2017-4-13

Individual Final Report

EIE3105 Integrated Project

YANG, TIANYU
13102841D

Contents

Hardware Description

1.1 System Functions Description

This project is to design an autonomous robot to facility several system requirements and specifications, which means that user system requirement is just a wish list and system specification is the list. Engineers need to finish the tasks and precise engineering data. The whole system includes several steps,

- **With the system specifications defined**
- **High Level Design can be developed**
- **Testing strategy and test data are consolidated**
- **Acceptance conditions are fixed**
- **Detailed designs are generated**
- **Coding**
- **Debugging**
- **Final testing and delivery**

The whole project is to make a model car run and be maneuvered as fast as possible. It is to follow track on the ground and realize some optional features, including auto detect and avoid collision, capability to drive servo. Besides, with wireless communication channel and bird view guidance and multiple robot's collaboration, it can facility a real autonomous robot and do what it can.

Besides, the robot can follow the track and run at least two laps and stay on track. The robot can shoot / maneuver a ball from specific location with the assistance of the remote navigation information. Multiple robots can perform as a team.

System Specifications of the autonomous robot

The system data are specified by the designers in order to demonstrate the creativity and innovation. For example, for hardware convenience, dc supply voltages for the electronic circuit and the dc motor are the same, such as 5v; for maximum speed, dc supply voltages for the electronic circuit and the dc motor are different.

System Hardware Specifications

Assume the core of the hardware has already

been specified, to be specific

(1) 32-bit MIPS microcontroller with USB 2.0;

(2) Ir led and phototransistor for floor sensor;

(3) Ultra sound transducer for collision detection;

(4) 3-10v PWM servo controller as motor drive;

(5) Optical coupler in wheel counter for feedback control;

(6) 802.11b/g Wi-Fi module

1.2 Schematic diagram explanation

The system hardware circuit is shown as below.

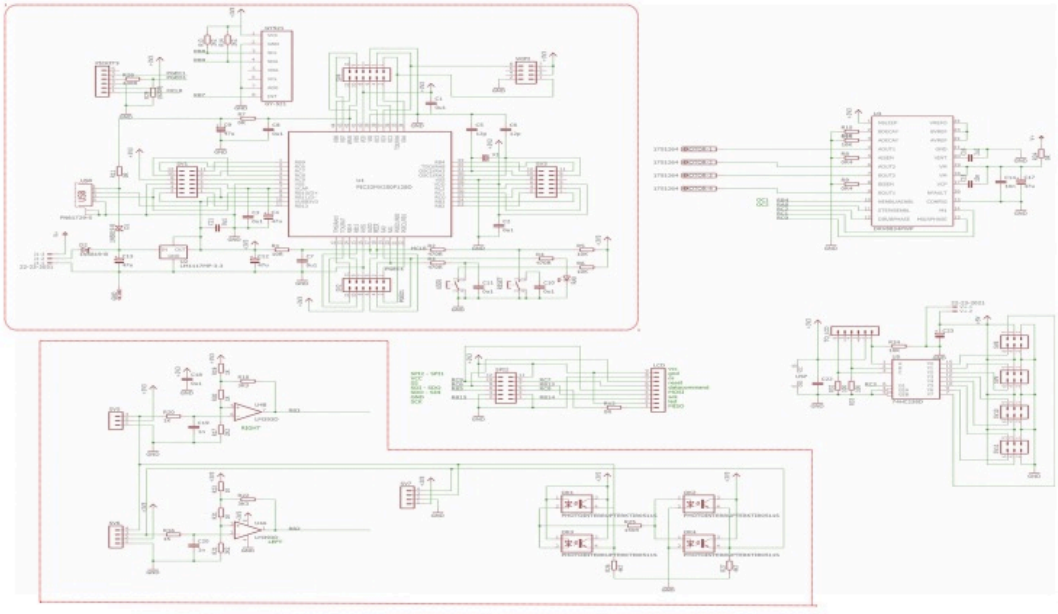


Figure 1.2.1 Schematic diagram

The hardware circuit design is use processor to interface peripherals. We use Microchip PIC32MX250F128D for its special features highlight and peripheral features.

Features highlight

RISC CPU 2.3v to 3.6v DC to 40MHz (Two 32-bit cores register files to reduce interrupt latency) Prefetch cache module to speed execution from flash Multiple interrupt vectors with individually programmable priority

Peripheral features

Configurable watchdog timer USB 2.0 compliant full speed device and On The Go (OTG) controller Two I2 C modules Two SPI modules Two UART modules Parallel master and slave port Hardware real time clock Five 16-bit timers/counters (two 16-bit pairs combine to create two 32-bit timers) Five capture inputs Five compare/PWM

outputs Five external interrupt pins

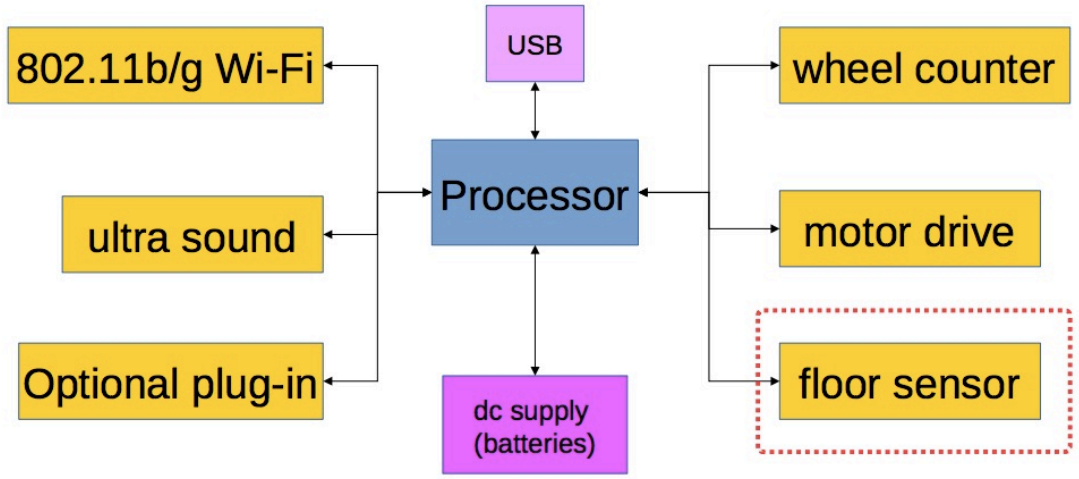


Figure 1.3.1 Diagram of different functional part

The schematic of the autonomic robot is including several parts above. 802.11b/g Wi-Fi is used to detect the situation of the car and the ball. Wheel counter is used as speed and orientation controller, which is to count the number of holes over a period (generated by the timer interrupt) to get the data of the speed. Motor drive is driven by PWM (Pulse width Modulation) signal from the Output Capture Module of the CPU. Last but not the least, floor sensor is used to detect the white line on the floor by using Ir LEDs to detect light color and dark color

1.3 Suggested hardware debugging techniques

During the hardware configuration, we need soldering and testing the electrical components. It needs several special techniques, which is shown as below.

1.3.1 Line tracing board

To test the line tracing board, we use a PCB board to test photodetectors and use magnifier to see whether it is working or not. If one or several of them are not lighting, we will solder it again and check whether it is broken. If it has broken, we will replace it by new one. Then we will make it work and detect the voltage of different pins.

1.3.2 Motor drive circuitry

We will voltage measurement tools to detect motor driving circuitry. If it is working, the voltage should be normal, else there are some connecting programs between the motor and PIC board.

1.3.3 Wheel rotation feedback

We use a program in Demo 2 to detect the wheel running. When the wheel is running, the number will change from 1 to 0 and 0 to 1 continuously.

2. System Software

2.1 System software development environment

System software development environment refers to the collection of hardware and software tools a system developer uses to build software systems. As technology improves and user expectations grow, an environment's functionality tends to change and the set of software tools available to developers has expanded considerably. For Autonomous Robots, we need a Real Time Embedded System, different from Batch Processing Type System, owing three main critical advantages, to be specific, immediate response to inputs, enable timing constraints and on-time output delivery. A Real Time Embedded System contains hard real time system and soft real time system. For hard real time system, all the time constraints must be met, and as for soft real time system, only Most of the time constraints will be met.

