



Instituto Politécnico Nacional
Escuela Superior de Ingeniería
Mecánica y Eléctrica
Unidad Zacatenco



MICROPROCESADORES

ALUMNO: LARA SANCHEZ VICTOR EZEQUIEL

BOLETA: 2021300885

GRUPO: 6CM4

PROYECTO “PLUMA”

Introducción.

Hoy en día uno de los aspectos más importantes en cualquier industria es el controlar a un sistema mediante servidores web, unas de las ventajas que puede ofrecer ser el acceso remoto para mover un sistema son la flexibilidad de manejar dicho sistema, eficiencia del trabajo, facilidad de trabajo, la seguridad, la posibilidad de integración de nuevas tecnologías y en si el avance tecnológico entre otras.

En el siguiente documento se propone un proyecto sobre cómo manejar de manera remota un sistema mediante un servidor web usando un microprocesador y tres servomotores.

Objetivo. (General)

El objetivo del proyecto es aprender a usar herramientas como lo son los microcontroladores que son herramientas que nos pueden proporcionar control, procesamiento de datos y automatización en una amplia variedad de aplicaciones, desde dispositivos.

Objetivo. (Específico)

El objetivo específico es construir un sistema para controlarlo de forma remota a un sistema mediante una página web usando un microprocesador como lo es la Raspberry pi.

Enfoque Teórico

Entender cómo funciona la comunicación entre un cliente y un servidor puede ser esencial para el desarrollo de la práctica y/o proyecto propuesto. Para esto explicaremos brevemente el funcionamiento.

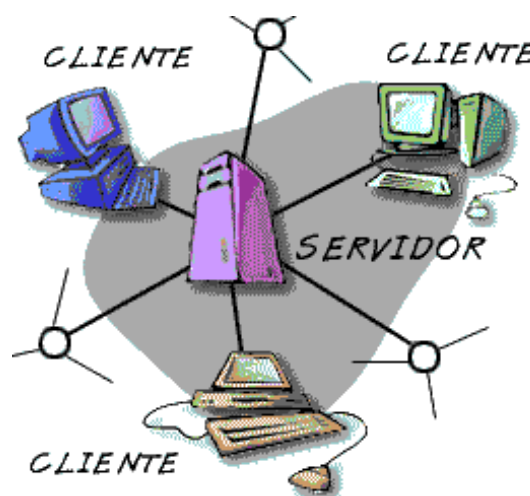
Comunicación Cliente-Servidor:

Cliente y Servidor:

En el contexto web, el "cliente" generalmente se refiere al navegador web que utiliza el usuario, mientras que el "servidor" es la computadora que aloja la aplicación o el sitio web.

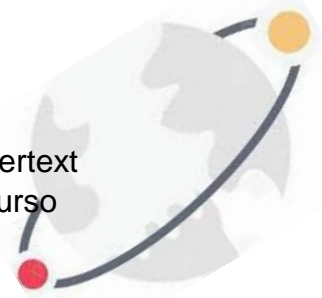
-Petición del Cliente:

Cuando un usuario ingresa una dirección web en el navegador y presiona "Enter", el navegador envía una solicitud al servidor para obtener la página web correspondiente.



HTTP Request (Solicitud HTTP):

La solicitud del cliente se realiza mediante el protocolo HTTP (Hypertext Transfer Protocol). Esta solicitud incluye información sobre el recurso solicitado, como la dirección URL y otros datos relevantes.



Procesamiento en el Servidor:

El servidor recibe la solicitud y procesa la información. Puede realizar tareas como acceder a bases de datos, ejecutar código y preparar la respuesta.

HTTP Response (Respuesta HTTP):

Una vez que el servidor ha procesado la solicitud, envía una respuesta de vuelta al cliente. Esta respuesta también utiliza el protocolo HTTP y contiene la información solicitada, que generalmente incluye código HTML, CSS, JavaScript y otros recursos.

Renderización en el Cliente:

El navegador del cliente recibe la respuesta y comienza a interpretarla. Renderiza la página web utilizando el código HTML y aplica estilos con CSS. Si hay código JavaScript, también se ejecuta para realizar acciones adicionales en el lado del cliente.

