AQA - Taller Vestuario

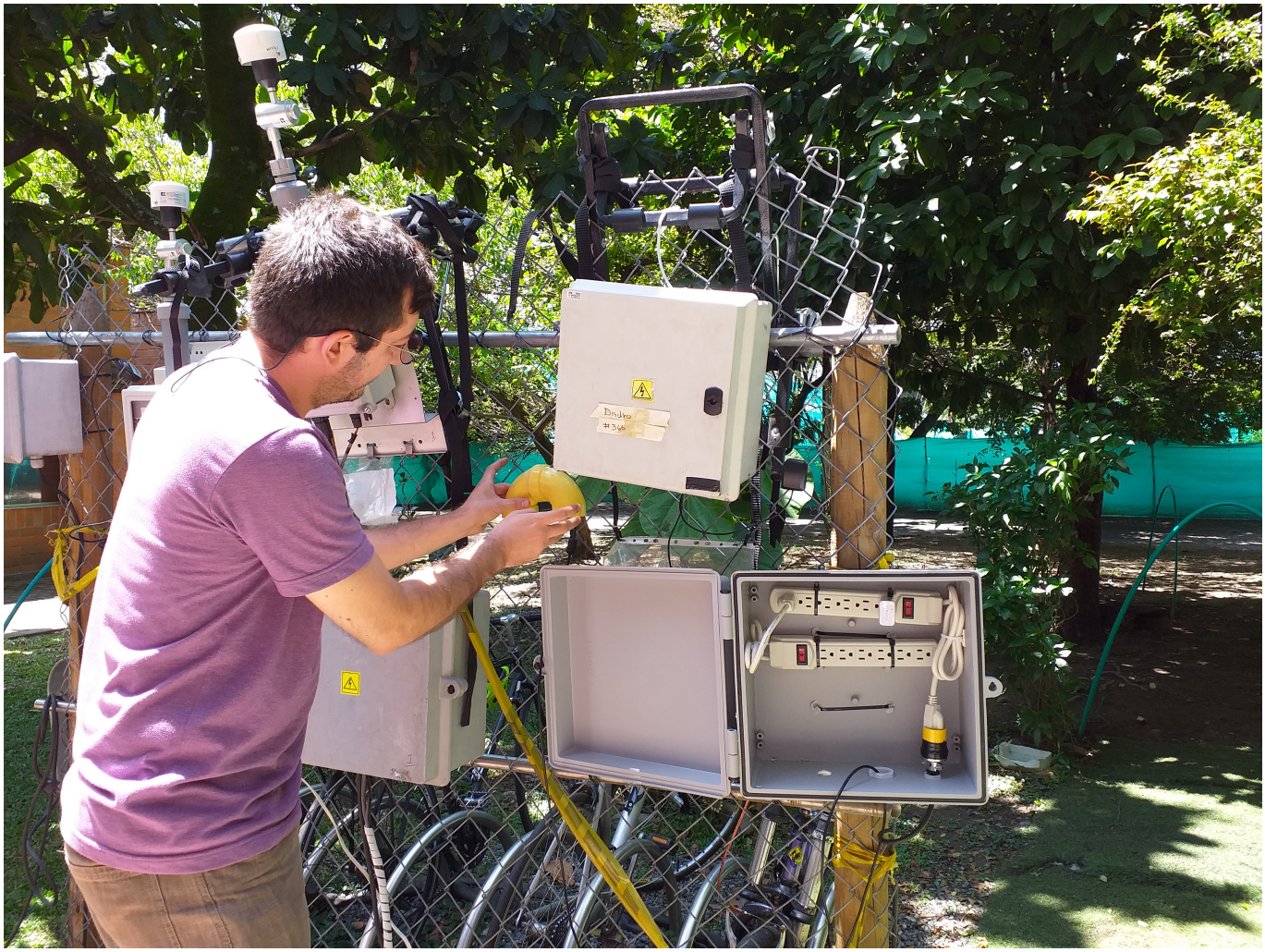
Prendas de vestuario + AQA

Bitácora

2019-10-17

- Instalación de sensores para proceso de calibración

marca Viakable
SAMSUNG
900t
2
3
1



- Visualización de los datos de los sensores instalados

Referentes

- https://twitter.com/arturo182/status/1162762166107353089?s=20

Ideas y materiales que se pueden usar para el taller

- instructable
- adrafruit
- Enlace externo
- Enlace externo

prendas

Falda led

Iluminación

Como conectar cinta de leds

código incial para hacer una animación en la matrix de leds

```
#include<FastLED.h>
#define LED_PIN     D3
#define LED_TYPE    WS2812B
#define COLOR_ORDER GRB
#define f false
#define t true

const uint8_t kMatrixWidth  = 8;
const uint8_t kMatrixHeight = 8;
#define NUM_LEDS (kMatrixWidth * kMatrixHeight)

int BRIGHTNESS = 60;   // this is half brightness
CRGB leds[kMatrixWidth * kMatrixHeight];

#define amarillo CRGB::Yellow
#define black CRGB::Black
#define rojo CRGB::Red

int loop_cnt = 0;
uint16_t speed = 20;
static uint16_t x;
static uint16_t y;
static uint16_t z;
uint16_t scale = 31;
uint8_t noise[kMatrixWidth][kMatrixHeight];

// Fill the x/y array of 8-bit noise values using the inoise8 function.
/*
void fillnoise8() {
  for(int i = 0; i < kMatrixWidth; i++) {
    int ioffset = scale * i;
    for(int j = 0; j < kMatrixHeight; j++) {
      int joffset = scale * j;
      noise[i][j] = inoise8(x + ioffset,y + joffset,z);
    }
  }
  z += speed;
}
*/
void setup() {
  LEDS.addLeds<LED_TYPE,LED_PIN,COLOR_ORDER>(leds,NUM_LEDS);
