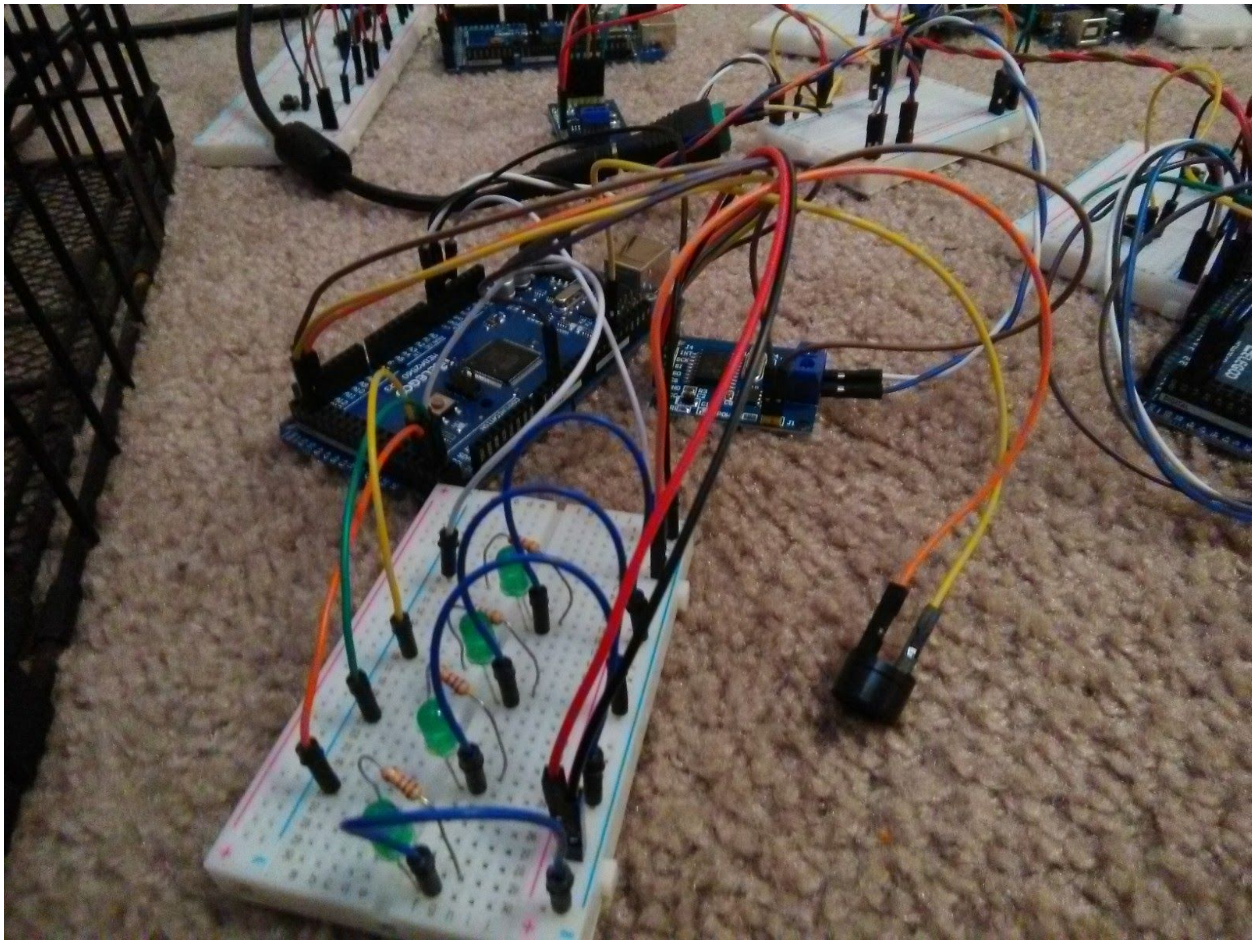
Node 0: This node consists of the actuators for the side mirror control. There are two servo motors and two stepper motors. The stepper motors are controlled using two ULN2003 Stepper Motor Drivers. Each servo motor is paired with a stepper motor to control the position of the mirror along two axes. The motor mounts are constructed using craft wood and hot glue.



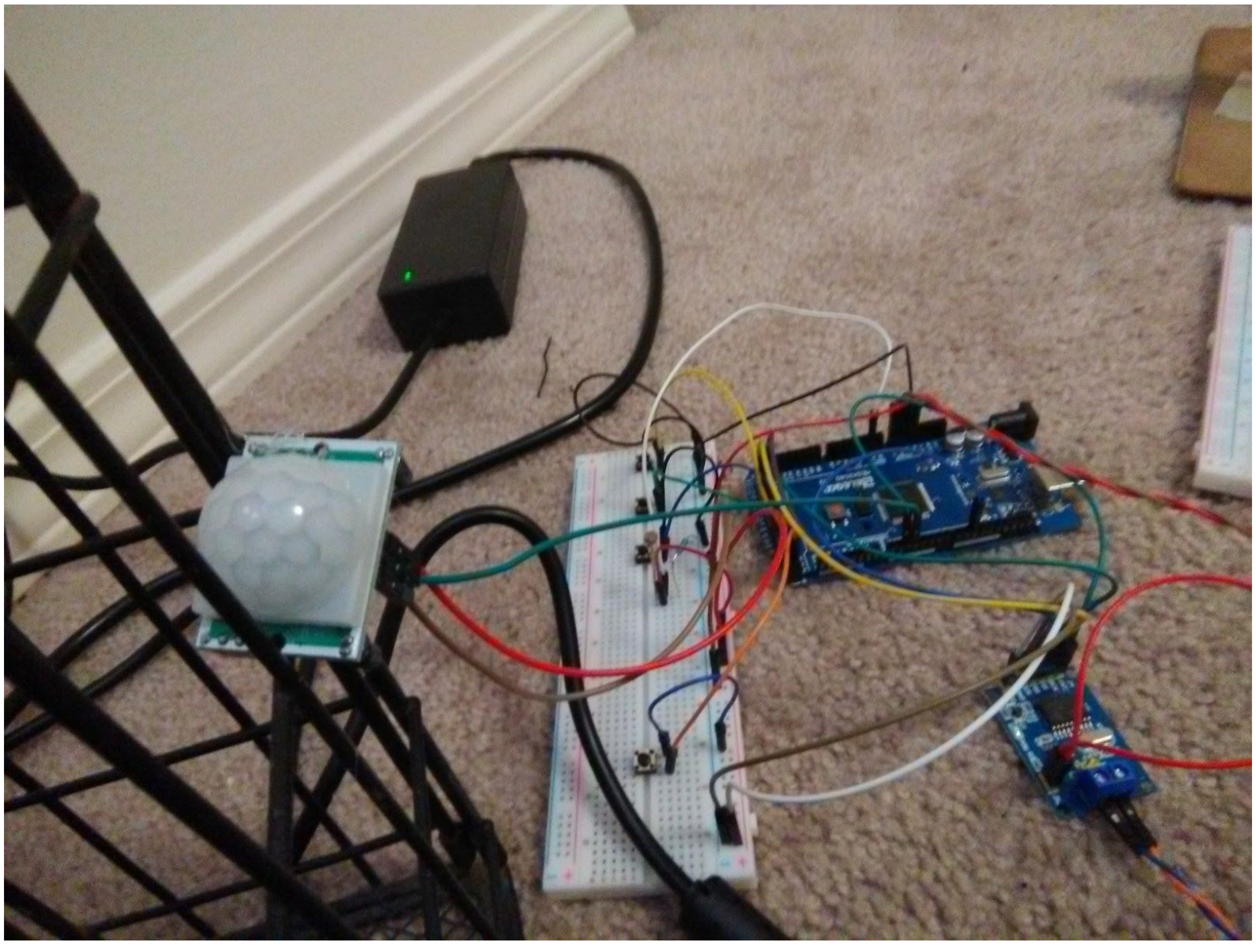
Node 1: This node consist of two push buttons and one analog joystick. The push buttons are used to select the left and right mirrors.