Comunicación Bidireccional (Opcional):

Sockets y WebSocket:

En algunas aplicaciones, la comunicación no se limita a solicitudes y respuestas únicas. Puede haber necesidad de una comunicación más continua y bidireccional. En estos casos, se utilizan tecnologías como WebSocket, que permite una conexión persistente y bidireccional entre el cliente y el servidor.



Envío de Datos desde el Cliente al Servidor:

En situaciones donde es necesario enviar datos desde el cliente al servidor (por ejemplo, formularios, actualizaciones en tiempo real), el cliente puede enviar una solicitud especial al servidor, generalmente mediante métodos como POST o PUT. Estos métodos permiten enviar datos junto con la solicitud.

Procesamiento y Respuesta del Servidor:

El servidor procesa los datos recibidos, realiza las acciones necesarias y puede enviar una respuesta de vuelta al cliente para confirmar la operación o proporcionar información adicional.

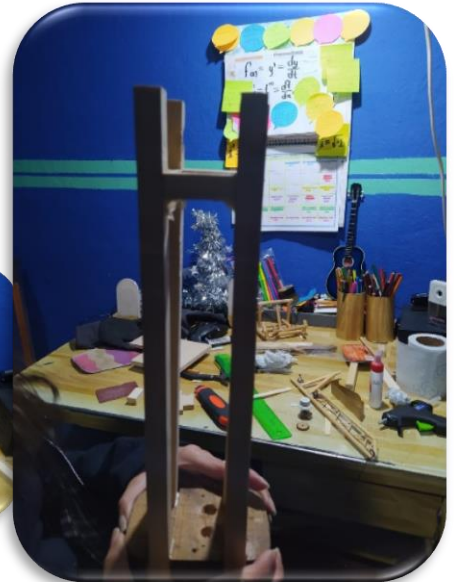


Desarrollo

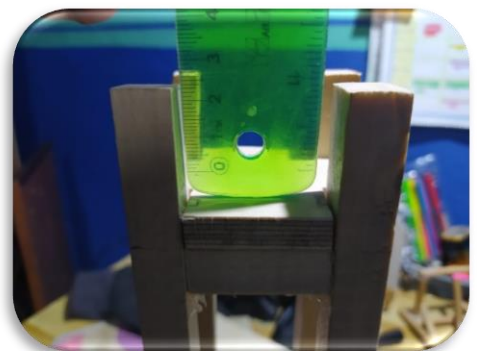
En este se pretende realizar una pluma a tamaño escala que funcione mediante un servidor web. Para esto se diseña con los siguientes materiales y equipo

- Una Raspberry pi 4
- 1 Servomotor MG996 180°
- 2 servomotores sg90 180°
- Un protoboard pequeño
- Jumperes de varios tamaños
- Un led
- Una fuente de 5 volts
- Palos de madera
- Pegamento de madera
- Barras de silicón
- Hilo cáñamo
- Alambre de calibre delgado

-Para la construcción de dicha estructura comenzaremos colocando 4 vigas de madera de unos 35 cm aproximadamente sobre una base de madera y uniéndolos con trozos de madera en la parte superior (a 30 cm de altura) como se muestra en la figura.

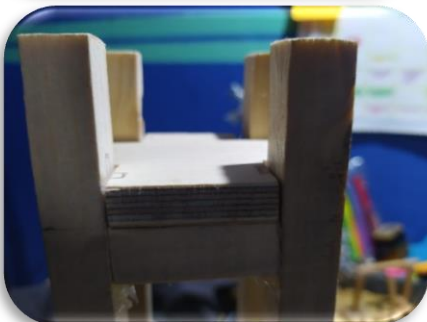


-Colocaremos un a base de madera pequeña donde pondremos nuestro servo de giro de base.



-Dejaremos el espacio suficiente para que el servomotor quede perfectamente fijado. Las medidas de la base inferior y superior son de 7x7 cm aproximadamente y una altura de 2,5 en la base superior.

-Una vez fijada la base superior fijaremos el servomotor mg996 (giro de base)





-Después cortaremos y pegaremos los palitos de paleta de manera cruzada como se muestra en la figura del lado derecho.

