Lil Latch E.
Remote control Vehicle

ECE 216

Introduction

- Team: Lil Latch E

- Project: Lil Latch Model-E

Overview

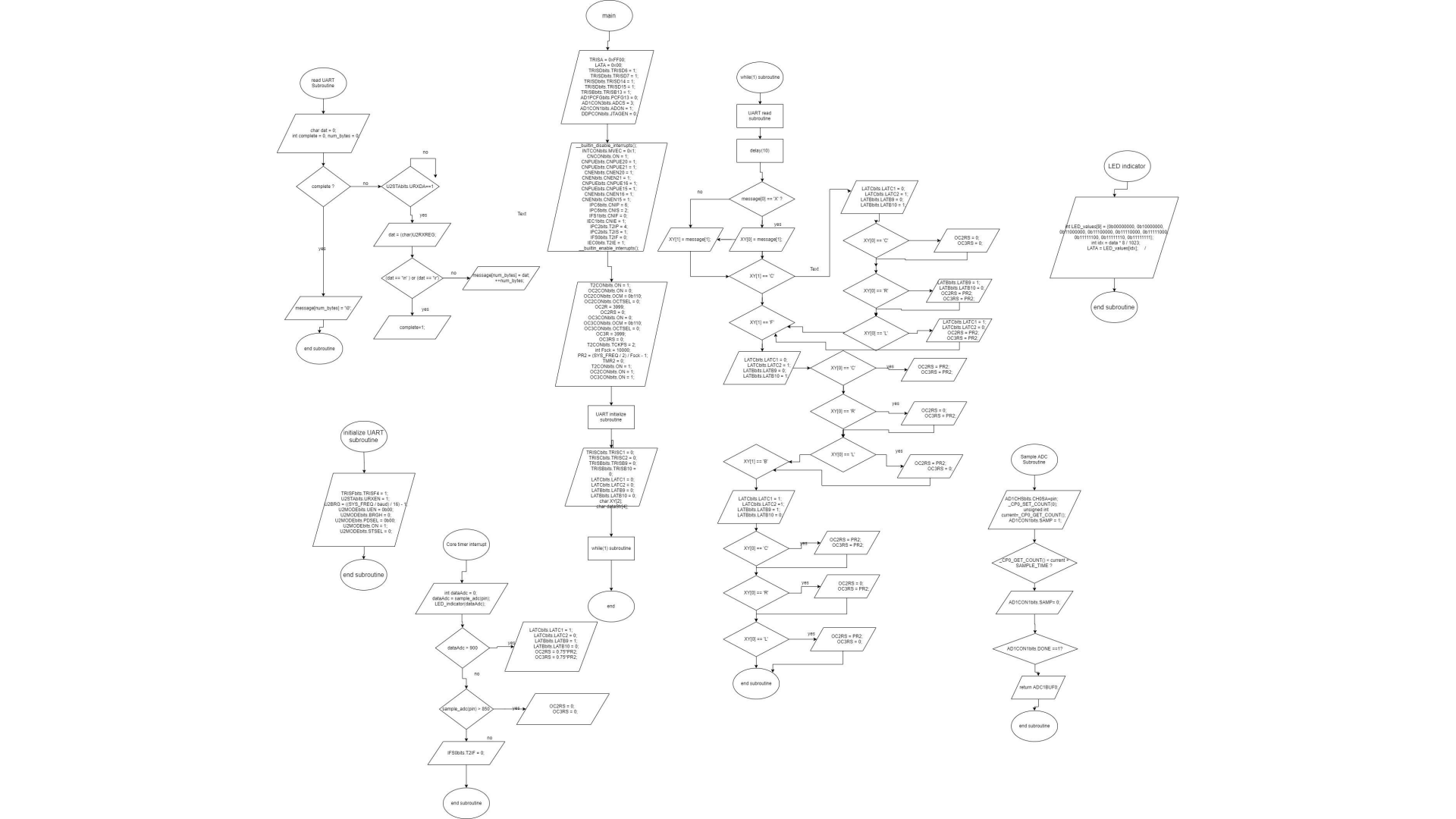
- Drive a remotely operated vehicle using a gamepad

