6. GUÍA PRÁCTICA DE LABORATORIO

6.1 Introducción.

La principal finalidad de este trabajo es el desarrollo de un entorno Ciber-físico que permita la realización de prácticas de laboratorio para alumnos del Máster Universitario en Ingeniería de Sistemas Automáticos y Electrónica Industrial de la EPSEVG de la Universidad Politécnica de Cataluña.

Para esto se contará con un prototipo de fines académicos, en el cual se podrá lograr interpretar un sistema físico y un sistema enfocado a un Servidor-Cliente, estos estarán conformados por aplicaciones de importancia para el desarrollo del entorno ciber-físico. Aplicaciones que están destinadas tanto para realizar eventos o alguna manipulación en el servidor Web o desde el sistema Físico.

Por lo tanto, al contar con una página diseñada para la aplicación se debe diseñar e implementar dentro de una ventana, específicamente en la ventana Control y Monitorización de la Página Web, conformado por comandos de controles de forma manual de los distintos actuadores del sistema con la ayuda de los entornos de programación mencionados en este trabajo, con el objetivo de obtener información del proceso del sistema físico en tiempo real.

Se debe tomar en cuenta para el diseño de la aplicación, la creación de estructuras de objetos en función a las librerías de socket.io y ArduinoJson, las mismas que ayudarán a comunicarse desde la plataforma del Servidor Web, la plataforma Raspberry y la plataforma Arduino de forma simultánea, logrando obtener como punto de partida una aplicación del internet de las cosas (IoT).

Como este trabajo está enfocado a un sistema Ciber-físico, se deberá diseñar un algoritmo de visión artificial, y aplicado al control de calidad. Dando un paso hacia adelante de las IoT e integrando sistemas computacionales.

6.2 Objetivos:

A partir de un sistema monitorizado y controlado automático se debe:

- Controlar vía remota actuadores del sistema de forma manual, de esta manera el usuario adquirirá controlar individualmente a los distintos actuadores del prototipo desde algún sitio de intranet o internet.
- Implementar comandos individuales para el control de los actuadores.
- Crear y enviar de cadena de objetos en base a ArduinoJson.
- Interpretar las aplicaciones del Servidor (Raspberry pi) y del sistema físico (Arduino)
- Leer y enviar cadena de objetos en la plataforma Arduino.

6.3 Estación de un CPS y aplicativo web.

6.3.1 Prototipo académico SECVIA.

El prototipo SECVIA presentado en la Figura 37, es un sistema académico de aprendizaje y que está orientado al servicio de control de calidad de engranajes mediante visión. Este trabajo consiste en dos partes importantes para el aprendizaje de un CPS; la manipulación de un sistema físico a escala y la implementación de modificaciones al software y hardware para el control y procesamiento de engranes rectos.

Uno de los objetivos es comprender el comportamiento de un CPS, cuando el usuario interactúa con el sistema físico desde cualquier punto de la red. El trabajo se centrará dentro de una red de una organización (intranet). Para el funcionamiento del prototipo se debe considerar las siguientes características técnicas presentadas en la Tabla 5.

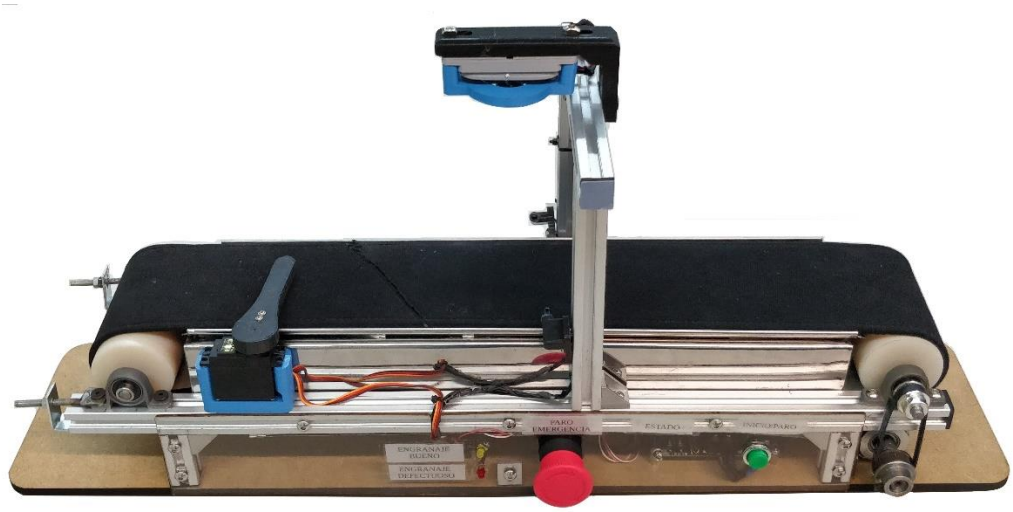

Figura 37. Estación de clasificación SECVIA

Ítem	Sistema	Alimentación de entrada
1	Control del prototipo SECVIA	6V - 12V DC, min. corriente de 5 Amperios
2	Microprocesador Rasperry pi	5V DC, min corriente de 2,5 Amperios

Tabla 5. Características de Funcionamiento del prototipo SECVIA

Las fuentes de alimentación de cada uno de los sistemas deben estar alimentadas individualmente, para evitar que el sistema de control generé variaciones de corriente y voltaje que puede afectar al microprocesador.