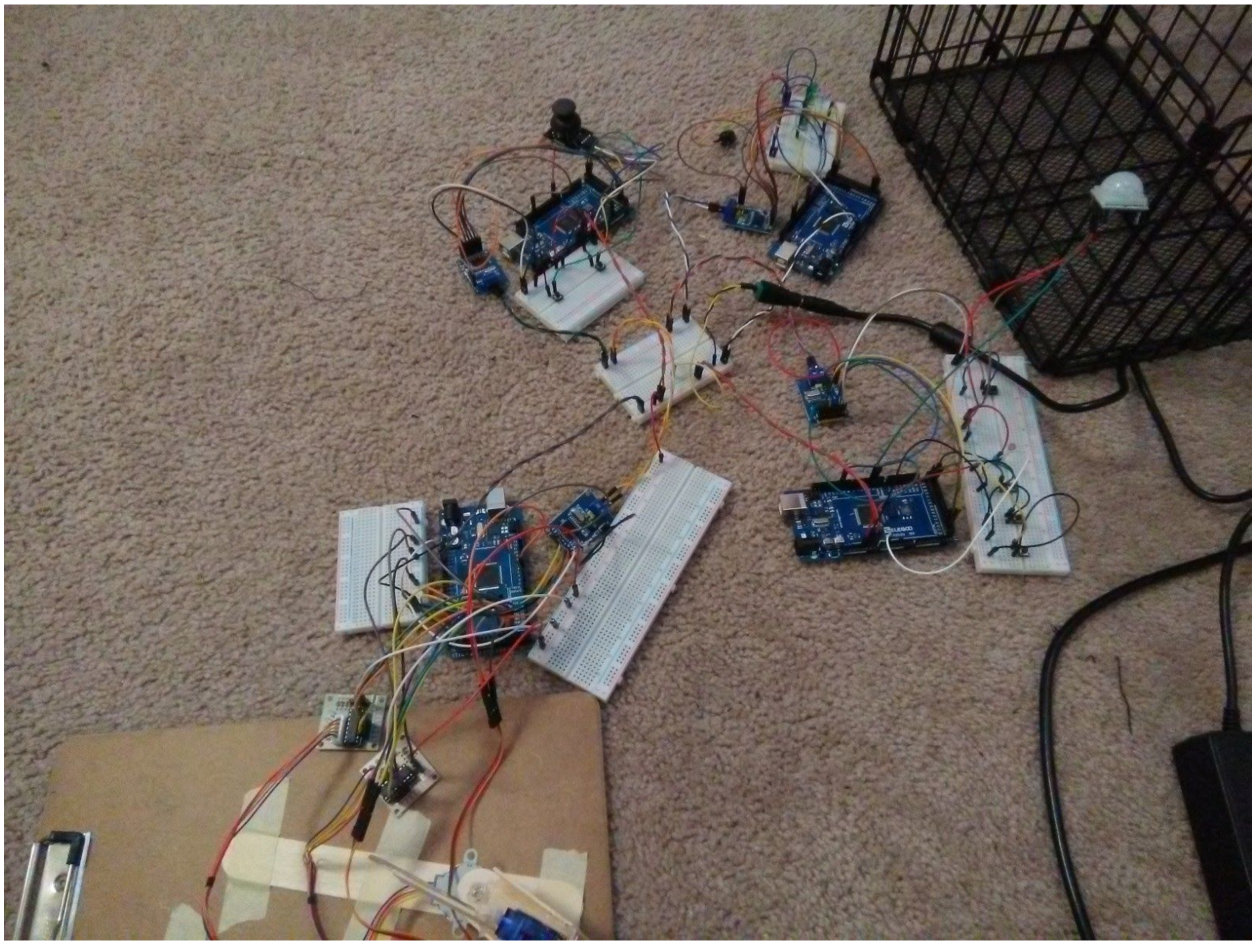
Node 2: This node has four LEDs with 1KΩ resistors to create the headlights. Two LEDs are used for the automatic setting and all four LEDs are used for the bright headlights setting. This node also has a passive buzzer to create the car alarm. The alarm oscillates between a 500 Hz signal for half a second, and a 250 Hz signal for half a second.



Node 3: This node has four push buttons. One button is used to arm and disarm the vehicle's security. The other three buttons are used for the headlights control. One turns the lights on and off, another turns the automatic setting on and off, and the last turns the bright headlights on and off. This node has a photoresistor in a voltage divider with a 1KΩ resistor to sense ambient light. There is also a HC-SR501 PIR motion sensor used to trigger the alarm on Node 2.



CAN Bus: All four nodes meet at a single breadboard that serves as the CAN bus. The breadboard has two sets of power and ground rails. One power rail is used as the CAN High data line and one ground rail is used as the CAN Low data line. Each node has a MCP2515 CAN controller connected to the ATmega processor using SPI protocol. These CAN controllers connect directly to the CAN High and Low lines. The other power and ground rails are connected to a 5V external power supply. The arduinos share a common ground on this rail and take input voltage from the power supply. The motors on Node 0 also take 5V directly from the power supply.



Assessment

During testing, we found that both the security and automatic headlights systems were functioning correctly. We found an issue with the mirror control system during the initial test. After adjusting one mirror and moving to another, the servos would begin twitching uncontrollably and interfere with the operation of the other motors. We solved this problem by powering each of the motors directly from the power supply instead of the 5V from the Arduino board. After fixing this issue, the system worked well without any issues. The motors now function reliably and respond well to the inputs from the analog joystick.

