Appendix C: Final Code

Contents:

1. Vision System

```
/*********************************Pixy/SPI Functions**********************/
void initializeSPI (void){
float WREN=0x06; //avoiding global variables
float WRITE=0x02;

       unsigned char Info;
    // Initialize SPI _ low level
    IEC1CLR=0x03800000;//disable SPI interrupts in IECx register
    SPI2CON=0;//stop and reset SPI module by clearing ON bit
    Info=SPI2BUF; //clear the receive buffer
    //SPI3CONbits.ENHBUF=0; //clear ENHBUF bit
    SPI2BRG=0;//write the baud rate register
    SPI2STATbits.SPIROV=0;//clear SPIROV bit
    SPI2CONbits.CKE=0;
    SPI2CONbits.CKP=1;
    SPI2CONbits.SMP=1;
    SPI2CONbits.MSTEN=1;//write desired settings to SPIxCON
    SPI2CONbits.ON=1;//enable SPI

    DDPCONbits.JTAGEN=0; //disable JTAG
    AD1PCFG = 0xFFFF;
    TRISGbits.TRISG7 = 0;
    //AD1PCFGbits.PCFG8=1; //enables pins as digital
    //AD1PCFGbits.PCFG10=1;
    //AD1PCFGbits.PCFG11=1;
    //AD1PCFGbits.PCFG14=1;


    // write enable Pixy
    TRISBbits.TRISB8 = 0;// Tristate Buffer to 0
    LATBbits.LATB8=0;
    SPI_SEND(WREN);//set write enable latch
    LATBbits.LATB8=1; //set high
    LATBbits.LATB8=0;//set low
    SPI_SEND(WRITE);         //write data to memory
}


unsigned char SPI_SEND (unsigned char data){
    IFS1bits.SPI2RXIF=0; //clears flag
    SPI2BUF= data;//place data to be sent in transmit buffer
    while(IFS1bits.SPI2RXIF==0); //wait for data to be shifted
        return(SPI2BUF);
    //return data received
}
```

```
float capture_pixy_data (void) {

    unsigned char X;
    float CC1, CC2, Y, Z, CN1, CN2, X1, X2, Y1, Y2, W1, W2, H1, H2, A1, A2 ;
    float dist_from_center;
    int counter = 0;

    while (counter < 6000) {

        X= SPI_SEND(0X5A);          //location in memory
        X= SPI_SEND(0X00);
        if (X == 0x55) {
            //LATBbits.LATB8=1; //set high
            //LATBbits.LATB8=0;//set low
            CC1= SPI_SEND(0X5A);                    // Color Code
            CC2= SPI_SEND(0X00);
            if (CC2 == 0x56) {              // check for 55 twice, not needed for CC
                Y= SPI_SEND(0X5A);                  // ????
                Z= SPI_SEND(0X00);
                CN1= SPI_SEND(0X5A);                // Channel Number
                CN2= SPI_SEND(0X00);
                X1= SPI_SEND(0X5A);                 // X Coordinate
                X2= SPI_SEND(0X00);
                Y1= SPI_SEND(0X5A);                 // Y Coordinate
                Y2= SPI_SEND(0X00);
                W1= SPI_SEND(0X5A);                 // Width
                W2= SPI_SEND(0X00);
                H1= SPI_SEND(0X5A);                 // Height
                H2= SPI_SEND(0X00);
                A1= SPI_SEND(0X5A);                 // Angle
                A2= SPI_SEND(0X00);

                if (H2 > 27) {                      // if CC is too close
                    return(7777);                   // return 7777
                }

                if (X1 == 1) {
                    X2 = X2 + 255;
                }
                dist_from_center = X2 - 160;        // right side positive values

                X = 0;// reset X
                return(dist_from_center);
            }
        }
        else //if (X == 0)
        {
            counter++;
        }
    }
    return(9000);
}
```

2. Motor Controls

```
/*********************************Motor Control Functions**********************/

void motorcontrol (float dist_from_center, float upper_bound, float lower_bound){

    LATEbits.LATE1 = 0;
    //define variable
    float left;                 // left motor OC3RS
    float right;                // right motor OC4RS


    /* CALCULATE DESIRED LEFT AND RIGHT SPEEDS */

    if (dist_from_center>0){  ///turn right
        LATEbits.LATE2 = 0;
        if (dist_from_center>80) {
            left = upper_bound;
            right = lower_bound/2;
        }
        else {
            left = (upper_bound-lower_bound)/80 * dist_from_center + lower_bound;
//converting x value in relation to upper/lower bounds
            right = (-lower_bound/2)/80 * dist_from_center + lower_bound;
        }
    }
    else if (dist_from_center<=0){  ///turn left
        LATEbits.LATE3 = 0;
        if (dist_from_center<-80) {
            right = upper_bound;
            left = lower_bound/2;
        }
        else {
            right= (upper_bound-lower_bound)/80 *(-dist_from_center)+lower_bound;
//converting x value in relation to upper/lower bounds
            left = (-lower_bound/2)/80 * (-dist_from_center) + lower_bound;
        }
    }

    /* IF SPEED IS BELOW THE LOWER BOUND, RAMP IT UP TO THE LOWER BOUND */

    if (left_speed == 0 || right_speed == 0) { // if either motor is below lower
bound, ramp up

        float D1, D2, D3;
        D1 = capture_pixy_data();
        D2 = capture_pixy_data();
        D3 = capture_pixy_data();
        if  ((D1 == 7777 || D2 == 7777) || D3 == 7777) {        // Check to ensure the
CC isn't too close
            left = 0;                                           // if too close, get
out
            right = 0;
        }
```

```
        LATEbits.LATE4 = 0;
        while (left_speed < lower_bound/2) {
            left_speed++;
            right_speed++;
            OC3RS = left_speed;
            OC4RS = right_speed;
            delay_us(600);
        }
    }

    /* ADJUST SPEEDS */

    while (left < left_speed) {
        left_speed--;
        OC3RS = left_speed;
        delay_us(600);
    }
    while (right < right_speed) {
        right_speed--;
        OC4RS = right_speed;
        delay_us(600);
    }
    while (left > left_speed) {
        left_speed++;
        OC3RS = left_speed;
        delay_us(600);
    }
    while (right > right_speed) {
        right_speed++;
        OC4RS = right_speed;
        delay_us(600);
    }

    LATEbits.LATE5 = 0;

}


void ramp_down(float lower_bound){


    // To ramp down PWM signal, run this while loop

    while (right_speed > left_speed){
            right_speed --; // If the duty cycle is not at max, increase
            OC4RS = right_speed; // Write new duty cycle
            delay_us(600);
    }
    while (left_speed > right_speed){