6.3.2 Entorno cliente-servidor.

Para el desarrollo de un cliente servidor se utiliza un microprocesador. En este proyecto se utilizará un Raspberry pi, quien va a ser quien corra el servidor web, además de interactuar con los diferentes dispositivos, debido a que contará con los diferentes softwares y programas instalados para el desarrollo de un sistema CPS.

Para el desarrollo de este prototipo SECVIA se debe contar con conocimientos básicos de entornos de programación HTML, JavaScript, C++, y C para Arduino. Establecido estos requerimientos básicos, la necesidad de diferenciar directorios que se enfocan al servidor y a la conexión del cliente, a continuación, se detalla de manera rápida como el usuario ingresará al ordenador del Raspberry pi desde la consola de otro ordenador.

1. Conocer la dirección ip del ordenador del usuario, si el usuario desconoce su dirección ip, en la consola del ordenador del usuario se aplica los siguientes comandos, recomendable el uso de un ordenador con Sistema operativo en Linux, ya que el sistema operativo del Raspberry pi está basado en el kernel de Linux.

```
ifconfig
```

En la Figura 38 se muestra el resultado de la dirección ip del usuario que está utilizando en la red.

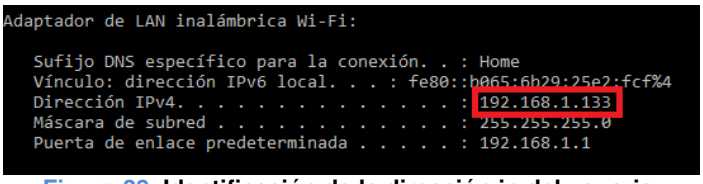

Figura 38. Identificación de la dirección ip del usuario.

2. El siguiente paso es realizar un mapeo de las direcciones IP de los diferentes dispositivos que están conectados en la red mediante el comando.

```
nmap -s 192.168.1.133/24
```

3. Identificado la dirección ip del microprocesador Raspberry pi, al aplicar el primer comando de la Tabla 6, se puede ingresar al sistema operativo del Raspberry pi. Mediante la consola o bien desde un explorador de archivos.

- Utilización de la consola. Al utilizar este método, el estudiante debe estar consciente que va a trabajar en adelante desde el terminal. Para eso debe conocer las diferentes funciones de uso en una consola. En la Tabla 6 se muestra las funciones básicas que se utilizará para el desarrollo del proyecto.

Comandos	Función
`sudo -sh pi@192.168.1.133`	Ingresa al sistema operativo del Raspberry pi luego de solicitar la contraseña correspondiente para el ingreso. Clave: raspberry
`cd Documentos`	Ingresa al directorio Documentos
`cd Documentos/UPC_TFM_Fernando_Jacome/ServidorWeb`	Ingresa al directorio del proyecto.
`ls`	Despliega todos los archivos del directorio.
`nano controlymoni.html`	Ingresa al documento mencionado, lugar donde se encuentra los diferentes entornos de programación que podrán ser modificados.

Tabla 6. Comandos útiles para consola de Raspberry pi

- Utilización de un explorador de archivos, mediante la utilización de la siguiente función fish://192.168.x.xx, que facilitará el ingreso y una fácil visualización del contenido de los diferentes ficheros que se encuentra en el sistema operativo Raspberry pi.

Hay que tomar en cuenta, que para acceder al sistema operativo y consiguiente a sus directorios del Raspberry pi, el dispositivo solicitará su nombre y su contraseña. Estos están definidos por defecto de fábrica User: pi; Password: raspberry.
En la Figura 39 se muestra los diferentes ficheros después de ingresar al ordenador de Raspberry pi. Para el desarrollo de las prácticas se utilizará el directorio UPC_TFM_Fernando_Jacome, que está contenida en Documentos del Raspberry pi.

Figura 39. **Escritorio de Raspberry pi.**

4. En directorio mencionado de la aplicación se despliega los directorios tanto para el servidor y el cliente. En la Figura 40 se muestra los diferentes ficheros y directorios de la aplicación. Los ficheros de la carpeta de ClienteSocket se encuentra los diferentes algoritmos para visión y la comunicación con el sistema físico. En la carpeta ServidorWeb, contiene los diferentes ficheros que permitirán enviar los diferentes eventos al sistema físico y bien visualizar dentro de un subsitio web estos cambios de los eventos.
   Y, por último, la carpeta TFM_ARDUINO contiene el algoritmo del sistema físico que interactuará con los diferentes sensores y actuadores
   De aquí en adelante nos enfocaremos en la carpeta del ServidorWeb para realizar las diferentes prácticas.

