
TELESCOPIO CON AUTOENFOQUE

Sofía Guevara Montoya¹ and Andres Felipe Vargas Londoño¹

¹*Departamento de Física, Universidad Nacional de Colombia, Ciudad Universitaria, Bogotá, Colombia*

(28 de junio de 2022)

Resumen

El proyecto resuelve el enfoque de un telescopio con ayuda de un motor, un Arduino UNO, un controlador TEXAS L293D y los botones que indican el arranque y cambio de dirección del motor. En el trayecto Python va evaluando cada imagen y escoge la de mayor enfoque.

Palabras clave: Telescopio, imagen, circuito, lentes, enfoque, arduino, motor, dirección.

Descripción del montaje

Montaje mecánico

La Figura 1 muestra el montaje mecánico que se realizó. El trípode fue ensamblado por nosotros y adecuamos el motor en la perilla de enfoque, para que moviera el enfoque del telescopio. El montaje contiene un ocular de 26 mm, una lente de 60 mm, un trípode y diferentes materiales como tornillos para el ensamble.

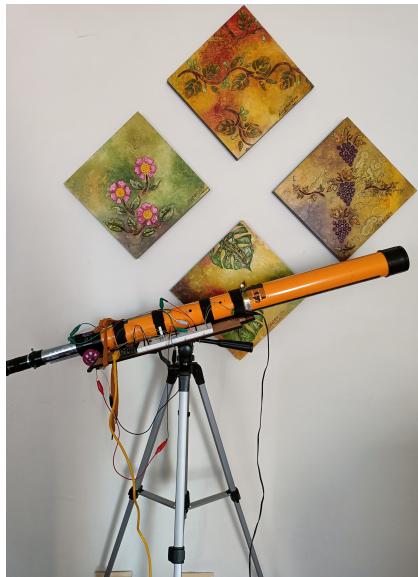


Figura 1: Montaje mecánico

Descripción del circuito

En la figura 2 vemos el circuito mucho más detallado y en la figura 3 visualizamos exactamente como funciona el circuito. Podemos verlo detalladamente en el [simulador](#). Allí encontramos directionPin que indica cuando debe parar el motor. El controlPin1 y controlPin2 son los pines en donde el Arduino indica cómo se tiene que mover el motor. En la práctica onOffPin y directionPin son los pulsadores que manejamos, en donde el Arduino hace la lectura una única vez, sin embargo en el simulador usamos interruptores para mostrar que una vez se oprime onOffPin se debe mantener prendido el motor y cuando directionPin es activado se debe apagar. Las compuertas AND que vemos representan la lógica que tiene al Arduino pero en la práctica no empleamos esas compuertas. La construcción de la simulación fue mejor entendida gracias a [1] y el entendimiento del controlador TEXAS L293D a [2].