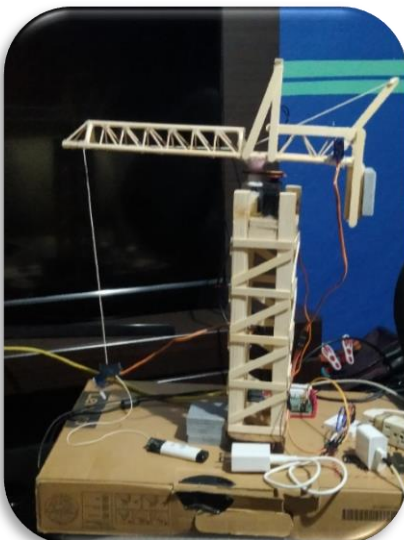


-Colocaremos el servomotor MG996 en la base superior con un engrane en su superficie.

Después colocaremos a su misma altura un eje con potenciómetro y le colocaremos un engrane a la altura del engrane del servomotor de giro de base.



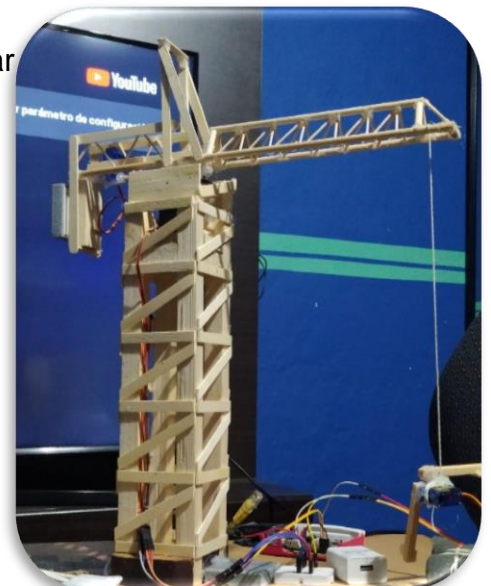
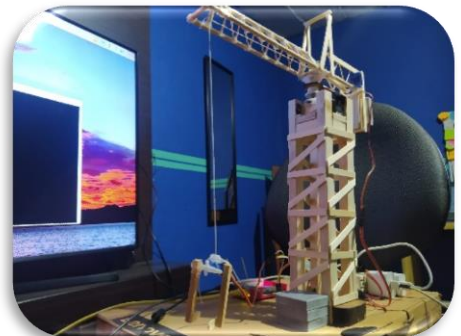
-Posteriormente se hace el armado del brazo de la pluma con ayuda de palillos para los dientes y colocaremos un servomotor sg90 en la parte trasera y le colocaremos un palito de paleta y en la punta le amarraremos el hilo para bajar y subir la pinza. Fijaremos un orificio de con alambre en la punta de la pluma para que pase por que el hilo que bajara la pinza.



-Uniremos las 2 partes nivelándolas desde el engranaje y colocaremos y tercer servomotor en la punta del hilo (sg90). Y le pegaremos dos palitos de paleta cuadrados para simular la pinza.

Nivelaremos la estructura colocándole un contrapeso en la parte trasera y en la base.

-Finalmente colocaremos los detalles estincos necesarios para tapar los servomotores y tendremos el producto final como se muestra en el lado derecho.



The image shows three overlapping terminal windows on a Raspberry Pi desktop. The desktop background is a sunset over water. The top window shows the root directory with a file listing. The middle window shows the user navigating into the 'nodetest' directory. The bottom window shows the user listing files in the 'nodetest' directory, including 'arbolito.js', 'grua.js', 'nuevointento.js', 'public', 'blinnk.js', 'led_rgb.js', 'package.json', 'server.js', 'grua_Final.js', 'node_modules', and 'package-lock.json'.

```
pi@raspberrypi:~$ ls
blink      Downloads  Music      segments
Bookshelf  hello2-main  nodetest   server
Desktop    int         Pictures   Templates
Documents  luma.examples  raspberry-pi-assembler  Videos
pi@raspberrypi:~$ cd nodetest/
pi@raspberrypi:~/nodetest$ ls
arbolito.js  grua.js      nuevointento.js  public
blinnk.js    led_rgb.js   package.json      server.js
grua_Final.js  node_modules package-lock.json
pi@raspberrypi:~/nodetest$
```

Primero
abriremos la
terminal

Para analizar el
código buscaremos
su ubicación y lo
abriremos desde
Visual Studio

SERVER

En esta sección se
declaran las
bibliotecas a utilizar
previamente ya
instaladas.