  FastLED.setBrightness(BRIGHTNESS);
```

```
  // Initialize our coordinates to some random values
  x = random16();
  y = random16();
  z = random16();
}

#define ESCENAS 8

CRGB matrix[ESCENAS][8][8] = {
  {
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
  },
  {
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
  },
  {
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
```

```
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
  },
  {
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
  },
    {
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
  },
  {
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
```

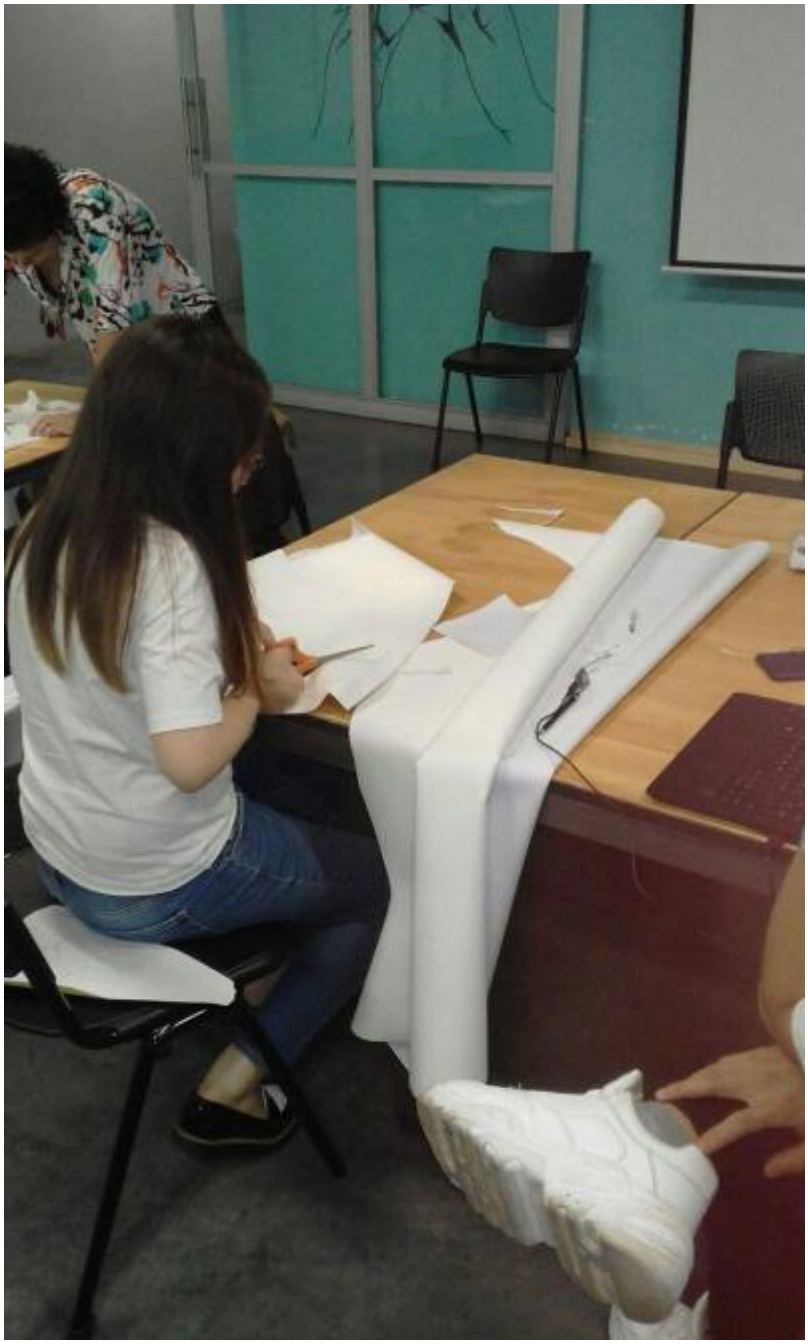
```
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
  },
  {
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
    {CRGB::Green, CRGB::Green, CRGB::Green,CRGB::Green, CRGB::Green,
CRGB::Green, CRGB::Green,CRGB::Green},
  },
  {
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
    {CRGB::Black, CRGB::Black, CRGB::Black,CRGB::Black, CRGB::Black,
CRGB::Black, CRGB::Black,CRGB::Black},
  },
```

```
};

void loop() {
  /*fillnoise8();*/

  for(int i = 0; i< kMatrixHeight; i++) {
    for(int j = 0; j< kMatrixWidth; j++) {
      leds[i*kMatrixWidth + j] = matrix[loop_cnt%ESCENAS][i][j];
    }
  }
  FastLED.show();
  delay(500);
  for(int i = 0; i< kMatrixHeight; i++) {
    for(int j = 0; j<kMatrixWidth; j++) {
      leds[i*kMatrixWidth + j] = CRGB::Black;
    }
  }
  delay(500);
  loop_cnt++;
}
```