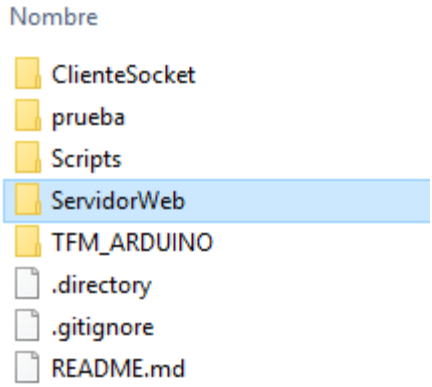

Figura 40. **Ficheros del proyecto SECVIA**

5. Al desplegar el directorio mencionado en el anterior apartado, el directorio cuenta con ficheros de las herramientas y librerías para que el servidor funcione correctamente, los estilos de la aplicación que se encuentran dentro del directorio de la carpeta **public**. En la Figura 41 se muestra los diferentes ficheros para crear el estilo, estilos dinámicos y contenidos del sitio web.

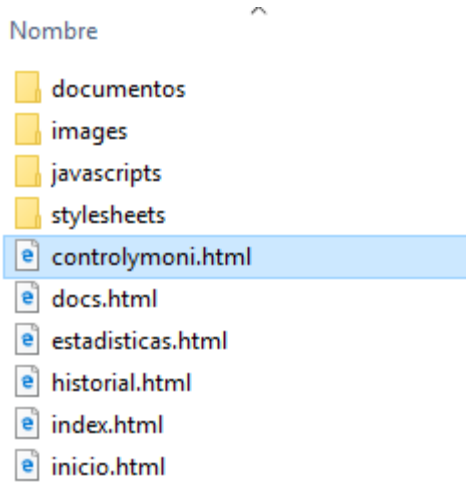

Figura 41. **Ficheros de estilos y contenidos del sitio web de la aplicación.**

Establecido e identificado los diferentes ficheros de la aplicación, ahora bien, queda en enfocarse en diseñar e insertar el contenido y comandos deseados por usuario. Esta guía práctica contará con dos laboratorios para lograr comprender el funcionamiento de un CPS.

6.4 Guía de laboratorio #1. Controlar un sistema de forma Manual.

6.4.1 Implementación de comandos en la página web.

A. Creación de comandos en el entorno de programación HTML.

El diseño actual del subsitio web control y monitorización cuenta con comandos para trabajar de forma automática tal como se muestra en la Figura 42, sin interferir en el estado y posición que se encuentre cada actuador, como objetivo principal dentro de esta parte de la guía práctica, es aprender a integrar comandos individuales de cada actuador asociado al sistema.

Figura 42. **Diseño de comandos para un sistema automático**

Ahora bien, para crear una tabla dentro de un fichero HTML, en primer lugar, hay que Interpretar el número de columnas y filas que cuenta la tabla insertada del sistema automático, para este caso 3 filas por 7 columnas. Tomando en cuenta que la estructura para crear tablas en documentos con extensión html, a continuación, se observa la sintaxis de este entorno de programación:

Columna 1	Column 2	Column 3
A	B	C
D	E	F

Tabla 7. Estructura de una Tabla en un Sitio Web.

```
<TABLE BORDER>
        <TR>
                <TD>Column 1</TD> <TD>Column 2</TD> <TD> Column 3</TD>
        </TR>
        <TR>
                <TD>A</TD> <TD>B</TD> <TD>C</TD>
        </TR>
        <TR>
                <TD>D</TD> <TD>E</TD> <TD>F</TD>
        </TR>
</TABLE>
```

Se puede observar que para construir en HTML una fila se utiliza la terminología `<TR></TR>`, consiguientemente dentro de la fila se inserta las columnas usado la terminología `<TD></TD>`. Cabe notar que dentro de cada terminología de columna se utilizará el campo para crear botones o contenido texto.
Para este ejercicio se creará una columna en cada fila y los diferentes comandos a utilizar son:

Tipo	**Nombre del Botón**	**Identificador del botón (id)**	**Identificador del texto (id)**
Button	SISTEMA MANUAL	id="manual"	
Button	MOTOR ADELANTE	id="motoradelante"	Id="motoradelantelabel
Button	MOTOR ATRÁS	id="motoratras"	id="motoratraslabel"
Button	CAPTURAR Y PROCESAR ENGRANE	id="capturarengrane"	id="textcapturarengrane"
Button	CLASIFICADOR BUENO	id="clasificacionbueno"	id="engranebueno"
Button	CLASIFICADOR MALO	id="clasificacionmalo"	id="engranemalo"

Tabla 8. Comandos e identificadores para un sistema manual

Los nombres e identificadores se pueden denominar de la manera como el usuario lo desee, para este laboratorio se recomienda utilizar dicha nomenclatura presentado en la Tabla 8, con el fin que no exista confusión con los distintos entornos de programación.