Crearemos 3 variables una
para cada servo y
asignaremos un numero de pin
(Gpio.) en la Raspberry.

Le asignaremos una
posición inicial a cada
servo con la función
servoWrite

```

30
31
32
33
34
35 http.listen(8080); //listen to port 8080
36
37
38 function handler (req, res) { //what to do on requests to port 8080
39   fs.readFile(__dirname + '/public/indexgrua.html', function(err, data) {
40     if (err) {
41       res.writeHead(404, {'Content-Type': 'text/html'}); //displ
42       return res.end("404 Not Found");
43     }
44     res.writeHead(200, {'Content-Type': 'text/html'}); //write HTML
45     res.write(data); //write data from rgb.html
46     return res.end();
47   });

```

En la siguiente parte se configura todo sobre la conexión de la pag. web

```

grua.js - nodetest - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  > NODETEST
    > node_modules
    > public
      > arbolito.html
      > index.html
      > index2.html
      > indexgrua.html
      > led_rgb.html
    JS arbolito.js
    JS blink.js
    JS Grua_Final.js
    JS grua.js
    JS led_rgb.js
    JS nuevointento.js
    {} package-lock.json
    {} package.json
    JS server.js
  > OUTLINE
    > node_modules
    > public
      > arbolito.html
      > index.html
      > index2.html
      > indexgrua.html
      > led_rgb.html
    JS arbolito.js
    JS blink.js
    JS Grua_Final.js
    JS grua.js
    JS led_rgb.js
    JS nuevointento.js
    {} package-lock.json
    {} package.json
    JS server.js
  > grua.js
    > io.sockets.on('connection') callback > io.sockets.on('connection') callback > socket.on('state1')
    56
    57
    58 io.sockets.on('connection', function (socket) {
    59   io.sockets.on('connection', function (socket) {
    60     var buttonState = 0; //variable to store button state
    61
    62     socket.on('state1', function (data) {
    63       buttonState = data;
    64       console.log(data); //output data from Websocket connection
    65
    66       if ((data == 2500) && (a==750))
    67       {
    68         function delay(ms) {
    69           return new Promise(resolve => setTimeout(resolve, ms));
    70
    71         async function loopWithDelay1() {
    72           for (let i = 750; i < data+1; i++) {
    73             console.log('iteracion',i);
    74             sv1.servoWrite(i);
    75             // Introduce un retraso de 1 segundo (1000 milisegundos)
    76             await delay(2);
    77             a=i; //es el valor donde se quedo el servo
    78             console.log('valor de ',a);
    79           }
    80         }
    81         loopWithDelay1();
    82       }
    83     }
    84   }
    85 }

```

En esta sección tenemos el cuerpo de mi programa principal es donde mi servidor recibe el dato mediante los sockets.

Por medio de los sockets la variable estado mandará un dato a la variable data y a su vez imprimirá el dato en la terminal

Posteriormente dependiendo del dato recibido entrara o no al cuerpo del "if".

Si el valor recibido es 2500, y si la posición del servo 1 es 750 (para bajar y subir la pinza) entrará al ciclo for y recorrerá de 750 a 2500 con un delay de 2 microsegundos y al mismo tiempo se escribirá el valor en el servomotor para que este se mueva paulatinamente.

```

indexgrua.html JS grua.js
JS grua.js > io.sockets.on('connection') callback > io.sockets.on('connection') callback > socket.on('state1')
85
86
87
88 if ((data == 750) && (a==2500))
89 {
90   function delay(ms) {
91     return new Promise(resolve => setTimeout(resolve, ms));
92
93   async function loopWithDelay1() {
94     for (let i = 2500; i > data; i--) {
95       console.log('iteracion',i);
96       sv1.servoWrite(i);
97       // Introduce un retraso de 1 segundo (1000 milisegundos)
98       await delay(2);
99       a=i; //es el valor donde se quedo el servo
100       console.log('valor de ',a);
101     }
102   }
103   loopWithDelay1();
104 }
105
106
107
108

```

Algo similar ocurrirá con el mismo servo, pero esta vez en lugar de incrementar el valor para que este llegue de 750 a 2500 será inversamente de 2500 a 750 para que la pinza baje, cuando reciba el dato de 750 los sockets. Y así podremos bajar y subir la pinza

