Apartado técnico y especificaciones para Manual para el topoanálisis del hogar

Cassiopea Acebes Prado

28 de febrero de 2022

Índice

1. Introducción

En este documento se recoge la documentación técnica de la obra *Manual para el topoanálisis del hogar* para la pieza *Manual para el topoanálisis del hogar*, independientemente de su formato, para la generación, disposición, proyección y control de la proyección de las normas.

También incluye una serie de recomendaciones (Sección *Recomendaciones*) y anexos fotográficos y de código (Sección *Resultados*). Estos se utilizarán para explicar el funcionamiento normal de los componentes; para dar posibles adaptaciones de cara a la instalación física de la pieza; facilitar la cita, referencia o mención de la documentación técnica, diseño o funcionamiento de los componentes; y posibilitar la recuperación de piezas de código para su uso en la instalación.

2. Desarrollo

2.1. Proceso Generación normas

2.1.1. Introducción

En esta subsección se va a documentar el software para la generación de las normas de la pieza *Manual para el topoanálisis del hogar*. Se tratarán los requisitos del componente, su funcionamiento y sus dos salidas, en la Sección de *Resultados* se adjuntará una copia del código básico.

2.1.2. Requisitos

Este componente debe hacer lo siguiente:

- Tomar como *input* las normas recopiladas.
- Generar, a partir de las normas originales, nuevas normas.
- Dar una salida por escrito y por vídeo de las nuevas.

Se debe tener en cuenta que las nuevas normas son de naturaleza aleatoria, es decir que no tienen que tener un sentido lógico, y que se pueden adaptar las normas originales para poderse utilizar en el *software*.

2.1.3. Funcionamiento

Las normas recopiladas pasan por un proceso de normalización para su posterior procesamiento, para ello se determina una fórmula general o estructura sintáctica. Esta fórmula se utiliza en todos los puntos del procesamiento, se compone de:

Adv.Neg. + verbo + sujeto + CCL + CCT + CCM + Consecuencia

Donde **Adv.Neg.** se trata de un adverbio de negación, **CCL** de un complemento circunstancial de lugar, **CCT** un complemento circunstancial de tiempo, **CCM** un cc de modo y **consecuencia** una estructura que actúa como consecuencia a la norma. No todas las normas contienen todos los componentes, por lo que se permite que haya componentes vacíos.

Una vez tomadas las partes de las normas recopiladas, estas pasan por una doble selección aleatoria. En primer lugar se selecciona una estructura sintáctica para la nueva norma, a las diferentes estructuras se las llama *seeds* y contienen variaciones de la estructura original, añadiendo o quitando componentes. Una vez hecha la selección de la estructura de los componentes, cada uno de ellos se selecciona aleatoriamente. Esto permite que se genere una variedad ciertamente orgánica a las normas generadas, algunos ejemplos de las *seeds* son:

```
negacion + verbo + sujeto + ccl
negacion + verbo + consecuencia
negacion + verbo + sujeto + ccl +
    ccm
```

Se explica línea por línea el código del componente, realizado en *Python* en su versión 3.8.5:

Para poder elegir números aleatorios, y de esta forma seleccionar las partes de las normas aleatoriamente, se requiere de un módulo específico llamado 'secrets'. Se le llama con la línea 8 y para utilizar sus funciones se le trata como un objeto.

```
import secrets
```

Se van a generar archivos con normas, para que cada vez que se ejecute el generador no se sobreescriba el anterior archivo se incluirá la fecha y hora en que se crearon. Para esto se utiliza el módulo 'datetime', que puede dar la fecha y hora momentáneamente.

```
import datetime
```

Se toman las variables para construir las frases, para ello se lee un archivo y se forma un array (una 'lista') con las líneas de texto de cada archivo.

Las líneas de lectura de los archivos tienen tres partes:

- **negacion =**
  Se asigna el array de la lectura del archivo a la variable 'negacion'.
- **open('partes-normas/negacion','r')**
  Lee el archivo 'negacion' de la carpeta 'partes-normas'.
- **.read().splitlines()**
  Asigna cada línea del archivo a un ítem del array.

