```
/*
 *
 * Smart Robot Car V3
 * - Bluetooth control version
 *
 */

#include <SoftwareSerial.h>

//////////////////////
//  <BT>     <UNO>
//   TX <-----> RX
//   RX <-----> TX
//////////////////////
SoftwareSerial btSerial(10, 11);  // RX, TX(UNO)

////////////////////////////////////////////////////////////////
// Note:  ENA and ENB must be connected to PWD supported pins
//
#define ENA   6    // PWD
#define EN1   7
#define EN2   3

#define EN3   4
#define EN4   2
#define ENB   5    // PWD

/////////////////////////////////////
// Ultrasonic sensor
/////////////////////////////////////
int TRIG_pin = 12;  // 센서 Trig 핀, D12
int ECHO_pin = 13;  // 센서 Echo 핀, D13

#define blinkLED  8  // for crash warning

//////////////////////
// Car direction
//
#define CAR_DIR_FW  0   // forward
#define CAR_DIR_BK  1   // backward
#define CAR_DIR_LT  2   // left turn
#define CAR_DIR_RT  3   // right turn
#define CAR_DIR_ST  4   // stop

//////////////////////////////////
// Default direction and speed
//
int g_carDirection = CAR_DIR_ST;
int g_carSpeed = 230; // 60% of max speed for testing

////////////////////////////////////////////////
```

```
// Note : confirm HIGH/LOW for correct movement
//
void car_forward()
{
  digitalWrite(EN1, HIGH);
  digitalWrite(EN2, LOW);
  analogWrite(ENA, g_carSpeed);

  digitalWrite(EN3, HIGH);
  digitalWrite(EN4, LOW);
  analogWrite(ENB, g_carSpeed);
}

void car_backward()
{
  digitalWrite(EN1, LOW);
  digitalWrite(EN2, HIGH);
  analogWrite(ENA, g_carSpeed);

  digitalWrite(EN3, LOW);
  digitalWrite(EN4, HIGH);
  analogWrite(ENB, g_carSpeed);
}

void car_left()
{
  digitalWrite(EN1, LOW);
  digitalWrite(EN2, HIGH);
  analogWrite(ENA, g_carSpeed);

  digitalWrite(EN3, HIGH);
  digitalWrite(EN4, LOW);
  analogWrite(ENB, g_carSpeed);
}

void car_right()
{
  digitalWrite(EN1, HIGH);
  digitalWrite(EN2, LOW);
  analogWrite(ENA, g_carSpeed);

  digitalWrite(EN3, LOW);
  digitalWrite(EN4, HIGH);
  analogWrite(ENB, g_carSpeed);
}

void car_stop()
{
  analogWrite(ENA, 0);
  analogWrite(ENB, 0);
}
```

```
//////////////////////////
// Execute car moving
//
void update_Car()
{
  switch ( g_carDirection ) {
    case CAR_DIR_FW:
        car_forward();
        break;
    case CAR_DIR_BK:
        car_backward();
        break;
    case CAR_DIR_LT:
        car_left();
        break;
    case CAR_DIR_RT:
        car_right();
        break;
    case CAR_DIR_ST:
        car_stop();
        break;
    default :
        ;
  }
  return;
}


//////////////////////////////////
//  Class - Serial Protocol
//
class _CommProtocol
{
private:
  unsigned char protocolPool[28];
  int bufPoint;

public:
  _CommProtocol()
  {
  }

  void addPool(unsigned char cByte)
  {
    if (bufPoint < 28)
    {
      if (bufPoint == 0 and cByte != 0x0c)
        return;  // invalid code

      protocolPool[bufPoint++]=cByte;
      //Serial.print("bufPoint -> ");
      //Serial.println(bufPoint);
```

```
  }
}

void clearPool()
{
  bufPoint = 0;
  memset(protocolPool, 0x00, 28);
  Serial.println("clearPool");
}

bool isValidPool()
{
  if (bufPoint >= 28)
  {
    //Serial.print("protocol length : ");

    if (protocolPool[0] == 0x0c && protocolPool[14] == 0x0c)
    {
      //Serial.println(protocolPool.length());
      return true;
    }
    else
    {
      clearPool();
      Serial.println("isValidPool 28 OVER");
    }
  }
  return false;
}

unsigned char getMotorLValue()
{
  unsigned char szProto[14];
  memcpy(szProto, protocolPool, 14);
  if (szProto[0] == 0x0C &&
    szProto[1] == 0x00 &&
    szProto[2] == 0x80 &&
    szProto[3] == 0x04 &&
    szProto[4] == 0x02)
  {
    unsigned char l = szProto[5];// -0x32;
    return l;
  }
  return 0x00;
}

unsigned char getMotorRValue()
{
  unsigned char szProto[14];
  memcpy(szProto, &protocolPool[14], 14);
  if (szProto[0] == 0x0C &&
    szProto[1] == 0x00 &&
```

```
      szProto[2] == 0x80 &&
      szProto[3] == 0x04 &&
      szProto[4] == 0x01)
    {
      unsigned char l = szProto[5];// -0x32;
      return l;
    }
    return 0x00;
  }
};    // class(_CommProtocol)

////////////////////////////////////////////
// Create an instance of class(_CommProtocol)
//
_CommProtocol SerialCommData;