1. Python program to interface with the Gamepad on the Raspberry Pi
2. UART communication from Raspberry Pi to PIC32
3. C program to map gamepad controls to actions to direct the vehicle
4. Motors control direction to move forward, left, right, back and diagonal
5. Based on motor rotations, read from the two 48 counts/rev encoders
6. If vehicle too close to an object back up, then return control to user
7. Hardware assembly

Approach

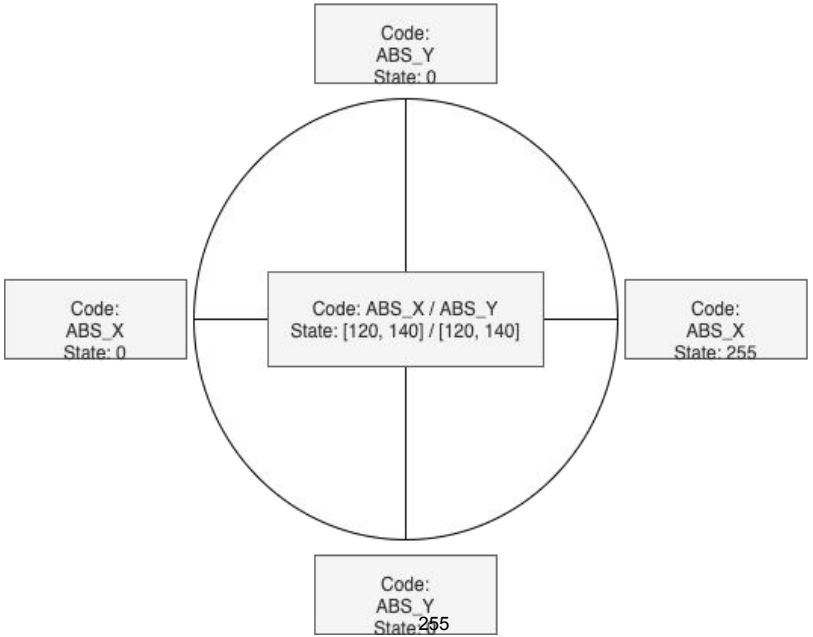
Software Breakdown

- Python (Raspberry Pi 3)
  - Gamepad
  - Communication
- C (PIC 32)
  - Interrupts
  - UART
  - ADC
  - Motor Control
    - PWM
    - GPIO
  - Gamepad input parse