Software architecture refers to the high level structures of a software system, the discipline of creating such structures, and the documentation of these structures. These structures are needed to reason about the software system. Each structure comprises software elements, relations among them, and properties of both elements and relations. The architecture of a software system is a metaphor, analogous to the architecture of a building. Software architecture choices include specific structural options from possibilities in the design of software. Time intervals control is particular import in

architecture design. In order to have a good control of the time intervals, three issues must be noticed, that (1) interrupt process is required; (2) Architecture is an event driven approach; and (3) Interrupt Timers are heavily used. To achieve this, multi-tasks approaches, such as the interrupt handler and the interlock flags will be adopted. The design process is divided into different functional modules, and we need to identify modules that are timing critical, then decouple the modules after checking the dependence of the module and any possibility of interlocking. The role of architecture in development of the software is just like a table of contents in a report. Generally, optimal detailed plan requires iterations, and the final and formal design will be fixed after the iterations. To be specific, informal designs are conducted to try out in particular the uncertainties. When the Iteration of the informal designs, only critical portions are tested in order to save time and results formal systems design. After that, subsystems will be identified. To Establish framework for the subsystems control and communications, system structuring, control modelling and modular decomposition will be conducted in order.

2.2 System software requirement in Demo 5

Objective:

You have developed a set of capabilities for your autonomous robot with the on board WiFi module in Demo4. In Demo5, the final demonstration, you are required to team up with your classmates to complete a series of tasks. In this demonstration, you are

going to enrich the capability set in order to complete the challenging tasks within the shortest time. All members of a team must participate in the demonstrations. In a demonstration, THREE autonomous robots from each team are going to perform ball manipulation, shooting and intercept actions.

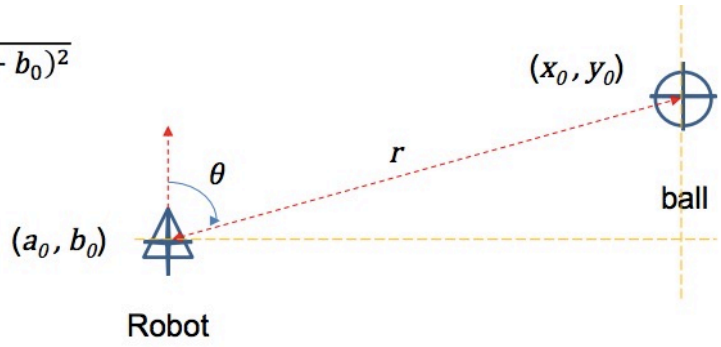
The Tasks:

Robot_1 manipulates the ball from the START position to the LAUNCH_1 pad and shoots the ball from the LAUNCH_1 pad into the LAUNCH_2 area. Robot_2 moves out from HOLD_2 area and shoots the ball from LAUNCH_2 area towards robot_3. Robot_3 is only allowed to move out from HOLD_3 area after robot_2 has shot the ball. Robot_3 must intercept the ball before the ball stops.

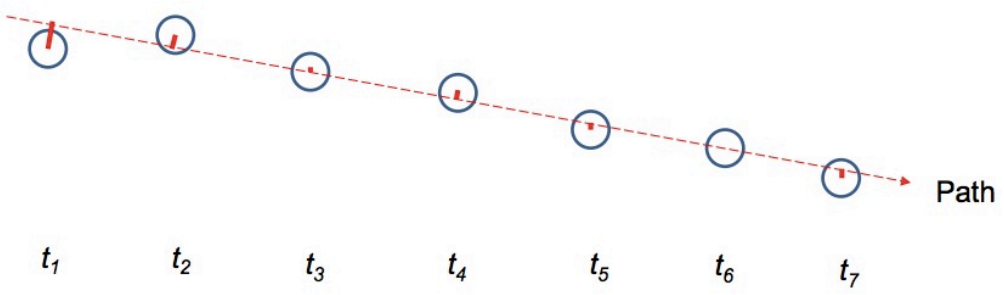
2.3 High level design of Demo 5

$$r = \sqrt{(x_0 - a_0)^2 + (y_0 - b_0)^2}$$

$$\theta = \cot^{-1}\left(\frac{y_0 - b_0}{x_0 - a_0}\right)$$



As the formula above, after receiving the situation of the robot and ball from Wi-Fi, we can get the relative position for these two items.

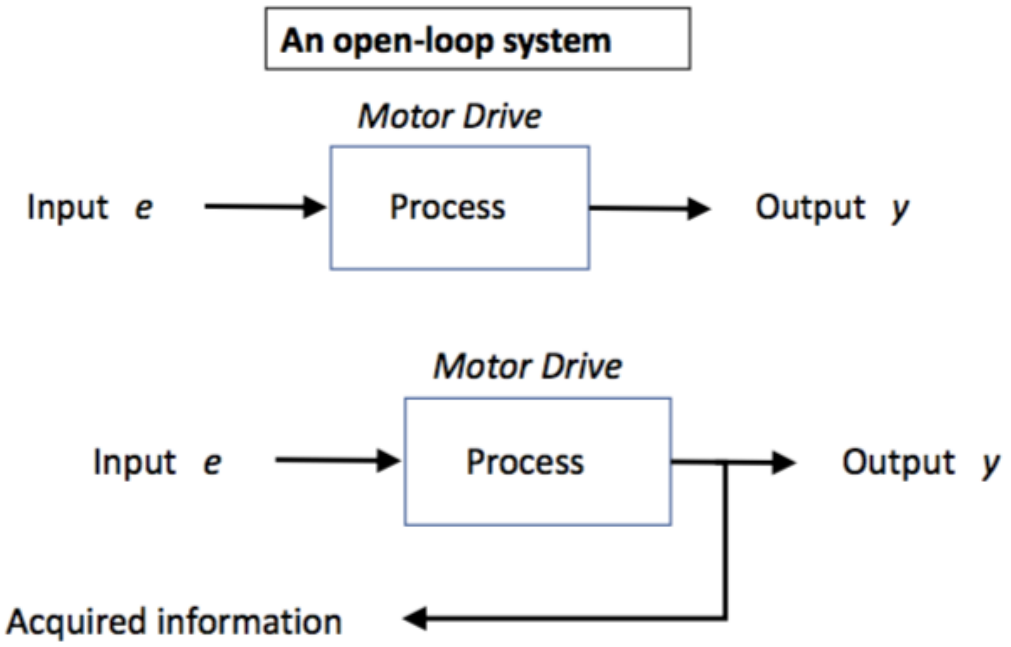


After that, we can assume that the ball is running in a straight line, so that we can calculate the speed of the ball by time interval and displacement.

Later, after analyzing the practical situation to remove noise and measurement error, we can gain the equation of time and the predicted situation of the ball. and then we can intercept the rolling ball.

PID Control

Driving the robot to a location is a process acquiring external information to correct the drive effort and arrive the destination, which can be represented in functional block diagram below.



To achieve this, Feedback Principle is very important which enable this function. The Feedback System acquires external information to correct the drive effort with On/Off control and arrive the destination. Separately, the control output signal changes to umax / umin or On / Off value, depending on the process variable greater or less than the set point. This control owns the advantage of simple, but also the disadvantage is it oscillates or 'rings' around the set point / reference.

