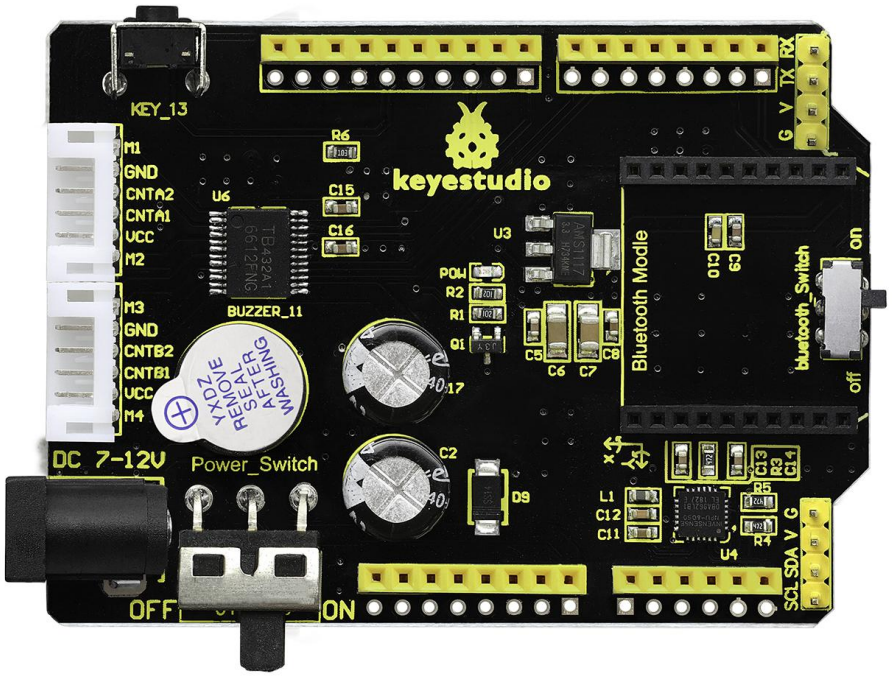
Keyestudio Balance Car Shield V3



Description:

You might envy those who can easily drive the gravity balance car on the street. So cool! How to DIY a mini balance car on your own?

You are able to combine this shield with UNO R3 control board to make a balance car based on Arduino development platform.

The shield comes with a XBEE Bluetooth socket, so you can connect the Bluetooth APP to easily control the balance car. With the APP, you can choose both key control and gravity control mode, besides, you can also adjust the balance angle and change PID parameters.

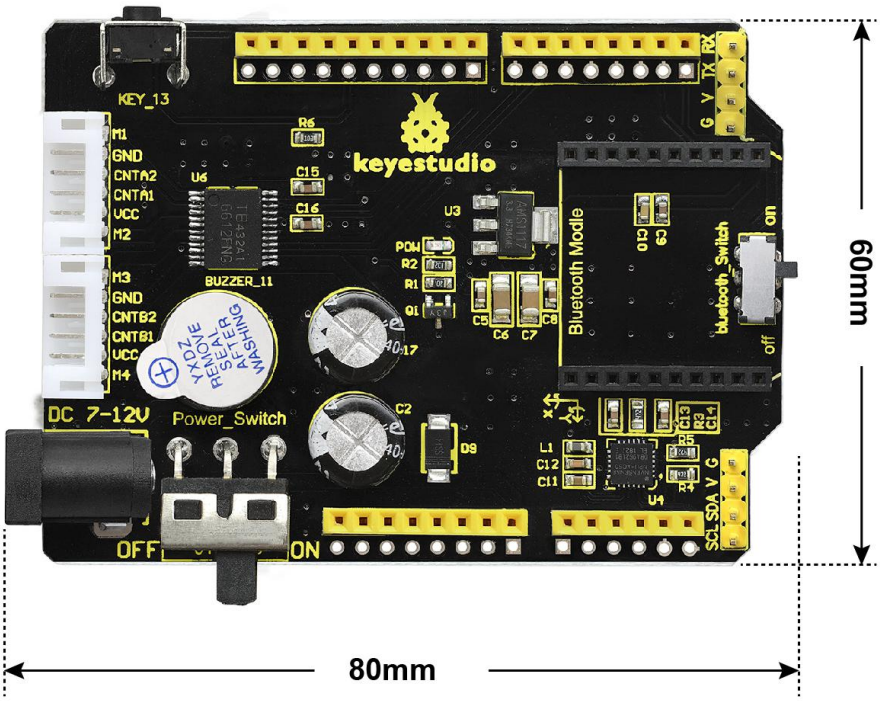
The shield is also compatible with a 12V permanent magnet brush motor with a reduction ratio of 1:30 and a locked torque of 7.5kg.cm.

After metal gear reduction output, it has stronger horsepower than ordinary motors. It has low power consumption, high torque and quick start, etc.

For easy settings, the balance car shield integrates a power switch, XBEE communication switch, built-in active buzzer (D11 control), D13 button controlled device, a serial communication and an I2C communication pin.