main
TRISA = 0xFF00;
LATA = 0x00;
TRISDbits.TRISD8 = 1;
TRISDbits.TRISD7 = 1;
TRISDbits.TRISD14 = 1;
TRISDbits.TRISD15 = 1;
TRISBbits.TRISB13 = 1;
AD1PCFGbits.PCFG13 = 0;
AD1CON3bits.ADCS = 3;
AD1CON1bits.ADON = 1;
DDPCONbits.JTAGEN = 0;
__builtin_disable_interrupts();
INTCONbits.MVEC = 0x1;
CNCONbits.ON = 1;
CNPUEbits.CNPUE29 = 1;
CNPUEbits.CNPUE21 = 1;
CNEnbits.CNEN20 = 1;
CNENbits.CNEN21 = 1;
CNPUEbits.CNPUE16 = 1;
CNPUEbits.CNPUE15 = 1;
CNENbits.CNEN16 = 1;
CNENbits.CNEN15 = 1;
IPC6bits.CNIP = 6;
IPC6bits.CNIS = 2;
IFS1bits.CNIF = 0;
IEC1bits.CNIE = 1;
IPC2bits.T2IP = 4;
IPC2bits.T2IS = 1;
IFS0bits.T2IF = 0;
IEC0bits.T2IE = 1;
__builtin_enable_interrupts();
T2CONbits.ON = 1;
OC2CONbits.ON = 0;
OC2CONbits.OCM = 0b110;
OC2CONbits.OCTSEL = 0;
OC2R = 3999;
OC2RS = 0;
OC3CONbits.ON = 0;
OC3CONbits.OCM = 0b110;
OC3CONbits.OCTSEL = 0;
OC3R = 3999;
OC3RS = 0;
T2CONbits.TCKPS = 2;
int Fsck = 10000;
PR2 = (SYS_FREQ / 2) / Fsck - 1;
TMR2 = 0;
T2CONbits.ON = 1;
OC2CONbits.ON = 1;
OC3CONbits.ON = 1;
UART intialize subroutine
TRISCbits.TRISC1 = 0;
TRISCbits.TRISC2 = 0;
TRISBbits.TRISB8 = 0;
TRISBbits.TRISB10 = 0;
LATCbits.LATC1 = 0;
LATCbits.LATC2 = 0;
LATBbits.LATB9 = 0;
LATBbits.LATB10 = 0
char XY[2];
char data[8][4];
while(1) subroutine
end
while(1) subroutine
UART read subroutine
delay(10)
message[0] == 'X' ?
no
yes
XY[1] = message[1];
XY[0] = message[1];
XY[1] == 'C'
Text
XY[1] == 'F'
LATCbits.LATC1 = 0;
LATCbits.LATC2 = 1;
LATBbits.LATB9 = 0;
LATBbits.LATB10 = 1;
XY[0] == 'C'
yes
OC2RS = PR2;
OC3RS = PR2;
XY[0] == 'R'
yes
OC2RS = 0;
OC3RS = PR2
XY[1] == 'B'
XY[0] == 'L'
yes
OC2RS = PR2;
OC3RS = 0;
LATCbits.LATC1 = 1;
LATCbits.LATC2 = 0;
LATBbits.LATB9 = 1;
LATBbits.LATB10 = 0;
XY[0] == 'C'
yes
OC2RS = PR2;
OC3RS = PR2;
XY[0] == 'R'
yes
OC2RS = 0;
OC3RS = PR2;
XY[0] == 'L'
yes
OC2RS = PR2;
OC3RS = 0;
end subroutine
LATCbits.LATC1 = 0;
LATCbits.LATC2 = 1;
LATBbits.LATB9 = 0;
LATBbits.LATB10 = 1;
XY[0] == 'C'
yes
OC2RS = 0;
OC3RS = 0;
XY[0] == 'R'
yes
LATBbits.LATB9 = 1;
LATBbits.LATB10 = 0;
OC2RS = PR2;
OC3RS = PR2;
XY[0] == 'L'
yes
LATCbits.LATC1 = 1;
LATCbits.LATC2 = 0;
OC2RS = PR2;
OC3RS = PR2;
read UART Subroutine
char dat = 0;
int complete = 0, num_bytes = 0;
complete ?
no
yes
U2STAbits.URXDA==1
no
yes
dat = (char)U2RXREG;
(dat == '\n') or (dat == '\r')
no
yes
message[num_bytes] = dat;
++num_bytes;
complete=1;
message[num_bytes] = '\0';
end subroutine
Text
initialize UART subroutine
TRISFbits.TRISF4 = 1;
U2STAbits.URXEN = 1;
U2BRG = ((SYS_FREQ / baud) / 16) - 1;
U2MODEbits.LEN = 0b00;
U2MODEbits.BRGH = 0;
U2MODEbits.PDSEL = 0b00;
U2MODEbits.ON = 1;
U2MODEbits.STSEL = 0;
end subroutine
Core timer interrupt
int dataAdc = 0;
dataAdc = sample_adc(pin);
LED_indicator(dataAdc);
dataAdc = 900
yes
LATCbits.LATC1 = 1;
LATCbits.LATC2 = 0;
LATBbits.LATB9 = 1;
LATBbits.LATB10 = 0;
OC2RS = 0.75*PR2;
OC3RS = 0.75*PR2;
no
sample_adc(pin) > 850
yes
OC2RS = 0;
OC3RS = 0;
no
IFS0bits.T2IF = 0;
end subroutine
LED indicator
int LED_values[9] = {0b00000000, 0b10000000, 0b11000000, 0b11100000, 0b11110000, 0b11111000, 0b11111100, 0b11111110, 0b11111111};
int idx = data * 8 / 1023;
LATA = LED_values[idx];
/
end subroutine
Sample ADC Subroutine
AD1CHSbits.CH0SA=pin;
_CP0_SET_COUNT(0);
unsigned int current= _CP0_GET_COUNT();
AD1CON1bits.SAMP = 1;
_CP0_GET_COUNT() < current + SAMPLE_TIME ?
AD1CON1bits.SAMP= 0;
AD1CON1bits.DONE ==1?
return ADC1BUF0;
end subroutine