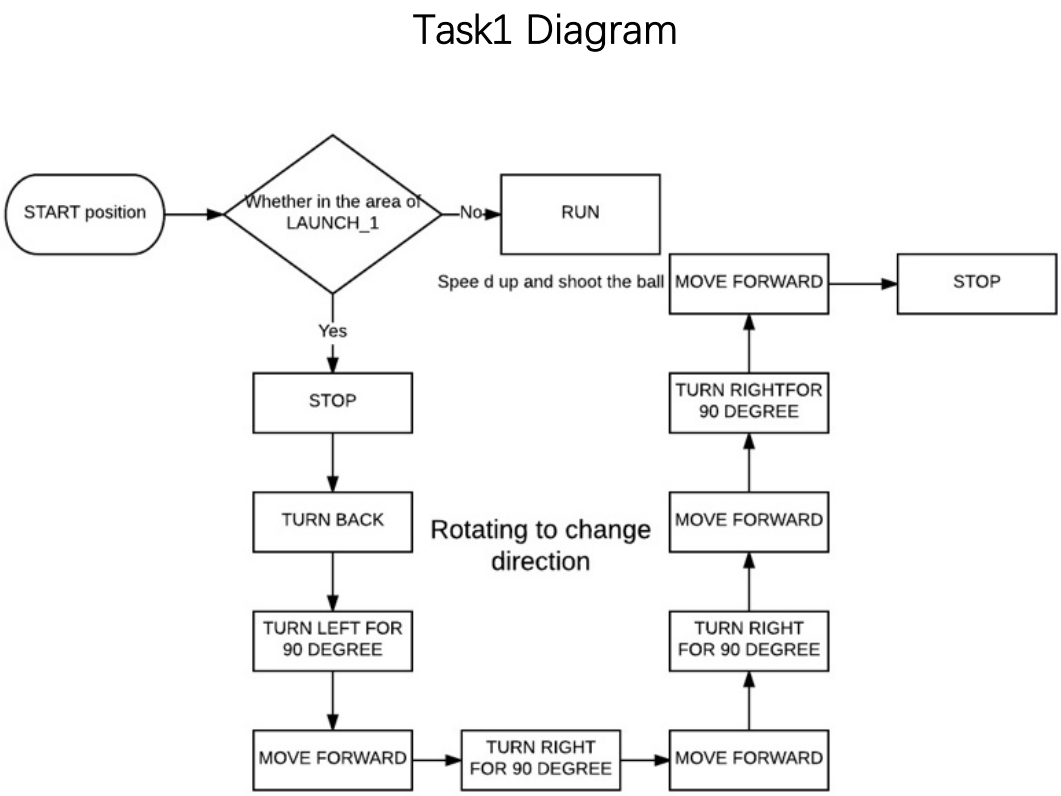
Apart from using hysteresis in the controller to reduce the phenomenon of signal oscillation, using a small gain when the error e is small and using a large gain when the error e is large. The relationship between the control signal and a small control error u = u0 + Kc.

To resolve the steady state error problem with the Proportional control, the integral or reset action should be introduced. This integral action eliminates the problem of any remaining steady state error. The integral, I part able to find the correct value for u0 automatically, in response to any set point without having to know the process static gain. Compare with the On/Off controller, the constant level of u0 found in the P controller is replaced by the integral. The integral is effectively proportional to the area under the curve between the process variable y and the set point r.

Also, need to remark, in practice, the derivative is taken of the process variable instead

of the control error. And the set point r is normally constant with abrupt changes (adjustment) and have no contribution to the derivative.

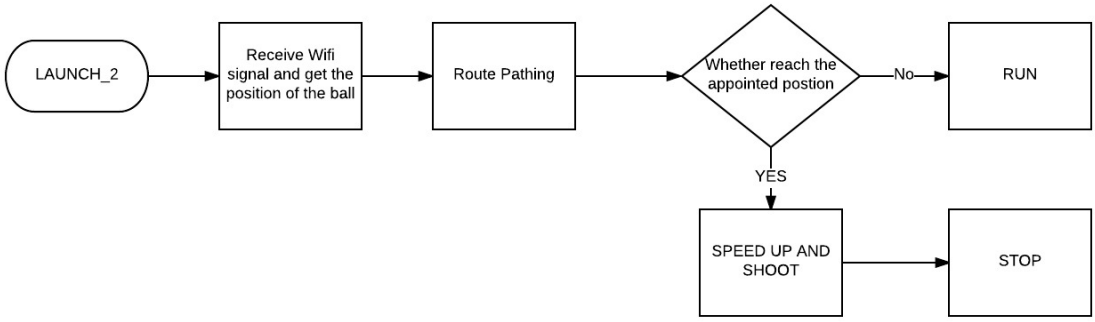
2.4 Flow chart



Flow Chart 1

Firstly, our Robot_1 start pulling the ball from the START position and detect the position of LAUNCH_1 by Wi-Fi continuously and automatically. If the robot arrives the appointed area, it will stop and turn back. Later, the robot will rotate for 360 degrees to adjust the shooting direction. Finally, it will speed up to shoot the ball to the LAUNCH_2.

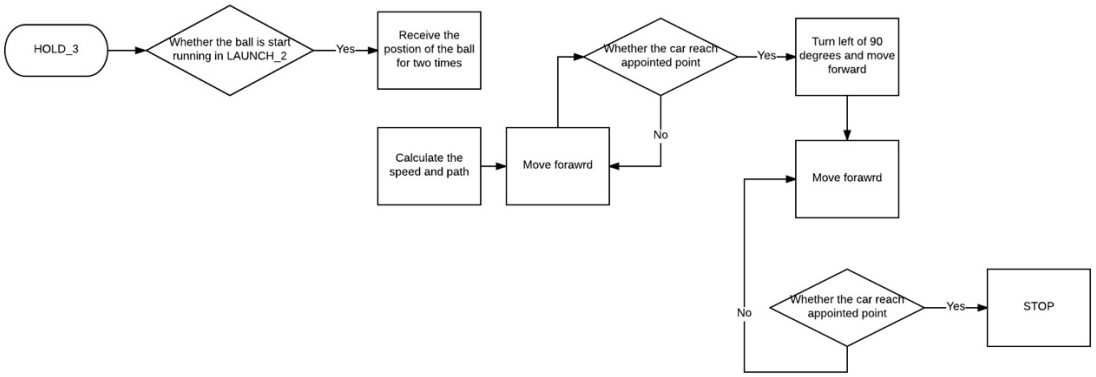
Task 2 Diagram



Flow Chart 2

Next, out Robot_2 will receive the position of ball by Wi-Fi. If the ball is in the area of LAUNCH_2 and it stop for a while, the Robot_2 will start moving by the route which has been assigned in the program. We just use two directions (Forward and turn left) to reach the position of ball. After calculation, the Robot_2 can get the order that how much it should go forward and how much it should turn left. Later, it will move following the path and when it move forward, it will speed up and shoot the ball.

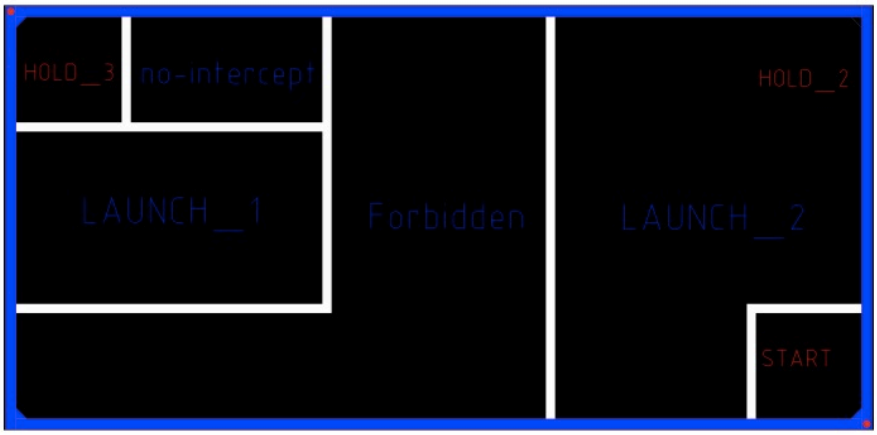
Task 3 diagram



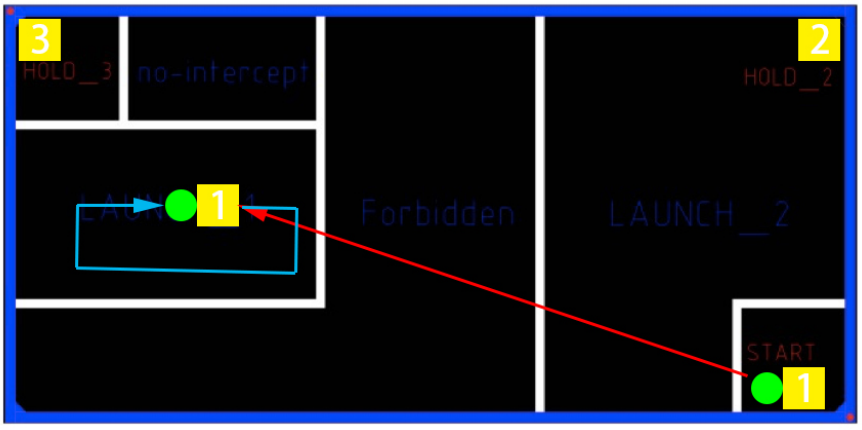
Flow Chart 3

Last but not the least, the Robot_3 will judge whether the ball has already shooting