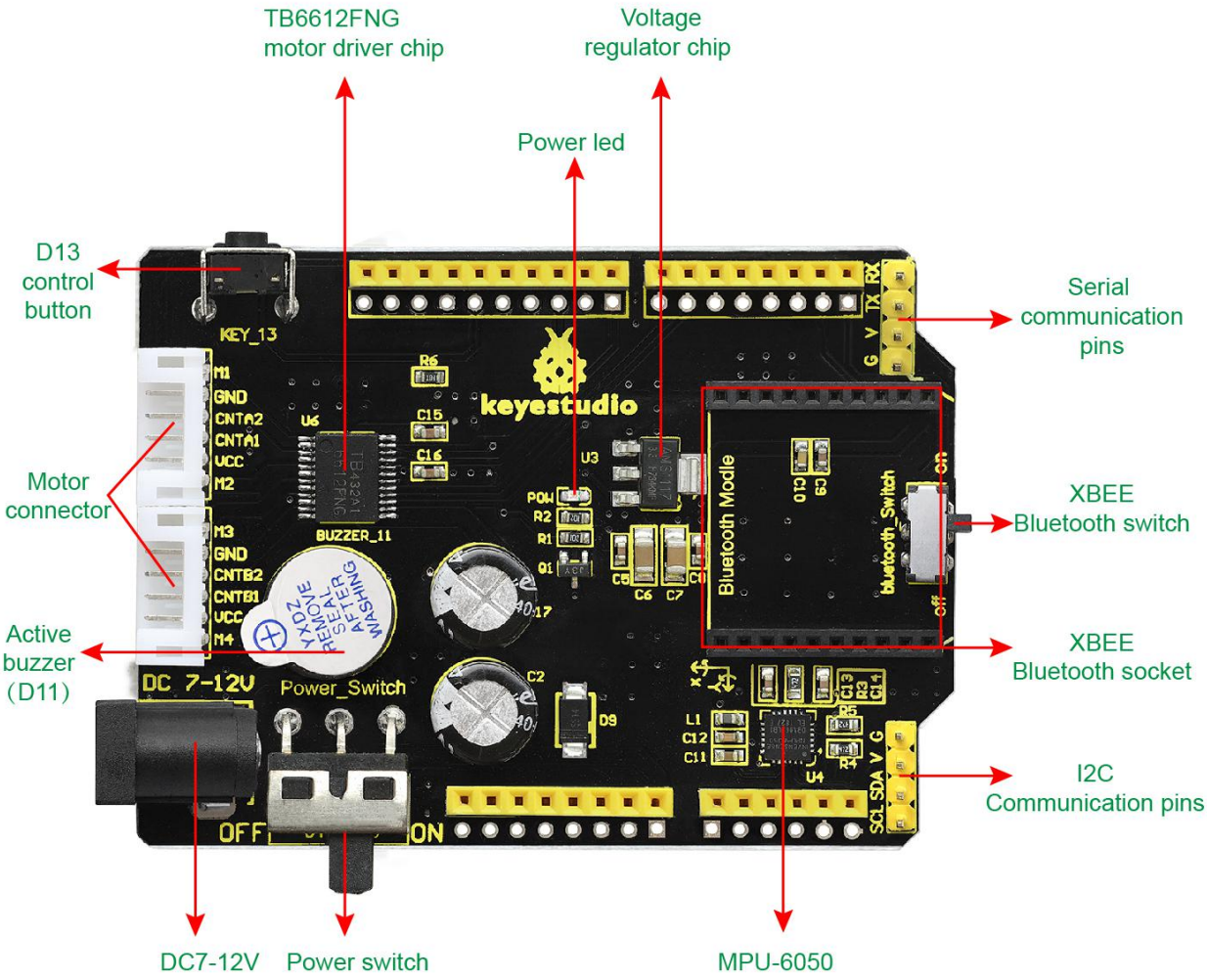
Technical Details:

- Operating Voltage: DC 7-12V
- Operating Current: 800mA
- Compatible with UNO R3 board
- Complying with ROHS
- Dimensions: 80mm*60mm*28mm
- Weight: 28.6g



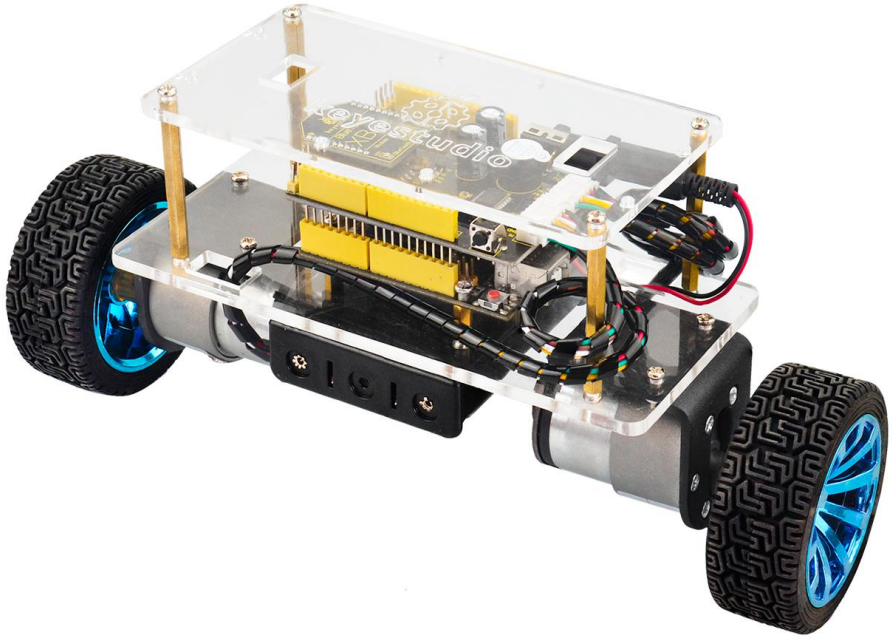
Element and Pin Interfaces:

Here is an explanation figure of what every element and interface of the board has:



Hookup Guide:

It is meaningless to test the shield alone. You should stack the shield onto the control board and connect other external devices to build a balance car. Check the picture below.