Algunas fotografias de la 5 sesión del taller



Código para el proyecto del corazon

- Wiki sobre la tira de leds
- tutorial inicial

Este código simula el movimiento del corazón

```
#include <FastLED.h>
FASTLED_USING_NAMESPACE

#if defined(FASTLED_VERSION) && (FASTLED_VERSION < 3001000)
#warning "Requires FastLED 3.1 or later; check github for latest code."
#endif

#define DATA_PIN    D6
#define CLK_PIN     D5
#define LED_TYPE    DOTSTAR
#define COLOR_ORDER BGR
#define NUM_LEDS    23
#define BRIGHTNESS  96
CRGB leds[NUM_LEDS];
```

```
int contador = 0;

void setup() {
    FastLED.addLeds<LED_TYPE,DATA_PIN,CLK_PIN,COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);
    FastLED.setBrightness(BRIGHTNESS);
    Serial.begin(115200);
}

void loop() {
  contador++;

  if (contador < 10) {
    normalHeart();
  } else {
    HeartTired();
  }

  if (contador > 20) { contador = 0; }
  Serial.println(contador);

}

void normalHeart() {
    for(int dot=0; dot<NUM_LEDS+1; dot++) {
      leds[dot] = CRGB::Green;
      FastLED.show();
      delay(30);
      leds[dot] = CRGB::Black;
      FastLED.delay(50);
    }

    for(int dot=0; dot<NUM_LEDS+1; dot++) {
      leds[NUM_LEDS - dot] = CRGB::Green;
      FastLED.show();
      delay(30);
      leds[NUM_LEDS - dot] = CRGB::Black;
      FastLED.delay(5);
    }
    delay(1000);
}

void HeartTired() {
    for(int dot=0; dot<NUM_LEDS+1; dot++) {
      leds[dot] = CRGB::Red;
      FastLED.show();
      delay(15);
      leds[dot] = CRGB::Black;
      FastLED.delay(10);
    }
```

```
    for(int dot=0; dot<NUM_LEDS+1; dot++) {
      leds[NUM_LEDS - dot] = CRGB::Red;
      FastLED.show();
      delay(15);
      leds[NUM_LEDS - dot] = CRGB::Black;
      FastLED.delay(2);
    }
    delay(500);
}
```