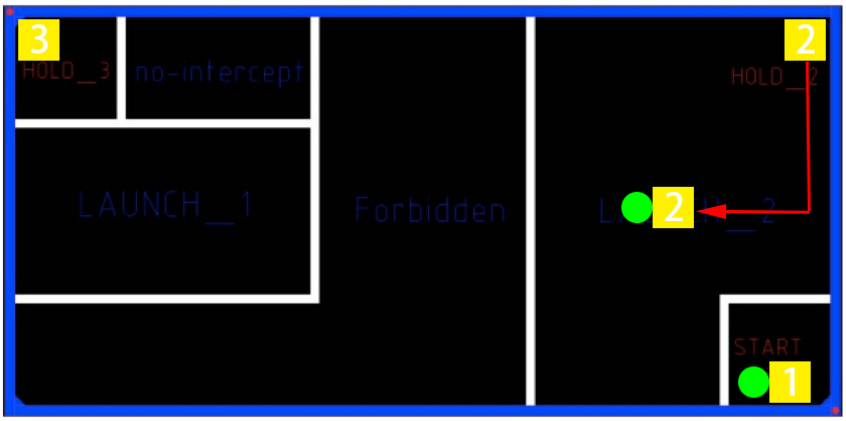
from LAUNCH_3. It will receive the position of the ball and Robot_2. When Robot_2 start running, it starts judge. If the ball moves forward to the direction of Robot_3, Robot_3 start measure the path of the ball by receive the time interval and position. After that, it starts moving forwards to the appointed point and stop. Finally, it will intercept the ball's movement.



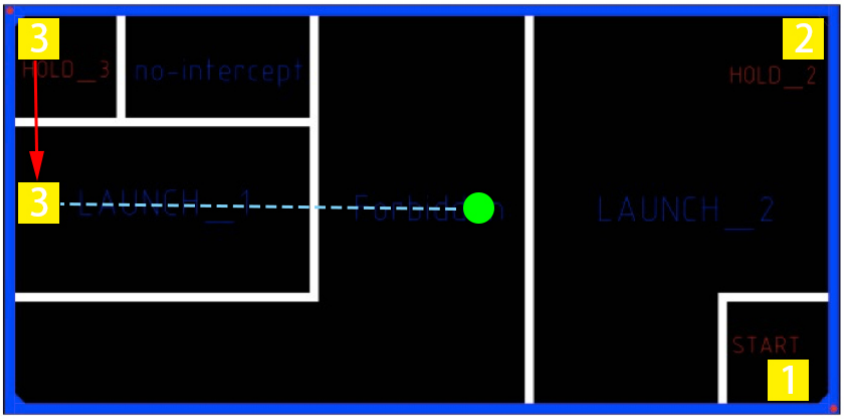
This is the whole schematic diagram of the Demo 5. The yellow triangle represents the autonomous robot and the green circle represents the ball.



Step 1: Red arrow represents the path that the robot pulling the ball. Then the blue arrow represents that the robot rotates 360 degrees and shoot the ball.



Step 2: Red arrow represents the path of Robot_2. It will start judge after the ball stops and then move following the path.



Step 3: Red arrow represents the path of Robot_3. The distance will depend on the path of the ball which is shown as blue dotted line.

3. Individual effort in the software implementation

3.1 Detailed design of the part of the software implemented

As for detailed design of the part of the software implemented, I design the software aimed for each demo, including demo 3, 4 and 5.

Demo 3

In demo 3, we modify and write the following program to implement the requirement of demo 3, including tracking the white line and make the car run.

The code is shown as bebow.

The content of Straight.cpp is shown as below.

```
#include <p32mx250f128d.h>

#define PERIOD 49999

namespace
{

int target = 5;                 // preset speed

class PID {
public:
```

```
        PID(int p, int in, int id, int d) {
                P = p;
                In = in; Id = id;       // I break "I" into two parts to avoid decimal
                D = d;
                acc = last = 0;
        }
        void reset(void) {acc = last = 0;}
        int control(int x, int target) {
                int error = x - target;
                int diff = x - last;
                last = x;
                acc += error;
                if (acc > 2000000000) acc = 2000000000;
                if (acc < -2000000000) acc = -2000000000;
                return - P * error - In * (acc >> Id) - D * diff;
        }
private:
        int P, In, Id, D, acc, last;
} pid_speed(2000,999,0,2000), pid_track(2000,999,0,2000);

int floor, flag;

class Counter {
public:
        Counter(void) { counts = last = 0; }
        void sample(short i) {
            short d = i - last;
            counts <<= 4;               // shift out old data
            counts |= d;                // record new data
            last = i;
        }
        int getCount(void) {                    // sum up data in "counts"
            int sum = 0, temp = counts;
```

```
            for (int i = 0; i < 8; i++) {
                sum += temp & 15;
                temp >>= 4;
            }
            return sum;
        }
private:
        short last;
        int counts;             // holds 8 pcs of 4-bit samples
} left, right;

unsigned short bound(int control) {
        if (control < 0) return 0;
        return control > PERIOD ? PERIOD : control;
}

} // anonymous namespace for privacy


int getFloorReading(void) { return floor; }
unsigned getLeft(void) { return left.getCount(); }
unsigned getRight(void) { return right.getCount(); }
bool checkFlag(void) { int f = flag; flag = 0; return f; }

extern "C"
{

void __attribute__ ((interrupt)) t3ISR(void) {    // don't forget isr.S
        static int count;
        if (++count == 8) { count = 0; flag = 1; }
        right.sample(TMR2);
        left.sample(TMR4);
        int rt = right.getCount(), lt = left.getCount();
```

```
    int control_speed = pid_speed.control(rt + lt, target);
    int left_num=0;
    int right_num=0;
  for(int i=1;i<4;i++)
  {
    if((floor&(1<<i))==0)
    {
        right_num++;
    }
    else break;
  }
  for(int i=6;i>3;i--)
      {
       if((floor&(1<<i))==0)
       {
            left_num++;
       }
       else break;
      }

    int control_track=pid_track.control(rt-lt,(right_num-left_num)/2);
    OC1RS = bound(control_speed - control_track);
    OC2RS = bound(control_speed + control_track);
    LATCSET = 1<<9;                    // turn on LED
    TMR5 = 0;
    T5CONbits.ON = 1;
    IEC0bits.T5IE = 1; //!
    IFS0bits.T3IF = 0;
}

void __attribute__ ((interrupt)) t5ISR(void) {
    SPI2CONbits.CKP = 1;        // load data
    LATCCLR = 1<<9;                    // turn off LED
```

```
        IEC1bits.SPI2RXIE = 1;              // enable interrupt receive done
        SPI2BUF = 0;                        // dummy write
        IEC0bits.T5IE = 0;                  // disable delay timer
        T5CONbits.ON = 0;
        IFS0bits.T5IF = 0;
}

void __attribute__ ((interrupt)) spi2ISR(void) {
        SPI2CONbits.CKP = 0;
        floor = SPI2BUF;                // read
        IEC1bits.SPI2RXIE = 0;              // disable interrupt
        IFS1bits.SPI2RXIF = 0;
}

}
```