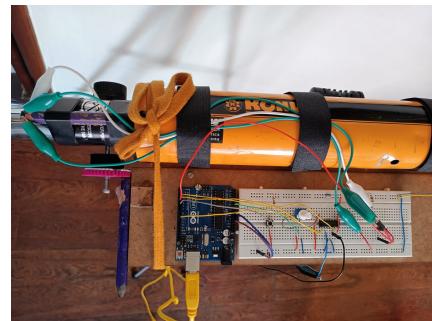


Figura 2: Circuito

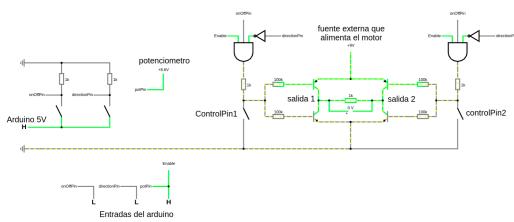


Figura 3: Simulación del circuito

Funcionamiento general

Primero se compila el código del Arduino y después el de Python, una vez hecho esto se arranca con el botón OnOffSwitchPin. El tubo de autoenfoque esta en su máxima distancia, al oprimir el botón, el motor se enciende y traslada el tubo hasta su menor distancia. A medida que ocurre esto Python va evaluando cada imagen que recibe empleando el método de Canny, este revisa los bordes que tiene una imagen. Cuando hay una imagen desenfocada no encuentra tantos bordes como cuando tiene la imagen mas enfocada. Al accionar el botón de cambio de dirección (directionSwitchPin) le indica a Python que ya no tome mas datos, para que él envíe el tiempo en el que obtuvo el mayor enfoque. Por lo cual el arduino recibe ese tiempo y se devuelve el tiempo total del movimiento del ocular menor el tiempo recibido por Python, llegando así a la imagen con mayor enfoque.

En las figuras 4 y 5 vemos un ejemplo del resultado de compilar la simulación. La imagen a la izquierda es la imagen con la que comienza el telescopio y a la derecha encontramos la imagen que analiza Python, es decir, la más enfocada.



Figura 4: Comparación entre imágenes tomadas de un letrero.



Figura 5: Comparación entre imágenes tomadas de edificios.

La siguiente imagen muestra además una referencia de los lugares observados sin emplear el telescopio, se optó por objetos distantes dado que la calidad de la imagen era mejor en estos casos:



Figura 6: Vista de los lugares observados a ojo desnudo.

La Figura 7 muestra una escala de las distancias manejadas en la práctica. El punto donde se ubicó el telescopio se encuentra en el barrio Paulo VI, mientras que el edificio que se muestra en la Figura 8 es la Clínica Universitaria Colombia, aproximadamente a unos 2,11 km de distancia del punto de medición.



Figura 7: Referencia de distancias manejadas en la práctica.

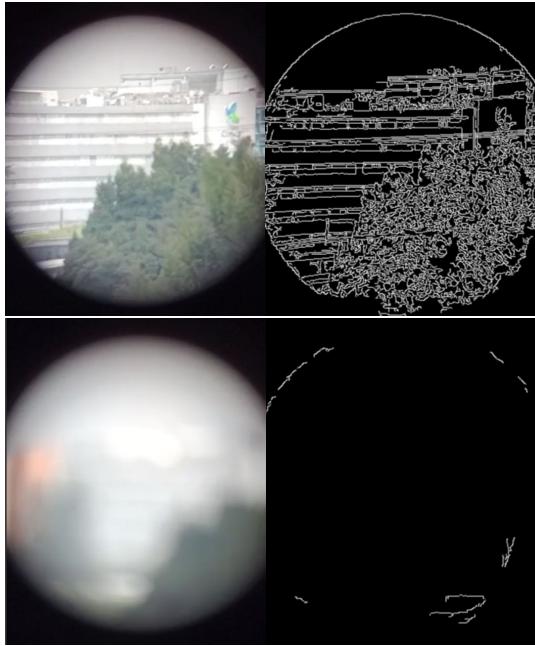


Figura 8: Comparación de Canny para imágenes enfocadas y desenfocadas.

En las imágenes de la Figura 8 vemos los resultados arrojados por el método Canny para una toma enfocada y otra desenfocada, esto para la simula-

ción compilada de la imagen inferior de la Figura 5. Como vemos en las imágenes de la derecha, se muestran los bordes que detecta el programa, viendo algo mucho mas definido en la toma enfocada.

El metodo Canny analiza la imagen en escala de grises, lo que le permite evaluar el contraste de la imagen revisando los gradientes de intensidad entre píxeles adyacentes. Una imagen enfocada (con muchos bordes) tiene gradientes de intensidad altos, por esto los números arrojados por el programa son mayores para imágenes con mejor enfoque, mientras que en imágenes desenfocadas los gradientes de intensidad son menores.