Conclusions

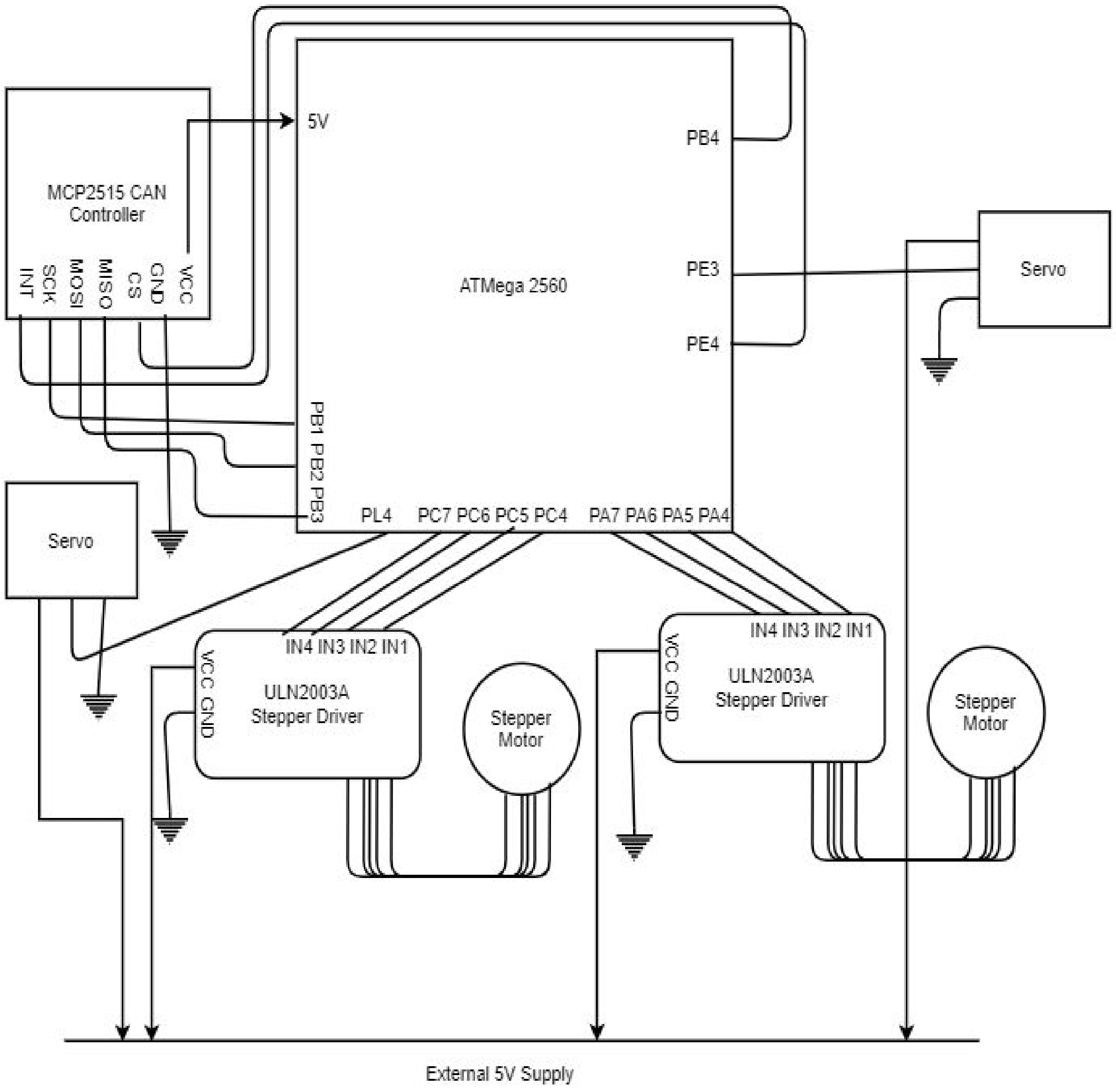
Through this project, we were able to get a deeper understanding of how a controller area network works, and gain an understanding of the procedures used to design a system on a CAN bus. We were also able to provide proof of the TTUCAN protocol software's functionality, and its ability to be used in hardware applications. We have found that the software functions as desired with a small system such as this one. In order to create a more complex system, it may not be viable to construct the bus using a single breadboard. This software should be used with standard CAN wiring to create larger systems with more nodes for further testing.

References

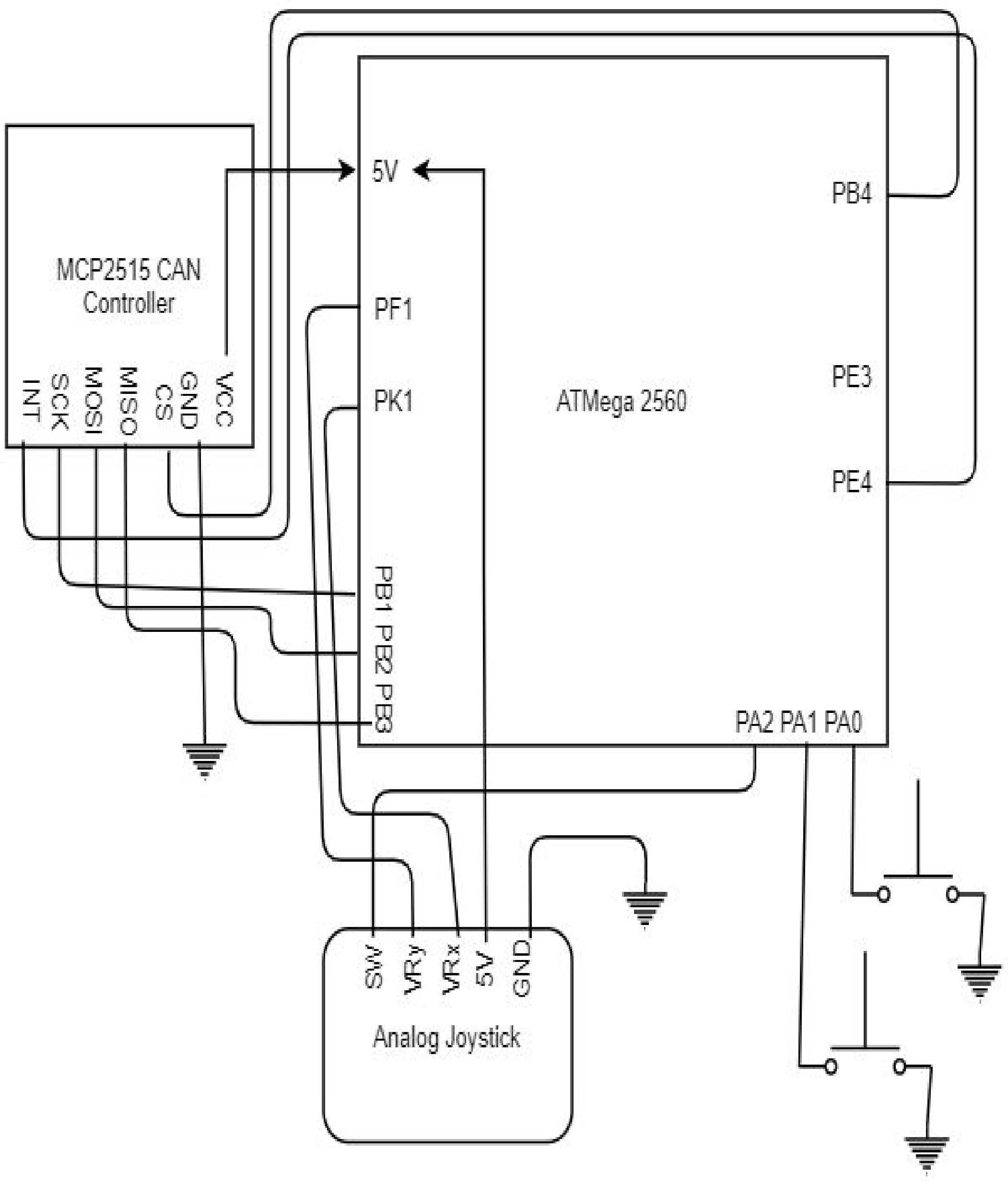
TTUCAN Software Library: https://github.com/saidm5797/TTUCAN