            left_speed --; // If the duty cycle is not at max, increase
            OC3RS = left_speed; // Write new duty cycle
            delay_us(600);
    }
```

```
    while (right_speed > 0) { // Ramp up mode (Ramp up to 50%)   ensuring both will
turn off
            right_speed --; // If the duty cycle is not at max, increase
            left_speed --;
            OC4RS = right_speed; // Write new duty cycle
            OC3RS = left_speed;
            delay_us(600);
        }                               // End of ramp down

}


void PWMConfigure(void){

    TRISDbits.TRISD2 = 0; // Make RD2 an output (OC3, PWM pin) //added from, main code
    TRISDbits.TRISD3 = 0; // Make RD3 an output (OC4, PWM pin)

    LATDbits.LATD3 = 0;
    LATDbits.LATD2 = 0;

//******************* Output Compare for PWM_left/OC3 ***********************//

    OC3CON = 0x0000;           // Turn off the OC3 when performing the setup
    OC3CONbits.OCM = 6;        // Configure for PWM mode without Fault pin enabled
    OC3CONbits.OCTSEL = 0;     // Timer2 is clock source for Output Compare 1
    OC3CONbits.OC32 = 0;       // 16-bit mode (Output Compare)

//******************* Output Compare for PWM_right/OC2 **********************//

    OC4CON = 0x0000;           // Turn off the OC4 when performing the setup
    OC4CONbits.OCM = 6;        // Configure for PWM mode without Fault pin enabled
    OC4CONbits.OCTSEL = 0;     // Timer2 is clock source for Output Compare 2
    OC4CONbits.OC32 = 0;       // 16-bit mode (Output Compare)

//******************** Set up Timer2 for both OC3 & OC2 *********************//

    T2CON = 0x0000;                 // Clear Timer 3 Control bits
    TMR2 = 0x0000;
    T2CONbits.ON = 0;               // Clear ON Control bit to clear timer
    T2CONbits.T32 = 0;              // 16-bit mode (Timer 2 and Timer 3)

//************************ Initializations OC3/OC2 ***************************//

    OC3R = 0x0000;                  // Initialize primary Compare register
    OC3R = 0;                       // Give OC3R initial duty cycle
                                    // (value/160 * 100 is duty cycle in %)
    OC3RS = 0x0000;                 // Initialize secondary Compare register

    OC4R = 0x0000;                  // Initialize primary Compare register
    OC4R = 0;                       // Give OC2R initial duty cycle
                                    // (value/160 * 100 is duty cycle in %)
    OC4RS = 0x0000;                 // Initialize secondary Compare register

    T2CONbits.TCKPS0 = 1;           //
    T2CONbits.TCKPS1 = 0;           // Set Prescale on Timer2 to 1:32
```

```
    T2CONbits.TCKPS2 = 1;           //

    PR2 = 160;                      // Set PR to 160
                                    // This gives PWM signal with 1.9 kHz freq

//***********************   Turn on Timers & OCs    ***************************//

    T2CONbits.TON = 1;              // Enable time base on Timer3
    T2CONbits.ON = 1;               // Set ON Control bit high to enable timer
    OC3CONbits.ON = 1;              // Turn on OC3
    OC4CONbits.ON = 1;              // Turn on OC4

}
```

3. Bluetooth Integration

Edits to main code:

```
void setup(void);
void loop(void);

void initializeSPI(void);
unsigned char SPI_SEND (unsigned char data);
void PWMConfigure(void);
void ramp_down(float lower_bound);
float capture_pixy_data (void);
void motorcontrol (float dist_from_center, float upper_bound, float lower_bound);


int bluetooth_data = 6000;
int enable = 0;
int right_speed = 0;
int left_speed = 0;

void main(void)
{
    setup();
    PWMConfigure();
    initializeSPI();

    while(1)
        loop();
}
```

Edits to loop():

```
      // ********************** Declare Variables ************************** //
    float dist_from_center;
    float upper_bound=100;
    float lower_bound=60;
    float D1, D2, D3;
```

```
    upper_bound= upper_bound*160 /100; // Not sure if we can use float. Use int
    lower_bound= lower_bound*160 /100;

  //Process any ACI commands or events
  aci_loop();

  if (bluetooth_data == 1) {                        // BLUETOOTH ENABLED
      LATEbits.LATE0 = 0; //TURN LED 0 ON
      enable = 1;
      ramp_down(lower_bound);
  }

  else if (bluetooth_data == 0) {                   // BLUETOOTH DISABLED
      LATEbits.LATE0 = 1; //TURN LED 0 OFF
      enable = 0;
      ramp_down(lower_bound);
  }


/*******************************BLUETOOTH COMMANDS*************************/
      if (bluetooth_data == 3) {                    // LEFT MOTOR ON
        LATEbits.LATE2 = 0; //TURN LED 0 ON

        unsigned int Pwm_left = 0;       // Pwm_left controls left motor
        while (Pwm_left < lower_bound) { // Ramp up mode (Ramp up to 50%)
            Pwm_left ++;
            OC3RS = Pwm_left;
            delay_us(600);      // End of ramp up
        }
      }
      else if (bluetooth_data == 2) {               // LEFT MOTOR OFF
        LATEbits.LATE2 = 1; //TURN LED 0 OFF

        unsigned int Pwm_left = lower_bound;     // Pwm_right controls right motor
        while (Pwm_left > 0) {                          // Ramp down to zero
            Pwm_left --;
            OC3RS = Pwm_left;                           // Write new duty cycle
            delay_us(600);
        }
      }
      else if (bluetooth_data == 5) {               // BOTH MOTORS ON
        LATEbits.LATE3 = 0; //TURN LED 0 ON
        unsigned int Pwm_left = 0;      // Pwm_left controls left motor
        unsigned int Pwm_right = 0;     // Pwm_right controls right motor
        while (Pwm_right < ((upper_bound+lower_bound)/2)) { // Ramp up mode (Ramp up
to 50%)   ensuring both will turn off
            Pwm_right++; // If the duty cycle is not at max, increase
            Pwm_left++;
            OC4RS = Pwm_right; // Write new duty cycle
            OC3RS = Pwm_left;
            delay_us(600);
        }
      }
      else if (bluetooth_data == 4) {               // BOTH MOTORS OFF
        LATEbits.LATE3 = 1; //TURN LED 0 OFF
```

```
        unsigned int Pwm_left = lower_bound;      // Pwm_left controls left motor
        unsigned int Pwm_right = lower_bound;     // Pwm_right controls right motor