Para la insertar los comandos tipo Button en la tabla construida en el fichero html, estos comandos deberán estar ubicados en la columna número tres, desplazando la columna de OBSERVACIÓN a la columna número cuatro. En la columna cuatro se implementará el espacio para los diferentes textos informativos y para esto se utiliza los identificadores de texto. Dicho espacio contendrá textos informativos tanto para el sistema automático como para el sistema manual.

Una vez insertado los comandos mencionados se podrá obtener una interfaz de control y monitorización dentro del subsitio web como se muestra a continuación en la Figura 43**,** **c**oncluyendo así con el entorno de programación en HTML.

Figura 43. **Diseño del control de un sistema de forma automática y manual**

B. Crear entorno dinámico en JavaScript.

Para crear un sitio web dinámico a los diferentes comandos y contenidos del documento HTML. Se debe recordar los distintos identificadores mencionados en el entorno de programación HTML con el fin de ligarlos con el entorno JavaScript. Estos identificadores llamarán al comando ligado y actuarán dinámicamente en el sitio web y a la vez se creará objetos para enviará mediante un buffer al sistema físico.

Esto se logrará con dos ficheros en el entorno JavaScript **app.js** y **main.js.** Ficheros correspondientes al servidor y al cliente respectivamente.

- **Crear objetos en el servidor bajo la librería socket.io.**

Dentro del fichero **app.js** el primer paso es crear variables para leer y guardar su valor correspondiente. Variables que pertenecerán al servidor de la librería socket.io, considerando el valor a tomar cada variable (int, bool), de la forma presentada:

```
var idservidor_1=0;
var idservidor_2=false;
```

EL fichero cuenta con un arreglo de objetos bajo las librerías Json, este arreglo es de tipo cadena (String) denominada **dispositivo,** esta variable es la encargada de guardar la cadena de objetos que se crea automáticamente después de realizar algún evento dentro la sitio web hacia el cliente y consecuente enviar al sistema físico. En la variable deberá contar con los diferentes objetos incluido su valor, valor que puede tomar como int, bool, string, array.

```
var dispositivo = {objeto_1: 1, objeto_2: false };
```

Para desarrollar la práctica como recomendación se debe utilizar los siguientes objetos, y que serán interpretados en la plataforma Arduino, tal como se presenta en la Tabla 9.

OBJETO	TIPO
dispositivo:	int
modofunc	bool
motor	int
presencia:	bool
estado:	int
imagenprocesada	bool
clasificador	bool
sentidomotor	int
direcclasificacion	int
capturarengrane	bool

Tabla 9. Creación de objetos

Teniendo en cuenta estas variables, mediante de la función **io.on** permitirá establecer la conexión y enviar las variables del servidor que son capturadas desde el arreglo Json con su objeto, como se indica en la sintaxis.

```
io.on('connection', function (socket) {
   if (dispositivo.objeto_1 != null) {
      dispositivo.objeto_1 = idservidor_1
                          }
  if (dispositivo.objeto_2 != null) {
      dispositivo.objeto_2 = idservidor_2;
                          }
```

Para el envío de estos objetos en la función **io.on**, luego de haber guardado las variables mediante el método **push(),** que permite agregar uno o más elementos a un vector (`arreglodispositivo`), y al final emitiendo todo este vector a la conexión del servidor.

```
arreglodispositivo.push(dispositivo);
                dispositivo = {};
socket.emit("nombre_funcion", arreglodispositivo);
                arreglodispositivo = [];
```

Para obtener lectura de algún evento del cliente desde el sistema físico, el procedimiento es leer esa cadena de caracteres que se obtiene a partir de la generación de la estructura por ArduinoJson, la herramienta **socket.on** contendrá una función de datos, estos datos deben ser asociados con los diferentes objetos correspondientes y enviados a todos los clientes con ayuda de las herramientas de socket.on

```
socket.on("datos", function (data) {
       var sizedatos = data.length;
        for (i = 0; i < sizedatos; i++) {
           if (idservidor_1 != null) {
         idservidor_1 = data[i].objeto_1;
                          }
           if (idservidor_2 != null) {
         idservidor_2 = data[i].objeto_1;
                          }
                  }
   socket.broadcast.emit('datos', data);
   });
```