Lo mismo realizaremos con el servo 2 solo que en este caso en lugar de subir y bajar la cuerda de la pinza lo usaremos para mover el servomotor horizontalmente ala derecha e izquierda

```
indexgrua.html JS grua.js
JS grua.js > io.sockets.on('connection') callback > io.sockets.on('connection') callback > socket.on('state2') ca
108
109
110
111
112 socket.on('state2', function (data) {
113     buttonState = data;
114     console.log(data); //output data from WebSocket connection to co
115
116     if ((data == 2450) && (b==550))
117     {
118         function delay(ms) {
119             return new Promise(resolve => setTimeo
120
121
122         async function loopWithDelay1() {
123             for (let i = 550; i < data+1; i++) {
124                 console.log('iteracion',i);
125                 sv2.servoWrite(i);
126                 // Introduce un retraso de 1 segundo
127                 await delay(2);
128                 b=i;//es el valor donde se quedo el s
129                 console.log('valor de ',b);
130             }
131         }
132         loopWithDelay1();
133     }
134 }
135
```

```
134
135 }
136
137
138
139 if ((data == 550) && (b==2450))
140 {
141     function delay(ms) {
142         return new Promise(resolve => setTimeout(resolve, ms));
143     }
144
145     async function loopWithDelay1() {
146         for (let i = 2450; i+1 > data; i--) {
147             console.log('iteracion',i);
148             sv2.servoWrite(i);
149             // Introduce un retraso de 1 segundo (1000 milisegundos)
150             await delay(2);
151             b=i;//es el valor donde se quedo el servo
152             console.log('valor de ',a);
153         }
154     }
155     loopWithDelay1();
156 }
157
158
159
160
```

```
160
161
162 socket.on('state3', function (data) {
163     buttonState = data;
164     console.log(data); //output data from WebSocket connection to co
165     sv3.servoWrite(data);
166 });
167
168
169
170
171 });
172
173
174 process.on('SIGINT', function () {
175     sv1.servoWrite(750);
176     sv2.servoWrite(550);
177     sv3.servoWrite(1500);
178     process.exit();
179
180
181 });
182 //
183
```

Finalmente, con el tercer servo (pinza) ya no usaremos el ciclo "for" para mover paulatinamente la pinza ya que esta tiene que apretar al objeto a mover, y usaremos solamente el valor recibido para escribir el valor en el servo.

Cuando el proceso se está iniciando el valor de los servos se escribirán en el valor de la imagen

Archivo HTML

La forma en la que mandamos valores será con el estilo de botones dando un click para mandar el valor al servo

indexgrua.html x JS grua.js

```
public > indexgrua.html > html > body > script
1 <!DOCTYPE html>
2 <html>
3 <body style="background-color: rgb(72, 13, 167);">
4 <title>Proyecto de grua</title>
5 <h1>Control de Grua </h1>
6
7
8 <p>Giro de Base</p>
9 <button type="button" id="state" onclick="sv20()" style="background-color: #18e4ff;">Izquierda max</button>
10 <button type="button" id="state" onclick="sv24()" style="background-color: #18e4ff;">Derecha max</button>
11
12 <p>Bajar y subir Carga </p>
13 <button type="button" id="state" onclick="sv12()" style="background-color: rgb(181, 249, 119);">Bajar Carga</button>
14 <button type="button" id="state" onclick="sv10()" style="background-color: rgb(181, 249, 119);">Levantar Carga</button>
15 <script src="/socket.io/socket.io.js"></script>
16
17 <p>Pinza </p>
18 <button type="button" id="state" onclick="sv33()" style="background-color: rgb(235, 244, 146);">Abrir Pinza</button>
19 <button type="button" id="state" onclick="sv32()" style="background-color: rgb(235, 244, 146);">Cerrar Pinza</button>
20 <button type="button" id="state" onclick="sv31()" style="background-color: rgb(235, 244, 146);">Cerrar Pinza</button>
21 <button type="button" id="state" onclick="sv30()" style="background-color: rgb(235, 244, 146);">Cerrar Pinza max</button>
22
23
24
25
```

La comunicación con los botones y vales será por medio de las variables svXX y la variable state.