////////////////////////////////////////////////////////////
// Parse and change the serial input value to MOVE command
//
void process_SerialCommModule()
{
  if (SerialCommData.isValidPool())
  {
    char motorLR[2];

    motorLR[0] = (char)SerialCommData.getMotorLValue();
    motorLR[1] = (char)SerialCommData.getMotorRValue();
    SerialCommData.clearPool();

    //
    Serial.print("Left [");
    Serial.print(motorLR[0],DEC);
    Serial.print("] Right [");
    Serial.print(motorLR[1],DEC);
    Serial.println("]");
    //

    char szCmdValue = '5';
    // set MOVE commands
    if (motorLR[0] == 0 && motorLR[1] == 0) {  // (0,0) stop
      szCmdValue = '5';
    }
    else
    {
      int nSpeed;
      nSpeed = max(abs(motorLR[0]), abs(motorLR[1]));

      // Set direction
      if (motorLR[0] > 0 && motorLR[1] > 0)   // (+,+) forward
      {
        szCmdValue = '2';
        g_carSpeed = 255.0f * ((float)nSpeed / 100.0f);
```

```
      }
      else if (motorLR[0] < 0 && motorLR[1] < 0)   // (-,-) backward
      {
        szCmdValue = '8';
        g_carSpeed = 255.0f * ((float)nSpeed / 100.0f);
      }
      else if (motorLR[0] < 0 && motorLR[1] > 0)   // (-,+) left turn
      {
        szCmdValue = '4';
        g_carSpeed = 255.0f * ((float)((float)nSpeed*1.66f) / 100.0f);
      }
      else if (motorLR[0] > 0 && motorLR[1] < 0)   // (+,-) right turn
      {
        szCmdValue = '6';
        g_carSpeed = 255.0f * ((float)((float)nSpeed*1.66f) / 100.0f);
      }
    }
    //
    Serial.print("speed ");
    Serial.print(g_carSpeed);
    Serial.print(" ");
    Serial.println(szCmdValue);
    //

    // Set the direction and speed with command
    controlByCommand(szCmdValue);
  }
}

////////////////////////////////////////////////
// Hint : Command codes come from keypad numbers
//
void controlByCommand(char doCommand)
{
  switch ( doCommand ) {
    case '+' :    // speed up
      g_carSpeed += 20;
      g_carSpeed = min(g_carSpeed, 255);
      break;
    case '-' :    // speed down
      g_carSpeed -= 20;
      g_carSpeed = max(g_carSpeed, 75);
      break;
    case '2' :    // forward
      g_carDirection = CAR_DIR_FW;
      break;
    case '5' :    // stop
      g_carDirection = CAR_DIR_ST;
      break;
    case '8' :    // backward
      g_carDirection = CAR_DIR_BK;
      break;
```

```
    case '4' :   // left
      g_carDirection = CAR_DIR_LT;
      break;
    case '6' :   // right
      g_carDirection = CAR_DIR_RT;
      break;
    default  :
      ;
  }
  return;
}


void setup()
{
  Serial.begin(9600);   // PC serial monitor debugging
  btSerial.begin(9600);   // bluetooth serial connection

  //init car control board
  pinMode(ENA, OUTPUT);  // ENA
  pinMode(EN1, OUTPUT);  // EN1
  pinMode(EN2, OUTPUT);  // EN2

  pinMode(ENB, OUTPUT);  // ENB
  pinMode(EN3, OUTPUT);  // EN3
  pinMode(EN4, OUTPUT);  // EN4

  pinMode(blinkLED, OUTPUT); // for crash check
  pinMode(TRIG_pin, OUTPUT);
  pinMode(ECHO_pin, INPUT);

  //
  Serial.print("direction value ");
  Serial.println(g_carDirection);
  Serial.print("speed pwm value ");
  Serial.print(g_carSpeed);
  Serial.println("");
  //
}


void loop()
{
  //alert_Bump();  // blink LED

  if (btSerial.available()) {

    unsigned char cByte;

    cByte = btSerial.read();

    SerialCommData.addPool(cByte);  // store the serial input to Buffer
```

```
    process_SerialCommModule();  // parse and change the input value to MOVE

    update_Car();                // execute car MOVE

  }

}

//////////////////////////////////////////
//  Ultrasonic sensor : calculate distance
//  Blink LED if distance < 20
//
bool alert_Bump()
{
  long duration, cm;

  digitalWrite(TRIG_pin, HIGH);           // 센서에 Trig 신호 입력
  delayMicroseconds(10);                 // 10us 정도 유지
  digitalWrite(TRIG_pin,LOW);             // Trig 신호 off
  duration = pulseIn(ECHO_pin,HIGH);       // Echo pin: HIGH->Low 간격을 측정
  cm = microsecondsToCentimeters(duration); // 거리(cm)로 변환

  if (cm < 20)
  {
    Serial.print("cm -> ");
    Serial.println(cm);

    digitalWrite(blinkLED, HIGH);

    return true;
  }
  else
    digitalWrite(blinkLED, LOW);

  return false;
}

long microsecondsToCentimeters(long microseconds)
{
  return microseconds/29/2;
}
```