- **Añadir eventos dinámicos de la ventana de la página Web**

Ahora bien, al sitio web se realizará un entorno dinámico para capturar y enviar las diversas respuestas enviadas desde el servidor. Esto se trabajará en el fichero **main.js,** donde se debe crear nuevas variables con el objetivo de obtener y guardar la información del id de los diferentes comandos y textos del fichero .html, mediante la siguiente sintaxis al utilizar en JavaScript es:

```
var id_1 = document.getElementById(id1);
var id_2 = document.getElementById(id2);
```

A la vez se debe considerar una variable tipo vector (Array), variable que guardará los diferentes objetos, y luego enviar a todos los elementos con su respectivo valor al servidor de este proyecto.

```
var arreglodispositivo = [];
```

Para leer los eventos de cada elemento se utiliza la herramienta `elemento.addEventListener()`, herramienta que ayudan a obtener los eventos desde el sitio web. En la Tabla 10, se presenta las diferentes actividades que se puede realizar después de la lectura de un evento.

Herramientas	**Tipo**	**Acción.**
dispositivo.objeto_1	int, bool,array	Valor correspondiente para enviar hacia sistema físico
id_2.checked	bool	Habilitación / deshabilitación permanente un elemento
id_3.disabled	bool	Habilitación / deshabilitación elemento
id_4.style.backgroundColor	Formato .hex	Color de Fondo de los elementos
id_5.style.display	"block" or "none"	Muestra texto de los elementos

Tabla 10. Opciones de la Herramienta addEventListener

El objeto_1 llama al mismo objeto y su valor correspondiente que está contenida en la variable **dispositivo** (Arreglos Json) del fichero app.js, el tipo de valor que puede contener es int, bool, array, etc.

Establecidos los puntos importantes del evento de un elemento la sintaxis a utilizar luego que dicho elemento ha realizado un pulso (“on click”) en el botón de la página web, y al final enviar mediante un vector los objetos hacia el sistema físico.

```
id_1.addEventListener("click", function () {
                var dispositivo = {};
            dispositivo.dispositivo = 1;
            dispositivo.objeto_1 = true;
              dispositivo.objeto_2 = 1;
                 id_1.disabled = true;
            id_2.style.display = "none";
        id_3.style.backgroundColor = "#f4f4f4";
    arreglodispositivo.push(dispositivo);
      socket.emit('datos', arreglodispositivo);
              arreglodispositivo = [];
```

```
                    }

id_2.addEventListener("click", function () {
                var dispositivo = {};
                    ...
                    ...
                    ...
    arreglodispositivo.push(dispositivo);
      socket.emit('datos', arreglodispositivo);
              arreglodispositivo = [];
                        }
```

De la misma manera que el fichero **app.js** si algún evento enviado desde el sistema físico, el aplicativo se pone en funcionamiento y se debe interpretar mediante la actuación del evento en la página web, la herramienta **socket.on** contendrá una función de datos, datos que deben ser asociados con los diferentes objetos correspondientes y compartidos a todos los clientes con la función `socket.broadcast.emit`.

```
socket.on("datos", function (data) {
       var sizedatos = data.length;
        for (i = 0; i < sizedatos; i++) {
         if (data[i].objeto_1 != null) {
               id_1.disabled = true;
       id_2.style.display = "none";
 id_3.style.backgroundColor = "#f4f4f4";
                          }
         if (data[i].objeto_2  != null) {
                ...
                ...
                        }
          }
  socket.broadcast.emit('datos', data);
  });
```

6.4.2 Implementación en sistema físico (ARDUINO)

A. Interpretación de los objetos (ArduinoJson)

Para interpretar en la plataforma Arduino los diferentes cambios que se realizó dentro del entorno del servidor y el sitio web, nos basaremos en un asistente de ArduinoJson, que permite la interpretación de los objetos y su valor que llega en la cadena de caracteres por el puerto serial del Arduino comunicación denominada UART, cadena que es enviada desde el microprocesador Raspberry pi.
Para esto se debe recordar los nombres de los objetos y el tipo de valor que se indicó en la Tabla 9. En el sitio web https://arduinojson.org/, cuenta con una ventana de asistencia para crear el buffer de salida y entrada.
En la sección INPUT de este asistente se debe ingresar los elementos según la Sintaxis basado en Json:

```
{
"objeto_1": "gps",
"objeto_2": 1,
"objeto_3": false
}
```

El asistente refleja resultados del tamaño del Buffer (JsonBuffer size), **JSON_OBJECT_SIZE(3);** indicando el tamaño del buffer puede entrar o salir 3 objetos:

En este proyecto se debe ingresar los datos correspondientes al objeto y su tipo de valor dentro del arreglo de Json. Tal como se muestra en la siguiente sintaxis de la aplicación.

```
{
"dispositivo":1,
"modofunc":true,
"motor":255,
"presencia":true,
"estado":0,
"imagenprocesada":true,
"clasificador":true,
"sentidomotor":1,
"direcclasificacion":1,
"capturarengrane":false
}
```