indexgrua.html x JS grua.js

public > indexgrua.html > html > body > script

```
25
26 <script>
27   var socket = io.connect();
28
29   //servo 3
30   function sv33 (){
31
32     socket.emit("state3", 2500); //send button state to server
33   }
34   function sv32 (){
35
36     socket.emit("state3", 2000); //send butt
37   }
38   function sv31 (){
39
40     socket.emit("state3", 1750); //send butt
41   }
42   function sv30 (){
43
44     socket.emit("state3", 1500); //send butt
45   }
46
47
48
49
```

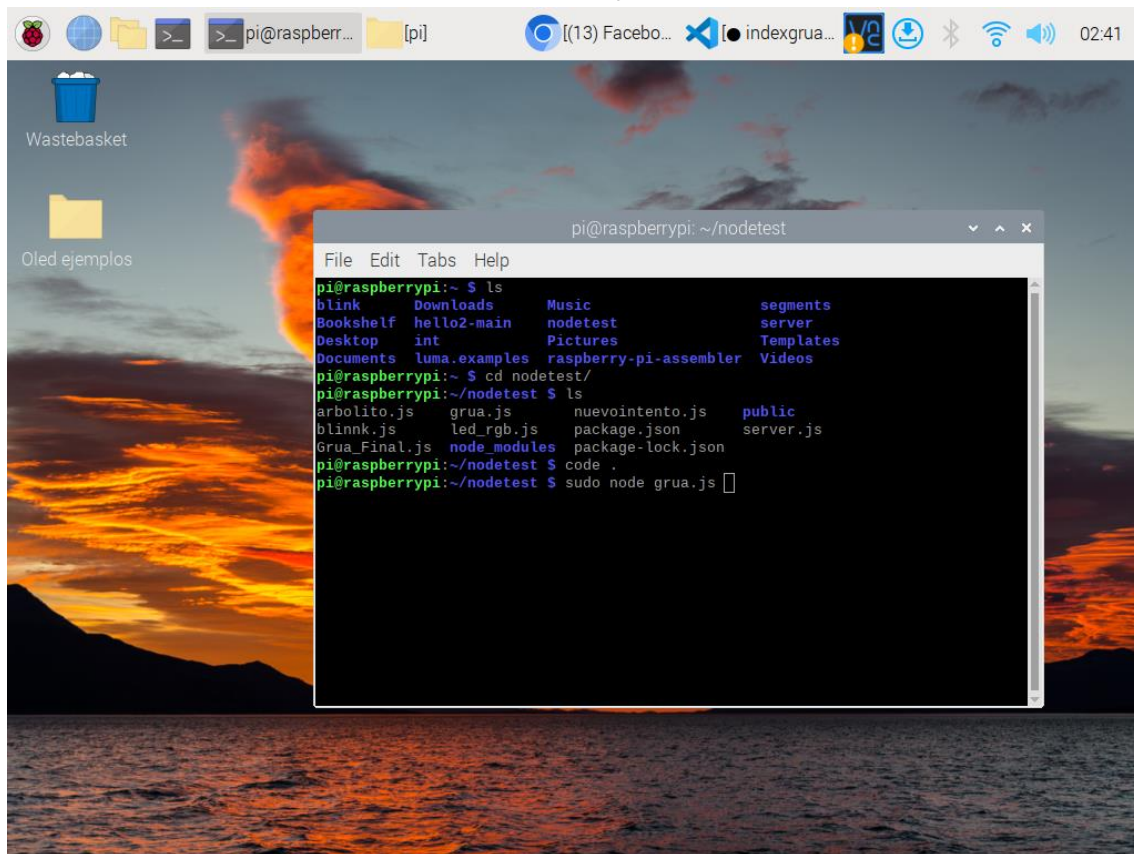
indexgrua.html JS grua.js

public > indexgrua.html > html > body > script

```
40
49
50 //servo 2
51 function sv20 (){
52   socket.emit("state2", 550); //send button state to server
53 }
54
55
56
57 function sv24 (){
58
59   socket.emit("state2", 2450); //send button state to server
60 }
61
62
63 //servo 1
64 function sv10 (){
65   socket.emit("state1", 2500); //send button state to server
66 }
67
68
69
70 function sv12 (){
71   socket.emit("state1", 750); //send button state to server
72 }
73
74 </script>
75
76 </body>
77 </html>
```

Dependiendo del valor presionado en la página mandara determinado valor al servo mediante la variable sv XX.