Reference Code for Balance Car:

```
**********************************************************
/*
Contact information
-------------------
*/

#include <Wire.h>
#include "Kalman.h"
#include "ComPacket.h"
#include "I2C.h"
#include <EEPROM.h>
//#include "music.h"
#include "MyEEprom.h"
#include <MPU6050.h>
#include <PinChangeInt.h> // for RC reciver

//#define USEEEPROM     //uncomment to use eeprom for storing PID parameters
```

```
#define RCWork true

#define mpuAddress 0x68

//Rc receiver   //2 channels
#define UP_DOWN_IN_PIN   16
#define  LEFT_RIGHT_IN_PIN  17
#define  AUX_IN_PIN 14  //is it use RC ,high level :no
//bool RCWork = false;

volatile uint8_t bUpdateFlagsRC = 0;
#define UP_DOWN_FLAG   0b10
#define LEFT_RIGHT_FLAG 0b100

uint32_t UpDownStart;
uint32_t LeftRightStart;

volatile uint16_t UpDownEnd = 0;
volatile uint16_t LeftRightEnd = 0;

int UpDownIn;
int LeftRightIn;

Vector  rg, zzl;
Vector_accel calibrate_Accel;
MPU6050 mpu;


//Speaker Mode
#define SPK_OFF  0x00
#define SPK_ON  0x01
#define SPK_ALARM 0x02
```

```
Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;
ComPacket SerialPacket;

/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int16_t tempRaw;

double gyroXangle, gyroYangle; // Angle calculate using the gyro only
double compAngleX, compAngleY; // Calculated angle using a complementary filter
double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

uint32_t timer;
uint8_t i2cData[14]; // Buffer for I2C data

#define BUZZER 6
#define LED 13
char val;
                    //motor  define
#define PWM_L 11  //M1
#define PWM_R 9   //M2
#define DIR_L1 10
#define DIR_L2 12
#define DIR_R1 8
#define DIR_R2 7
                    //encoder define
#define SPD_INT_R 3   //interrupt
#define SPD_PUL_R 4
#define SPD_INT_L 2   //interrupt
#define SPD_PUL_L 5
```

```
int Speed_L, Speed_R;
int Position_AVG;
int Position_AVG_Filter;
int Position_Add;
int pwm, pwm_l, pwm_r;
double Angle_Car;
double Gyro_Car;

double KA_P, KA_D;
double KP_P, KP_I;
double K_Base;
struct EEpromData SavingData;
struct EEpromData ReadingData;

int Speed_Need, Turn_Need;
int Speed_Diff, Speed_Diff_ALL;

uint32_t LEDTimer;
bool blinkState = false;

uint32_t calTimer;
uint32_t delayTimer;
uint32_t  lampTimer;



byte speakMode;

//Music play
int tonelength;
int tonePin = 13;
int tonecnt = 0;
```

```
uint32_t  BuzzerTimer;


void setup() {
    //mpu.begin(MPU6050_SCALE_250DPS, MPU6050_RANGE_2G);
    Serial.begin(9600);
    Init();
    //mySerial.begin(9600);
    Wire.begin();
    TWBR = ((F_CPU / 400000L) - 16) / 2; // Set I2C frequency to 400kHz


    i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
    i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro filtering, 8 KHz sampling
    i2cData[2] = 0x00; // Set Gyro Full Scale Range to ▯▯ 250deg/s
    i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ▯▯ 2g


    while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers at once
    while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope reference and disable sleep mode


    while (i2cRead(0x75, i2cData, 1));
    if (i2cData[0] != 0x68) { // Read "WHO_AM_I" register
        Serial.print(F("Error reading sensor"));
        while (1);
    }


    delay(200); // Wait for sensor to stabilize


                /* Set kalman and gyro starting angle */
    while (i2cRead(0x3B, i2cData, 6));
    accX = (i2cData[0] << 8) | i2cData[1];
    accY = (i2cData[2] << 8) | i2cData[3];
```

```
    accZ = (i2cData[4] << 8) | i2cData[5];

    // atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2
    // It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
    double roll = atan2(accY, accZ) * RAD_TO_DEG;
    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
    double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
    double pitch = atan2(-accX, accZ) * RAD_TO_DEG - calibrate_Accel.pitch;
#endif

    kalmanX.setAngle(roll); // Set starting angle
    kalmanY.setAngle(pitch);
    gyroXangle = roll;
    gyroYangle = pitch;
    compAngleX = roll;
    compAngleY = pitch;

    /*
    if( analogRead(AUX_IN_PIN) > 1000)
    {
    RCWork = true;
    }
    else
    {
    RCWork = false;
    }
    */

    // using the PinChangeInt library, attach the interrupts
    // used to read the channels
```

```
Serial.print("RCWork: "); Serial.println(RCWork);
if (RCWork == true)
{
    PCintPort::attachInterrupt(UP_DOWN_IN_PIN, calcUpDown, CHANGE);