De esta forma la variable 'negacion' terminaría funcionando como:

negacion['no', 'jamás',(...)]

Aligerando así el código del generador

```
negacion = open('partes-normas/
    negacion','r').read().splitlines
    ()
verbo = open('partes-normas/verbo','
    r').read().splitlines()
sujeto = open('partes-normas/sujeto'
    ,'r').read().splitlines()
ccl = open('partes-normas/ccl','r').
    read().splitlines()
cct = open('partes-normas/cct','r').
    read().splitlines()
ccm = open('partes-normas/ccm','r').
    read().splitlines()
consecuencia = open('partes-normas/
    consecuencia','r').read().
    splitlines()
```

Para hacer más visual el código, se va a crear un alias de la función 'choice()' del módulo secrets llamado 'random()', así cada vez

que se quiera seleccionar un elemento aleatorio de un array de datos aparecerá como 'random(y)' en vez de como 'secrets.choice(y)'. Esto es puramente visual.

```
def random(a):
    return secrets.choice(a)
```

Ahora se definen las funciones que actúan como seeds, cada seed es una estructura de normas, de esta forma las seeds (denominadas seed0, seed1,...seed10) tienen diferentes componentes, como una negación, un complemento circunstancial de un tipo u otro o una consecuencia. De esta forma se crean frases con diversas estructuras, lo que hace menos repetitivas las normas.

El funcionamiento de una seed N es el siguiente:

- **def seedN():**
  Es la declaración de la función.
- **x=”**
  Inicializa la variable x, en la que se añadirán las partes de la norma.
- **x += random(negacion) + ' '**
  Añade a la variable x vacía una negación aleatoria y añade un espacio.
- **x += random(verbo) + ' '**
  Añade a la negación y al espacio un verbo aleatorio con un espacio delante.
- **x += random(sujeto) + ' '**
  Añade a la negación y al verbo un sujeto. Añade un espacio delante, aunque no es necesario.
- **return x.strip('_ ')**
  Devuelve la norma guardada en la variable x, le quita un posible '_' del archivo de negaciones. Al devolver la variable esta se convierte en la salida de la funcion, es como si fuera su resultado.

Se pueden añadir y quitar partes a las seeds para dar variedad a las frases.

```
def seed0():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    x += random(ccl) + ' '
    x += random(cct) + ' '
    x += random(ccm) + ' '
    x += random(consecuencia) + ' '
    return x.strip('\_ ')


def seed1():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    return x.strip('\_ ')


def seed2():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    x += random(ccl) + ' '
    x += random(ccm) + ' '
    return x.strip('\_ ')

def seed3():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    return x.strip('\_ ')

def seed4():
    x=''
    x += random(negacion) + ' '
```

```
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    x += random(cct) + ' '
    x += random(ccm) + ' '
    return x.strip('\_ ')

def seed5():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(consecuencia) + ' '
    return x.strip('\_ ')

def seed6():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    x += random(cct) + ' '
    return x.strip('\_ ')

def seed7():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    x += random(ccm) + ' '
    return x.strip('\_ ')

def seed8():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    x += random(ccm) + ' '
    return x.strip('\_ ')

def seed9():
    x=''
    x += random(negacion) + ' '
    x += random(verbo) + ' '
    x += random(sujeto) + ' '
    x += random(ccl) + ' '
    x += random(cct) + ' '
    return x.strip('\_ ')
```

```
def seed10():
    x=''
    x += random(negacion) + ' '
    x += random(sujeto) + ' '
    x += random(ccl) + ' '
    x += random(cct) + ' '
    x += random(ccm) + ' '
    return x.strip('\_ ')
```

Ahora se crea el nombre del archivo, que tendrá la estructura:

output-YYYY-MM-DD_HH.MM.SS.mmmmm.txt

- **'generadas/output-'**
  Guarda en la carpeta 'generadas', el nombre del archivo comienza por 'output'.
- **str(datetime.datetime.now()).replace(' ', '_').replace(':', '.')**
  Obtiene la fecha y hora y las formatea.
- **'.txt'**
  Terminación del archivo, al ser este de texto.

