

```

/*the hookup:
  ADJD-S311 Breakout ----- Arduino
  -----
    LED -----D4
    3.3V -----3.3V
    GND ----- GND
    SCL ----- A5
    SDA ----- A4
    GND ----- GND
    SLP ----- Not connected
    CLK ----- Not connected
*/
#include <Wire.h>  // We use Wire.h to talk I2C to the sensor
#include<Servo.h>

//pour la partie moteur
int E1 = 6; //M1 Speed Control
int E2 = 5; //M2 Speed Control
int M1 = 8; //M1 Direction Control
int M2 = 7; //M2 Direction Control

//pour la partie sonar
const int numOfReadings = 10;           // number of readings to
take/ items in the array
int readings[numOfReadings];           // stores the distance
readings in an array
int arrayIndex = 0;                     // arrayIndex of the current
item in the array
int total = 0;                          // stores the cumulative total
int averageDistance = 0;                // stores the average value

// setup pins and variables for SRF05 sonar device

int echoPin = 2;                        // SRF05 echo pin (digital
2)
int initPin = 3;                        // SRF05 trigger pin (digital
3)
unsigned long pulseTime = 0;            // stores the pulse in Micro
Seconds
unsigned long distance = 0;              // variable for storin

//define servo stuff
#define SERVO_PIN 9 //digital pin 9

```

```

#define SERVOMIN 0    //minimum angle 0 degrees
#define SERVOMAX 45   //maximum angle 45 degrees

//define rotation sensor stuff
#define ROTATIONPIN 0 //analog pin 0
// ADJD-S311's I2C address, don't change
#define ADJD_S311_ADDRESS 0x74

#define RED 0
#define GREEN 1
#define BLUE 2
#define CLEAR 3

// ADJD-S311's register list
#define CTRL 0x00
#define CONFIG 0x01
#define CAP_RED 0x06
#define CAP_GREEN 0x07
#define CAP_BLUE 0x08
#define CAP_CLEAR 0x09
#define INT_RED_LO 0xA
#define INT_RED_HI 0xB
#define INT_GREEN_LO 0xC
#define INT_GREEN_HI 0xD
#define INT_BLUE_LO 0xE
#define INT_BLUE_HI 0xF
#define INT_CLEAR_LO 0x10
#define INT_CLEAR_HI 0x11
#define DATA_RED_LO 0x40
#define DATA_RED_HI 0x41
#define DATA_GREEN_LO 0x42
#define DATA_GREEN_HI 0x43
#define DATA_BLUE_LO 0x44
#define DATA_BLUE_HI 0x45
#define DATA_CLEAR_LO 0x46
#define DATA_CLEAR_HI 0x47
#define OFFSET_RED 0x48
#define OFFSET_GREEN 0x49
#define OFFSET_BLUE 0x4A
#define OFFSET_CLEAR 0x4B
int t1=0;
// Pin definitions:
int sdaPin = A4; // serial data, hardwired, can't change
int sclPin = A5; // serial clock, hardwired, can't change

```

```

int ledPin = 4;  // LED light source pin, any unused pin will work

// RGB LED pins, should all be PWM output pins:
int redledPin = 9;
int greenledPin = 10;
int blueledPin = 11;
int rgbPins[3] = {redledPin, greenledPin, blueledPin};

// initial values for integration time registers
unsigned char colorCap[4] = {9, 9, 2, 5};  // values must be between 0 and 15
unsigned int colorInt[4] = {2048, 2048, 2048, 2048};  // max value for these is 4095
unsigned int colorData[4];  // This is where we store the RGB and C data values
signed char colorOffset[4];  // Stores RGB and C offset values

Servo myServo;
int pos = 0;
int com=0;
char val = '6';
int r=0;
int g=0;
int b=0;
int c=0;
void setup()
{
  pinMode(ledPin, OUTPUT);  // Set the sensor's LED as output
  digitalWrite(ledPin, HIGH);  // Initially turn LED light source on

  for (int i=0; i<3; i++)
  {  // Set up the RGB LED pins
    pinMode(rgbPins[i], OUTPUT);
    digitalWrite(rgbPins[i], LOW);
  }
  //setup moteur
  int i;
  for(i=5;i<=8;i++)
    pinMode(i, OUTPUT);
  // initialize the serial port, lets you view the
  // distances being pinged if connected to computer
  myServo.attach(SERVOPIN);
  Serial.begin(9600);

  Wire.begin();
  delay(1);  // Wait for ADJD reset sequence