    PCintPort::attachInterrupt(LEFT_RIGHT_IN_PIN, calcLeftRight, CHANGE);
}
PCintPort::attachInterrupt(SPD_INT_L, Encoder_L, FALLING);
PCintPort::attachInterrupt(SPD_INT_R, Encoder_R, FALLING);

timer = micros();
LEDTimer = millis();
lampTimer = millis();

// tonelength = sizeof(tune)/sizeof(tune[0]);

zzl = mpu.calibrateGyro_zzl();
Serial.print("calibrateGyro_zzl = ");
Serial.print(zzl.YAxis);

calibrate_Accel = mpu.calibrate_accel_pitch_and_roll();
Serial.print(" calibrate_Accel = ");
Serial.print(calibrate_Accel.pitch);

digitalWrite(BUZZER, HIGH);
delay(1000);
digitalWrite(BUZZER, LOW);
delay(1000);

while (true)
{
```

```
        val = Serial.read();
        if (val == 0x53)
        {
            digitalWrite(BUZZER, HIGH);
            delay(500);
            digitalWrite(BUZZER, LOW);
            delay(500);
            digitalWrite(BUZZER, HIGH);
            delay(500);
            digitalWrite(BUZZER, LOW);
            delay(500);
            digitalWrite(BUZZER, HIGH);
            delay(500);
            digitalWrite(BUZZER, LOW);
            delay(500);
            Serial.println("keyes");
            break;
        }
    }
}

void loop() {

    double DataAvg[3];
    double AngleAvg = 0;

    DataAvg[0] = 0; DataAvg[1] = 0; DataAvg[2] = 0;

    while (1)
    { //Serial.println("here");
        if (UpdateAttitude())
        {
```

```
            DataAvg[2] = DataAvg[1];
            DataAvg[1] = DataAvg[0];
            DataAvg[0] = Angle_Car;
            AngleAvg = (DataAvg[0] + DataAvg[1] + DataAvg[2]) / 3;
            if (AngleAvg < 40 || AngleAvg > -40) {
                PWM_Calculate();
                Car_Control();
            }
        }
        //UserComunication();
        SerialEvent();
        ProcessRC();
        MusicPlay();
    }


}

void SerialEvent()
{
    char a;
if (Serial.available())
    {
        a = Serial.read();
        //myserial.print(a);
        //myserial.print("");
        switch (a)
        {
        case 0x55:
            //myserial.println(a);
            Speed_Need = 0; Turn_Need = -10;
```

```
            //Serial.print(Speed_Need); Serial.println(Turn_Need);
            break;
        case 0x44:
            //myserial.println(a);
            Speed_Need = 0; Turn_Need = 10;
            //Serial.print(Speed_Need); Serial.println(Turn_Need);
            break;
        case 0x4c:
            //myserial.println(a);
            Speed_Need = 10; Turn_Need = 0;
            //Serial.print(Speed_Need); Serial.println(Turn_Need);
            break;
        case 0x52:
            //myserial.println(a);
            Speed_Need = -10; Turn_Need=0;
            //Serial.print(Speed_Need); Serial.println(Turn_Need);
            break;
        case 0x53:
            //myserial.println(a);
            Speed_Need = 0; Turn_Need = 0;
            //Serial.print(Speed_Need); Serial.println(Turn_Need);
            break;

        default:
            break;
        }
    }
}

bool StopFlag = true;

void PWM_Calculate()
```

```
{

    float ftmp = 0;
    //Serial.print(" Speed_L = ");
    //Serial.print(Speed_R);
    //Serial.print(" Speed_R = ");
    //Serial.print(Speed_L);

    //Serial.print(" ftmp = ");
    //Serial.print(ftmp);

    //Serial.print(" S_L+S_R= ");
    //Serial.print(Speed_L + Speed_R);


    ftmp = (Speed_L + Speed_R) * 0.5;

    //Serial.print(" ftmp = ");
    //Serial.print(ftmp);

    if (ftmp > 0)
        Position_AVG = ftmp + 0.5;
    else
        Position_AVG = ftmp - 0.5;

    Speed_Diff = Speed_L - Speed_R;
    Speed_Diff_ALL += Speed_Diff;

    //Serial.print(" Speed_Diff_- = ");
    //Serial.print(Speed_Diff);
    //Serial.print(" Position_AVG = ");
    //Serial.print(Position_AVG);
```

```
    //Serial.print(" Speed_Diff_ALL = ");
    //Serial.print(Speed_Diff_ALL);

    //  Position_AVG_Filter *= 0.5;     //speed filter
    //  Position_AVG_Filter += Position_AVG * 0.5;
    Position_AVG_Filter = Position_AVG;

    Position_Add += Position_AVG_Filter;  //position
                                          //Serial.print(Speed_Need);     Serial.print("\t");
Serial.println(Turn_Need);

    Position_Add += Speed_Need;  //

    Position_Add = constrain(Position_Add, -800, 800);

    //Serial.print(" Position_Add = ");
    //Serial.println(Position_Add);

    //   Serial.print(Position_AVG_Filter);  Serial.print("\t"); Serial.println(Position_Add);
    //Serial.print((Angle_Car-5 + K_Base)* KA_P);  Serial.print("\t");