Appendix 1: Circuit Diagrams

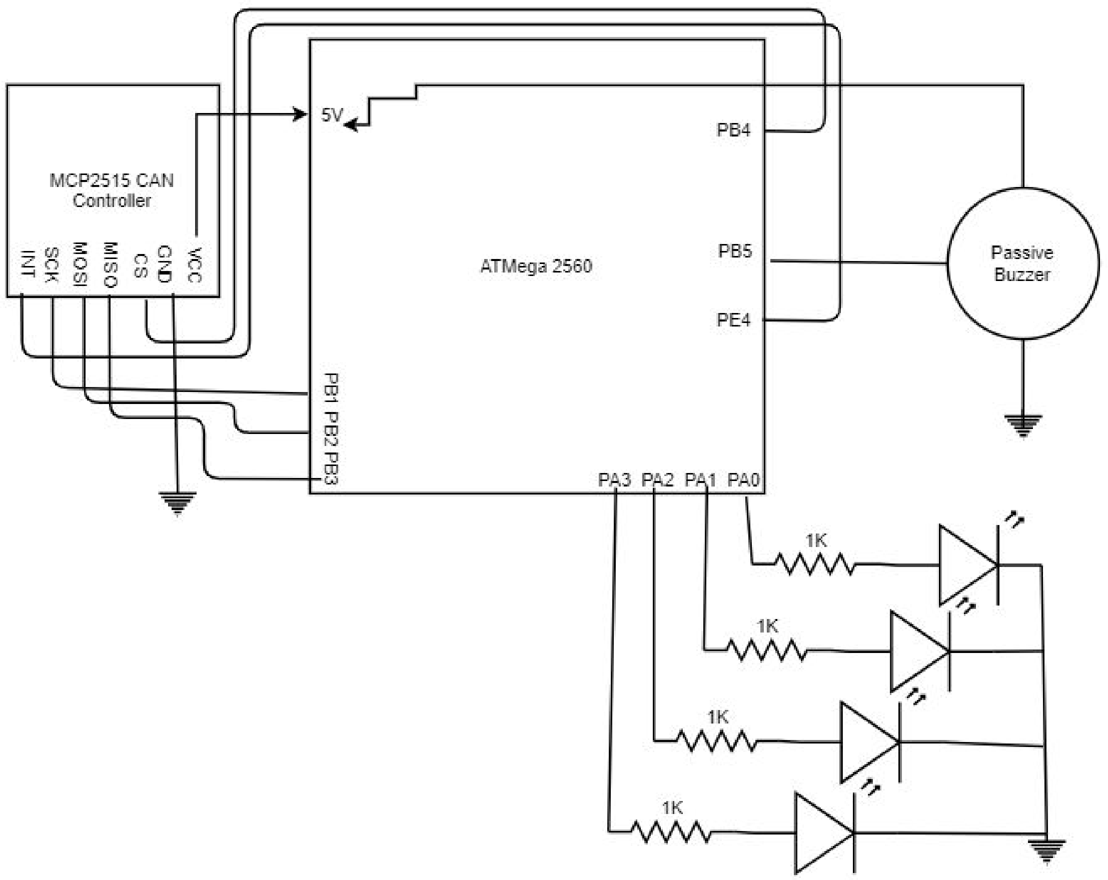
Node 0:



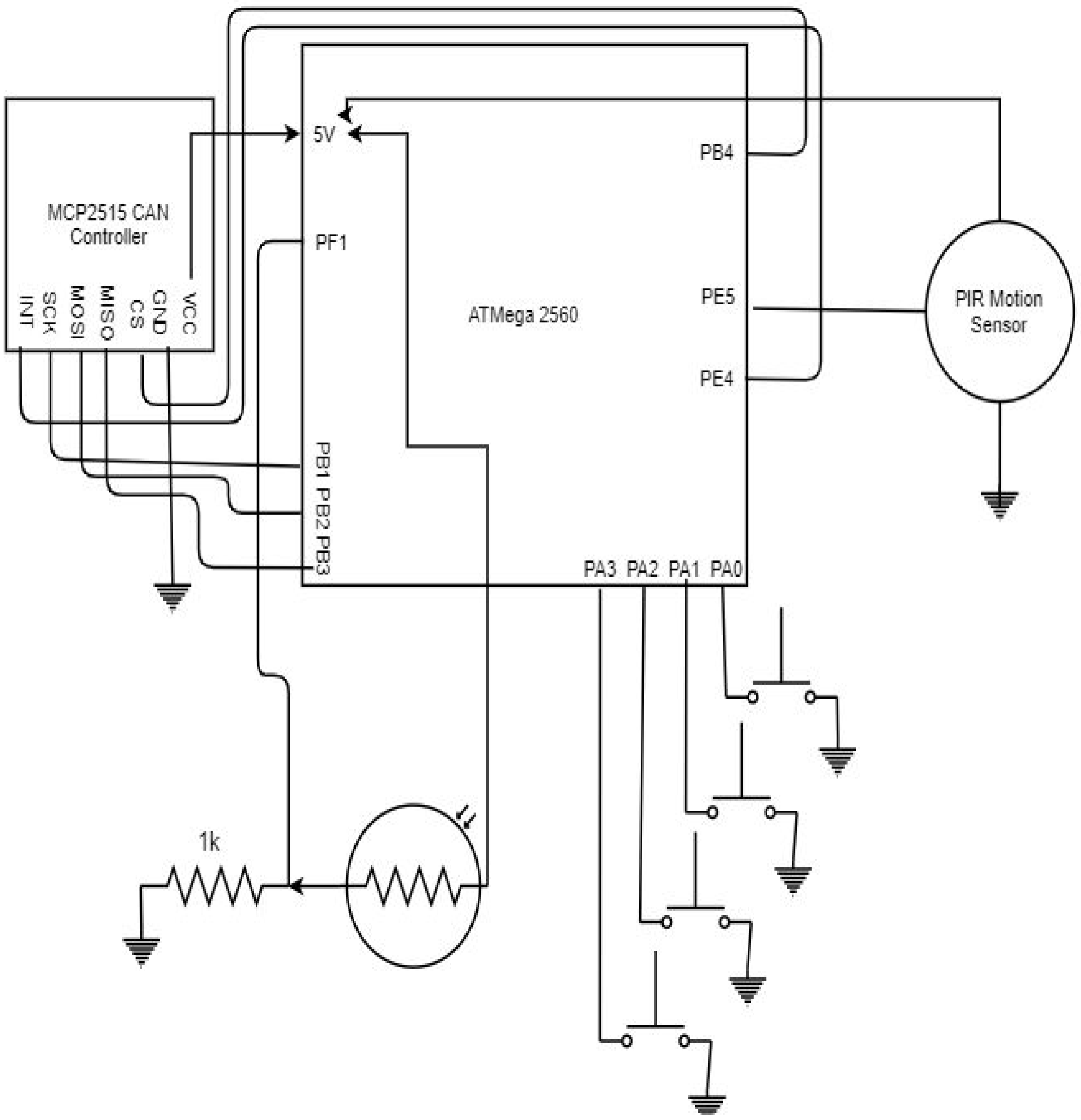
Node 1:


MCP2515 CAN Controller
INT
SCK
MOSI
MISO
CS
GND
VCC
5V
PF1
PK1
PB1 PB2 PB3
ATMega 2560
PB4
PE3
PE4
PA2 PA1 PA0
SW
VRy
VRx
5V
GND
Analog Joystick

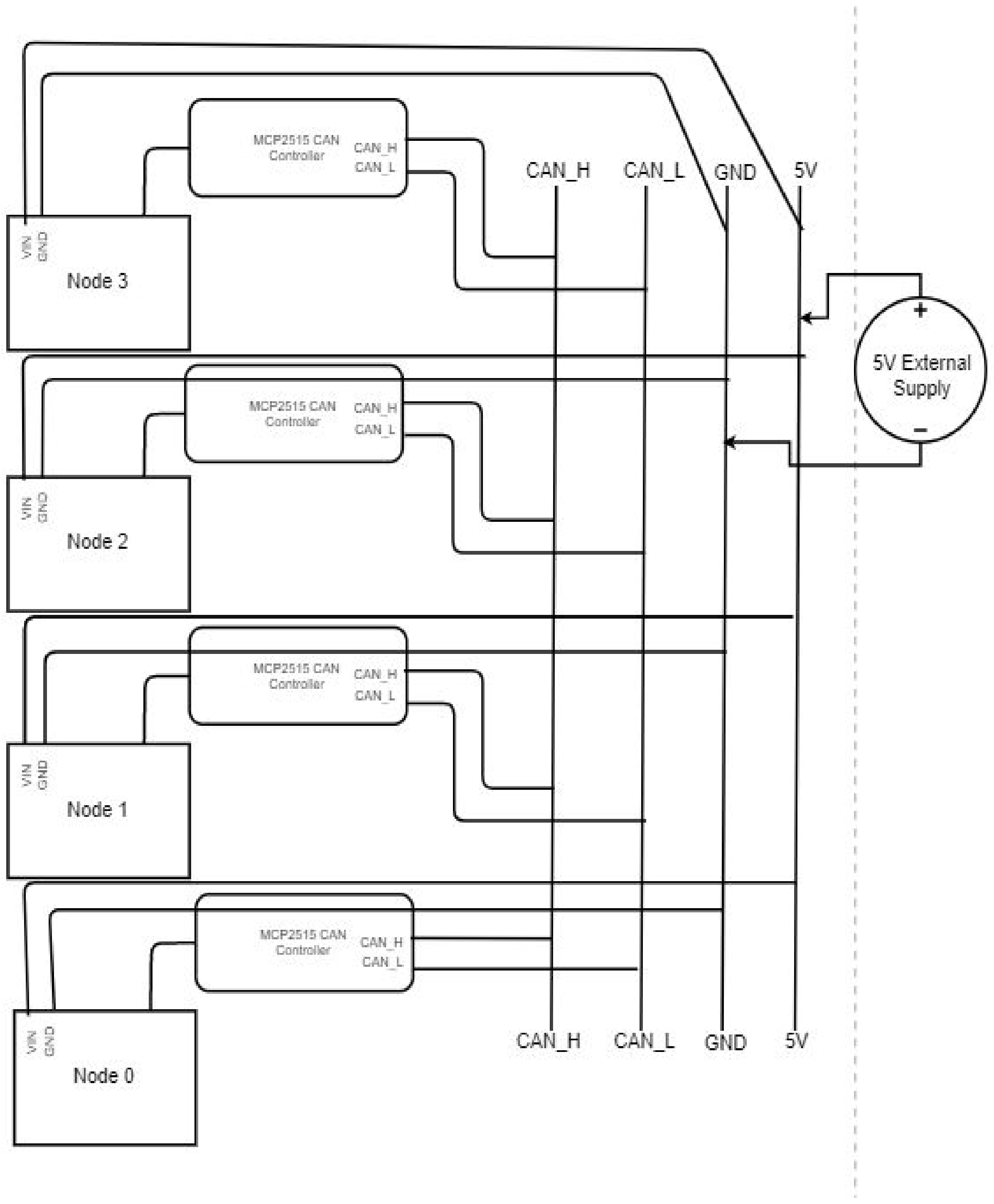
Node 2:


MCP2515 CAN Controller
INT
SCK
MOSI
MISO
CS
GND
VCC
5V
ATMega 2560
PB4
PB5
PE4
PB1
PB2
PB3
PA3
PA2
PA1
PA0
Passive Buzzer
1K
1K
1K
1K

Node 3:


MCP2515 CAN Controller
INT
SCK
MOSI
MISO
CS
GND
VCC
5V
PF1
ATMega 2560
PB4
PE5
PE4
PB1 PB2 PB3
PA3 PA2 PA1 PA0
PIR Motion Sensor
1k

CAN Bus:


MCP2515 CAN Controller
CAN_H
CAN_L
CAN_H
CAN_L
GND
5V
VIN
GND
Node 3
MCP2515 CAN Controller
CAN_H
CAN_L
5V External Supply
+
–
VIN
GND
Node 2
MCP2515 CAN Controller
CAN_H
CAN_L
VIN
GND
Node 1
MCP2515 CAN Controller
CAN_H
CAN_L
VIN
GND
Node 0
CAN_H
CAN_L
GND
5V