Este código usa la libreria FastLed Painter

```
/**
   Simple animation demo for using the FastLED painter library
   This demo does the Knight Rider scanner effect with just a few lines of
code
   The speed and fadespeed need to be adjusted for different processor
speeds
   Chosen settings work nicely on 60-pixels and an Arduino Duemilenove (Uno)
*/

#define NUMBEROFPIXELS 105 //Number of LEDs on the strip
#define PIXELPIN 2 //Pin where WS281X LED strip data-line is connected

#include "Arduino.h"
#include <FastLED.h>
#include <FastLEDPainter.h>

#define DATA_PIN    D2
#define CLK_PIN     D1
#define LED_TYPE    DOTSTAR
#define COLOR_ORDER BGR
#define NUM_LEDS    105
#define BRIGHTNESS  96
CRGB leds[NUM_LEDS];

//create one canvas and one brush with global scope
FastLEDPainterCanvas pixelcanvas = FastLEDPainterCanvas(NUM_LEDS); //create
canvas, linked to the FastLED library (canvas must be created before the
brush)
FastLEDPainterBrush pixelbrush = FastLEDPainterBrush(&pixelcanvas); //crete
brush, linked to the canvas to paint to


void setup() {
//initilize FastLED library
FastLED.addLeds<LED_TYPE,DATA_PIN,CLK_PIN,COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);
```

```
  Serial.begin(115200);
  Serial.println(" ");
  Serial.println(F("FastLED Painter simple demo"));

  //check if ram allocation of brushes and canvases was successful (painting
will not work if unsuccessful, program should still run though)
  //this check is optional but helps to check if something does not work,
especially on low ram chips like the Arduino Uno
  if (pixelcanvas.isvalid() == false) Serial.println(F("canvas allocation
problem (out of ram, reduce number of pixels)"));
  else  Serial.println(F("canvas allocation ok"));

  if (pixelbrush.isvalid() == false) Serial.println(F("brush allocation
problem"));
  else  Serial.println(F("brush allocation ok"));

  //initialize the animation, this is where the magic happens:

  CHSV brushcolor; //the brush and the canvas operate on HSV color space
only
  brushcolor.h = 0; //zero is red in HSV. Library uses 0-255 instead of
0-360 for colors (see https://en.wikipedia.org/wiki/HSL_and_HSV)
  brushcolor.s = 255; //full color saturation
  brushcolor.v = 130; //about half the full brightness

  pixelbrush.setSpeed(1200); //set the brush movement speed (4096 means to
move one pixel per update)
  pixelbrush.setColor(brushcolor); //set the brush color
  pixelbrush.setFadeSpeed(130); //fading speed of pixels (255 is maximum
fading speed)
  pixelbrush.setFadeout(true); //do brightness fadeout after painting
  pixelbrush.setBounce(true); //bounce the brush when it reaches the end of
the strip

  //this sets up the brush to paint pixels in red, the pixels fade out after
they are painted (the brush is the size of one pixel and can only one pixel
per brush update, see other examples to paint multiple pixels at once)

}

void loop() {
  FastLED.clear(); //always need to clear the pixels, the canvas' colors
will be added to whatever is on the pixels before calling a canvas update

  pixelbrush.paint(); //paint the brush to the canvas (and update the brush,
i.e. move it a little)
  pixelcanvas.transfer(); //transfer the canvas to the LEDs

  FastLED.show();
}
```

Leds con plantower

platformio.ini

```
;PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:d1_mini]
platform = espressif8266
board = d1_mini
framework = arduino
lib_deps = FastLED, PMS Library,
https://github.com/DedeHai/FastLEDPainter.git
```