The content of main.cpp is shown as below.

```
#include <p32mx250f128d.h>


extern "C" { // function provided by cdc stack


void USBDeviceInit(void);
void USBDeviceTasks(void);
void CDCDeviceTasks(void);
void putsUSBUSART(char *data);   // including trailing zero

} //extern "C"
```

```
extern "C"        // handlers required by cdc stack
{
void CDCRxChars(char *c, int length) {}        // you got a message from cdc
void CDCTxReadyNotify(void) {}                 // you may send message to cdc
}//extern


void USBTasks(void) {
        USBDeviceTasks();
        U1OTGIR = 0xFF;
        IFS1bits.USBIF = 0;
        CDCDeviceTasks();
}

int t3FLAG;

int main(void) {
        USBDeviceInit();
        while (1) {
            USBTasks();
            if (t3FLAG) {
                putsUSBUSART((char*)"Hello World!\r\n");
                t3FLAG = 0;                // Watch out! Here lies a bug.
            }
        }
}

extern "C" void __attribute__ ((interrupt)) t3ISR(void) {
        static int count;
        if (++count == 800) {   // 2 second
            count = 0;
            t3FLAG = 1;                // cannot print string at interrupt leve
```

```
                                    // so set a flag and let main() print the message
        LATAINV=1;                  // toggle LED
    }
    IFS0bits.T3IF = 0;
}
```

Demo 4

The code of demo 4 is shown as below, we use this C++ program to implement the task of demo 4.

```
#include <p32mx250f128d.h>
#define PERIOD 49999
int target;
int count,flag,floor;
//////////////////////////////////// USB STUFF
int speedSum, speedDif,ls,rs,sum,dif;
int ballx,bally,carx,cary,goalx,goaly,startx,starty,targetx,targety;
int lastCarx,lastCary,lastballx,lastbally;
int waitcount,waitcount2;
int hit=0;
int state=0;
int diff;

//"A2toD" to getPData are additional

int A2toD(char c)//ASCII 2 NUM
{
    if(c>0x60)
    return (c-0x61+10);
    else return c-0x30;
}
void getAData(char* n)//msg A
```

```
{
        int a=0,b=0,c=0,d=0;
        for(int i=0;i<3;i++){
            a=A2toD(n[6-i]);b=A2toD(n[9-i]);
            c=A2toD(n[12-i]);d=A2toD(n[15-i]);
            goalx|=a<<(4*i);goaly|=b<<(4*i);
            startx|=c<<(4*i);starty|=d<<(4*i);
        }
}
void getPData(char* n)//msg B
{
        lastCarx=carx;lastCary=cary;
        int a=0,b=0,c=0,d=0;
        for(int i=0;i<3;i++){
            a=A2toD(n[6-i]);b=A2toD(n[9-i]);
            c=A2toD(n[12-i]);d=A2toD(n[15-i]);
            ballx|=a<<(4*i);bally|=b<<(4*i);
            carx|=c<<(4*i);cary|=d<<(4*i);
        }
}



unsigned short bound(int control) {


        if (control < 0) return 0;
        return control > PERIOD ? PERIOD : control;
}


//Additional Methods
void findTarget(void)//should be used after the ball is stopped
{
        float tan=(goaly+79-bally)/(goalx-ballx);//dy of goal is 160pixel..?
```

```
        targetx=ballx-(int)10;targety=bally-(int)10*tan;//target is behind the ball pointing to middle
goal
}
bool result=false;
bool ballInPosition(void){

        int rangeXLeft=600, rangeXRight=0, rangeYTop=0, rangeYBottom=0;
        //&&(bally>rangeYBottom)
        if(ballx<rangeXLeft) result=true;
        else result=false;
        return result;
}
bool carReady=false;
bool carInPosition(void){
        if(carx>680){
            carReady=true;
        }
        return carReady;
}




extern "C" { // function provided by cdc stack

void USBDeviceInit(void);
void USBDeviceTasks(void);
void CDCDeviceTasks(void);
bool USBCDCTxREADY(void);
void putUSBUSART(char *data, unsigned char length);
void putsUSBUSART(char *data);
void USBTasks(void) {
        USBDeviceTasks();
```

```
        U1OTGIR = 0xFF;
        IFS1bits.USBIF = 0;
        CDCDeviceTasks();
}

} //extern "C"

////////////////////////////////////
////////////////////////////////////
namespace
{

class PID {
public:
        PID(int p, int in, int id, int d) {
                P = p;
                In = in; Id = id;       // I break "I" into two parts to avoid decimal
                D = d;
                acc = last = 0;
        }
        void reset(void) {acc = last = 0;}
        int control(int x, int target) {
                int error = x - target;
                int diff = x - last;
                last = x;
                acc += error;
                if (acc > 2000000000) acc = 2000000000;
                if (acc < -2000000000) acc = -2000000000;
                return - P * error - In * (acc >> Id) - D * diff;
        }
private:
        int P, In, Id, D, acc, last;
} pid_speed(20000,3,1,21450), pid_track(20000,3,1,21450);
```

```
class Counter {
public:
        Counter(void) { counts = last = 0; }
        void sample(short i) {
            short d = i - last;
            counts <<= 4;               // shift out old data
            counts |= d;                // record new data
            last = i;
        }
        int getCount(void) {                    // sum up data in "counts"
            int sum = 0, temp = counts;
            for (int i = 0; i < 8; i++) {
                sum += temp & 15;
                temp >>= 4;
            }
            return sum;
        }
private:
        short last;
        int counts;             // holds 8 pcs of 4-bit samples
} left, right;
}

static class RingBuffer {
public:
        RingBuffer(void) { read = write = 0; }
        char get(void) {
            char c = buffer[read];
            if (read == SIZE-1) read = 0;
            else read++;
            return c;
        }
```

```
    void save(char c) {
        buffer[write] = c;
        if (write == SIZE-1) write = 0;
        else write++;
    }
    int empty(void) { return read == write; }
private:
    enum {SIZE=128};        // an alternative to #define
    char buffer[SIZE];
    int read, write;
} tx;

static class ReadBuffer : public RingBuffer {
public:
    ReadBuffer(void) { free();}
    void save(char c) {
        RingBuffer::save(c);        // send to cdc (buffered) as usual
        if (!hold)                  // strip UDP message into buffer
            switch (ipd) {
            case IPD:               // phase: read length
                if (c == ':') ipd = len ? -1 : 0;
                else {
                    int i = c - '0';
                    if (i > 9) ipd = 0;
                    else len = len * 10 + i;
                }
                break;
            case -1:                // phase: save message
                buffer[index++] = c;
                if (!--len) {
                    buffer[index] = 0;
                    hold = true;    // message is ready for processing
                }
```

```
                    break;
                default:                    // phase: look for header "IPD,"
                    ipd = ipd << 8 | c;
                }
        }
        char *getString(void) { return hold ? buffer : 0; }
        void free(void) { ipd = len = index = 0; hold = false; }
private:
        enum {SIZE=100, IPD='I'<<24|'P'<<16|'D'<<8|',};
        int index, ipd, len;
        char buffer[SIZE];
        bool hold;
} rx;

extern "C"      // function required by cdc stack
{

void CDCRxChars(char *c, int length) {      // you got a message from cdc
        for (int i = 0; i < length; i++) {
            if (c[i] != '\n') {                     // skip linefeed
                tx.save(c[i]);
                if (c[i] == '\r') tx.save('\n');// add linefeed after carriage return
            }
        }
        IEC1bits.U2TXIE = 1;
}


void CDCTxReadyNotify(void) {                   // you may send message to cdc
        static char buffer[64];
        unsigned char len = 0;
        while (!rx.empty()) buffer[len++]= rx.get();
        if (len) putUSBUSART(buffer, len);
```

```
}



} //extern "C"
bool checkFlag(void) { int f = flag; flag = 0; return f; }
extern "C"
{
void __attribute__ ((interrupt)) t3ISR(void) {
        static int count;
        if (++count == 8) { count = 0; flag = 1; }
            right.sample(TMR2);
            left.sample(TMR4);
            int rt = right.getCount(), lt = left.getCount();
            int control_speed;
            int control_track;


                if (ballx>630)
                {
                    target=4;
                    LATCSET=0b101;
                    LATCSET=0b011;
                    diff=0;
                    //diff=(startx-goalx)/(starty-goaly/2)*(startx-ballx)-bally;
                    waitcount=0;
                    waitcount2=0;


                }
                if (ballx<=630 && ballx>570 && waitcount<=200)
                {
                    LATCCLR=0b100;LATCCLR=0b010;
                    target=2;
                    diff=0;
                    waitcount++;
```

```
}
if (ballx<=630 && ballx>570 && waitcount>200)
{
    LATCSET=0b101;
    LATCSET=0b011;
    target=4;
    diff=0;

}


if (ballx<=570 && waitcount2<=300)
{
    LATCCLR=0b100;LATCCLR=0b010;
    target=2;
    diff=0;
    waitcount2++;

}
if (ballx<=570 && waitcount2>300)
{

    LATCSET=0b101;
    LATCSET=0b011;
    target=10;
    diff=0;
    //print(4444, bally, carx,cary);
}
if (carx<540)
{
    target=4;
    LATCCLR=0b100;LATCCLR=0b010;
```

```
            //      print(5555, bally, carx,cary);


            }



        control_speed = pid_speed.control(rt + lt, target);
        control_track = pid_track.control(rt - lt, diff);
        OC2RS=bound(control_speed + control_track);
        OC1RS=bound(control_speed - control_track);

        LATCSET = 1<<9;                     // turn on LED
        TMR5 = 0;
        T5CONbits.ON = 1;
        IEC0bits.T5IE = 1; //!
        IFS0bits.T3IF = 0;

}


void __attribute__ ((interrupt)) u2ISR(void) {
    while (U2STAbits.URXDA) rx.save(U2RXREG);
    IFS1bits.U2RXIF = 0;
    if (IEC1bits.U2TXIE) {
        while (!U2STAbits.UTXBF) {
            if (tx.empty()) { IEC1bits.U2TXIE = 0; break; }
            else U2TXREG = tx.get();
        }
        IFS1bits.U2TXIF = 0;
    }
}
void __attribute__ ((interrupt)) t5ISR(void) {
    SPI2CONbits.CKP = 1;        // load data
```

```
        LATCCLR = 1<<9;                 // turn off LED
        IEC1bits.SPI2RXIE = 1;          // enable interrupt receive done
        SPI2BUF = 0;                    // dummy write
        IEC0bits.T5IE = 0;              // disable delay timer
        T5CONbits.ON = 0;
            //checkState();
        IFS0bits.T5IF = 0;
}

void __attribute__ ((interrupt)) spi2ISR(void) {
        SPI2CONbits.CKP = 0;
        floor = SPI2BUF;            // read
        IEC1bits.SPI2RXIE = 0;          // disable interrupt
        IFS1bits.SPI2RXIF = 0;
}

} // ISRs
//namespace
//{

void bits(int b, char *c) {             // integer to bit pattern: 00001111
        int mask = 1;
        for (int i = 0; i < 8; i++) {
            c[i] = b & mask ? '1' : '0';
            mask <<= 1;
        }
}

void detoChar(int u, char *c) {     // unsigned short to ASCII string
        int p=u;
        for (int i = 2; i >= 0; i--){
            c[i]=p%10+'0';                          // u must be unsigned short
            p/=10;
```

```
        }
}


char string[] = "                              \r";

void print(int ax, int ay, int bx,int by)
{
            detoChar(ax, &string[0]);
            detoChar(ay, &string[4]);
            detoChar(bx, &string[8]);
            detoChar(by, &string[12]);
            putsUSBUSART(string);
}

int hextodeci(char c)
{
        if(c<='9'&&c>='0')return c-'0';
        else return c-'a'+10;
}

int main(void)
{
        static int count;
        char *s, buffer[128];
        USBDeviceInit();
        while (1)
        {
            USBTasks();
            //checkState();
            if(!ballInPosition()){
            if (checkFlag())
            {      // flag set every 20 ms
```