En la siguiente línea a la de la inicialización de nombre_archivo se 'crea' (abre) un archivo vacío con ese nombre, la opción 'w' indica que se va a escribir.

```
nombre\_archivo = 'generadas/output
   -'+str(datetime.datetime.now()).
   replace(' ', '\_').replace(':',
   '.')+'.txt'
output = open(nombre\_archivo, 'w')
```

Ahora se utiliza un bucle para que se añadan 50 normas generadas aleatoriamente al archivo abierto. Ahora se explicarán las partes de la línea de output.(...):

- **output.write(...)**
  Lo que se encuentre dentro del paréntesis se va a escribir en el archivo abierto.
- **locals()[...]() + '\n'**
  locals() se trata de una 'lista' con las funciones del programa, entre otras funciones de python se encuentran las seeds. Para seleccionar una seed en concreto se utilizan los corchetes, de forma que entre ellos se escribe el nombre de la función que se quiera utilizar. Si a este locals()[...] se le añade un paréntesis '()', se ejecutará la función que se haya escrito en los corchetes. Por ejemplo 'locals()[seed5]()' ejecutará la función seed5(). Esto se utiliza porque interesa que lo que se ejecute (lo que hay entre los corchetes) sea aleatorio. Y esto es más sencillo si se llama a la seed desde locals(). Por último, el (+ '\n') sirve para que haya un salto de línea al final de cada norma.
- **'seed' + str(secrets.randbelow(11))**
  Esto es lo que se encontraría dentro de los corchetes, de igual forma que con el nombre del archivo se le añade a la palabra 'seed' un número aleatorio por debajo de 11. Este número debe ser convertido a formato texto para poderse adjuntar a seed, de ahí que se utilice el str(...).

En conclusión, por cada iteración del bucle se añade al archivo el 'resultado' (la norma correspondiente) de una de las funciones seed. Siendo esta elegida aleatoriamente.

```
for i in range(50):
    output.write(locals()['seed' +
   str(secrets.randbelow(11))]()+'\
   textbackslash n')
```

Por último, se cierra el archivo abierto con close(). Esto se suele utilizar para que en archivos grandes la memoria del ordenador no se sobrecargue, en este caso no resulta del todo necesario pero se trata de una buena práctica de programación.

```
output.close()
```

2.1.4. Salidas

Para este componente existen dos salidas; una en forma de archivo de texto, considerada como la predeterminada, que crea el archivo que referencia el código anterior con cincuenta (50) normas; y una salida por consola, utilizada en la proyección de las normas en conjunto con el siguiente componente (Subsección *Proceso Proyección Manual*).

Para este segundo componente se sustituirían las dos últimas líneas (que referencian a la creación de un archivo) por esto:

```
  for i in range(20000):
        locals()['seed' + str(
    secrets.randbelow(11))]()+"\n"
```

Esta pieza de código hace que se ejecute la función *seedX*, con X un número aleatorio, y añada al final de cada ejecución un salto de línea. Se requiere también de añadir la siguiente línea a cada función de *seed*, para que la nueva norma salga por consola.

```
print(x.strip("_ "))
```

2.2. Proceso Proyección normas

2.2.1. Introducción

En esta subsección se va a documentar el software y el montaje del hardware para la proyección de las normas de la pieza *Manual para el topoanálisis del hogar*. Se tratarán los requisitos del componente, su montaje y el funcionamiento de su *software* en la Sección de *Resultados* se adjuntará una copia del código y pictogramas del montaje.

2.2.2. Requisitos

Este componente debe poder hacer lo siguiente:

- Detectar personas en una determinada zona (zona de detección ó ZDD).
- En caso de detectar a alguien en la ZDD, abrir el paso a la luz de un proyector.
- Cuando deje de detectar a alguien, cerrar el paso de la luz del proyector.