main.cpp

```
/**
   Simple animation demo for using the FastLED painter library
   This demo does the Knight Rider scanner effect with just a few lines of
code
   The speed and fadespeed need to be adjusted for different processor
speeds
   Chosen settings work nicely on 60-pixels and an Arduino Duemilenove (Uno)
*/

#include "Arduino.h"
#include <SoftwareSerial.h>
#include "PMS.h"
#include <FastLED.h>
#include <FastLEDPainter.h>

#define P_TOWER_RX D2
#define P_TOWER_TX 6
// Plantower
SoftwareSerial plantower_serial(P_TOWER_RX, P_TOWER_TX);
PMS pms(plantower_serial);
PMS::DATA data;

// Leds
#define DATA_PIN    D7
#define CLK_PIN     D6
```

```
#define LED_TYPE    DOTSTAR
#define COLOR_ORDER BGR
#define NUM_LEDS    30
#define BRIGHTNESS  96
CRGB leds[NUM_LEDS];

//create one canvas and one brush with global scope
FastLEDPainterCanvas pixelcanvas = FastLEDPainterCanvas(NUM_LEDS); //create
canvas, linked to the FastLED library (canvas must be created before the
brush)
FastLEDPainterBrush pixelbrush = FastLEDPainterBrush(&pixelcanvas); //crete
brush, linked to the canvas to paint to

CHSV brushcolor; //the brush and the canvas operate on HSV color space only

void setup() {
  //initilize FastLED library
  FastLED.addLeds<LED_TYPE,DATA_PIN,CLK_PIN,COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);

  plantower_serial.begin(9600);
  Serial.begin(115200);
  Serial.println(" ");
  Serial.println(F("FastLED Painter simple demo"));

  //check if ram allocation of brushes and canvases was successful (painting
will not work if unsuccessful, program should still run though)
  //this check is optional but helps to check if something does not work,
especially on low ram chips like the Arduino Uno
  if (pixelcanvas.isvalid() == false) Serial.println(F("canvas allocation
problem (out of ram, reduce number of pixels)"));
  else  Serial.println(F("canvas allocation ok"));

  if (pixelbrush.isvalid() == false) Serial.println(F("brush allocation
problem"));
  else  Serial.println(F("brush allocation ok"));

  //initialize the animation, this is where the magic happens:

  brushcolor.h = 0; //zero is red in HSV. Library uses 0-255 instead of
0-360 for colors (see https://en.wikipedia.org/wiki/HSL_and_HSV)
  brushcolor.s = 255; //full color saturation
  brushcolor.v = 130; //about half the full brightness

  pixelbrush.setSpeed(1200); //set the brush movement speed (4096 means to
move one pixel per update)
  pixelbrush.setColor(brushcolor); //set the brush color
  pixelbrush.setFadeSpeed(130); //fading speed of pixels (255 is maximum
fading speed)
  pixelbrush.setFadeout(true); //do brightness fadeout after painting
  pixelbrush.setBounce(true); //bounce the brush when it reaches the end of
```

```
the strip

  //this sets up the brush to paint pixels in red, the pixels fade out after
they are painted (the brush is the size of one pixel and can only one pixel
per brush update, see other examples to paint multiple pixels at once)

}

void loop() {

  Serial.println("Leyendo sensor");
  if (pms.readUntil(data)){
    Serial.print("PM2.5 ");Serial.println(data.PM_AE_UG_2_5);
    // brushcolor.h = data.PM_AE_UG_2_5;
    // pixelbrush.setColor(brushcolor);
    // // pixelbrush.setFadeSpeed(data.PM_AE_UG_2_5*3);
    Serial.print("PM10 ");Serial.println(data.PM_AE_UG_10_0);
    // saveDataForAverage(data.PM_AE_UG_2_5,data.PM_AE_UG_10_0);
  }
  else Serial.println("ERROR leyendo sensor");

  Serial.println("Encendiendo leds");
  FastLED.clear(); //always need to clear the pixels, the canvas' colors
will be added to whatever is on the pixels before calling a canvas update

  pixelbrush.paint(); //paint the brush to the canvas (and update the brush,
i.e. move it a little)
  pixelcanvas.transfer(); //transfer the canvas to the LEDs

  FastLED.show();
}
```

Leds + plantower: cambia color segun lectura del sensor

platformio.ini

```
[env:d1_mini]
platform = espressif8266
board = d1_mini
framework = arduino
lib_deps = FastLED, PMS Library,
https://github.com/DedeHai/FastLEDPainter.git, Time
```