    pwm = (Angle_Car - 5 + K_Base)* KA_P   //P
        + Gyro_Car * KA_D //D
        + Position_Add * KP_I    //I
        + Position_AVG_Filter * KP_P; //P
                                  // Serial.println(pwm);

                                  // if(Speed_Need ==0)
                                  //   StopFlag = true;

    if ((Speed_Need != 0) && (Turn_Need == 0))
    {
        if (StopFlag == true)
```

```
        {
            Speed_Diff_ALL = 0;
            StopFlag = false;
        }
        pwm_r = int(pwm + Speed_Diff_ALL);
        pwm_l = int(pwm - Speed_Diff_ALL);

    }
    else
    {
        StopFlag = true;
        pwm_r = pwm + Turn_Need; //
        pwm_l = pwm - Turn_Need;

    }

    Speed_L = 0;
    Speed_R = 0;
}

void Car_Control()
{
    if (pwm_l<0)
    {
        digitalWrite(DIR_L1, HIGH);
        digitalWrite(DIR_L2, LOW);
        pwm_l = -pwm_l;  //cchange to positive
    }
    else
    {
        digitalWrite(DIR_L1, LOW);
        digitalWrite(DIR_L2, HIGH);
```

```
    }

    if (pwm_r<0)
    {
        digitalWrite(DIR_R1, LOW);
        digitalWrite(DIR_R2, HIGH);
        pwm_r = -pwm_r;
    }
    else
    {
        digitalWrite(DIR_R1, HIGH);
        digitalWrite(DIR_R2, LOW);
    }
    if (Angle_Car > 45 || Angle_Car < -45)
    {
        pwm_l = 0;
        pwm_r = 0;
    }
    // pwm_l = pwm_l;  //adjust Motor different
    //  pwm_r = pwm_r;
    // Serial.print(pwm_l);  Serial.print("\t"); Serial.println(pwm_r);
    analogWrite(PWM_L, pwm_l>180 ? 180 : pwm_l);
    analogWrite(PWM_R, pwm_r>180 ? 180 : pwm_r);

    /*
    Serial.println("pwm:"); Serial.print(pwm);
    Serial.print("\t");
    Serial.print("pwm_L:"); Serial.print(pwm_l);
    Serial.print("pwm_R:"); Serial.print(pwm_r);
    */
}
```

```
void UserComunication()
{
    MySerialEvent();
    if (SerialPacket.m_PackageOK == true)
    {
        SerialPacket.m_PackageOK = false;
        switch (SerialPacket.m_Buffer[4])
        {
        case 0x01:break;
        case 0x02: UpdatePID(); break;
        case 0x03: CarDirection(); break;
        case 0x04: SendPID(); break;
        case 0x05:
            SavingData.KA_P = KA_P;
            SavingData.KA_D = KA_D;
            SavingData.KP_P = KP_P;
            SavingData.KP_I = KP_I;
            SavingData.K_Base = K_Base;
            WritePIDintoEEPROM(&SavingData);
            break;
        case 0x06:  break;
        case 0x07:break;
        default:break;
        }
    }

}

void UpdatePID()
{
    unsigned int Upper, Lower;
```

```
    double NewPara;
    Upper = SerialPacket.m_Buffer[2];
    Lower = SerialPacket.m_Buffer[1];
    NewPara = (float)(Upper << 8 | Lower) / 100.0;
    switch (SerialPacket.m_Buffer[3])
    {
    case 0x01:KA_P = NewPara; Serial.print("Get KA_P: \n"); Serial.println(KA_P); break;
    case 0x02:KA_D = NewPara; Serial.print("Get KA_D: \n"); Serial.println(KA_D); break;
    case 0x03:KP_P = NewPara; Serial.print("Get KP_P: \n"); Serial.println(KP_P); break;
    case 0x04:KP_I = NewPara; Serial.print("Get KP_I: \n"); Serial.println(KP_I); break;
    case 0x05:K_Base = NewPara; Serial.print("Get K_Base: \n"); Serial.println(K_Base); break;
    default:break;
    }
}

void CarDirection()
{
    unsigned char Speed = SerialPacket.m_Buffer[1];
    switch (SerialPacket.m_Buffer[3])
    {
    case 0x00: Speed_Need = 0; Turn_Need = 0; break;
    case 0x01: Speed_Need = -Speed; break;
    case 0x02: Speed_Need = Speed; break;
    case 0x03: Turn_Need = Speed; break;
    case 0x04: Turn_Need = -Speed; break;
    default:break;
    }
}

void SendPID()
{
    static unsigned char cnt = 0;
```

```
unsigned char data[3];
switch (cnt)
{
case 0:
    data[0] = 0x01;
    data[1] = int(KA_P * 100) / 256;
    data[2] = int(KA_P * 100) % 256;
    AssemblyAndSend(0x04, data);
    cnt++; break;
case 1:
    data[0] = 0x02;
    data[1] = int(KA_D * 100) / 256;
    data[2] = int(KA_D * 100) % 256;
    AssemblyAndSend(0x04, data);
    cnt++; break;
case 2:
    data[0] = 0x03;
    data[1] = int(KP_P * 100) / 256;
    data[2] = int(KP_P * 100) % 256;
    AssemblyAndSend(0x04, data);
    cnt++; break;
case 3:
    data[0] = 0x04;
    data[1] = int(KP_I * 100) / 256;
    data[2] = int(KP_I * 100) % 256;
    AssemblyAndSend(0x04, data);
    cnt++; break;
case 4:
    data[0] = 0x05;
    data[1] = int(K_Base * 100) / 256;
    data[2] = int(K_Base * 100) % 256;
    AssemblyAndSend(0x04, data);
```

```
        cnt = 0; break;