Por las características del proyector este componente es puramente de hardware, haciendo uso de una PLC Arduino Uno con los módulos de Servo Motor SG90, Sensor ultrasonidos HC-SR04 y una placa de pruebas modular.

2.2.3. Funcionamiento

Para la proyección de las normas se va a utilizar un proyector estándar configurado para dar la salida del Generador de normas por consola. La imagen que va a tener el proyector es una fuente de vídeo con texto integrado, el cual se va generando dinámicamente. Para abrir y cerrar la imagen se va a utilizar un trozo rectangular (o de una forma similar para mejorar la absorción de luz) de cartulina negra. Este trozo de cartulina irá montada en el servo, que se accionará con la arduino y el sensor.

Para unir los distintos métodos se utilizará la placa de pruebas (*protoboard*), conectándose mediante cables macho-macho de la siguiente forma (una 'pista' es una de las filas de la placa):

- Pista 1:
  - Arduino: +5V.
  - Sensor: VCC.
  - Servo: Cable Rojo.
- Pista 2:
  - Arduino: Gnd.
  - Sensor: Gnd.
  - Servo: Cable Marrón.
- Pista 3:
  - Arduino: D3.
  - Sensor: Trig.
- Pista 4:
  - Arduino: D2.
  - Sensor: Echo.
- Pista 5:
  - Arduino: D7.
  - Servo: Cable amarillo.

También se debe de resolver el problema de la detección, este problema de apariencia sencilla tiene infinidad de soluciones, desde la más elemental; limitarse a detectar algo cuando pasa de cierta distancia (x <d),

hasta el uso de múltiples sensores e incluso cámaras. La solución que se ha apostado es la de utilizar un cierto rango de sensibilidad, a efectos de evitar las detecciones por ruido, que permite hacer detecciones cuando la distancia detectada se sale de este rango. Dado:

d $\rightarrow$ distancia a la pared, o la distancia detectada base

x $\rightarrow$ distancia leída en cada iteración (se supone que esta función se ejecuta todo el rato en bucle)

p $\rightarrow$ permisividad de la fórmula, permite establecer un rango en el que la detección se mantenga nula (reduce el movimiento errático del brazo).

Se utiliza la siguiente fórmula para la detección de personas u objetos:

$$s = \left(\frac{x-d}{p}\right)^2 \begin{cases} s < 1 & False \\ s >= 1 & True \end{cases}$$

Donde False es que no hay detección y True que detecta un cuerpo.

El cociente (s) es el que determina la detección. Si (s) es inferior a 1, se considera que no hay nada/nadie en la zona de detección; si (s) es igual o mayor que 1, se considera que hay un objeto o persona en la zona de detección. Esta aproximación a una detección efectiva utiliza el cuadrado del valor para añadirle sensibilidad al resultado sin reducir la velocidad del cálculo, que es lo que interesa en este detector.

Para la obtención de los casos del cociente (s) y el divisor interno se ha calculado que a una distancia base de 300cm, considerando una diferencia de ruido de +-10cm, no hay nadie siendo detectado. Esta fórmula funciona tanto si hay alguien que de alguna forma cancele el ruido (caso x ->∞) como si este alguien se encuentra muy cerca del detector (caso x <300+-10cm).

Se expone el código del programa que hace funcionar el detector y el servo:

Se definen los pines que se van a utilizar y se declaran las variables globales, también se incluye la librería de control del servo:

```
#include <Servo.h>

#define echoPin 2 // Pin D2 (Echo)
#define trigPin 3 // Pin D3 (Trig)
#define servPin 7 // Pin D7 (Serv)

long duration; // Tiempo de viaje
   del sonido
int distancia; // Distancia
   detectada en cada instante

long s; // Cociente del detector
int base = 1184; // Distancia base
   del detector (zona de detección
   vacía)
int permisividad = 20; //Varianza de
    la distancia
int giro = 90; // Grados de giro del
    servo
int giro = 50; // Grados de giro del
    servo
```