        while (Pwm_right > 0) { // Ramp up mode (Ramp up to 50%)   ensuring both will
turn off
            Pwm_right --; // If the duty cycle is not at max, increase
            Pwm_left --;
            OC4RS = Pwm_right; // Write new duty cycle
            OC3RS = Pwm_left;
            delay_us(600);
        }
      }
      else if (bluetooth_data == 7) {               // RIGHT MOTOR ON
        LATEbits.LATE4 = 0; //TURN LED 0 ON

        unsigned int Pwm_right = 0;       // Pwm_left controls left motor
        while (Pwm_right < lower_bound) { // Ramp up mode (Ramp up to 50%)
            Pwm_right ++;
            OC4RS = Pwm_right;
            delay_us(600);      // End of ramp up
        }
      }
      else if (bluetooth_data == 6) {               // RIGHT MOTOR OFF
        LATEbits.LATE4 = 1; //TURN LED 0 OFF

        unsigned int Pwm_right = lower_bound;     // Pwm_right controls right motor
        while (Pwm_right > 0) {                         // Ramp down to zero
            Pwm_right --;
            OC4RS = Pwm_right;                          // Write new duty cycle
            delay_us(600);
        }

  }

  /******************************** PIXY CODE ************************************/

  if (enable == 0) {
      LATEbits.LATE6 = 0; //TURN LED 0 ON

       if (PORTEbits.RE7 == 0){   // toggle switch is on
        dist_from_center = capture_pixy_data();
        if (dist_from_center == 9000) {
            ramp_down(lower_bound);
           // LATEbits.LATE0 = 1;
        }
        else if (dist_from_center == 7777) {                        // if CC too close
            D1 = capture_pixy_data();
            D2 = capture_pixy_data();
            //D3 = capture_pixy_data();
            if  ((D1 == 7777 && D2 == 7777)) {      // D3 removed
               // LATEbits.LATE0 = 0;
                ramp_down(lower_bound);
            }
        }
        else motorcontrol(dist_from_center, upper_bound, lower_bound);
```

```
   }
       else {
           LATEbits.LATE0=1;
           ramp_down (lower_bound); //necessary?
       }
  }
  else LATEbits.LATE6 = 1; //TURN LED 0 OFF
```

Edits to aci_loop():

```
bluetooth_data = 6000;

case ACI_EVT_DATA_RECEIVED:
        printf("Pipe Number: ");
        printf("%d\r\n",aci_evt->params.data_received.rx_data.pipe_number);

        if (PIPE_UART_OVER_BTLE_UART_RX_RX == aci_evt-
>params.data_received.rx_data.pipe_number)
          {
//****************************ADDED CODE****************************************//
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x30){ // A 0
                bluetooth_data = 0;
                LATEbits.LATE1 = 0;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x31){ // A 1
                bluetooth_data = 1;
                LATEbits.LATE1 = 1;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x32){ // A 2
                bluetooth_data = 2;
                LATEbits.LATE2 = 0;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x33){ // A 3
                bluetooth_data = 3;
                LATEbits.LATE2 = 1;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x34){ // A 4
                bluetooth_data = 4;
                LATEbits.LATE3 = 0;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x35){ // A 5
                bluetooth_data = 5;
                LATEbits.LATE3 = 1;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x36){ // A 6
                bluetooth_data = 6;
                LATEbits.LATE4 = 0;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x37){ // A 7
                bluetooth_data = 7;
                LATEbits.LATE4 = 1;
            }
```

4. Complete Code

```
#include <xc.h>
#include "../configbits.h"
#include "./NRFspi.h"
#include "../SDlib.h"
#include "lib_aci.h"
#include "aci_setup.h"
#include "uart_over_ble.h"
#include "services.h"

#ifdef SERVICES_PIPE_TYPE_MAPPING_CONTENT
    static services_pipe_type_mapping_t
        services_pipe_type_mapping[NUMBER_OF_PIPES] =
SERVICES_PIPE_TYPE_MAPPING_CONTENT;
#else
    #define NUMBER_OF_PIPES 0
    static services_pipe_type_mapping_t * services_pipe_type_mapping = NULL;
#endif

///* Store the setup for the nRF8001 in the flash of the AVR to save on RAM */
static hal_aci_data_t setup_msgs[NB_SETUP_MESSAGES]  = SETUP_MESSAGES_CONTENT;

// aci_struct that will contain
// total initial credits
// current credit
// current state of the aci (setup/standby/active/sleep)
// open remote pipe pending
// close remote pipe pending
// Current pipe available bitmap
// Current pipe closed bitmap
// Current connection interval, slave latency and link supervision timeout
// Current State of the the GATT client (Service Discovery)
// Status of the bond (R) Peer address
static struct aci_state_t aci_state;

/*
Temporary buffers for sending ACI commands
*/
static hal_aci_evt_t  aci_data;
//static hal_aci_data_t aci_cmd;

/*
Timing change state variable
*/
static bool timing_change_done          = false;

/*
Used to test the UART TX characteristic notification
*/
static uart_over_ble_t uart_over_ble;
static uint8_t         uart_buffer[20];
static uint8_t         uart_buffer_len = 0;
static uint8_t         dummychar = 0;
```

```
/*
Initialize the radio_ack. This is the ack received for every transmitted packet.
*/
//static bool radio_ack_pending = false;

/* Define how assert should function in the BLE library */
void __ble_assert(const char *file, uint16_t line)
{
//  Serial.print("ERROR ");
//  Serial.print(file);
//  Serial.print(": ");
//  Serial.print(line);
//  Serial.print("\n");
  while(1);
}
/*
Description:

In this template we are using the BTLE as a UART and can send and receive packets.
The maximum size of a packet is 20 bytes.
When a command it received a response(s) are transmitted back.
Since the response is done using a Notification the peer must have opened
it(subscribed to it) before any packet is transmitted.
The pipe for the UART_TX becomes available once the peer opens it.
See section 20.4.1 -> Opening a Transmit pipe
In the master control panel, clicking Enable Services will open all the pipes on the
nRF8001.

The ACI Evt Data Credit provides the radio level ack of a transmitted packet.