Appendix 2: Program Listing

Node 0:

```
// Actuators for mirror position control
#include <mcp_can.h>
#include <TTU_IsoTp.h>
#include <SPI.h>
#include <TTUCAN.h>


long unsigned int rxId, txId;
unsigned char len = 0;
unsigned char rxBuf[8];
char msgString[128];                    // Array to store serial string
byte data[8];


//define variables
int on_off = 0, left = 0, right = 0;
int move_x = 0, move_y = 0;
//variables for stepper motor control
int sequence[] = {0x80, 0xC0, 0x40, 0x60, 0x20, 0x30, 0x10, 0x90, 0x00};
int _stepr=0, _stepl=0;
int countr=0, countl=0;
int rdir=1, ldir=1, prevRdir=1, prevLdir=1;
```

```
#define CAN0_INT 2                              // Set INT to pin 2

TTUCAN CAN0(10, CAN0_INT, 0, 0);                              // Set CS to pin 10


void leftServo();

void rightServo();

void leftStepper();

void rightStepper();

void postProcess(byte *data, INT32U receiveID); //update variables from data array


void setup()

{

  Serial.begin(115200);

  DDRE &= 0xEF; //set PE4 as input for CAN interrupt

  DDRA = 0xF0; //left stepper

  DDRC = 0xF0; //right stepper

  //use OC3A - PE3 as pwm pin left servo output

  DDRE = 0b00001000;

  //use OC5B - PL4 as pwm pin right servo output

  DDRL = 0b00010000;

  //fast pwm 10 bit

  TCCR3A = 0b10000011; //OCR3A

  TCCR5A = 0b00100011; //OCR5B

  //prescale of 256; 60 cnts = 1ms

  TCCR3B = 0b00001100; //OCR3A

  TCCR5B = 0b00001100; //OCR5B
  //set initial servo positions
  OCR3A = 105; //move_y 6 steps, low is 160, high is 50, increment 18
  OCR5B = 95; //move_y 6 steps, low is 160, high is 30, increment 21
  delay(1500);

  CAN0.TTU_begin();
}

void loop()
{
  if(!(PORTE | 0xEF))                    // If CAN0_INT pin is low, read receive buffer
  {
    CAN0.receive_Msg(&rxId, &len, rxBuf);      // Read data: len = data length, buf = data byte(s)
    Serial.println();
    postProcess(rxBuf, rxId); //post process data to use information received
  }
  if(on_off){
    if(left){
      leftServo();
      leftStepper();
    }
    else if(right){
      rightServo();
      rightStepper();
    }
  }
}

void postProcess(byte *data, INT32U receiveID){
  //each case is for a different ID
  //users should plan ahead and know the contents of each message
  //in order to process information correctly
  int _size=0;
  switch(receiveID){
    case 0x008: //sent from node 3 to node 1
      memcpy(&on_off, data, sizeof(on_off));
      _size += sizeof(on_off);
      memcpy(&right, (data +_size), sizeof(right));
      _size += sizeof(right); //starting index of data array
      memcpy(&left, (data +_size), sizeof(left));
      Serial.print("Power ");
      Serial.println(on_off);
      Serial.print("right ");
      Serial.println(right);
      Serial.print("left ");
      Serial.println(left);
      break;
    case 0x009: //sent from node 2 to node 3 (use addFilter() to be able to see messages sent to another
node)
```

```
      memcpy(&move_x, data, sizeof(move_x));
      _size += sizeof(move_x); //starting index of data array
      memcpy(&move_y, (data +_size), sizeof(move_y));
      Serial.print("X ");
      Serial.println(move_x);
      Serial.print("Y ");
      Serial.println(move_y);
      break;
  }
}

void leftServo(){
  //servo needs value between 50-160 for high time of pwm signal
  int increment = 18;
  if(move_y == 1){
    if(OCR3A > 51){
      OCR3A -= increment;
    }
  }
  else if(move_y == -1){
    if(OCR3A < 159){
      OCR3A += increment;
    }
  }
  move_y=0; //reset
```

```
}

void rightServo(){
  //servo needs value between 30-160 for high time of pwm signal
  int increment = 21;
  if(move_y == 1){
    if(OCR5B > 32){
      OCR5B -= increment;
    }
  }
  else if(move_y == -1){
    if(OCR5B < 158){
      OCR5B += increment;
    }
  }
  move_y=0; //reset
}

void leftStepper(){
 int increment = 375; //6 steps
 if(move_x != 0){
  if(move_x == 1){
    ldir = 1;
    if(prevLdir == 0){
      countl = 2250-countl;
      prevLdir = 1;
    }
  }
  else{
    ldir = 0;
    if(prevLdir == 1){
      countl = 2250-countl;
      prevLdir = 0;
    }
  }
  if(countl < 2250){
   countl+=increment;
   for(int i=0;i<increment;i++){
    PORTA = sequence[_stepl];
    if(ldir){
     _stepl++;
     }else{
     _stepl--;
     }
     if(_stepl>7){
     _stepl=0;
     }
     if(_stepl<0){
     _stepl=7;
     }
```

```
    delay(1);
   }
  }
 }
 move_x=0; //reset
}

void rightStepper(){
 int increment = 375; //6 steps
 if(move_x != 0){
  if(move_x == 1){
    rdir = 1;
    if(prevRdir == 0){
      countr = 2250-countr;
      prevRdir = 1;
    }
  }
  else{
    rdir = 0;
    if(prevRdir == 1){
      countr = 2250-countr;
      prevRdir = 0;
    }
  }
  if(countr < 2250){
   countr+=increment;
   for(int i=0;i<increment;i++){
    PORTC = sequence[_stepr];
    if(rdir){
     _stepr++;
     }else{
     _stepr--;
     }
     if(_stepr>7){
     _stepr=0;
     }
     if(_stepr<0){
     _stepr=7;
     }
    delay(1);
   }
  }
 }
 move_x=0; //reset
}
```