    default:break;
    }
}



void AssemblyAndSend(char type, unsigned char *data)
{
    unsigned char sendbuff[6];
    sendbuff[0] = 0xAA;
    sendbuff[1] = type;
    sendbuff[2] = data[0];
    sendbuff[3] = data[1];
    sendbuff[4] = data[2];
    sendbuff[5] = data[0] ^ data[1] ^ data[2];
    for (int i = 0; i < 6; i++)
    {
        Serial.write(sendbuff[i]);
    }
}

void Init()
{
    pinMode(BUZZER, OUTPUT);
    pinMode(SPD_PUL_L, INPUT);//
    pinMode(SPD_PUL_R, INPUT);//
    pinMode(PWM_L, OUTPUT);//
    pinMode(PWM_R, OUTPUT);//
    pinMode(DIR_L1, OUTPUT);//
    pinMode(DIR_L2, OUTPUT);
    pinMode(DIR_R1, OUTPUT);//
```

```
    pinMode(DIR_R2, OUTPUT);

    pinMode(AUX_IN_PIN, INPUT);
    pinMode(UP_DOWN_IN_PIN, INPUT);//
    pinMode(LEFT_RIGHT_IN_PIN, INPUT);//
                                        //init variables
    Speed_L = 0;
    Speed_R = 0;
    Position_AVG = 0;
    Position_AVG_Filter = 0;
    Position_Add = 0;
    pwm = 0; pwm_l = 0; pwm_r = 0;
    Speed_Diff = 0; Speed_Diff_ALL = 0;

    //KA_P = 30.0;
    //KA_D = 1.0;
    //KP_P = 0;       //255
    //KP_I = 0.0;
    //K_Base = 1.5;

    //KA_P = 30.0;
    //KA_D = 1.3;
    //KP_P = 11;       //255
    //KP_I = 0.01;
    //K_Base = 1.7;

    //KA_P = 30.0;
    //KA_D = 1.0;
    //KP_P = 20;       //255  good
    //KP_I = 0.01;
    //K_Base = 1.5;
```

```
KA_P = 30.0;
KA_D = 1.0;
KP_P = 20;
KP_I = 0.10;
K_Base = 6.7;

//KA_P = 30.0;
//KA_D = 1.0;
//KP_P = 10;
//KP_I = 0.10;
//K_Base = 2.5;

//KA_P = 30.0;
//KA_D = 1.0;
//KP_P = 10;
//KP_I = 0.20;
//K_Base = 2.5;

ReadingData.KA_P = KA_P;
ReadingData.KA_D = KA_D;
ReadingData.KP_P = KP_P;
ReadingData.KP_I = KP_I;
ReadingData.K_Base = K_Base;

Speed_Need = 0;
Turn_Need = 0;
ReadFromEEprom(&ReadingData);
KA_P = ReadingData.KA_P;
KA_D = ReadingData.KA_D;
KP_P = ReadingData.KP_P;
KP_I = ReadingData.KP_I;
K_Base = ReadingData.K_Base;
```

```
    Serial.print("PID Data is"); Serial.print("\t");
    Serial.print(KA_P); Serial.print("\t");
    Serial.print(KA_D); Serial.print("\t");
    Serial.print(KP_P); Serial.print("\t");
    Serial.print(KP_I); Serial.print("\t");
    Serial.print(K_Base); Serial.println("\t");

}



void Encoder_L()   //car up is positive car down  is negative
{
    if (digitalRead(SPD_PUL_L))
        Speed_L += 1;
    else
        Speed_L -= 1;
    //  Serial.print("SPEED_L:    ");
    // Serial.println(Speed_L);
}

void Encoder_R()    //car up is positive car down  is negative
{
    if (!digitalRead(SPD_PUL_R))
        Speed_R += 1;
    else
        Speed_R -= 1;

    // Serial.print("SPEED_R:    ");
    // Serial.println(Speed_R);
}
```

```
void MySerialEvent()
{
    uchar c = '0';
    uchar tmp = 0;
    if (Serial.available()) {
        c = (uchar)Serial.read();
        //Serial.println("here");
        for (int i = 5; i > 0; i--)
        {
            SerialPacket.m_Buffer[i] = SerialPacket.m_Buffer[i - 1];
        }
        SerialPacket.m_Buffer[0] = c;

        if (SerialPacket.m_Buffer[5] == 0xAA)
        {
            tmp = SerialPacket.m_Buffer[1] ^ SerialPacket.m_Buffer[2] ^ SerialPacket.m_Buffer[3];
            if (tmp == SerialPacket.m_Buffer[0])
            {
                SerialPacket.m_PackageOK = true;
            }
        }
    }
}

int UpdateAttitude()
{
    if ((micros() - timer) >= 10000)
    {    //10ms
         /* Update all the values */
        while (i2cRead(0x3B, i2cData, 14));
        accX = ((i2cData[0] << 8) | i2cData[1]);
        accY = ((i2cData[2] << 8) | i2cData[3]);
```

```
accZ = ((i2cData[4] << 8) | i2cData[5]);
tempRaw = (i2cData[6] << 8) | i2cData[7];
gyroX = (i2cData[8] << 8) | i2cData[9];
gyroY = (i2cData[10] << 8) | i2cData[11];
gyroZ = (i2cData[12] << 8) | i2cData[13];

//Serial.print(" gyroY1 = ");
//Serial.print(gyroY);

//Serial.print(" gyroY_cculate = ");
//Serial.print(gyroY - zzl.YAxis);

/*****************************/

//      Wire.beginTransmission(mpuAddress);
//#if ARDUINO >= 100
//      Wire.write(MPU6050_REG_GYRO_XOUT_H);
//#else
//      Wire.send(MPU6050_REG_GYRO_XOUT_H);
//#endif
//      Wire.endTransmission();
//
//      Wire.beginTransmission(mpuAddress);
//      Wire.requestFrom(mpuAddress, 6);
//
//      while (Wire.available() < 6);
//
//#if ARDUINO >= 100
//      uint8_t xha = Wire.read();
//      uint8_t xla = Wire.read();
//      uint8_t yha = Wire.read();
//      uint8_t yla = Wire.read();
```

```
//      uint8_t zha = Wire.read();