*/
/* protos*/
void setup(void);
void loop(void);

void initializeSPI(void);
unsigned char SPI_SEND (unsigned char data);
void PWMConfigure(void);
void ramp_down(float lower_bound);
float capture_pixy_data (void);
void motorcontrol (float dist_from_center, float upper_bound, float lower_bound);


int bluetooth_data = 6000;
int enable = 0;
int right_speed = 0;
int left_speed = 0;

void main(void)
{
    setup();
    PWMConfigure();
    initializeSPI();

    while(1)
```

```
        loop();
}



void setup (void)
{
  serial_init(57600);
  TRISEbits.TRISE0 = 0;
  TRISEbits.TRISE1 = 0;
  TRISEbits.TRISE2 = 0;
  TRISEbits.TRISE3 = 0;
  TRISEbits.TRISE4 = 0;
  TRISEbits.TRISE5 = 0;
  TRISEbits.TRISE6 = 0;

  LATEbits.LATE0 = 1;
  LATEbits.LATE1 = 1;
  LATEbits.LATE2 = 1;
  LATEbits.LATE3 = 1;
  LATEbits.LATE4 = 1;
  LATEbits.LATE5 = 1;
  LATEbits.LATE6 = 1;

//  //Wait until the serial port is available (useful only for the Leonardo)
//  //As the Leonardo board is not reseted every time you open the Serial Monitor
//  #if defined (__AVR_ATmega32U4__)
//    while(!Serial)
//    {}
//    delay(5000);  //5 seconds delay for enabling to see the start up comments on the
serial board
//  #elif defined(__PIC32MX__)
//    delay(1000);
//  #endif

//  Serial.println(F("Arduino setup"));
//  Serial.println(F("Set line ending to newline to send data from the serial
monitor"));

  printf("there was a print here\r\n");

  /* setup toggles on f0 and f1*/
  TRISFbits.TRISF0 = 0;
  TRISFbits.TRISF1 = 0;
#define blue LATFbits.LATF0 = 1; Nop(); LATFbits.LATF0 = 0;
#define vio LATFbits.LATF1 = 1; Nop(); LATFbits.LATF1 = 0;


  /**
  Point ACI data structures to the the setup data that the nRFgo studio generated for
the nRF8001
  */
  if (NULL != services_pipe_type_mapping)
  {
```

```
    aci_state.aci_setup_info.services_pipe_type_mapping =
&services_pipe_type_mapping[0];
  }
  else
  {
    aci_state.aci_setup_info.services_pipe_type_mapping = NULL;
  }
  aci_state.aci_setup_info.number_of_pipes    = NUMBER_OF_PIPES;
  aci_state.aci_setup_info.setup_msgs         = setup_msgs;  //????????
  aci_state.aci_setup_info.num_setup_msgs     = NB_SETUP_MESSAGES;

  /*
  Tell the ACI library, the MCU to nRF8001 pin connections.
  The Active pin is optional and can be marked UNUSED
  */

  aci_state.aci_pins.board_name = BOARD_DEFAULT; //See board.h for details
REDBEARLAB_SHIELD_V1_1 or BOARD_DEFAULT
  aci_state.aci_pins.reqn_pin   = LATBbits.LATB8; //SPI chip select
  aci_state.aci_pins.rdyn_pin   = PORTDbits.RD0; //ready sig (also int!!)

  /* SPI hadled in hardware*/
  //aci_state.aci_pins.mosi_pin   = LATFbits.LATF5;
  //aci_state.aci_pins.miso_pin   = LATFbits.LATF4;
  //aci_state.aci_pins.sck_pin    = LATBbits.LATB14;
//
//  aci_state.aci_pins.spi_clock_divider      = SPI_CLOCK_DIV8;//SPI_CLOCK_DIV8  =
2MHz SPI speed
//                                                          //SPI_CLOCK_DIV16 =
1MHz SPI speed
//
  NRFspi_init(2000000ul);

  aci_state.aci_pins.reset_pin              = LATBbits.LATB12; //4 for Nordic board,
UNUSED for REDBEARLAB_SHIELD_V1_1
  aci_state.aci_pins.active_pin             = LATBbits.LATB11;  // siad unused
  aci_state.aci_pins.optional_chip_sel_pin  = LATBbits.LATB8;

  aci_state.aci_pins.interface_is_interrupt = false; //Interrupts still not available
in Chipkit
  aci_state.aci_pins.interrupt_number       = 1;

  //We reset the nRF8001 here by toggling the RESET line connected to the nRF8001
  //If the RESET line is not available we call the ACI Radio Reset to soft reset the
nRF8001
  //then we initialize the data structures required to setup the nRF8001
  //The second parameter is for turning debug printing on for the ACI Commands and
Events so they be printed on the Serial
  lib_aci_init(&aci_state, false);
  printf("Setup done\r\n");
  blue
}

//void uart_over_ble_init(void)
*********************************************************COMMENTED OUT TO CLEAN CODE
```

```
//{
//  uart_over_ble.uart_rts_local = true;
//}

bool uart_tx(uint8_t *buffer, uint8_t buffer_len)
{
  bool status = false;

  if (lib_aci_is_pipe_available(&aci_state, PIPE_UART_OVER_BTLE_UART_TX_TX) &&
      (aci_state.data_credit_available >= 1))
  {
    status = lib_aci_send_data(PIPE_UART_OVER_BTLE_UART_TX_TX, buffer, buffer_len);
    if (status)
    {
      aci_state.data_credit_available--;
    }
  }

  return status;
}

bool uart_process_control_point_rx(uint8_t *byte, uint8_t length)
{
  bool status = false;
  aci_ll_conn_params_t *conn_params;

  if (lib_aci_is_pipe_available(&aci_state, PIPE_UART_OVER_BTLE_UART_CONTROL_POINT_TX)
)
  {
  //  Serial.println(*byte, HEX);
    printf("%x\r\n", *byte);
    switch(*byte)
    {
      /*
      Queues a ACI Disconnect to the nRF8001 when this packet is received.