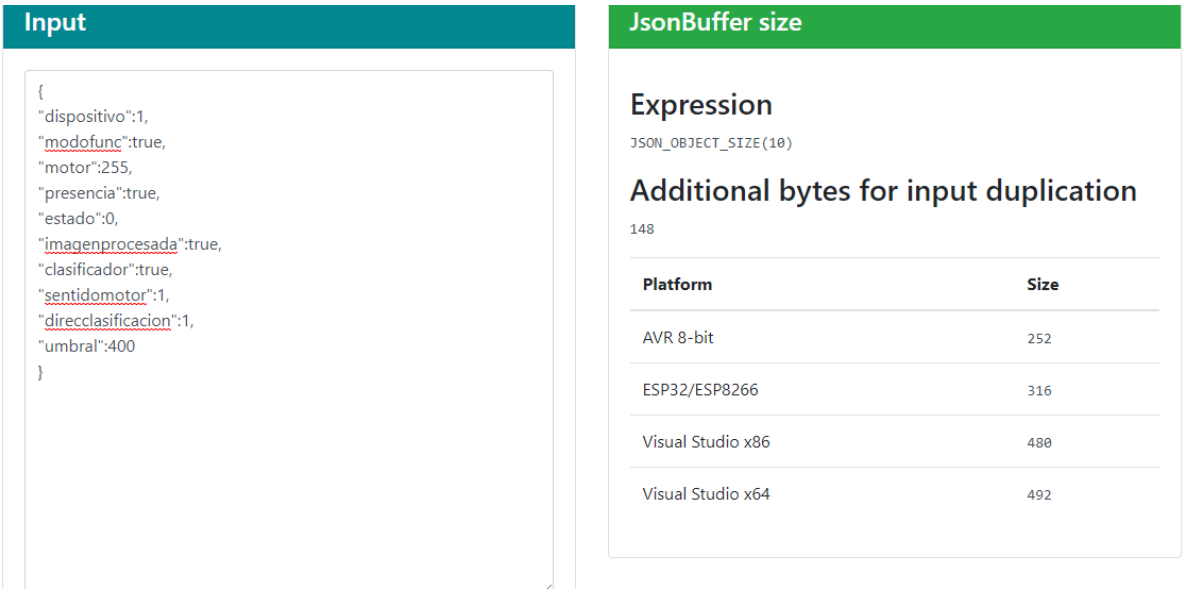

Figura 44. Asistente Arduinojson.org.

En la Figura 44 se muestra el resultado del tamaño de los objetos de Json, adicionalmente muestra el tamaño de bytes que en las diferentes plataformas que se puede utilizar ArduinoJson. En nuestro caso la plataforma a utilizar es AVR 8-bit dentro del grupo ATMEGA, el tamaño que el microcontrolador utilizará es 263 bytes de un total de 32Kbytes.

Para desarrollar el fichero del Arduino(.ino), el primer paso es incluir la librería ArduinoJson.h, que facilitará leer la cadena de caracteres y asignar los diferentes objetos y sus valores correspondientes a cada uno. Luego de declarar variables que ayudara a la programación de los actuadores o sensores, se debe abrir comunicación en el void setup a una velocidad de 115200 baudios, y asignar si las variables son de tipo GPIO e inicializar las diferentes variables auxiliares. El siguiente paso es el void loop, campo para programar las diferentes acciones de los sensores y actuadores.
Dicho esto, seguiremos la siguiente sintaxis según el ejemplo presentado para el encendido/apagado de un led desde el sistema físico o desde el sitio web.

```
#include "ArduinoJson.h"
int inicio = 7;
int paro = 9;
int ledstate = 8;
```

Lo primero es declarar el tamaño de los objetos “bufferSize”, para este ejemplo el tamaño del Objeto en JSON es dos, a continuación, se declara un buffer dinámico para los datos de

salida “Buffersalida”, escribiendo los objetos dentro de la variable “bufferSize”. Por último, declaramos la variable “datosalidajson”, variable que guardado el objeto y su valor creado luego de un evento que se obtuvo en el sistema físico.

```
const size_t bufferSize =JSON_OBJECT_SIZE(2);
DynamicJsonBuffer Buffersalida(bufferSize);
JsonObject& datosalidajson = Buffersalida.createObject();
```

Adicional se debe tomar en cuenta las diferentes variables auxiliares a utilizar en el código de programación.

```
unsigned long tiempoinicio = 0;
char d;
int flag = 0;
char serialequipo2=0;
String datospuertoserial = "";
bool haycadena = false;
bool iniciocadena = false, fincadena = false;
```

Inicializamos el Void setup. En este campo se debe abrir la comunicación por el puerto serial a velocidad (115200 baudios) para el envió y recepción de la cadena de caracteres, e indicar las diferentes variables GPIO de entradas o salidas.

```
void setup() {
  Serial.begin(115200);
  pinMode(ledstate, OUTPUT);
  pinMode(inicio, INPUT);
  pinMode(paro, INPUT);
  }
```