Código fuente

Arduino

A continuación se muestra el setup y loop usado por el arduino, este establece comunicación con Python, indicándole cuando arranca su movimiento y cuando finaliza. Gracias a el libro [3] entendimos como funcionaban los códigos y de acuerdo a esto lo escribimos.

```

1 void setup() {
2   Serial.begin(9600);
3   Serial.setTimeout(1000);
4   pinMode(controlPin1, OUTPUT);
5   pinMode(controlPin2, OUTPUT);
6   pinMode(enablePin, OUTPUT);
7   pinMode(directionSwitchPin, INPUT);
8   pinMode(onOffSwitchStateSwitchPin, INPUT);
9   pinMode(potPin, INPUT);
10  analogWrite(enablePin, 0);
11 }
12
13 void loop() {
14   //Lee el estado de prendido y la dirección
15   onOffSwitchState = digitalRead(
16     ↪ onOffSwitchStateSwitchPin);
17   directionSwitchState = digitalRead(
18     ↪ directionSwitchPin);
19
20   //Prende el motor
21   if (onOffSwitchState == HIGH &&
22     ↪ previousOnOffSwitchState == LOW){
23     motorSpeed = analogRead(potPin)/4;
24     delay(100);
25     Serial.println(1); //Comunica a Python su
26     ↪ inicio
27     time_start = millis();
28     digitalWrite(controlPin1, LOW);
29     digitalWrite(controlPin2, HIGH);
30     analogWrite(enablePin, motorSpeed);
31     previousOnOffSwitchState = HIGH;
32   }
33 }
```

```

30 //Para el motor
31 if (directionSwitchState == HIGH &&
     ↪ previousDirectionSwitchState == LOW){
32   analogWrite(enablePin, 0);
33   time_parada = millis() - time_start;
34   Serial.println(0); //Comunica a Python su fin
35   while (Serial.available() == 0){
36     delay(1000);
37   }
38
39 //Tiempos
40 float time_max = 1000.0*Serial.parseFloat();
41 float move_time = time_parada - time_max;
42
43 //Cambio de dirección
44 motorSpeed = analogRead(potPin)/4;
45 digitalWrite(controlPin1, HIGH);
46 digitalWrite(controlPin2, LOW);
47 analogWrite(enablePin, motorSpeed);
48 time_end = millis();
49
50 //Condición para que el motor sepa hasta donde
     ↪ retornar de acuerdo al tiempo enviado
     ↪ por Python
51 while (millis() - time_end < move_time){}
52 analogWrite(enablePin, 0);
53 while (Serial.available() == 0){}
54 previousDirectionSwitchState = HIGH;
55 done = 1;
56
57 }
58 }
```

Listing 1: Código del Arduino para el movimiento del motor y su cambio de dirección.