      May cause some of the UART packets being sent to be dropped
      */
      case UART_OVER_BLE_DISCONNECT:
        /*
        Parameters:
        None
        */
        lib_aci_disconnect(&aci_state, ACI_REASON_TERMINATE);
        status = true;
        break;


      /*
      Queues an ACI Change Timing to the nRF8001
      */
      case UART_OVER_BLE_LINK_TIMING_REQ:
        /*
        Parameters:
        Connection interval min: 2 bytes
        Connection interval max: 2 bytes
```

```
        Slave latency:           2 bytes
        Timeout:                 2 bytes
        Same format as Peripheral Preferred Connection Parameters (See nRFgo studio ->
nRF8001 Configuration -> GAP Settings
        Refer to the ACI Change Timing Request in the nRF8001 Product Specifications
        */
        conn_params = (aci_ll_conn_params_t *)(byte+1);
        lib_aci_change_timing( conn_params->min_conn_interval,
                               conn_params->max_conn_interval,
                               conn_params->slave_latency,
                               conn_params->timeout_mult);
        status = true;
        break;

      /*
      Clears the RTS of the UART over BLE
      */
      case UART_OVER_BLE_TRANSMIT_STOP:
        /*
        Parameters:
        None
        */
        uart_over_ble.uart_rts_local = false;
        status = true;
        break;


      /*
      Set the RTS of the UART over BLE
      */
      case UART_OVER_BLE_TRANSMIT_OK:
        /*
        Parameters:
        None
        */
        uart_over_ble.uart_rts_local = true;
        status = true;
        break;
    }
  }

  return status;
}

void aci_loop()
{

    bluetooth_data = 6000;

  static bool setup_required = false;

  // We enter the if statement only when there is a ACI event available to be
processed
  if (lib_aci_event_get(&aci_state, &aci_data))
  {
```

```
    aci_evt_t * aci_evt;
    aci_evt = &aci_data.evt;
    switch(aci_evt->evt_opcode)
    {
      /**
      As soon as you reset the nRF8001 you will get an ACI Device Started Event
      */
      case ACI_EVT_DEVICE_STARTED:
      {
        aci_state.data_credit_total = aci_evt->params.device_started.credit_available;
        switch(aci_evt->params.device_started.device_mode)
        {
          case ACI_DEVICE_SETUP:
            /**
            When the device is in the setup mode
            */
            printf("Evt Device Started: Setup\r\n");
            setup_required = true;
            break;

          case ACI_DEVICE_STANDBY:
            printf("Evt Device Started: Standby\r\n");
            //Looking for an iPhone by sending radio advertisements
            //When an iPhone connects to us we will get an ACI_EVT_CONNECTED event
from the nRF8001
            if (aci_evt->params.device_started.hw_error)
            {
              delay_ms(20); //Handle the HW error event correctly.
            }
            else
            {
              lib_aci_connect(0/* in seconds : 0 means forever */, 0x0050 /*
advertising interval 50ms*/);
              printf("Advertising started : Tap Connect on the nRF UART app\r\n");
            }

            break;
        }
      }
      break; //ACI Device Started Event

      case ACI_EVT_CMD_RSP:
        //If an ACI command response event comes with an error -> stop
        if (ACI_STATUS_SUCCESS != aci_evt->params.cmd_rsp.cmd_status)
        {
          //ACI ReadDynamicData and ACI WriteDynamicData will have status codes of
          //TRANSACTION_CONTINUE and TRANSACTION_COMPLETE
          //all other ACI commands will have status code of ACI_STATUS_SCUCCESS for a
successful command
//          Serial.print(F("ACI Command "));
//          Serial.println(aci_evt->params.cmd_rsp.cmd_opcode, HEX);
//          Serial.print(F("Evt Cmd respone: Status "));
//          Serial.println(aci_evt->params.cmd_rsp.cmd_status, HEX);

          printf("ACI Command ");
```

```
          printf("%x\r\n",aci_evt->params.cmd_rsp.cmd_opcode);
          printf("Evt Cmd respone: Status ");
          printf("%x\r\n",aci_evt->params.cmd_rsp.cmd_status);
        }
        if (ACI_CMD_GET_DEVICE_VERSION == aci_evt->params.cmd_rsp.cmd_opcode)
        {
          //Store the version and configuration information of the nRF8001 in the
Hardware Revision String Characteristic
          lib_aci_set_local_data(&aci_state,
PIPE_DEVICE_INFORMATION_HARDWARE_REVISION_STRING_SET,
            (uint8_t *)&(aci_evt->params.cmd_rsp.params.get_device_version),
sizeof(aci_evt_cmd_rsp_params_get_device_version_t));
        }
        break;

      case ACI_EVT_CONNECTED:
        printf("Evt Connected");
        uart_over_ble.uart_rts_local = true; //this line was called in
uart_over_ble_init();****************************************************************
****
        //uart_over_ble_init();
        timing_change_done              = false;
        aci_state.data_credit_available = aci_state.data_credit_total;

        /*
        Get the device version of the nRF8001 and store it in the Hardware Revision
String
        */
        lib_aci_device_version();
        break;

      case ACI_EVT_PIPE_STATUS:
        printf("Evt Pipe Status");
        if (lib_aci_is_pipe_available(&aci_state, PIPE_UART_OVER_BTLE_UART_TX_TX) &&
(false == timing_change_done))
        {
          lib_aci_change_timing_GAP_PPCP(); // change the timing on the link as
specified in the nRFgo studio -> nRF8001 conf. -> GAP.
                                            // Used to increase or decrease bandwidth
          timing_change_done = true;

          char hello[]="Connected to Caddiellac";
          uart_tx((uint8_t *)&hello[0], strlen(hello));
          printf("Sending :");
          printf("hello\r\n");
        }
        break;

      case ACI_EVT_TIMING:
        printf("Evt link connection interval changed");
        lib_aci_set_local_data(&aci_state,
                               PIPE_UART_OVER_BTLE_UART_LINK_TIMING_CURRENT_SET,
                               (uint8_t *)&(aci_evt->params.timing.conn_rf_interval),
/* Byte aligned */
```

```
PIPE_UART_OVER_BTLE_UART_LINK_TIMING_CURRENT_SET_MAX_SIZE);
        break;

      case ACI_EVT_DISCONNECTED:
        printf("Evt Disconnected/Advertising timed out\r\n");
        lib_aci_connect(0/* in seconds  : 0 means forever */, 0x0050 /* advertising
interval 50ms*/);
        printf("Advertising started. Tap Connect on the nRF UART app\r\n");
        break;

      case ACI_EVT_DATA_RECEIVED:
        printf("Pipe Number: ");
        printf("%d\r\n",aci_evt->params.data_received.rx_data.pipe_number);

        if (PIPE_UART_OVER_BTLE_UART_RX_RX == aci_evt-
>params.data_received.rx_data.pipe_number)