main.cpp

```
#include "Arduino.h"
#include <SoftwareSerial.h>
#include "PMS.h"
#include <FastLED.h>
```

```
#include <FastLEDPainter.h>
#include <Time.h>

#define P_TOWER_RX D2
#define P_TOWER_TX 6

// Plantower
SoftwareSerial plantower_serial(P_TOWER_RX, P_TOWER_TX);
PMS pms(plantower_serial);
PMS::DATA data;

int pm2_5 = 0;
int *dir_pm2_5;

// Leds
#define DATA_PIN D7
#define CLK_PIN D6
#define LED_TYPE DOTSTAR
#define COLOR_ORDER BGR
#define NUM_LEDS 26 // estos 10 es para que cuando lea el sensor no se vean
los leds prendidos
#define BRIGHTNESS 96
CRGB leds[NUM_LEDS];

//create one canvas and one brush with global scope
FastLEDPainterCanvas pixelcanvas = FastLEDPainterCanvas(NUM_LEDS);  //create
canvas, linked to the FastLED library (canvas must be created before the
brush)
FastLEDPainterBrush pixelbrush = FastLEDPainterBrush(&pixelcanvas); //crete
brush, linked to the canvas to paint to

CHSV brushcolor; //the brush and the canvas operate on HSV color space only

int color = 0;
int speed = 400;

int time_hours = 0;
int time_minutes = 0;
int time_seconds = 0;

void readPlantower();
void animateLeds();
void updateValues();

void setup() {
  //initilize FastLED library
  FastLED.addLeds<LED_TYPE, DATA_PIN, CLK_PIN, COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);

  plantower_serial.begin(9600);
  Serial.begin(115200);
```

```
  Serial.println(" ");
  Serial.println(F("FastLED Painter simple demo"));

  //check if ram allocation of brushes and canvases was successful (painting
will not work if unsuccessful, program should still run though)
  //this check is optional but helps to check if something does not work,
especially on low ram chips like the Arduino Uno
  if (pixelcanvas.isvalid() == false)
    Serial.println(F("canvas allocation problem (out of ram, reduce number
of pixels)"));
  else
    Serial.println(F("canvas allocation ok"));

  if (pixelbrush.isvalid() == false)
    Serial.println(F("brush allocation problem"));
  else
    Serial.println(F("brush allocation ok"));

  //initialize the animation, this is where the magic happens:
  brushcolor.h = color; //zero is red in HSV. Library uses 0-255 instead of
0-360 for colors (see https://en.wikipedia.org/wiki/HSL_and_HSV)
  brushcolor.s = 255;   //full color saturation
  brushcolor.v = 130;   //about half the full brightness

  pixelbrush.setSpeed(speed);      //set the brush movement speed (4096
means to move one pixel per update)
  pixelbrush.setColor(brushcolor); //set the brush color
  pixelbrush.setFadeSpeed(130);    //fading speed of pixels (255 is maximum
fading speed)
  pixelbrush.setFadeout(true);     //do brightness fadeout after painting
  pixelbrush.setBounce(true);      //bounce the brush when it reaches the
end of the strip

  // set pointer
  dir_pm2_5 = &pm2_5;

  // setup time
  setTime(time_hours, time_minutes, time_seconds, 30, 11, 19);
}

void loop() {
  readPlantower();
}

void readPlantower() {
  int sec = second();
  if (sec == 0 || sec == 15 || sec == 20 || sec == 30 || sec == 40 || sec ==
50 ||
      sec == 5 || sec == 10 || sec == 35 || sec == 35 || sec == 45 || sec ==
55) {
    pms.wakeUp();
```

```
    if (pms.read(data)) {
      pm2_5 = data.PM_AE_UG_2_5;
      delay(500);
      updateValues();
    }
    Serial.print("Leyendo pm 2.5 ");
    Serial.println(*dir_pm2_5);
  }
  else {
    Serial.print("durmiendo. Ultimo valor ");
    Serial.println(*dir_pm2_5);
    pms.sleep();
    animateLeds();
  }
  // Serial.print("sec: ");
  // Serial.println(sec);
}

void animateLeds(){
  // Serial.println("Encendiendo leds");
  FastLED.clear();        //always need to clear the pixels, the canvas'
colors will be added to whatever is on the pixels before calling a canvas
update
  pixelbrush.paint();     //paint the brush to the canvas (and update the
brush, i.e. move it a little)
  pixelcanvas.transfer(); //transfer the canvas to the LEDs
  FastLED.show();
}

void updateValues() {
  // varia en los que varia
  /*
    233 minimo
    1225 maximo
  */
  if (*dir_pm2_5 < 400){
    brushcolor.h = 90; // aire bueno verde
    pixelbrush.setColor(brushcolor);
    pixelbrush.setFadeout(true);
    pixelbrush.setBounce(true);
  } else if (*dir_pm2_5 > 400 ) {
    brushcolor.h = 200; // aire regular azul
    pixelbrush.setColor(brushcolor);
    pixelbrush.setFadeout(true);
    pixelbrush.setBounce(true);
  } else if (*dir_pm2_5 > 800) {
    brushcolor.h = 155; // aire malo ?
    pixelbrush.setColor(brushcolor);
    pixelbrush.setFadeout(true);
    pixelbrush.setBounce(true);
  } else if (*dir_pm2_5 > 1200) {
```

```
    brushcolor.h = 0; // magenta aire podrido
    pixelbrush.setColor(brushcolor);
    pixelbrush.setFadeout(true);
    pixelbrush.setBounce(true);
  }
}
```

From:
https://wiki.unloquer.org/ -

Permanent link:
https://wiki.unloquer.org/proyectos/vestuario_aqa?rev=1575612256

Last update: **2019/12/06 06:04**