Python

Los siguientes son los códigos empleados en Python, estos evalúan la mejor imagen y le comunican al arduino en qué tiempo fue encontrada.

```

1 from tools import lectura_info
2 from tools import get_max
3 import serial
4 import time
5 import pandas as pd
6
7 #Comunicación con el arduino
8 ser = serial.Serial('/dev/ttyACM0', 9600,
     ↪ timeout=1)
9 #Comienza la función de lectura
10 parcial = lectura_info(ser)
11 #Lee la tabla que está en el archivo data.txt
12 data = pd.read_table('data.txt', header=None,
     ↪ sep="\t", names=['time', "focus"])
13 #Obtiene el tiempo con el valor máximo de
     ↪ enfoque
14 time_max, focus_level=get_max(data)
15 #Le envía al arduino el tiempo que encontró
16 ser.write(time_max)
17 print("finished focusing at", time_max, "with
     ↪ focus level =", focus_level)
18 ser.close()

1 import pandas as pd
2 import cv2
3 import time
4 import math
5 import serial
6
7 #Función que evalúa el máximo enfoque, retorna
     ↪ su valor y el tiempo en el que obtuvo
     ↪ ese enfoque.
8 def get_max(data_frame):
9   index_max = data_frame['focus'].idxmax()
10  time_max=data_frame.loc[index_max, 'time']
11  focus_level=data_frame.loc[index_max, 'focus'
     ↪ ]
12  return(time_max, focus_level)
13
14 #Función que cuando el arduino arranca comienza
     ↪ a evaluar los bordes de cada imagen que
     ↪ se transmiten desde el telescopio.
15 def lectura_info(ser):
16
17   tiempo_lectura = 0
18   focusing = ser.read().decode('ascii')
19
20   #Inicialización y creación de archivo data.
     ↪ txt
21   raw_video = cv2.VideoCapture(1)
22   start = time.time()
23   previous = 0
24   f = open('data.txt', 'r+')
25   f.truncate(0)
26   f.close()
27   #display
28
29   while focusing != '0':
30     focusing = ser.read().decode('ascii')
31
32     if focusing == '1':
33       tiempo_lectura = 1
34       #Cuando el arduino arranca comienza
     ↪ a evaluar la imagen
35     if tiempo_lectura == 1:
36       partial = time.time() - start
37       current = int(math.floor(partial))
38       ret, frame = raw_video.read()
39       gray_video = cv2.cvtColor(frame, cv2.
     ↪ COLOR_BGR2GRAY)
40
41       #create border images with canny and
     ↪ save contrast values
42       canny = cv2.Canny(gray_video,0,100)
43       canny_var = cv2.Canny(gray_video
     ↪ ,0,100).var()
44
45       cv2.imshow("Canny", canny)
46
47       if (current % 1 == 0) & (current !=
     ↪ previous):
48         escritura = str(round(partial,3)
     ↪ ) + "\t" + str(round(canny_var,3)) + "\n"
49         print(escritura)
50         with open("data.txt", "a") as
     ↪ testfile1:
51           testfile1.write(escritura)
52           previous = current
53
```

Listing 2: Código de Python que implementa las funciones.

```

54         if cv2.waitKey(1) & 0xFF == ord('q'):
55             break
56 #La toma de datos termina cuando el telescopio
57 #ha recorrido todo su trayecto y toca el
58 #boton que cambia la direccion.
59 raw_video.release()
cv2.destroyAllWindows()
return(partial)

```

Listing 3: Código de Python en el que se encuentran las funciones usadas.

Dificultades

La primera dificultad encontrada fue el montaje mecánico ya que el motor tenia que tener la suficiente fuerza para mover el enfoque del telescopio. Después de varios intentos escogimos una ubicación del motor estratégica y tuvimos que ayudarlo con una cuerda para que no se despegara de las poleas que tenia, esto es un aspecto que queda por mejorar en el proyecto. Lo anterior hacia que, al tener un montaje inestable, la imagen se corriera un poco. También tuvimos dificultades con el código, pues fue difícil establecer la comunicación que queríamos entre Python y el Arduino, sin embargo esto resultó satisfactoriamente, llegando a el mejor código que pudimos emplear. Además, el tiempo empleado en el

proyecto fue mucho mas alto del que esperábamos, ya que esas dificultades requirieron de tiempo.

Conclusiones

Se logró crear el montaje propuesto, obteniendo imágenes con un buen nivel de enfoque de objetos lejanos empleando Arduino y Python. A pesar de las dificultades mecánicas el proceso de enfoque requiere intervención humana únicamente para asegurar buen contacto entre el motor y el pistón del telescopio. Se evidenció además la alta confiabilidad de la método Canny para determinar la calidad de la imagen, y lograr así el óptimo posicionamiento de las lentes.

Referencias

- [1] Engineers Garage. L293 and l293d internal structure. <https://www.engineersgarage.com/l293d-pin-description-and-working/>. Accessed: 2022-06-10.
- [2] Texas Instruments. L293d, quadruple half-h drivers. <https://www.ti.com/product/L293D>. Accessed: 2022-06-10.
- [3] ARDUINO. *30 Arduino projects book*. EN, 2015.