          {
            //****************************ADDED
CODE****************************************
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x30){ //IF I
SEND A 0
                bluetooth_data = 0;
                LATEbits.LATE1 = 0;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x31){ //IF I
SEND A 1
                bluetooth_data = 1;
                LATEbits.LATE1 = 1;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x32){ //IF I
SEND A 2
                bluetooth_data = 2;
                LATEbits.LATE2 = 0;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x33){ //IF I
SEND A 3
                bluetooth_data = 3;
                LATEbits.LATE2 = 1;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x34){ //IF I
SEND A 4
                bluetooth_data = 4;
                LATEbits.LATE3 = 0;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x35){ //IF I
SEND A 5
                bluetooth_data = 5;
                LATEbits.LATE3 = 1;
            }
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x36){ //IF I
SEND A 6
                bluetooth_data = 6;
                LATEbits.LATE4 = 0;
            }
```

```
            if (aci_evt->params.data_received.rx_data.aci_data[0] == 0x37){ //IF I
SEND A 7
                bluetooth_data = 7;
                LATEbits.LATE4 = 1;
            }

            //**************************END ADDED
CODE**************************************
           /* printf(" Data(Hex) :
");*****************************************************************COMMENT START
            int i;
            for(i=0; i<aci_evt->len - 2; i++)
            {
              printf("%X ",(char)aci_evt->params.data_received.rx_data.aci_data[i]);
              uart_buffer[i] = aci_evt->params.data_received.rx_data.aci_data[i];
             // Serial.print(F(" "));
            }
            uart_buffer_len = aci_evt->len - 2;
           // Serial.println(F(""));
            if (lib_aci_is_pipe_available(&aci_state, PIPE_UART_OVER_BTLE_UART_TX_TX))
            {
              /*Do this to test the loopback otherwise comment it out*/
              /*
              if (!uart_tx(&uart_buffer[0], aci_evt->len - 2))
              {
                Serial.println(F("UART loopback failed"));
              }
              else
              {
                Serial.println(F("UART loopback OK"));
              }
              */
            //}
*********************************************************************************C
OMMENT END

        }
        if (PIPE_UART_OVER_BTLE_UART_CONTROL_POINT_RX == aci_evt-
>params.data_received.rx_data.pipe_number)
        {
          uart_process_control_point_rx(&aci_evt-
>params.data_received.rx_data.aci_data[0], aci_evt->len - 2); //Subtract for Opcode
and Pipe number
        }
        break;

      case ACI_EVT_DATA_CREDIT:
        aci_state.data_credit_available = aci_state.data_credit_available + aci_evt-
>params.data_credit.credit;
        break;

      case ACI_EVT_PIPE_ERROR:
        //See the appendix in the nRF8001 Product Specication for details on the error
codes
        printf("ACI Evt Pipe Error: Pipe #:");
```

```
        printf("%d\r\n",aci_evt->params.pipe_error.pipe_number);
        printf("  Pipe Error Code: 0x");
        printf("%x \r\n",aci_evt->params.pipe_error.error_code);

        //Increment the credit available as the data packet was not sent.
        //The pipe error also represents the Attribute protocol Error Response sent
from the peer and that should not be counted
        //for the credit.
        if (ACI_STATUS_ERROR_PEER_ATT_ERROR != aci_evt->params.pipe_error.error_code)
        {
          aci_state.data_credit_available++;
        }
        break;

      case ACI_EVT_HW_ERROR:
        printf("HW error: ");
        printf("%d \r\n",aci_evt->params.hw_error.line_num);

        uint8_t counter;
        for(counter = 0; counter <= (aci_evt->len - 3); counter++)
        {
          printf("%d ", aci_evt->params.hw_error.file_name[counter]); //uint8_t
file_name[20];  //was a arduino write???
        }
        printf("\r\n");
        lib_aci_connect(0/* in seconds, 0 means forever */, 0x0050 /* advertising
interval 50ms*/);
        printf("Advertising started. Tap Connect on the nRF UART app");
        break;

    }
  }
  //else
  //{
    //Serial.println(F("No ACI Events available"));
    // No event in the ACI Event queue and if there is no event in the ACI command
queue the arduino can go to sleep
    // Arduino can go to sleep now
    // Wakeup from sleep from the RDYN line
  //}

  /* setup_required is set to true when the device starts up and enters setup mode.
   * It indicates that do_aci_setup() should be called. The flag should be cleared if
   * do_aci_setup() returns ACI_STATUS_TRANSACTION_COMPLETE.
   */
  if(setup_required)
  {
    if (SETUP_SUCCESS == do_aci_setup(&aci_state))
    {
      setup_required = false;
      vio
    }
  }
}
```

```
bool stringComplete = false;  // whether the string is complete
uint8_t stringIndex = 0;      //Initialize the index to store incoming chars

void serialEvent(void);  // prototype for fnc below

void loop() {

        // ********************** Declare Variables ************************** //
    float dist_from_center;
    float upper_bound=100;
    float lower_bound=60;
    float D1, D2, D3;
    upper_bound= upper_bound*160 /100; // Not sure if we can use float. Use int
    lower_bound= lower_bound*160 /100;

  //Process any ACI commands or events
  aci_loop();

  if (bluetooth_data == 1) {                        // BLUETOOTH ENABLED
      LATEbits.LATE0 = 0; //TURN LED 0 ON
      enable = 1;
      ramp_down(lower_bound);
  }

  else if (bluetooth_data == 0) {                   // BLUETOOTH DISABLED
      LATEbits.LATE0 = 1; //TURN LED 0 OFF
      enable = 0;
      ramp_down(lower_bound);
  }


/*******************************BLUETOOTH COMMANDS*************************/
      if (bluetooth_data == 3) {                    // LEFT MOTOR ON
        LATEbits.LATE2 = 0; //TURN LED 0 ON

        unsigned int Pwm_left = 0;       // Pwm_left controls left motor
        while (Pwm_left < lower_bound) { // Ramp up mode (Ramp up to 50%)
            Pwm_left ++;
            OC3RS = Pwm_left;
            delay_us(600);      // End of ramp up
        }
      }
      else if (bluetooth_data == 2) {               // LEFT MOTOR OFF
        LATEbits.LATE2 = 1; //TURN LED 0 OFF

        unsigned int Pwm_left = lower_bound;     // Pwm_right controls right motor
        while (Pwm_left > 0) {                          // Ramp down to zero
            Pwm_left --;