```

```
    initADJD_S311(); // Initialize the ADJD-S311, sets up cap and int registers
}
```

```
void loop(void){
    while (Serial.available() < 1) {
        } // Wait until a character is received
    char val= Serial.read();
    Serial.println(val, DEC);
    switch(val)
    {
        case '6'://wifi_control
            handcontrol();
            break;
        case '7'://suivre une ligne
            suivligne();
            break;
        case '8'://marche automatiquement
            autodirection() ;
            break;
        case '0'://arret
            stop();
            break;
    }
}

}
```

```
void handcontrol()
{
    while(1){
        while (Serial.available() < 1) {
            } // Wait until a character is received
        char val=Serial.read();
        Serial.println(val, DEC);
        int leftspeed = 150; //255 is maximum speed
        int rightspeed = 150;
        switch(val) // Perform an action depending on the command
        {
            case '1'://Move Forward
                forward (leftspeed,rightspeed);
                break;
            case '2'://Move Backwards
                reverse (leftspeed,rightspeed);
                break;
            case '3'://Turn Left
```

```

        left (leftspeed,rightspeed);
        break;
    case '4'://Turn Right
        right (leftspeed,rightspeed);
        break;
    case '0':
        stop();
        break;
    case '7':
        suivligne();
        break;
    case '8':
        autodirection();
        break;
    }
}
}
void suivligne()
{
    while (Serial.available() < 1) {
    int leftspeed = 100; //255 is maximum speed
    int rightspeed = 100;
        getRGBC();
        printADJD_S311Values();

        r=colorData[0];
        g=colorData[1];
        b=colorData[2];
        c=colorData[3];
        r=r/c;
        g=g/c;
        b=b/c;
        //int rgb=r+g+b;
        if(r<0.7&&g<0.6&&b<0.6){
            //val='1';
            //Serial.println(val, DEC);
            forward (leftspeed,rightspeed);
            t1=0;
        }
        else if(r>=0.7||g>=0.6||b>=0.6){
            //val='3';
            //Serial.println(val, DEC);
            if(t1<=5) {
                left (leftspeed,rightspeed);

```

```

        t1=t1+1;
        delay(50);
    }
    else if(t1>5&&t1<=17) {
        right (leftspeed,rightspeed);
        t1=t1+1;
        delay(50);
    }

}
if(t1>=18){
    t1=0;
}
} // Wait until a character is received
char val=Serial.read();
Serial.println(val, DEC);
switch(val) // Perform an action depending on the command
{
    case '0':
        stop();
        break;
    case '7':
        suivligne();
        break;
    case '8':
        autodirection();
        break;
    case '6':
        handcontrol();
        break;
}
}

```

```

void autodirection()
{
    while (Serial.available() < 1) {
        int left_low=255;
        int right_low=255;
        int left_high=255;
        int right_high=255;
        int angle;
        sonar_dist();
        int distance=averageDistance;
        if(distance>30&&com==0){

```

```

    forward(left_high,right_high);
    sonar_dist();
    distance=averageDistance;
}

else if(distance<=30&&com==0) {
    stop();
    com=1;
    val=scan();
    myServo.write(90);
    while(distance<=50&&com==1) {
        switch(val) // Perform an action depending on the command
        {
            case '1'://Move Forward
                forward (left_high,right_high);
                break;
            case '2'://Move Backwards
                reverse (left_high,right_high);
                break;
            case '3'://Turn Left
                left (left_low,right_low);
                break;
            case '4'://Turn Right
                right (left_low,right_low);
                break;
            case '0':
                stop();
                break;
        }
        sonar_dist();
        distance=averageDistance;
    }
}

else if(distance>30&&com!=0){
    if(val==50||val==51||val==52||val==49){
        stop();
        com=0;
    }
}

}

char val=Serial.read();
Serial.println(val, DEC);
switch(val) // Perform an action depending on the command
{

```

```

        case '0':
            stop();
            break;
        case '7':
            suivligne();
            break;
        case '8':
            autodirection();
            break;
        case '6':
            handcontrol();
            break;
    }
}

```

/* printADJD_S311Values() reads, formats, and prints all important registers of the ADJD-S311.