Node 1:

```
// Control interface for mirror position control

#include <mcp_can.h>

#include <TTU_IsoTp.h>

#include <SPI.h>

#include <TTUCAN.h>


long unsigned int txId;

byte data[8];


//define variables

int on_off = 0, left = 0, right = 0;

int move_x = 0, move_y = 0;


#define CAN0_INT 2                              // Set INT to pin 2

TTUCAN CAN0(10, CAN0_INT, 1, 0);                              // Set CS to pin 10


INT32U preProcess(byte *data, INT32U to_addr, INT32U descriptor); //create data array from variables


void setup()

{

  Serial.begin(115200);

  DDRE &= 0xEF; //set PE4 as input for CAN interrupt
  DDRA = 0xF8; //A0/A1 are inputs for switches
  PORTA = 0x07; //pull up resistors on A0/A1

  // Set MUX 4-0 as 00001 for ADC1 (and ADC9)
  ADMUX = 0b01000001; //ADC1, 5V VCC as AREF
  // Enable ADC, 16 prescale
  ADCSRA = 0b10010100;
  ADCSRB = 0b00000000; // for ADC1

  CAN0.TTU_begin();
}

void loop()
{
  //Define Variables
  int left_mask = 0xFE;
  int right_mask = 0xFD;
  int onOff_mask = 0xFB;
  int value_ADC1 = 0, value_ADC9 = 0;
  int prev_x=0, prev_y=0;
  float volts_x =0.0, volts_y = 0.0;

  while(1){
    if((PINA | left_mask) == left_mask){    //adjust left mirror
      delay(250); //wait 250ms for debouncing
      Serial.print("left ");
      if(on_off){  //only update if the device is on
        left = 1;
        right = 0;
      }
      Serial.println(left);
    }
    if((PINA | right_mask) == right_mask){   //adjust right mirror
      delay(250); //wait 250ms for debouncing
      Serial.print("right ");
      if(on_off){  //only update if the device is on
        left = 0;
        right = 1;
      }
      Serial.println(right);
    }
    if((PINA | onOff_mask) == onOff_mask){   //power on and off
      delay(250); //wait 250ms for debouncing
      Serial.print("Power ");
      on_off ^= 0x01; //toggle power
      Serial.println(on_off);
      if(!on_off){
        left = 0;
        right = 0;
      }
```

```
}
if(on_off){
 // read channel ADC1
 ADCSRB = 0b00000000; // set channel to ADC1
 ADCSRA |= 0b01000000; // start ADC conversion
 while(! (ADCSRA & 0b00010000)); //wait until conversion ends
 value_ADC1 = ADC;
 volts_x = float (value_ADC1)*5.0/1023.0;
 Serial.print("X axis");Serial.print("\t");Serial.println(volts_x);

 // read channel ADC9
 ADCSRB = 0b00001000; // set channel to ADC8
 ADCSRA |= 0b01000000; // starting Adc Conversions
 while(! (ADCSRA & 0b00010000)); //wait until conversion ends
 value_ADC9 = ADC;
 volts_y = float (value_ADC9)*5.0/1023.0;
 Serial.print("y axis");Serial.print("\t");Serial.println(volts_y);
 if(volts_x < 1){
  move_x = 1;
 }
 else if(volts_x > 4){
  move_x = -1;
 }
 else{
  move_x = 0;
  }
  if(volts_y < 1){
    move_y = 1;
  }
  else if(volts_y > 4){
    move_y = -1;
  }
  else{
    move_y = 0;
  }
  Serial.print("x ");
  Serial.println(move_x);
  Serial.print("y ");
  Serial.println(move_y);
  if((move_x != prev_x) || (move_y != prev_y)){
    Serial.println("Sent");
    prev_x = move_x;
    prev_y = move_y;
    txId = preProcess(data, 0, 1); //get ID for motion control info message
    CAN0.send_Msg(txId, 0, data, sizeof(data)); //send motion control info
    delay(250);
  }
}
txId = preProcess(data, 0, 0); //get ID for on_off status message
CAN0.send_Msg(txId, 0, data, sizeof(data)); //send on_off status
```

```
 }
}

INT32U preProcess(byte *data, INT32U to_addr, INT32U descriptor){
  //each case is for a different ID
  //users should plan ahead and know the contents of each message
  //in order to process information correctly
  INT32U transmitID = CAN0.buildTransmitID(to_addr, descriptor);
  int _size=0;

  switch(transmitID){
    case 0x008: //sent from node 1 to node 0
      //reset data array to all zeros
      memset(data,0,sizeof(data));
      //begin filling data array
      memcpy(data, (byte *)&on_off, sizeof(on_off));
      _size += sizeof(on_off);
      memcpy((data+_size), (byte *)&right, sizeof(right));
      _size += sizeof(right);
      memcpy((data+_size), (byte *)&left, sizeof(left));
      break;
    case 0x009: //sent from node 1 to node 0
      //reset data array to all zeros
      memset(data,0,sizeof(data));
      //begin filling data array
```

```
     memcpy(data, (byte *)&move_x, sizeof(move_x));
     _size += sizeof(move_x);
     memcpy((data+_size), (byte *)&move_y, sizeof(move_y));
     break;
  }
  return transmitID;
}
```

Node 2:

```
// Actuators for car alarm and automatic headlights

#include <mcp_can.h>

#include <TTU_IsoTp.h>

#include <SPI.h>

#include <TTUCAN.h>


long unsigned int rxId, txId;

unsigned char len = 0;

unsigned char rxBuf[8];

char msgString[128];                    // Array to store serial string

byte data[8];


//define variables

int alarm, headlight_on, auto_on, brights_on;

int rti_count = 0, cycles = 0, Hzsignal = 1;

float volts;



#define CAN0_INT 2                          // Set INT to pin 2

TTUCAN CAN0(10, CAN0_INT, 2, 0);                          // Set CS to pin 10


void postProcess(byte *data, INT32U receiveID); //update variables from data array
```

```
ISR(TIMER1_COMPA_vect){
 OCR1A += 16384;
 rti_count++;
 if(Hzsignal == 1){
  PORTB ^= 0b00100000;   //toggle PB5
  if(rti_count == 2){
    cycles++;              //count how many cycles to control duration
    rti_count=0;
  }
  if(cycles == 250){       //500Hz signal for 0.5 sec
     cycles=0;
     Hzsignal=0; //change to 250Hz signal
     rti_count=0;
  }
 }
 else{
   if(rti_count==2 || rti_count==4){
    PORTB ^= 0b00100000;   //toggle PB5
   }
   if(rti_count==4){
    cycles++;             //count how many cycles to control duration
    rti_count=0;
   }
   if(cycles == 125){       //250Hz signal for 0.5 sec
     cycles=0;
```

```
        Hzsignal=1; //change to 500Hz signal
        rti_count=0;
      }
    }
}

void setup()
{
  Serial.begin(115200);
  DDRE &= 0xEF; //set PE4 as input for CAN interrupt
  DDRA = 0x0F; //LEDs output
  PORTA = 0x00;
  cli(); //clear interrupts
  //use OC1A - PE3 as pwm pin output for buzzer
  DDRB |= 0x20; //PB5 as output for buzzer signal
  //output compare match interrupt on OC1A
  TCCR1A = 0b00000000;
  //prescale of 1;
  TCCR1B = 0b00000001;
  OCR1A = 16384; //interrupt once each millisecond
  sei(); //enable interrupts
  CAN0.TTU_begin();
}

void loop()
```

```
{

 if(!(PORTE | 0xEF))                    // If CAN0_INT pin is low, read receive buffer
 {
   CAN0.receive_Msg(&rxId, &len, rxBuf);      // Read data: len = data length, buf = data byte(s)

   if((rxId & 0x80000000) == 0x80000000)     // Determine if ID is standard (11 bits) or extended (29
bits)
     sprintf(msgString, "Extended ID: 0x%.8lX  DLC: %1d  Data:", (rxId & 0x1FFFFFFF), len);
   else
     sprintf(msgString, "Standard ID: 0x%.3lX       DLC: %1d  Data:", rxId, len);

   Serial.print(msgString);

   if((rxId & 0x40000000) == 0x40000000){    // Determine if message is a remote request frame.
     sprintf(msgString, " REMOTE REQUEST FRAME");
     Serial.print(msgString);
   } else {
     for(byte i = 0; i<len; i++){
       sprintf(msgString, " 0x%.2X", rxBuf[i]);
       Serial.print(msgString);
     }
   }
   Serial.println();
   postProcess(rxBuf, rxId); //post process data to use information received
```

```
}
if(alarm){
  TIMSK1 = 0x02; //enable output compare match interrupt
}
else{
   TIMSK1 = 0x00; //enable output compare match interrupt
}
if(headlight_on){
  if(auto_on){
    if(volts < 0.30){
      PORTA = 0x0A; //lights on
    }
    else{
      PORTA = 0x00; //lights off
    }
  }
  else if(brights_on){
    PORTA = 0x0F; //all lights on
  }
  else{
    PORTA = 0x0A; //lights on
  }
}
else{
  PORTA = 0x00; //lights off
```

```
 }
}


void postProcess(byte *data, INT32U receiveID){
  //each case is for a different ID
  //users should plan ahead and know the contents of each message
  //in order to process information correctly
  int _size;
  switch(receiveID){
    case 0x118: //sent from node 3 to node 2
      memcpy(&alarm, data, sizeof(alarm));
      _size = sizeof(alarm); //starting index of data array
      memcpy(&headlight_on, (data +_size), sizeof(headlight_on));
      Serial.print("alarm ");
      Serial.println(alarm);
      Serial.print("headlight ");
      Serial.println(headlight_on);
      break;
    case 0x119: //sent from node 3 to node 2
      memcpy(&volts, data, sizeof(volts));
      _size = sizeof(volts); //starting index of data array
      memcpy(&auto_on, (data +_size), sizeof(auto_on));
      _size +=sizeof(auto_on); //increment index of array by size of count to access the next variable
      memcpy(&brights_on, (data +_size), sizeof(brights_on));
      break;
  }
}
```