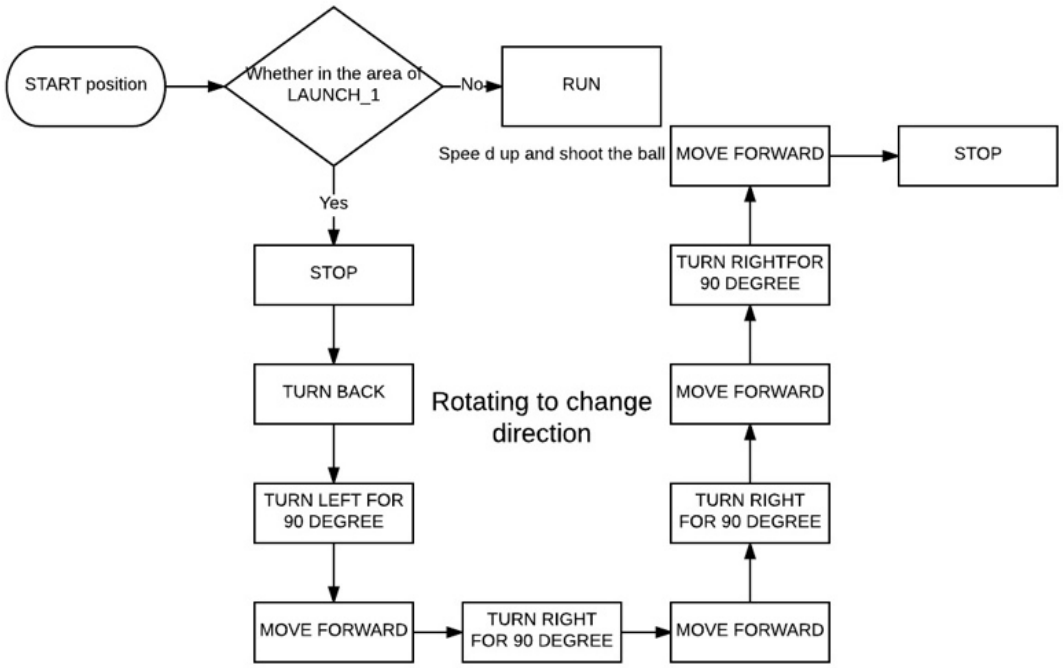
```
                //print(123, 123, 123,123);// toggle LED indicator every second
                if (++count == 50) { count = 0; LATAINV = 1; }
    }
    }
    else{
        if (checkFlag())
                {    // flag set every 20 ms
                    //print(123, 123, 123,123);// toggle LED indicator every second
                    if (++count == 50) { count = 0; LATAINV = 1; }
                }
    }
//  if (USBCDCTxREADY())
//  {
        s = rx.getString();

        if (s)
        {
            unsigned char len = 0;
            while ((buffer[len] = s[len])) len++;
            lastballx=ballx;lastbally=bally;lastCarx=carx;lastCary=cary;
            if(buffer[0=='P']){

ballx=hextodeci(buffer[4])*16*16+hextodeci(buffer[5])*16+hextodeci(buffer[6]);

bally=hextodeci(buffer[7])*16*16+hextodeci(buffer[8])*16+hextodeci(buffer[9]);

carx=hextodeci(buffer[10])*16*16+hextodeci(buffer[11])*16+hextodeci(buffer[12]);

cary=hextodeci(buffer[13])*16*16+hextodeci(buffer[14])*16+hextodeci(buffer[15]);}
            if(buffer[0=='A']){

goalx=hextodeci(buffer[4])*16*16+hextodeci(buffer[5])*16+hextodeci(buffer[6]);
```

```
        goaly=hextodeci(buffer[7])*16*16+hextodeci(buffer[8])*16+hextodeci(buffer[9]);

        startx=hextodeci(buffer[10])*16*16+hextodeci(buffer[11])*16+hextodeci(buffer[12]);

        starty=hextodeci(buffer[13])*16*16+hextodeci(buffer[14])*16+hextodeci(buffer[15]);}
                        print(ballx, bally, carx,cary);
                        //if (ballx>600){target=3;}
                        //else{target=0;}
                        //print(ballx, bally, carx,cary);
                        rx.free();

                }
        //   }

        }
}
```

Demo 5

In demo 5, I design the schematic diagram to realize the software implement.