Parsing Gamepad Inputs w/ Python

Raw Input from D-Pad

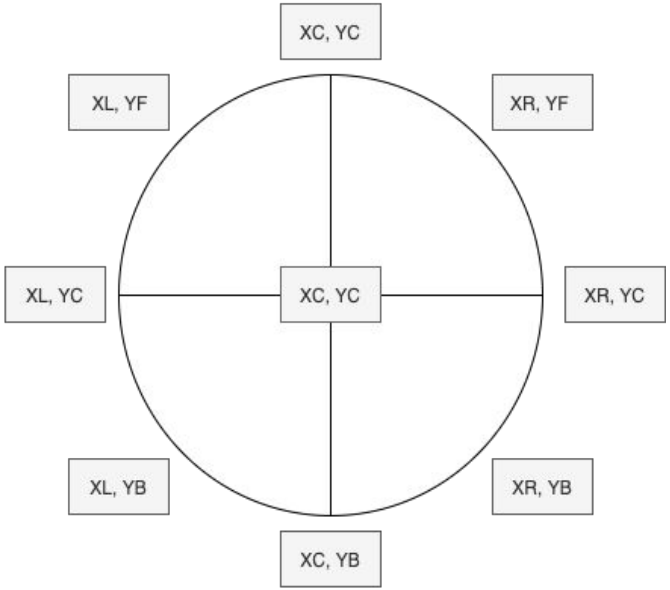


Code Parsing for UART Transmission

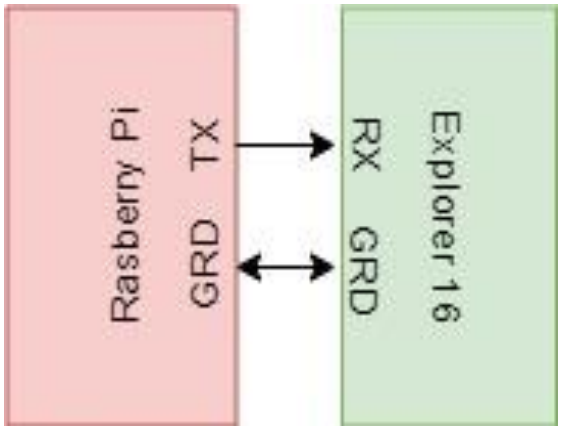
Process

Taking the digital-converted values of the analog gamepad inputs and parsing into two-byte, coordinate-like words for transmission