It doesn't perform any measurements, so you'll need to call getRGBC() to print new values.

```

*/
void printADJD_S311Values()
{
    Serial.println("\t\t Red \t Green \t Blue \t Clear");
    Serial.print("Data: \t\t ");
    for (int i=0; i<4; i++)
    {
        Serial.print(colorData[i]);
        Serial.print("\t ");
    }
    Serial.println();
    Serial.print("Caps: \t\t ");
    for (int i=0; i<4; i++)
    {
        Serial.print(readRegister(CAP_RED+i), DEC);
        Serial.print("\t ");
    }
    Serial.println();
    Serial.print("Int: \t\t ");
    for (int i=0; i<4; i++)
    {
        Serial.print(readRegisterInt(INT_RED_LO+(i*2)), DEC);
        Serial.print("\t ");
    }
    Serial.println();
}

```



```

Serial.print("Offset: \t ");
for (int i=0; i<4; i++)
{
    Serial.print((signed char) readRegister(OFFSET_RED+i), DEC);
    Serial.print("\t ");
}
Serial.println();
}

```

/* initADJD_S311() - This function initializes the ADJD-S311 and its capacitor and integration registers

The vaules for those registers are defined near the top of the code. the colorCap[] array defines all capacitor values, colorInt[] defines all integration values.

```

*/
void initADJD_S311()
{
    /*sensor gain registers, CAP_...
    to select number of capacitors.
    value must be <= 15 */
    writeRegister(colorCap[RED] & 0xF, CAP_RED);
    writeRegister(colorCap[GREEN] & 0xF, CAP_GREEN);
    writeRegister(colorCap[BLUE] & 0xF, CAP_BLUE);
    writeRegister(colorCap[CLEAR] & 0xF, CAP_CLEAR);

    /* Write sensor gain registers INT_...
    to select integration time
    value must be <= 4096 */
    writeRegister((unsigned char)colorInt[RED], INT_RED_LO);
    writeRegister((unsigned char)((colorInt[RED] & 0x1FFF) >> 8), INT_RED_HI);
    writeRegister((unsigned char)colorInt[BLUE], INT_BLUE_LO);
    writeRegister((unsigned char)((colorInt[BLUE] & 0x1FFF) >> 8),
INT_BLUE_HI);
    writeRegister((unsigned char)colorInt[GREEN], INT_GREEN_LO);
    writeRegister((unsigned char)((colorInt[GREEN] & 0x1FFF) >> 8),
INT_GREEN_HI);
    writeRegister((unsigned char)colorInt[CLEAR], INT_CLEAR_LO);
    writeRegister((unsigned char)((colorInt[CLEAR] & 0x1FFF) >> 8),
INT_CLEAR_HI);
}

```

/* calibrateClear() - This function calibrates the clear integration registers of the ADJD-S311.

```

*/

```

```

int calibrateClear()
{
    int gainFound = 0;
    int upperBox=4096;
    int lowerBox = 0;
    int half;

    while (!gainFound)
    {
        half = ((upperBox-lowerBox)/2)+lowerBox;
        //no further halving possible
        if (half==lowerBox)
            gainFound=1;
        else
        {
            writeInt(INT_CLEAR_LO, half);
            performMeasurement();
            int halfValue = readRegisterInt(DATA_CLEAR_LO);

            if (halfValue>1000)
                upperBox=half;
            else if (halfValue<1000)
                lowerBox=half;
            else
                gainFound=1;
        }
    }
    return half;
}

```

/* calibrateColor() - This function calibrates the RG and B integration registers.

*/

```

int calibrateColor()
{
    int gainFound = 0;
    int upperBox=4096;
    int lowerBox = 0;
    int half;

    while (!gainFound)
    {
        half = ((upperBox-lowerBox)/2)+lowerBox;
        //no further halving possible

```

```

    if (half==lowerBox)
    {
        gainFound=1;
    }
    else {
        writeInt(INT_RED_LO, half);
        writeInt(INT_GREEN_LO, half);
        writeInt(INT_BLUE_LO, half);

        performMeasurement();
        int halfValue = 0;

        halfValue=max(halfValue, readRegisterInt(DATA_RED_LO));
        halfValue=max(halfValue, readRegisterInt(DATA_GREEN_LO));
        halfValue=max(halfValue, readRegisterInt(DATA_BLUE_LO));

        if (halfValue>1000) {
            upperBox=half;
        }
        else if (halfValue<1000) {
            lowerBox=half;
        }
        else {
            gainFound=1;
        }
    }
}
return half;
}

```

/* calibrateCapacitors() - This function calibrates each of the RGB and C capacitor registers.