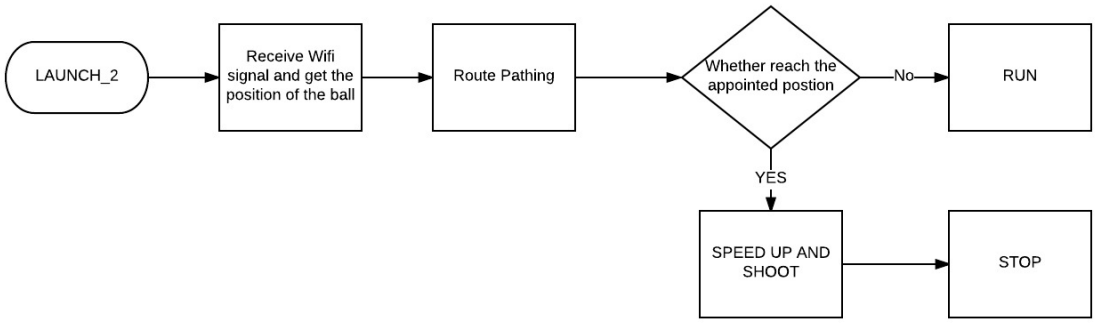
Task1 Diagram



Flow Chart 1

Firstly, our Robot_1 start pulling the ball from the START position and detect the position of LAUNCH_1 by Wi-Fi continuously and automatically. If the robot arrives the appointed area, it will stop and turn back. Later, the robot will rotate for 360 degrees to adjust the shooting direction. Finally, it will speed up to shoot the ball to the LAUNCH_2.

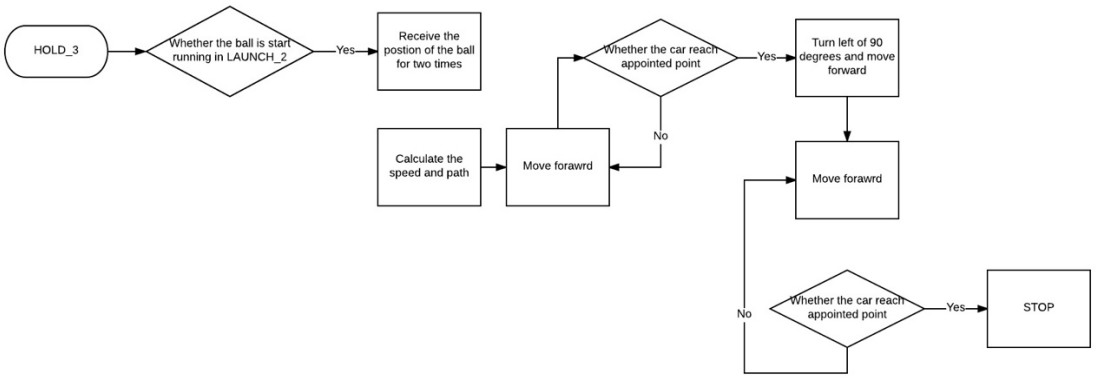
Task 2 Diagram



Flow Chart 2

Next, out Robot_2 will receive the position of ball by Wi-Fi. If the ball is in the area of LAUNCH_2 and it stop for a while, the Robot_2 will start moving by the route which has been assigned in the program. We just use two directions (Forward and turn left) to reach the position of ball. After calculation, the Robot_2 can get the order that how much it should go forward and how much it should turn left. Later, it will move following the path and when it move forward, it will speed up and shoot the ball.

Task 3 diagram



Flow Chart 3

Last but not the least, the Robot_3 will judge whether the ball has already shooting from LAUNCH_3. It will receive the position of the ball and Robot_2. When Robot_2 start running, it starts judge. If the ball moves forward to the direction of Robot_3, Robot_3 start measure the path of the ball by receive the time interval and position. After that, it starts moving forwards to the appointed point and stop. Finally, it will intercept the ball's movement.

The user.app to auto-connect with Wi-Fi module and gain the position of the ball. The code is shown as below.

```
#include "noos.h"
#include "udp.h"
```

```
#include "uart2.h"

using namespace noos;

void (*userTick)(void);

namespace
{

void usshort(unsigned u, char *c) {          // unsigned short to ASCII string
        int i, t, r;
        for (i = 0; i < 3; i++) c[i] = ' ';     // clear string
        c[i] = '0';                                     // default zero
        while (u) {                                     // u must be unsigned short < 4096
            t = u / 10;
            r = u - t * 10;
            u = t;
            c[i--] = '0' + r;
        }
}

const char heading[] =
            "\r\nSerial a1            a2            ball        orange     red          blue\n";
char display[] =
                "\r   xxx    xxx yyy    xxx yyy    xxx yyy    xxx yyy    xxx yyy    xxx yyy";
            //   01234567890 2345678901 3456789012 4567890123 5678901234 67

void updateDisplay(void) {
        usshort(udp::getSerial(), &display[3]);
        point p = udp::getA1();
        usshort(p.x, &display[8]);
        usshort(p.y, &display[12]);
        p = udp::getA2();
```

```
        usshort(p.x, &display[17]);
        usshort(p.y, &display[21]);
        p = udp::getBall();
        usshort(p.x, &display[26]);
        usshort(p.y, &display[30]);
        p = udp::getOrange();
        usshort(p.x, &display[35]);
        usshort(p.y, &display[39]);
        p = udp::getRed();
        usshort(p.x, &display[44]);
        usshort(p.y, &display[48]);
        p = udp::getBlue();
        usshort(p.x, &display[53]);
        usshort(p.y, &display[57]);
}

}//anonymous

void userMain(void) {
        LED(BLINK_FAST);
        LOOP (!udp::getSerial()){
            if (isEvent(BUTTON_PRESS)) LED(BLINK_SLOW);
            if (isEvent(BUTTON_RELEASE)) {
                if (LED(BLINK_FAST)) U2::print("AT+CIPSTART=\"UDP\",\"0\",0,3105,2\r\n");
                else U2::print("AT+CIPSTART=\"UDP\",\"0\",0,3106,2\r\n");
            }
        }
        LED(ON);
        userTick = [](){
            static int count = 0, mode = 0;
            if (count++ == 8) {
                count = 0;
                switch (mode) {
```

```
                case 0:
                    if (isEvent(BUTTON_PRESS)) {
                        mode = 2; U2::mute(true);
                    }
                    break;
                case 1:
                    updateDisplay(); print(display);
                    if (isEvent(BUTTON_PRESS)) {
                        mode = 0; U2::mute(false);
                    }
                    break;
                default:
                    mode = 1;
                    print((char*)heading);
                }
            }
        };
}
```

Line.app is the calculate the direction of the start position and end position. The code is shown as below.

```
#include "Line.h"

namespace
{

//http://www.embedded.com/electronics-blogs/programmer-s-toolbox/4219659/Integer-Square-Roots
int sqrt(unsigned a) {
    unsigned rem = 0;
    unsigned root = 0;
    for (int i=0; i < 16; i++) {
```

```
            root <<= 1;
            rem = ((rem << 2) + (a >> 30));
            a <<= 2;
            root++;
            if (root <= rem) {
                rem -= root;
                root++;
            } else root--;
        }
        return root >> 1;
}

point start, end;
int normal;

} // anonymous

namespace Line
{

void set(point s, point e) {
        start = s;
        end.x = e.x - s.x; end.y = e.y - s.y;
        normal = sqrt(end.x * end.x + end.y * end.y);
}

int distance(point p) {
        if (!normal) return 0;
        return ((p.x - start.x)*(end.y)- (p.y - start.y)*(end.x))/normal;
}

} //Line
```