            OC3RS = Pwm_left;                           // Write new duty cycle
            delay_us(600);
        }
      }
      else if (bluetooth_data == 5) {               // BOTH MOTORS ON
        LATEbits.LATE3 = 0; //TURN LED 0 ON
        unsigned int Pwm_left = 0;      // Pwm_left controls left motor
```

```
        unsigned int Pwm_right = 0;     // Pwm_right controls right motor
        while (Pwm_right < ((upper_bound+lower_bound)/2)) { // Ramp up mode (Ramp up
to 50%)   ensuring both will turn off
            Pwm_right++; // If the duty cycle is not at max, increase
            Pwm_left++;
            OC4RS = Pwm_right; // Write new duty cycle
            OC3RS = Pwm_left;
            delay_us(600);
        }
      }
      else if (bluetooth_data == 4) {               // BOTH MOTORS OFF
        LATEbits.LATE3 = 1; //TURN LED 0 OFF

        unsigned int Pwm_left = lower_bound;      // Pwm_left controls left motor
        unsigned int Pwm_right = lower_bound;     // Pwm_right controls right motor
        while (Pwm_right > 0) { // Ramp up mode (Ramp up to 50%)   ensuring both will
turn off
            Pwm_right --; // If the duty cycle is not at max, increase
            Pwm_left --;
            OC4RS = Pwm_right; // Write new duty cycle
            OC3RS = Pwm_left;
            delay_us(600);
        }
      }
      else if (bluetooth_data == 7) {               // RIGHT MOTOR ON
        LATEbits.LATE4 = 0; //TURN LED 0 ON

        unsigned int Pwm_right = 0;       // Pwm_left controls left motor
        while (Pwm_right < lower_bound) { // Ramp up mode (Ramp up to 50%)
            Pwm_right ++;
            OC4RS = Pwm_right;
            delay_us(600);      // End of ramp up
        }
      }
      else if (bluetooth_data == 6) {               // RIGHT MOTOR OFF
        LATEbits.LATE4 = 1; //TURN LED 0 OFF

        unsigned int Pwm_right = lower_bound;     // Pwm_right controls right motor
        while (Pwm_right > 0) {                         // Ramp down to zero
            Pwm_right --;
            OC4RS = Pwm_right;                          // Write new duty cycle
            delay_us(600);
        }

  }

  /********************************** PIXY CODE
************************************/

  if (enable == 0) {
      LATEbits.LATE6 = 0; //TURN LED 0 ON

       if (PORTEbits.RE7 == 0){   // toggle switch is on
        dist_from_center = capture_pixy_data();
        if (dist_from_center == 9000) {
```

```
            ramp_down(lower_bound);
           // LATEbits.LATE0 = 1;
        }
        else if (dist_from_center == 7777) {                        // if CC too close
            D1 = capture_pixy_data();
            D2 = capture_pixy_data();
            //D3 = capture_pixy_data();
            if  ((D1 == 7777 && D2 == 7777)) {      // D3 removed
               // LATEbits.LATE0 = 0;
                ramp_down(lower_bound);
            }
        }
        else motorcontrol(dist_from_center, upper_bound, lower_bound);

   }
       else {
           LATEbits.LATE0=1;
           ramp_down (lower_bound); //necessary?
       }


  }
  else LATEbits.LATE6 = 1; //TURN LED 0 OFF


  // print the string when a newline
arrives:*****************************************************DEBUGGING CODE
  if (stringComplete)
  {
    printf("Sending: ");
    printf("%c \r\n",(char *)&uart_buffer[0]);

    uart_buffer_len = stringIndex + 1;

    if (!lib_aci_send_data(PIPE_UART_OVER_BTLE_UART_TX_TX, uart_buffer,
uart_buffer_len))
    {
      printf("Serial input dropped\r\n");
    }

//*************************************************************************************
********END DEBUGGING CODE
    // clear the uart_buffer:
    for (stringIndex = 0; stringIndex < 20; stringIndex++)
    {
      uart_buffer[stringIndex] = ' ';
    }

    // reset the flag and the index in order to receive more data
    stringIndex    = 0;
    stringComplete = false;
  }

}
```

```
/* Copyright (c) 2014, Nordic Semiconductor ASA
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */


void PWMConfigure(void){

    TRISDbits.TRISD2 = 0; // Make RD2 an output (OC3, PWM pin) //added from, main code
    TRISDbits.TRISD3 = 0; // Make RD3 an output (OC4, PWM pin)

    LATDbits.LATD3 = 0;
    LATDbits.LATD2 = 0;

//******************* Output Compare for PWM_left/OC3 ************************//

    OC3CON = 0x0000;           // Turn off the OC3 when performing the setup
    OC3CONbits.OCM = 6;        // Configure for PWM mode without Fault pin enabled
    OC3CONbits.OCTSEL = 0;     // Timer2 is clock source for Output Compare 1
    OC3CONbits.OC32 = 0;       // 16-bit mode (Output Compare)

//******************* Output Compare for PWM_right/OC2 ***********************//

    OC4CON = 0x0000;           // Turn off the OC4 when performing the setup
    OC4CONbits.OCM = 6;        // Configure for PWM mode without Fault pin enabled
    OC4CONbits.OCTSEL = 0;     // Timer2 is clock source for Output Compare 2
    OC4CONbits.OC32 = 0;       // 16-bit mode (Output Compare)

//******************** Set up Timer2 for both OC3 & OC2 **********************//

    T2CON = 0x0000;                // Clear Timer 3 Control bits
    TMR2 = 0x0000;
    T2CONbits.ON = 0;              // Clear ON Control bit to clear timer
    T2CONbits.T32 = 0;             // 16-bit mode (Timer 2 and Timer 3)

//************************ Initializations OC3/OC2 ***************************//
```

```
    OC3R = 0x0000;                  // Initialize primary Compare register
    OC3R = 0;                       // Give OC3R initial duty cycle
                                    // (value/160 * 100 is duty cycle in %)
    OC3RS = 0x0000;                 // Initialize secondary Compare register

    OC4R = 0x0000;                  // Initialize primary Compare register
    OC4R = 0;                       // Give OC2R initial duty cycle
                                    // (value/160 * 100 is duty cycle in %)
    OC4RS = 0x0000;                 // Initialize secondary Compare register

    T2CONbits.TCKPS0 = 1;           //
    T2CONbits.TCKPS1 = 0;           // Set Prescale on Timer2 to 1:32
    T2CONbits.TCKPS2 = 1;           //

    PR2 = 160;                      // Set PR to 160
                                    // This gives PWM signal with 1.9 kHz freq

//***********************   Turn on Timers & OCs    ***************************//

    T2CONbits.TON = 1;              // Enable time base on Timer3
    T2CONbits.ON = 1;               // Set ON Control bit high to enable timer
    OC3CONbits.ON = 1;              // Turn on OC3
    OC4CONbits.ON = 1;              // Turn on OC4

}


void ramp_down(float lower_bound){