*/

void calibrateCapacitors()

```

{
    int  calibrationRed = 0;
    int  calibrationBlue = 0;
    int  calibrationGreen = 0;
    int calibrated = 0;

```

//need to store detect better calibration

int oldDiff = 5000;

while (!calibrated)

```

{
    // sensor gain setting (Avago app note 5330)
    // CAPs are 4bit (higher value will result in lower output)
    writeRegister(calibrationRed, CAP_RED);
    writeRegister(calibrationGreen, CAP_GREEN);
    writeRegister(calibrationBlue, CAP_BLUE);

    // int colorGain = _calibrateColorGain();
    int colorGain = readRegisterInt(INT_RED_LO);
    writeInt(INT_RED_LO, colorGain);
    writeInt(INT_GREEN_LO, colorGain);
    writeInt(INT_BLUE_LO, colorGain);

    int maxRead = 0;
    int minRead = 4096;
    int red    = 0;
    int green  = 0;
    int blue   = 0;

    for (int i=0; i<4 ;i ++)
    {
        performMeasurement();
        red    += readRegisterInt(DATA_RED_LO);
        green += readRegisterInt(DATA_GREEN_LO);
        blue   += readRegisterInt(DATA_BLUE_LO);
    }
    red    /= 4;
    green /= 4;
    blue   /= 4;

    maxRead = max(maxRead, red);
    maxRead = max(maxRead, green);
    maxRead = max(maxRead, blue);

    minRead = min(minRead, red);
    minRead = min(minRead, green);
    minRead = min(minRead, blue);

    int diff = maxRead - minRead;

    if (oldDiff != diff)
    {
        if ((maxRead==red) && (calibrationRed<15))
            calibrationRed++;
    }
}

```

```

        else if ((maxRead == green) && (calibrationGreen<15))
            calibrationGreen++;
        else if ((maxRead == blue) && (calibrationBlue<15))
            calibrationBlue++;
    }
    else
        calibrated = 1;

    oldDiff=diff;

    int rCal = calibrationRed;
    int gCal = calibrationGreen;
    int bCal = calibrationBlue;
}
}

```

/* writeInt() - This function writes a 12-bit value to the LO and HI integration registers */

```

void writeInt(int address, int gain)
{
    if (gain < 4096)
    {
        byte msb = gain >> 8;
        byte lsb = gain;

        writeRegister(lsb, address);
        writeRegister(msb, address+1);
    }
}

```

/* performMeasurement() - This must be called before reading any of the data registers. This commands the ADJD-S311 to perform a measurement, and store the data into the data registers.*/

```

void performMeasurement()
{
    writeRegister(0x01, 0x00); // start sensing
    while(readRegister(0x00) != 0)
        ; // waiting for a result
}

```

/* getRGBC() - This function reads all of the ADJD-S311's data registers and stores them into colorData[]. To get the

most up-to-date data make sure you call performMeasurement()
before calling this function.*/

```
void getRGBC()
{
    performMeasurement();

    colorData[RED] = readRegisterInt(DATA_RED_LO);
    colorData[GREEN] = readRegisterInt(DATA_GREEN_LO);
    colorData[BLUE] = readRegisterInt(DATA_BLUE_LO);
    colorData[CLEAR] = readRegisterInt(DATA_CLEAR_LO);
}
```

/* getOffset() - This function performs the offset reading
and stores the offset data into the colorOffset[] array.
You can turn on data trimming by uncommenting out the
writing 0x01 to 0x01 code.

```
*/
void getOffset()
{
    //digitalWrite(ledPin, LOW);  // turn LED off
    //delay(10);  // wait a tic
    writeRegister(0x02, 0x00); // start sensing
    while(readRegister(0x00) != 0)
        ; // waiting for a result
    //writeRegister(0x01, 0x01);  // set trim
    //delay(100);
    for (int i=0; i<4; i++)
        colorOffset[i] = (signed char) readRegister(OFFSET_RED+i);
    //digitalWrite(ledPin, HIGH);
}
```