Y por último la programación correspondiente dentro de un void loop. En este campo se deberá leer la cadena de caracteres y además de realizar las diferentes acciones de control con el prototipo. El primer paso será verificar si la cadena de caracteres llego forma completa, luego de haber sido guardado en la variable (d) tipo char, mendiante la identificación de los caracteres “{“ y “}” indicando el inicio y final de la cadena respectivamente, así logrando contar con todos los objetos para el funcionamiento correcto del proyecto.

```
void loop() {
    while (Serial.available() > 0) {
      haycadena = true;
      d = Serial.read();
      if (d == '{') {
        datospuertoserial = "";
        iniciocadena = true;
      }
      if (iniciocadena) {
        datospuertoserial.concat(d);
      }
      if (d == '}') {
        fincadena = true;
        break;
      }
    }
```

Luego de que haya comprobado si la cadena ha llegado completamente se procede a leer y asignar los objetos con su valor en diferentes variables. Con la ayuda de la herramienta `String(kv.key).compareTo(“ ”)`.

```
DynamicJsonBuffer Bufferentrada(bufferSize);
```

```
      JsonObject&datojson=
Bufferentrada.parseObject(datospuertoserial);
      if (datojson.success()) {
        for (auto kv : datojson) {
          if (String(kv.key).compareTo("led") == true) {
            if (kv.value.as<bool>() == true) {
                        digitalWrite(ledstate, HIGH);
                       // encendido el led
            } else {
                      digitalWrite(ledstate, LOW);
                        // led apagado
             }
            }
           }
          } else {
                  Serial.print(datospuertoserial);
          }
                   datospuertoserial = "";
         }
```

Se ha visto el control de encendido/apagado de un led desde el sitio web. Ahora bien, si el usuario desea interactuar desde el prototipo y verificar los cambios en el sitio web. En el campo de programación del Arduino se debe enviar la cadena caracteres con los objetos correspondientes con la herramienta `datosalidajson.printTo(Serial)`. Previamente guardado los diferentes objetos en la variable `datosalidajson`.

```
if (digitalRead(inicio) == HIGH) {
  flag = 1;
  delay(120);

  if (flag != 0) {
      // la función datossalidajson con el nombre del objeto
      // y su valor
     datosalidajson["led"] = true;
    datosalidajson.printTo(Serial);
    digitalWrite(ledstate, HIGH);
    flag = 0;
  }
}
if (digitalRead(paro) == HIGH) {
  flag = 1;
  delay(120);
  if (flag != 0) {
    datosalidajson["led"] = false;
    datosalidajson.printTo(Serial);
    digitalWrite(ledstate, LOW);
    flag = 0;
  }
}
```

Con el ejemplo ilustrado en los apartados anteriores, permitirá al estudiante conocer la estructura para leer y enviar los diferentes objetos desde el aplicativo web hacia el sistema físico o viceversa.

En la aplicación del prototipo SECVIA, se utilizará las diferentes herramientas expuestas en el ejemplo anterior y se identificar los objetos que envíen, reciban o bidireccional por el puerto serial, tal como muestra en la Tabla 11. Donde el sistema podrá trabajar en los modos: automático y manual.

Ítem	Objetos	Prototipo Envió por el puerto serial.	Aplicación Lee los objetos del puerto serial de entrada.	Valor	Funcionalidad
1	dispositivo:		dispositivo:	Int(1)	Indica al prototipo correspondiente.
2	modofunc:		modofunc:	bool(true)	Activación modo automático
				bool(false)	Activación modo manual.
3	motor:		motor:	Int (0-125)	Control de la velocidad del motor
4	presencia:	presencia:		Bool(true/false)	Detección de engranaje
5	estado:	estado:	estado:	Int(-1)	Paro de emergencia del prototipo
				Int(0)	Stanby del prototipo
				Int(1)	Sistema en funcionamiento
6	imagenprocesada:		imagenprocesada:	bool(true/false)	Procesado de engranaje
7	clasificador:		clasificador:	bool(true/false)	Clasificación de engranajes buenos y defectuosos.
8	sentidomotor:		sentidomotor:	Int(-1)	Motor hacia delante
				Int(0)	Motor parado
				Int(1)	Motor hacia atrás
9	direcclasificacion:		direcclasificacion:	Int(-1)	Servomotor dirección engranajes defectuosos.
				Int(0)	Movimiento en Stanby
				Int(1)	Servomotor dirección engranajes correctos.
10	umbral:		umbral:	Int(400)	Valor para la calibración del sensor de posición
11	capturarengrane:		capturarengrane:	bool(true/false)	Activa iluminación.

Tabla 11. Objetos utilizados en el prototipo SECVIA.