Esta línea inicia el servo en la vabiable servo, que se utilizará para su control:

```
Servo servo;
```

Se comienzan las salidas por consola por el el terminal serial, de esta forma se podrán ver y analizar las lecturas que tome el sensor, el estado del servo y otra información de

relevancia:

```
void setup() {
  Serial.begin(9600);
```

Para la toma de medidas se declaran los Pins del sensor como de salida y entrada y se le asigna al servo el pin definido

El brazo del servo se pone a cero:

```
  pinMode(trigPin, OUTPUT); // Sets
    the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets
    the echoPin as an INPUT
  servo.attach(servPin); //Servo al
    pin declarado

  servo.write(0); //Hace el cero del
     brazo del servo

}
```

Se declara el loop principal del programa, en este se va a primero tomar una medida por media, se le declara al valor 'average' el resultado de la media, luego se toma una medida real y se calcula el estado del brazo, en caso de ser positivo subir; en caso negativo, bajará.

Se hace una impresión por el monitor del estado del sensor, esto ayudará a la depuración y al ajuste en el montaje.

Con este estado se mueve el brazo del servo los grados anteriormente declarados:

```
void loop() {
  float average = calc_average();
  medir();
  bool state = detect(average, base)
    ;

  Serial.print("Cociente: ");
  Serial.print(s);
  Serial.print("  Distancia: ");
  Serial.print(distancia);
  Serial.print("  Average: ");
  Serial.println(average);

  mover(state, giro);
}
```

Esta función controla el movimiento del servo, toma un valor booleano, que se toma como el estado de la zona y los grados que se va a mover el servo. Segun el estado se abrirá el brazo o se pondrá a cero:

```
void mover(bool check,int grados){
  if (check) {
    //Serial.println("Accionando
    brazo.");
    servo.write(grados);
  } else {
    //Serial.println("Recogiendo
    brazo.");
    servo.write(0);
  }
}
```

Esta función controla la detección de personas en la zona de detección. Toma como valores la distancia medida en un momento determinado y la compara con la base declarada. Usando la fórmula anteriormente descrita emite un true para la detección y un false como estado natural:

```
bool detect(long detect, int base){
    s = pow(((detect - base)/
   permisividad), 2);
    if (s >= 1){
        return true;
    } else {
        return false;
    }
}
```

Esta función genera una media entre los 25 últimos valores detectados, de esta forma se reducen los picos y ruido en las medidas

tomadas, el valor 0.25 es un valor que se ha obtenido como el que menos errores produce en las pruebas de montaje:

```
float calc_average()
{
  static float average = -1.0 ;
  if (average == -1.0)  // special
   case for first reading.
  {
    average = medir() ;
    return average ;
  }
  average += 0.25 * (medir() -
   average) ;  // 0.1 for approx 10
   sample averate, 0.01 for 100
   sample etc.
  return average ;
}
```

Esta función controla el sensor y determina la distancia en todo momento, para ello baja el voltaje del pin de trigger, le da un pequeño delay y lo activa durante un breve período de tiempo, tras el cual vuelve a bajar su voltaje.

Una vez hecho esto lee la subida de voltaje por el pin de echo, este será el tiempo de la onda de sonido en ir y volver del sensor, se calcula la distnacia como la mitad de la duración multiplicado por un cierto cociente obtenido de: v(m/s)*1/1000(s/us)*100/1(cm/m) = 0.034(cm/us)

```
long medir(){
  digitalWrite(trigPin, LOW);
  delayMicroseconds(200);//200
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distancia = duration * 0.034 / 2;
  return distancia;
}
```

3. Conclusiones y Recomendaciones

Para los distintos componentes se han podido resolver adecuadamente los requerimientos y problemas planteados. Se recomienda para el correcto funcionamiento del componente de generación y del de proyección, el ajuste *in situ* de las distintas variables y constantes.