Node 3:

```
// Control interface for car alarm and automatic headlights

#include <mcp_can.h>

#include <TTU_IsoTp.h>

#include <SPI.h>

#include <TTUCAN.h>


long unsigned int txId;

byte data[8];


//define variables

int alarm = 0, headlight_on = 0, auto_on = 0, brights_on = 0;

float volts = 0.0;


#define CAN0_INT 2                              // Set INT to pin 2

TTUCAN CAN0(10, CAN0_INT, 3, 0);                              // Set CS to pin 10


INT32U preProcess(byte *data, INT32U to_addr, INT32U descriptor); //create data array from variables


ISR(INT5_vect){

  Serial.println("Alarm on!");

  alarm = 1;

  txId = preProcess(data, 2, 0); //get ID for on_off status message

  CAN0.send_Msg(txId, 0, data, sizeof(data)); //send on_off status
```

```
}

void setup()
{
  Serial.begin(115200);
  DDRE &= 0xCF; //set PE4/PE5 as inputs
  CAN0.TTU_begin();

  cli(); //clear interupts
  EICRB = 0x0C; //trigger on rising edge at PE5
  sei(); //enable interrupts

  DDRA = 0xF0; //A0-A3 are inputs for switches
  PORTA = 0x0F; //pull up resistors on A0-A3

  // Set MUX 4-0 as 00001 for ADC1 (and ADC9)
  ADMUX = 0b01000001; //ADC1, 5V VCC as AREF
  // Enable ADC, 16 prescale
  ADCSRA = 0b10010100;
  ADCSRB = 0b00000000; // for ADC1
}

void loop()
{
  //Define Variables
```

```
int armed = 0;
int armed_mask = 0xFE;
int headlight_mask = 0xFD;
int auto_mask = 0xFB;
int brights_mask = 0xF7;

int value_ADC1 = 0;

while(1){
  if((PINA | armed_mask) == armed_mask){    //adjust left mirror
    delay(250); //wait 250ms for debouncing
    Serial.print("armed ");
    armed ^= 0x01; //toggle armed status
    Serial.println(armed);
    if(armed){
      //enable external interrupt on PE5
      EIMSK = 0x20;
    }
    else{
      //disable external interrupt on PE5
      EIMSK = 0x00;
      alarm = 0;
      Serial.println("Alarm off.");
    }
  }
```

```
if((PINA | headlight_mask) == headlight_mask){   //adjust right mirror
  delay(250); //wait 250ms for debouncing
  Serial.print("headlights ");
  headlight_on ^= 0x01; //toggle headlights
  Serial.println(headlight_on);
  if(!headlight_on){
    auto_on = 0;
    brights_on = 0;
  }
}
if((PINA | auto_mask) == auto_mask){   //adjust right mirror
  delay(250); //wait 250ms for debouncing
  Serial.print("auto ");
  if(headlight_on){
    auto_on ^= 0x01; //toggle auto mode
    brights_on = 0;
  }
  Serial.println(auto_on);
}
if((PINA | brights_mask) == brights_mask){   //adjust right mirror
  delay(250); //wait 250ms for debouncing
  Serial.print("brights ");
  if(headlight_on){
    brights_on ^= 0x01; //toggle brights mode
    auto_on = 0;
```

```
      }
      Serial.println(brights_on);
    }
    if(auto_on){
      // read channel ADC1
      ADCSRA |= 0b01000000; // start ADC conversion
      while(! (ADCSRA & 0b00010000)); //wait until conversion ends
      value_ADC1 = ADC;
      volts = float (value_ADC1)*5.0/1023.0;
      Serial.print("voltage");Serial.print("\t");Serial.println(volts);
    }
    delay(100);
    txId = preProcess(data, 2, 0); //get ID for on_off status message
    CAN0.send_Msg(txId, 0, data, sizeof(data)); //send on_off status
    delay(100);
    txId = preProcess(data, 2, 1); //get ID for motion control info message
    CAN0.send_Msg(txId, 0, data, sizeof(data)); //send motion control info
  }
}


INT32U preProcess(byte *data, INT32U to_addr, INT32U descriptor){
  //each case is for a different ID
  //users should plan ahead and know the contents of each message
  //in order to process information correctly
  INT32U transmitID = CAN0.buildTransmitID(to_addr, descriptor);
  int _size;

  switch(transmitID){
    case 0x118: //sent from node 3 to node 2
      //reset data array to all zeros
      memset(data,0,sizeof(data));
      //begin filling data array
      memcpy(data, (byte *)&alarm, sizeof(alarm));
      _size = sizeof(alarm);
      memcpy((data+_size), (byte *)&headlight_on, sizeof(headlight_on));
      break;
    case 0x119: //sent from node 3 to node 2
      //reset data array to all zeros
      memset(data,0,sizeof(data));
      //begin filling data array
      memcpy(data, (byte *)&volts, sizeof(volts));
      _size = sizeof(volts);
      memcpy((data+_size), (byte *)&auto_on, sizeof(auto_on));
      _size += sizeof(auto_on);
      memcpy((data+_size), (byte *)&brights_on, sizeof(brights_on));
      break;
  }
  return transmitID;
}
```