//      uint8_t zla = Wire.read();
//#else
//      uint8_t xha = Wire.receive();
//      uint8_t xla = Wire.receive();
//      uint8_t yha = Wire.receive();
//      uint8_t yla = Wire.receive();
//      uint8_t zha = Wire.receive();
//      uint8_t zla = Wire.receive();
//#endif
//
//      rg.XAxis = xha << 8 | xla;
//      rg.YAxis = yha << 8 | yla;
//      rg.ZAxis = zha << 8 | zla;
//
//      Serial.print(" rg.YAxis = ");
//      Serial.print(rg.YAxis);

/*****************************/

/*****************************/
/*  Vector zzl = mpu.calibrateGyro_zzl();
Serial.print(" calibrateGyro_zzl = ");
Serial.print(zzl.YAxis);*/
/*****************************/


double dt = (double)(micros() - timer) / 1000000; // Calculate delta time
timer = micros();

// atan2 outputs the value of -□□ to □□ (radians) - see
http://en.wikipedia.org/wiki/Atan2
```

```
        // It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
        double roll = atan2(accY, accZ) * RAD_TO_DEG;
        double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
        double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
        double pitch = atan2(-accX, accZ) * RAD_TO_DEG - calibrate_Accel.pitch;
        //Serial.print(" pitch ="); Serial.print(pitch); Serial.println("  ");
        //calibrate_Accel = mpu.calibrate_accel_pitch_and_roll();
        //Serial.print(" calibrate_Accel = ");
        //Serial.print(calibrate_Accel.pitch);
#endif

        double gyroXrate = gyroX / 131.0; // Convert to deg/s
        double gyroYrate = (gyroY - zzl.YAxis) / 131.0; // Convert to deg/s
                                                        //Serial.print("   gyroYrate   =   ");
Serial.print(gyroYrate); Serial.print("  ");

#ifdef RESTRICT_PITCH
                                                        // This fixes the transition problem when
the accelerometer angle jumps between -180 and 180 degrees
        if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
            kalmanX.setAngle(roll);
            compAngleX = roll;
            kalAngleX = roll;
            gyroXangle = roll;
        }
        else
            kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman
filter

        if (abs(kalAngleX) > 90)
            gyroYrate = -gyroYrate; // Invert rate, so it fits the restriced accelerometer reading
```

```
        kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else
                                                    // This fixes the transition problem when
the accelerometer angle jumps between -180 and 180 degrees
        if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {
            kalmanY.setAngle(pitch);
            compAngleY = pitch;
            kalAngleY = pitch;
            gyroYangle = pitch;
        }
        else
            kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate the angle using a
Kalman filter

        if (abs(kalAngleY) > 90)
            gyroXrate = -gyroXrate; // Invert rate, so it fits the restriced accelerometer reading
        kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman
filter
#endif

                                                    // gyroXangle += gyroXrate * dt; //
Calculate gyro angle without any filter
                                                    // gyroYangle += gyroYrate * dt;
                                                    //gyroXangle += kalmanX.getRate() *
dt; // Calculate gyro angle using the unbiased rate
                                                    //gyroYangle += kalmanY.getRate() *
dt;

                                                    // compAngleX = 0.93 * (compAngleX +
gyroXrate * dt) + 0.07 * roll; // Calculate the angle using a Complimentary filter
                                                    //compAngleY = 0.93 * (compAngleY +
gyroYrate * dt) + 0.07 * pitch;

                                                    // Reset the gyro angle when it has
```

```
drifted too much
        if (gyroXangle < -180 || gyroXangle > 180)
            gyroXangle = kalAngleX;
        if (gyroYangle < -180 || gyroYangle > 180)
            gyroYangle = kalAngleY;

        /* Print Data */
#if 0 // Set to 1 to activate
        Serial.print(accX); Serial.print("\t");
        Serial.print(accY); Serial.print("\t");
        Serial.print(accZ); Serial.print("\t");

        Serial.print(gyroX); Serial.print("\t");
        Serial.print(gyroY); Serial.print("\t");