También se dan las recomendaciones de; ajustar la consola de salida del generador de normas en su salida a terminal, esto por los posibles problemas que haya con la codificación de las normas generadas; y de hacer pruebas de detección, movimiento y ajuste del brazo del servo del componente de proyección de normas.

4. Bibliografía

- Ultrasonic Sensor HC-SR04 with Arduino Tutorial. (s. f.). Arduino Project Hub. Recuperado 10 de febrero de 2022, de https://create.arduino.cc/projecthub /abdularbi17/ ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6
- Hernández, L. D. V. (2021, 23 marzo). Servomotor con Arduino tutorial de programación paso a paso. Programar fácil con Arduino. Recuperado 10 de febrero de 2022, de https://programarfacil.com/blog/arduino-blog/servomotor-con-arduino/

5. Resultados

```
import secrets
import datetime

negacion = open("partes-normas/negacion","r").read().splitlines()
verbo = open("partes-normas/verbo","r").read().splitlines()
sujeto = open("partes-normas/sujeto","r").read().splitlines()
ccl = open("partes-normas/ccl","r").read().splitlines()
cct = open("partes-normas/cct","r").read().splitlines()
ccm = open("partes-normas/ccm","r").read().splitlines()
consecuencia = open("partes-normas/consecuencia","r").read().splitlines()

def random(a):
    return secrets.choice(a)

def seed0():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    x += random(ccl) + " "
    x += random(cct) + " "
    x += random(ccm) + " "
    x += random(consecuencia) + " "
    return x.strip("_ ")


def seed1():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    return x.strip("_ ")


def seed2():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    x += random(ccl) + " "
    x += random(ccm) + " "
    return x.strip("_ ")
```

```
def seed3():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    #x += random(consecuencia) + " "
    return x.strip("_ ")

def seed4():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    x += random(cct) + " "
    x += random(ccm) + " "
    return x.strip("_ ")

def seed5():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(consecuencia) + " "
    return x.strip("_ ")

def seed6():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    x += random(cct) + " "
    return x.strip("_ ")

def seed7():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    x += random(ccm) + " "
    return x.strip("_ ")

def seed8():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
```

```
    x += random(ccm) + " "
    #x += random(consecuencia) + " "
    return x.strip("_ ")

def seed9():
    x=""
    x += random(negacion) + " "
    x += random(verbo) + " "
    x += random(sujeto) + " "
    x += random(ccl) + " "
    x += random(cct) + " "
    return x.strip("_ ")

def seed10():
    x=""
    x += random(negacion) + " "
    x += random(sujeto) + " "
    x += random(ccl) + " "
    x += random(cct) + " "
    x += random(ccm) + " "
    #x += random(consecuencia) + " "
    return x.strip("_ ")


nombre_archivo = "generadas/output-"+str(datetime.datetime.now()).replace
   (" ", "_").replace(":", ".")+".txt"
output = open(nombre_archivo, "w")


while True:
    output.write(locals()['seed' + str(secrets.randbelow(11))]()+"\n")

output.close()
```

Código básico para la salida por archivo del Generador de normas.

```
#include <Servo.h>

#define echoPin 2 // Pin D2 (Echo)
#define trigPin 3 // Pin D3 (Trig)
#define servPin 7 // Pin D7 (Serv)

long duration; // Tiempo de viaje del sonido
int distancia; // Distancia detectada en cada instante

long s; // Cociente del detector
int base = 1184; // Distancia base del detector (zona de detección vacía)
int permisividad = 20; //Varianza de la distancia
int giro = 90; // Grados de giro del servo

Servo servo;

void setup() {

  Serial.begin(9600);

  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
  servo.attach(servPin); //Servo al pin declarado

  servo.write(0); //Hace el cero del brazo del servo
}

void loop() {
  float average = calc_average();
  medir();
  bool state = detect(average, base);

  Serial.print("Cociente: ");
  Serial.print(s);
  Serial.print("  Distancia: ");
  Serial.print(distancia);
  Serial.print("  Average: ");
  Serial.println(average);

  mover(state, giro);
}


void mover(bool check,int grados){
  if (check) {
```

```
    //Serial.println("Accionando brazo.");
    servo.write(grados);
  } else {
    //Serial.println("Recogiendo brazo.");
    servo.write(0);
  }
}


bool detect(long detect, int base){
    s = pow(((detect - base)/permisividad), 2);
    if (s >= 1){
        return true;
    } else {
        return false;
    }
}


float calc_average()
{
  static float average = -1.0 ;
  if (average == -1.0)  // special case for first reading.
  {
    average = medir() ;
    return average ;
  }
  average += 0.25 * (medir() - average) ;  // 0.1 for approx 10 sample
   averate, 0.01 for 100 sample etc.
  return average ;
}

long medir(){
  digitalWrite(trigPin, LOW);
  delayMicroseconds(200);//200
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distancia = duration * 0.034 / 2;
  return distancia;
}
```