/* I2C functions...*/

```
// Write a byte of data to a specific ADJD-S311 address
void writeRegister(unsigned char data, unsigned char address)
{
    Wire.beginTransmission(ADJD_S311_ADDRESS);
    Wire.write(address);
    Wire.write(data);
    Wire.endTransmission();
}
```

```
// read a byte of data from ADJD-S311 address
unsigned char readRegister(unsigned char address)
{

```

```

    unsigned char data;

    Wire.beginTransmission(ADJD_S311_ADDRESS);
    Wire.write(address);
    Wire.endTransmission();

    Wire.requestFrom(ADJD_S311_ADDRESS, 1);
    while (!Wire.available())
        ; // wait till we can get data

    return Wire.read();
}

// Write two bytes of data to ADJD-S311 address and address+1
int readRegisterInt(unsigned char address)
{
    return readRegister(address) + (readRegister(address+1)<<8);
}

void stop(void) //Stop
{
    digitalWrite(E1,LOW);
    digitalWrite(E2,LOW);
}

void forward(char a,char b)
{
    analogWrite (E1,a);
    digitalWrite(M1,LOW);
    analogWrite (E2,b);
    digitalWrite(M2,LOW);
}

void reverse (char a,char b)
{
    analogWrite (E1,a);
    digitalWrite(M1,HIGH);
    analogWrite (E2,b);
    digitalWrite(M2,HIGH);
}

void left (char a,char b)
{
    analogWrite (E1,a);

```

```

    digitalWrite(M1,HIGH);
    analogWrite (E2,b);
    digitalWrite(M2,LOW);
}
void right (char a,char b)
{
    analogWrite (E1,a);
    digitalWrite(M1,LOW);
    analogWrite (E2,b);
    digitalWrite(M2,HIGH);
}

void sonar_dist(void)
{
    digitalWrite(initPin, HIGH);           // send 10 microsecond
    pulse                                  pulse
    delayMicroseconds(10);                 // wait 10 microseconds before
    turning off
    digitalWrite(initPin, LOW);            // stop sending the pulse
    pulseTime = pulseIn(echoPin, HIGH);    // Look for a return pulse,
    it should be high as the pulse goes low-high-low
    distance = pulseTime/58;               // Distance = pulse time
    / 58 to convert to cm.
    total= total - readings[arrayIndex];   // subtract the last distance
    readings[arrayIndex] = distance;       // add distance reading to
    array
    total= total + readings[arrayIndex];   // add the reading to the
    total
    arrayIndex = arrayIndex + 1;           // go to the next item in
    the array
    // At the end of the array (10 items) then start again
    if (arrayIndex >= numOfReadings) {
        arrayIndex = 0;
    }

    averageDistance = total / numOfReadings; // calculate the average
    distance
    Serial.println(averageDistance, DEC); // print out the average distance to the
    debugger

    delay(100);                           // wait 100 milli s
}

```



```

char scan(void)
{
    int val='0';
    int angle=90;
    int ang=0;
    int distance_sonar=0;
    int flag=0;
    int n=0;
    myServo.write(angle);
    sonar_dist();
    distance_sonar=averageDistance;
    while(n<=36&&distance_sonar<=30){
        sonar_dist();
        distance_sonar=averageDistance;
        while(distance_sonar<=30&&n<18){
            angle=angle+5;
            n=n+1;
            myServo.write(angle);
            sonar_dist();
            distance_sonar=averageDistance;
            //delay(100);
        }

        while(distance_sonar<=30&&n>=18){
            angle=angle+5;
            ang=90-5*(n-18);
            n=n+1;
            myServo.write(ang);
            sonar_dist();
            distance_sonar=averageDistance;
            //delay(100);
        }

        if(distance_sonar>30){
            flag=1;
            angle=myServo.read();
        }
    }
    if (flag==0){
        val=50;          //'2'
    }
    if (angle<90&&flag==1){
        val=51;          //'3'
    }
}

```

```
}  
if (angle>90&&flag==1){  
    val=52;        //'4'  
}  
if (angle==90&&flag==1){  
    val=49;        //'1'  
}  
  
return val;  
}
```