Finalmente, ejecutamos el comando sudo node y el nombre del archivo con la extensión js

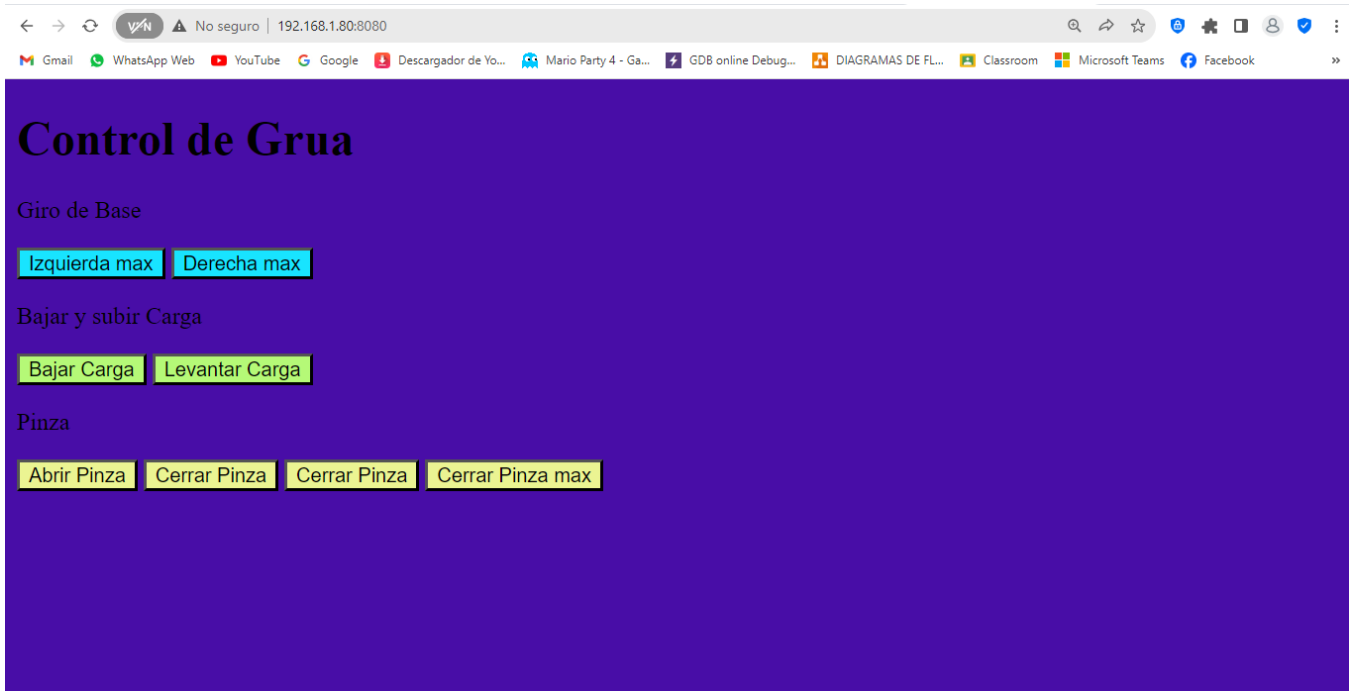


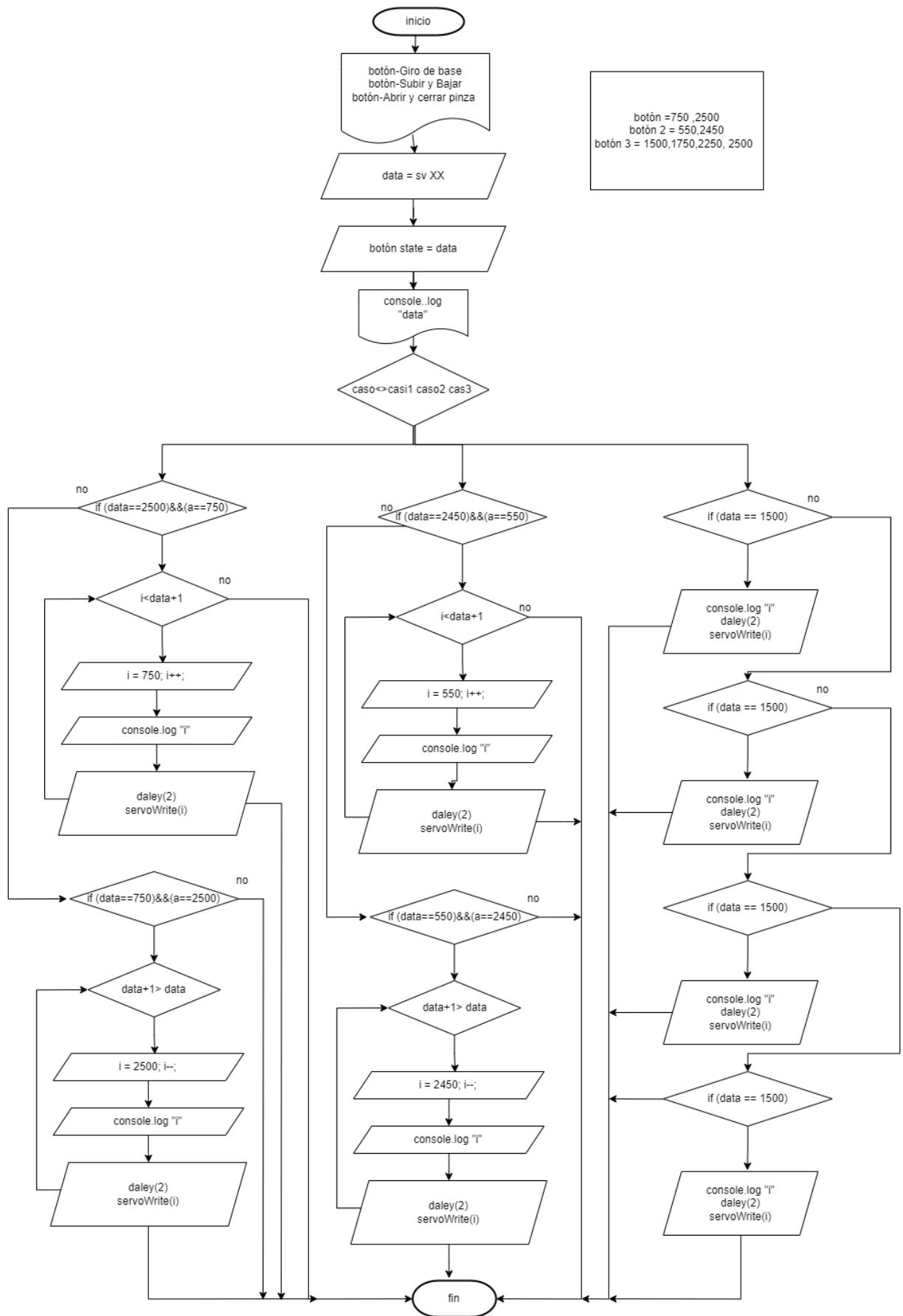
The screenshot shows a Raspberry Pi desktop with a sunset background. A terminal window titled 'pi@raspberrypi: ~/nodetest' is open, displaying the following commands and output:

```
pi@raspberrypi:~$ ls
blink      Downloads  Music      segments
Bookshelf  hello2-main  nodetest   server
Desktop    int        Pictures   Templates
Documents  luma.examples  raspberry-pi-assembler  Videos

pi@raspberrypi:~$ cd nodetest/
pi@raspberrypi:~/nodetest$ ls
arbolito.js  grua.js  nuevointento.js  public
blinnk.js    led_rgb.js  package.json      server.js
Grua_Final.js  node_modules  package-lock.json
pi@raspberrypi:~/nodetest$ code .
pi@raspberrypi:~/nodetest$ sudo node grua.js
```

Y en nuestro navegador colocaremos la dirección "ip" de conexión de la Raspberry con la terminación :8080 y nos mostrara la siguiente página con el control de la pluma.





Conclusión.

Como se puede observar la importancia de controlar servomotores con una Raspberry a través de un servidor ha demostrado ser una solución efectiva para la manipulación remota de dispositivos. Con la práctica y/o proyecto se puede corroborar que la manipulación remota ofrece a un sistema la flexibilidad de manejar dicho sistema, eficiencia del trabajo, facilidad de trabajo, la posibilidad de integración de nuevas tecnologías y en si el avance tecnológico entre otras.