udp.app and uart2.app is to gain the position of the car and the ball. The code is shown as below.

```
#include <p32mx250f128d.h>


static class RingBuffer {
public:
        RingBuffer(void) { read = write = 0; }
        char get(void) {
            char c = buffer[read];
            if (read == SIZE-1) read = 0;
            else read++;
            return c;
        }
        void save(char c) {
            buffer[write] = c;
            if (write == SIZE-1) write = 0;
            else write++;
        }
        int empty(void) { return read == write; }
private:
        enum {SIZE=128};        // an alternative to #define
        char buffer[SIZE];
        int read, write;
} tx;


namespace udp { bool parse(char*); }
static class ReadBuffer : public RingBuffer {
public:
        ReadBuffer(void) { ipd = len = index = 0; _mute = false; }
        void save(char c) {
            if (!_mute) RingBuffer::save(c);       // send to cdc (buffered) as usual
            switch (ipd) {                  // strip UDP message into buffer
```

```
            case IPD:                   // phase: read length
                if (c == ':') ipd = len ? -1 : 0;
                else {
                    int i = c - '0';
                    if (i > 9) ipd = 0;
                    else len = len * 10 + i;
                }
                break;
            case -1:                    // phase: save message
                buffer[index++] = c;
                if (!--len) {
                    ipd = index = buffer[index] = 0;
                    udp::parse(buffer);  // message is ready for processing
                }
                break;
            default:                    // phase: look for header "IPD,"
                ipd = ipd << 8 | c;
            }
        }
        void mute(bool b) { _mute = b; }
private:
        enum {SIZE=100, IPD='I'<<24|'P'<<16|'D'<<8|',};
        int index, ipd, len;
        char buffer[SIZE];
        bool _mute;
} rx;

extern "C" void __attribute__ ((interrupt)) u2ISR(void) {
        while (U2STAbits.URXDA) rx.save(U2RXREG);
        IFS1bits.U2RXIF = 0;
        if (IEC1bits.U2TXIE) {
            while (!U2STAbits.UTXBF) {
                if (tx.empty()) { IEC1bits.U2TXIE = 0; break; }
```

```
                else U2TXREG = tx.get();
            }
            IFS1bits.U2TXIF = 0;
        }
}

extern "C"      // function required by cdc stack
{

void CDCRxChars(char *c, int length) {          // you got a message from cdc
        for (int i = 0; i < length; i++) {
            if (c[i] != '\n') {                         // skip carriage return
                tx.save(c[i]);
                if (c[i] == '\r') tx.save('\n');// add linefeed after carriage return
            }
        }
        IEC1bits.U2TXIE = 1;
}

void putUSBUSART(char *data, unsigned char length);
void CDCTxReadyNotify(void) {                       // you may send message to cdc
        static char buffer[64];
        unsigned char len = 0;
        while (!rx.empty()) buffer[len++] = rx.get();
        if (len) putUSBUSART(buffer, len);
}

}

namespace U2
{

void print(const char *s) {       while (*s) tx.save(*s++); IEC1bits.U2TXIE = 1; }
```

```
void mute(bool b) { rx.mute(b); }


}
#include "udp.h"

namespace
{

int serial;
point ball, a1, a2, orange, red, blue;

int value(char c) {
        int v = c -'0';
        if (v < 10) return v;
        v = c - 'W';
        if (v < 10) return -1;
        if (v > 15) return -1;
        return v;
}

short value(char *s) { return value(s[0])<<8|value(s[1])<<4|value(s[2]); }

bool convert2(char *s, point *p1, point *p2) {
        point b, c;
        b = {value(&s[0]),value(&s[3])};
        c = {value(&s[6]),value(&s[9])};
        if ((b.x|b.y|c.x|c.y) < 0) return false;                    // number error
        *p1 = b; *p2 = c;
        return true;
}

bool parseA(char *s) {
        if (s[18]) return false;                                    // too long
```

```
        int cs = 0;
        for (int i = 4; i < 16; i++) cs ^= s[i];
        if (cs != (value(s[16])<<4|value(s[17]))) return false;  // cs error
        cs = value(&s[1]);
        if (cs == -1) return false; else serial = cs;
        return convert2(&s[4], &a1, &a2);
}

bool parseP(char *s) {
        for (int i = 18; i < 30; i++) if (!s[i]) return false;  // too short
        if (s[30]) return false;                                // too long
        int cs = 0;
        for (int i = 4; i < 28; i++) cs ^= s[i];
        if (cs != (value(s[28])<<4|value(s[29]))) return false;  // cs error
        cs = value(&s[1]);
        if (cs == -1) return false; else serial = cs;
        if (!convert2(&s[4], &ball, &orange)) return false;
        return convert2(&s[16], &red, &blue);
}

}
//Pfffb00b11c00c11c00c11c00c11cc
//01234567890 2345678901 3456789

namespace udp
{

bool parse(char *s) {
        for (int i = 0; i < 18; i++) if (!s[i]) return false;    // too short
        if (s[0] == 'A') return parseA(s);
        if (s[0] == 'P') return parseP(s);
        return false;
}
```

```
int getSerial(void) { return serial; }
point getBall(void) { return ball; }
point getOrange(void) { return orange; }
point getRed(void) { return red; }
point getBlue(void) { return blue; }
point getA1(void) { return a1; }
point getA2(void) { return a2; }

}
```

3.2 Presentation of the software in modular form

In the software, it includes several different functions to realize different control. It includes user.app to auto connect with Wi-Fi and wheel.app to facility the car run. Then I use main.app to control the main function of the car, which includes Timer3 interrupt to control the direction and speed of the car to reach the position of the ball. Besides, uart.app and udp.app is to connect with Wi-Fi and gain the interact position of the car and the ball. We use the if-justice to justice how the car will act and run under different situation.

3.3 Explanation of each software module

There are several different software modules in the cpp code and I will just introduce one of them. For example,

```
void save(char c) {
        if (!_mute) RingBuffer::save(c);       // send to cdc (buffered) as usual
        switch (ipd) {                  // strip UDP message into buffer
```

```
        case IPD:                   // phase: read length
            if (c == ':') ipd = len ? -1 : 0;
            else {
                int i = c - '0';
                if (i > 9) ipd = 0;
                else len = len * 10 + i;
            }
            break;
        case -1:                    // phase: save message
            buffer[index++] = c;
            if (!--len) {
                ipd = index = buffer[index] = 0;
                udp::parse(buffer);  // message is ready for processing
            }
            break;
        default:                    // phase: look for header "IPD,"
            ipd = ipd << 8 | c;
        }
    }
```

This function is to create buffer and save the message to the buffer. For the receiving data includes position in characters, it will divide it and save as character into different buffers.

References:

1. *Garlan & Shaw (1994). "An Introduction to Software Architecture" (PDF). Retrieved 2012-09-13.*
2. Fowler, M. (2003). "Design – Who needs an architect?". *IEEE Software*. **20** (5): 11–44.

Jansen, A.; Bosch, J. (2005). "Software Architecture as a Set of Architectural Design Decisions". *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. p. 109.

3. Ralph, P. and Wand, Y. (2009). A proposal for a formal definition of the design concept. In Lyytinen, K., Loucopoulos, P., Mylopoulos, J., and Robinson, W., editors, Design Requirements Workshop (LNBIP 14), pp. 103–136. Springer-Verlag, p. 109 doi:10.1007/978-3-540-92966-6_6.
4. *Freeman, Peter; David Hart (2004). "A Science of design for software-intensive systems". Communications of the ACM. **47** (8): 19–21 [20]. doi:10.1145/1012037.1012054.*
5. *Booch, Grady; et al. (2004). Object-Oriented Analysis and Design with Applications (3rd ed.). MA, USA: Addison Wesley. ISBN 0-201-89551-X. Retrieved 30 January 2015.*
6. *Suryanarayana, Girish (November 2014). Refactoring for Software Design Smells. Morgan Kaufmann. p. 258. ISBN 978-0128013977. Retrieved 31 January 2015.*
7. *Carroll, ed., John (1995). Scenario-Based Design: Envisioning Work and Technology in System Development. New York: John Wiley & Sons. ISBN 0471076597.*
8. *Bell, Michael (2008). "Introduction to Service-Oriented Modeling". Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. ISBN 978-0-470-14111-3.*
9. *Judith Bishop. "C# 3.0 Design Patterns: Use the Power of C# 3.0 to Solve Real-World Problems". C# Books from O'Reilly Media. Retrieved 2012-05-15. If you want to speed up the development of your .NET applications, you're ready for C# design patterns -- elegant, accepted and proven ways to tackle common programming problems.*
10. *Dijkstra, E. W. (1988). "On the cruelty of really teaching computing science". Retrieved 2014-01-10.*
11. *Knuth, Donald E. (1989). "Notes on the Errors of TeX" (PDF).*

12. *Ralph, P., and Wand, Y. A Proposal for a Formal Definition of the Design Concept. In, Lyytinen, K., Loucopoulos, P., Mylopoulos, J., and Robinson, W., (eds.), Design Requirements Engineering: A Ten-Year Perspective: Springer-Verlag, 2009, pp. 103-136*