Código base para el funcionamiento del componente de proyección de normas.

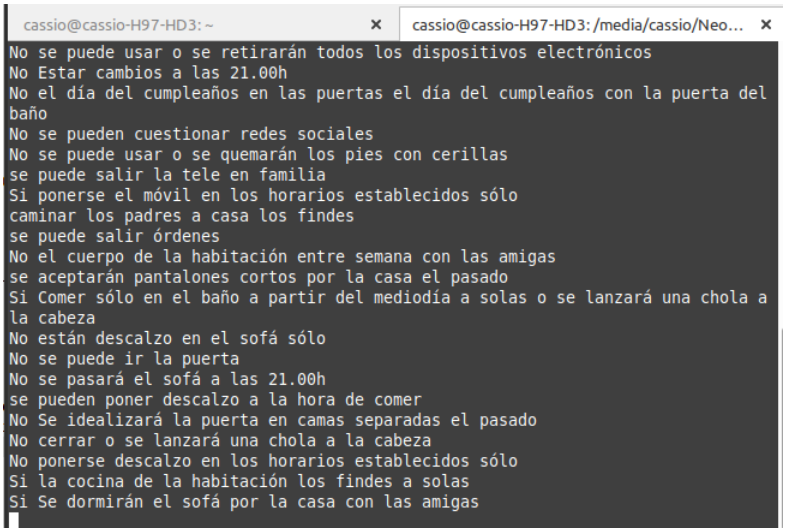

Figura 1: Normas generadas en un terminal en Linux

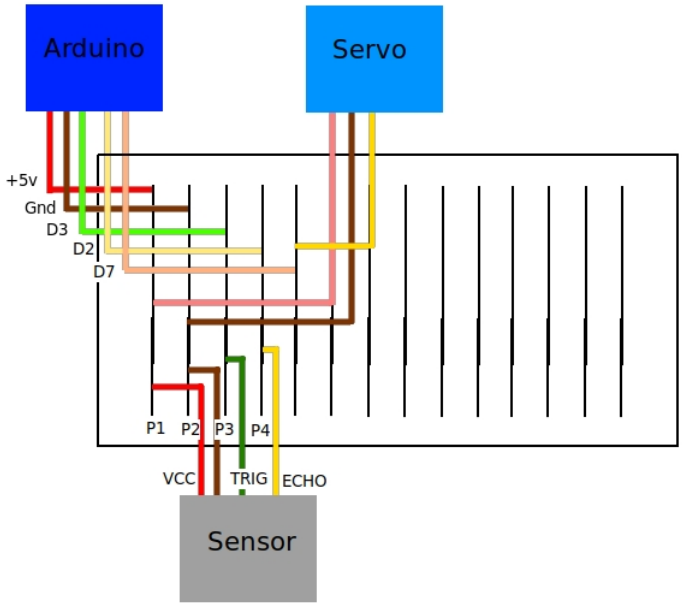

Figura 2: Modelo de la Placa de circuitos

Figura 3: Normas generadas siendo proyectadas en una prueba de montaje