Output to UART



UART Communication

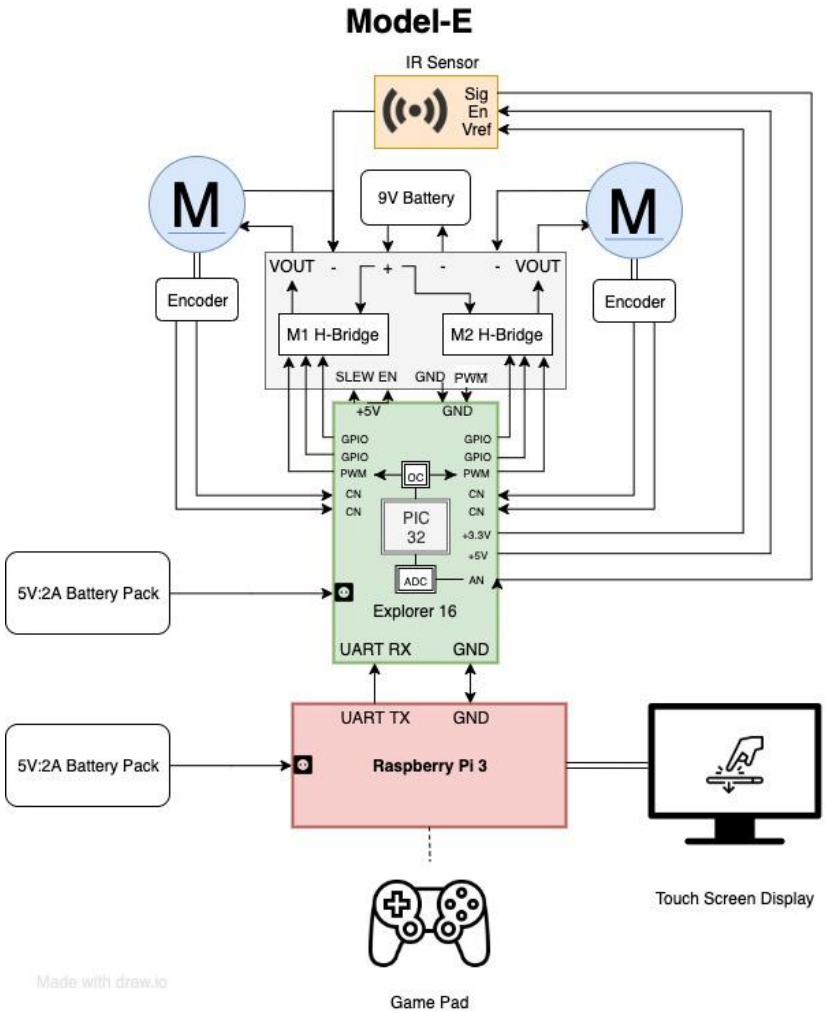


- PIC32 receives the X and Y Coordinate from the raspberry pi through TX-RX line
- Information is parsed on whether it is an X and Y Position
- Depending on X and Y position, output compare is adjusted forward, backward, left, right, and diagonal positions
- Also Depending on position X and Y Position, the GPIO pins for both motors are adjusted to have the wheels spinning in correct direction

Extra Credit: Distance Sensor

- Analog Input reading
- AD converter
- LED indicator

- Sample is read from AN pin 13
- ADC samples the sensor at a rate ~10 CPU ticks
- When done converting it returns 10 bit value and stores it in buffer
- When the sensor reaches 80% distance an interrupt is triggered
- The motors will then drive backwards shortly to move away from the obstacle
- The control is then return to the user gamepad



Hardware Assembly & Connection

- PIC32 MCU / Explorer 16 Board
- Raspberry Pi 3
- Motor Driver w/ H-Bridges
- Motor / Wheel
- I/O
  - Gamepad, Display, Sensors
- Wiring
  - Cables / Pins / Breadboards
- Frame
  - Battery placement

Testing

1. Testing Gamepad connecting to Raspberry Pi
    a. Printed statements to the console
2. Raspberry Pi to PIC32 through UART
    a. Oscilloscope
    b. LCD Display test message
    c. LEDs for different conditions
3. H-Bridge and motors
    a. Could move with just power
    b. Receiving from UART
4. Test fully assembled Model-E
    a. Test screen connections
    b. Test on the floor to adjust turning radius, speed, weight distribution

Distribution

- Python program to interface with the Gamepad Soma
- Output compare and GPIO code and set up Ronaldo
- Encoder read and CN Clara
- Coordinate speed and direction motors Clara
- UART communication Richard
- Receive UART send to motor Ronaldo
- Assembly of hardware & Connection Soma
- Extra credit: Distance Sensor Richard

Challenges & Lessons

- Have a clear outline of project timeline and project parts
- Start from the basics and build up
- Test all your hardware separately if something stops working
- The debugger is your best friend
- Use the LEDs and LCD display to debug
- Double check that your pins aren’t used anywhere else in your program