    // To ramp down PWM signal, run this while loop

    while (right_speed > left_speed){
            right_speed --; // If the duty cycle is not at max, increase
            OC4RS = right_speed; // Write new duty cycle
            delay_us(600);
    }
    while (left_speed > right_speed){
            left_speed --; // If the duty cycle is not at max, increase
            OC3RS = left_speed; // Write new duty cycle
            delay_us(600);
    }
    while (right_speed > 0) { // Ramp up mode (Ramp up to 50%)   ensuring both will
turn off
            right_speed --; // If the duty cycle is not at max, increase
            left_speed --;
            OC4RS = right_speed; // Write new duty cycle
            OC3RS = left_speed;
            delay_us(600);
        }                               // End of ramp down

}

/*********************************Pixy/SPI Functions**********************/
void initializeSPI (void){
```

```
float WREN=0x06; //avoiding global variables
float WRITE=0x02;

       unsigned char Info;
    // Initialize SPI _ low level
    IEC1CLR=0x03800000;//disable SPI interrupts in IECx register
    SPI2CON=0;//stop and reset SPI module by clearing ON bit
    Info=SPI2BUF; //clear the receive buffer
    //SPI3CONbits.ENHBUF=0; //clear ENHBUF bit
    SPI2BRG=0;//write the baud rate register
    SPI2STATbits.SPIROV=0;//clear SPIROV bit
    SPI2CONbits.CKE=0;
    SPI2CONbits.CKP=1;
    SPI2CONbits.SMP=1;
    SPI2CONbits.MSTEN=1;//write desired settings to SPIxCON
    SPI2CONbits.ON=1;//enable SPI

    DDPCONbits.JTAGEN=0; //disable JTAG
    AD1PCFG = 0xFFFF;
    TRISGbits.TRISG7 = 0;
    //AD1PCFGbits.PCFG8=1; //enables pins as digital
    //AD1PCFGbits.PCFG10=1;
    //AD1PCFGbits.PCFG11=1;
    //AD1PCFGbits.PCFG14=1;


    // write enable Pixy
    TRISBbits.TRISB8 = 0;// Tristate Buffer to 0
    LATBbits.LATB8=0;
    SPI_SEND(WREN);//set write enable latch
    LATBbits.LATB8=1; //set high
    LATBbits.LATB8=0;//set low
    SPI_SEND(WRITE);         //write data to memory
}

unsigned char SPI_SEND (unsigned char data){
    IFS1bits.SPI2RXIF=0; //clears flag
    SPI2BUF= data;//place data to be sent in transmit buffer
    while(IFS1bits.SPI2RXIF==0); //wait for data to be shifted
        return(SPI2BUF);
    //return data received

}

float capture_pixy_data (void) {

    unsigned char X;
    float CC1, CC2, Y, Z, CN1, CN2, X1, X2, Y1, Y2, W1, W2, H1, H2, A1, A2 ; //changed
to int from char
    float dist_from_center;
    int counter = 0;

    while (counter < 6000) {

        X= SPI_SEND(0X5A);          //location in memory
```

```
        X= SPI_SEND(0X00);
        if (X == 0x55) {
            //LATBbits.LATB8=1; //set high
            //LATBbits.LATB8=0;//set low
            CC1= SPI_SEND(0X5A);                    // Color Code
            CC2= SPI_SEND(0X00);
            if (CC2 == 0x56) {                  // check for 55 twice, not needed for
CC
                Y= SPI_SEND(0X5A);                  // ????
                Z= SPI_SEND(0X00);
                CN1= SPI_SEND(0X5A);                // Channel Number
                CN2= SPI_SEND(0X00);
                X1= SPI_SEND(0X5A);                 // X Coordinate
                X2= SPI_SEND(0X00);
                Y1= SPI_SEND(0X5A);                 // Y Coordinate
                Y2= SPI_SEND(0X00);
                W1= SPI_SEND(0X5A);                 // Width
                W2= SPI_SEND(0X00);
                H1= SPI_SEND(0X5A);                 // Height
                H2= SPI_SEND(0X00);
                A1= SPI_SEND(0X5A);                 // Angle
                A2= SPI_SEND(0X00);

                if (H2 > 27) {                      // if CC is too close
                    return(7777);                   // return 7777
                }

                if (X1 == 1) {
                    X2 = X2 + 255;
                }
                dist_from_center = X2 - 160;        // right side positive values

                X = 0;// reset X

                return(dist_from_center);
            }
        }
        else //if (X == 0)
        {
            counter++;
        }

    }

    return(9000);
}

void motorcontrol (float dist_from_center, float upper_bound, float lower_bound){

    LATEbits.LATE1 = 0;
    //define variable
    float left;                 // left motor OC3RS
    float right;                // right motor OC4RS
```

```
    /* CALCULATE DESIRED LEFT AND RIGHT SPEEDS */

    if (dist_from_center>0){  ///turn right; left motor increase. right motor constant
10%
        LATEbits.LATE2 = 0;
        if (dist_from_center>80) {
            left = upper_bound;
            right = lower_bound/2;
        }
        else {
            left = (upper_bound-lower_bound)/80 * dist_from_center + lower_bound;
//converting x value in relation to upper/lower bounds
            right = (-lower_bound/2)/80 * dist_from_center + lower_bound;
        }
    }
    else if (dist_from_center<=0){  ///turn right; left motor increase. right motor
constant 10%
        LATEbits.LATE3 = 0;
        if (dist_from_center<-80) {
            right = upper_bound;
            left = lower_bound/2;
        }
        else {
            right= (upper_bound-lower_bound)/80 *(-dist_from_center)+lower_bound;
//converting x value in relation to upper/lower bounds
            left = (-lower_bound/2)/80 * (-dist_from_center) + lower_bound;
        }
    }

    /* IF SPEED IS BELOW THE LOWER BOUND, RAMP IT UP TO THE LOWER BOUND */

    if (left_speed == 0 || right_speed == 0) { // if either motor is below lower
bound, ramp up

        float D1, D2, D3;
        D1 = capture_pixy_data();
        D2 = capture_pixy_data();
        D3 = capture_pixy_data();
        if  ((D1 == 7777 || D2 == 7777) || D3 == 7777) {        // Check to ensure the
CC isn't too close
            left = 0;                                           // if too close, get
out
            right = 0;
        }

        LATEbits.LATE4 = 0;
        while (left_speed < lower_bound/2) {
            left_speed++;
            right_speed++;
            OC3RS = left_speed;
            OC4RS = right_speed;
            delay_us(600);
        }
    }
```

```
    /* ADJUST SPEEDS */

    while (left < left_speed) {
        left_speed--;
        OC3RS = left_speed;
        delay_us(600);
    }
    while (right < right_speed) {
        right_speed--;
        OC4RS = right_speed;
        delay_us(600);
    }
    while (left > left_speed) {
        left_speed++;
        OC3RS = left_speed;
        delay_us(600);
    }
    while (right > right_speed) {
        right_speed++;
        OC4RS = right_speed;
        delay_us(600);
    }

    LATEbits.LATE5 = 0;

}
```