        Serial.print(gyroZ); Serial.print("\t");

        Serial.print("\t");
#endif

#if 0
        Serial.print(roll); Serial.print("\t");
        // Serial.print(gyroXangle); Serial.print("\t");
        // Serial.print(compAngleX); Serial.print("\t");
        Serial.print(kalAngleX); Serial.print("\t");

        Serial.print("\t");

        Serial.print(pitch); Serial.print("\t");
        // Serial.print(gyroYangle); Serial.print("\t");
        // Serial.print(compAngleY); Serial.print("\t");
        Serial.print(kalAngleY); Serial.println("\t");
#endif
```

```
#if 0 // Set to 1 to print the temperature
        Serial.print("\t");

        double temperature = (double)tempRaw / 340.0 + 36.53;
        Serial.print(temperature); Serial.print("\t");
#endif

        //Serial.print("\r\n");

        Angle_Car = kalAngleY;   //negative backward  positive forward
        Gyro_Car = gyroYrate;
        //Serial.print(Angle_Car);Serial.print("\t");Serial.println(Gyro_Car);

        return 1;
    }
    return 0;
}

void LEDBlink()
{
    if ((millis() - LEDTimer) > 500)
    {
        LEDTimer = millis();
        blinkState = !blinkState;
        // digitalWrite(LED, blinkState);

        // Serial.print("Angle_Car\t");
        // Serial.println(Angle_Car);//Serial.print("\t");Serial.println(Gyro_Car);
    }
}

bool isInMucis = false;
```

```
bool isInBuzzer = false;


void MusicPlay()
{
    /*
    if((Speed_Need !=0 ) || (Turn_Need !=0 )) //when move let's play music
    {
    if((millis() - BuzzerTimer) >= (380*duration[tonecnt]))
    {
    noTone(tonePin);
    tonecnt++;
    if(tonecnt>=tonelength)
    tonecnt = 0;
    BuzzerTimer = millis();
    tone(tonePin,tune[tonecnt]);
    isInMucis  = true;
    }
    }
    else if(isInMucis == true)
    {
    isInMucis = false;
    noTone(tonePin);
    }
    */
    if (Angle_Car > 45 || Angle_Car < -45) //if car fall down ,let's alarm
    {
        if ((millis() - LEDTimer) > 200)
        {
            LEDTimer = millis();
            blinkState = !blinkState;
            digitalWrite(LED, blinkState);
            isInBuzzer = true;
```

```
        }
    }
    else if (isInBuzzer == true)
    {
        digitalWrite(LED, 0);
        isInBuzzer = false;
    }
}


//rc receiver interrupt routine
//-----------------------------------------------------

void calcUpDown()
{
    // Serial.println("in up down here");
    if (digitalRead(UP_DOWN_IN_PIN) == HIGH)
    {
        UpDownStart = micros();
    }
    else
    {
        UpDownEnd = (uint16_t)(micros() - UpDownStart);
        bUpdateFlagsRC |= UP_DOWN_FLAG;
    }
}

void calcLeftRight()
{
    // Serial.println("in Left Right here");
    if (digitalRead(LEFT_RIGHT_IN_PIN) == HIGH)
    {
```

```
        LeftRightStart = micros();
    }
    else
    {
        LeftRightEnd = (uint16_t)(micros() - LeftRightStart);
        bUpdateFlagsRC |= LEFT_RIGHT_FLAG;
    }
}



#define  RC_ERROR 100
#define RC_CAR_SPEED  160
int RCTurnSpeed = 80;
void ProcessRC()
{
    if (bUpdateFlagsRC)
    {
        noInterrupts(); // turn interrupts off quickly while we take local copies of the shared
variables

        if (bUpdateFlagsRC & UP_DOWN_FLAG)
        {
            UpDownIn = UpDownEnd;
            if (abs(UpDownIn - 1500) > RC_ERROR)
            {
                if (UpDownIn > 1500) // car up
                    Speed_Need = RC_CAR_SPEED;
                else       // car down
                    Speed_Need = -RC_CAR_SPEED;
                //Serial.println("here up down");
            }
            else
```

```
            {
                Speed_Need = 0;
            }
        }

        if (bUpdateFlagsRC & LEFT_RIGHT_FLAG)
        {
            LeftRightIn = LeftRightEnd;
            if (abs(LeftRightIn - 1500) > RC_ERROR)
            {
                //  RCTurnSpeed = 100;//map(ThrottleIn,1000,2000,0,200);
                if (LeftRightIn > 1500)
                    Turn_Need = -RCTurnSpeed;  // right
                else   // left
                    Turn_Need = RCTurnSpeed;
                //Serial.println("here Left Right");
            }
            else
            {
                Turn_Need = 0;
            }
        }

        bUpdateFlagsRC = 0;

        interrupts();
    }
}
```

****

Test Result:

Upload the test code, you should be able to control the motion of balance car through Android APP.

Resources:

- Download the test code, libraries and Android APP：

https://fs.keyestudio.com/KS0377