En la Figura 45, se muestra el algoritmo que se debe seguir para la implementación del modo manual en el prototipo SECVIA, el algoritmo debe funcionar cuando el modo de funcionamiento es falso. Este valor deberá ser enviado desde aplicativo web e interpretado por el Arduino y trabajar en este campo.

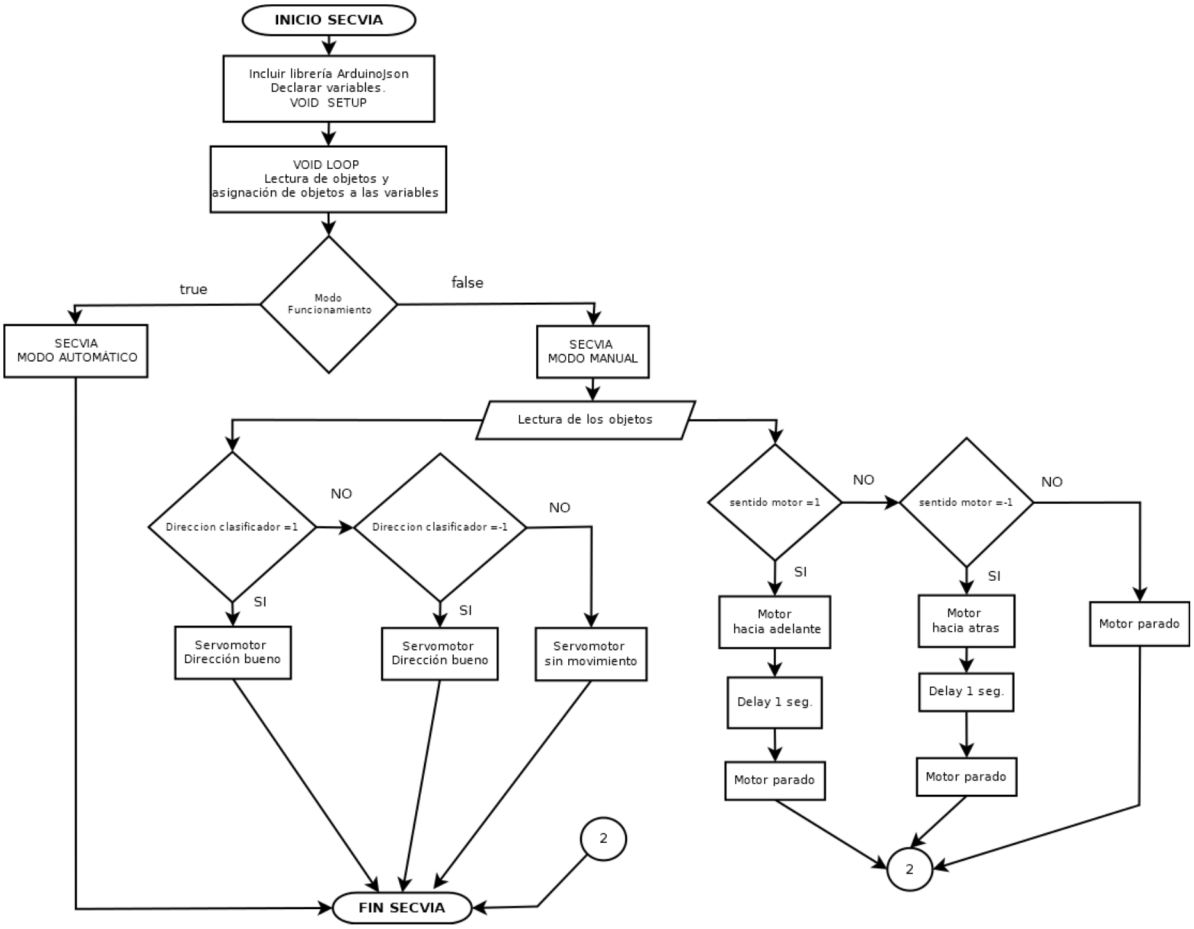

Figura 45. Funcionamiento del prototipo en modo manual.

Por último, se debe conocer los respectvos pines a utilizar para la asignación de los diferentes actuadores del prototipo. En el anexo A, se muestra e diagrama correspondiente y el diseño de la PCB a utilizar como shield en el Arduino Uno, para ello en la Tabla 12, se indica los diferentes puertos/pines para la programación de los actuadores, sensores y comandos de control.

GPIO	Denomin.	Asinación:	Tipo	Funcionalidad
Pin #6	red	OUTPUT	PWM	Combinación de valores para obtener el color deseado en el Diodo Led RGB.
Pin #3	blue	OUTPUT	PWM	
Pin #5	green	OUTPUT	PWM	
Pin #13	objbueno	OUTPUT	Digital	Led indicador para la clasificación de engranajes buenos.
Pin #8	objmalo	OUTPUT	Digital	Led indicador para la clasificación de engranajes malos.
Pin #A0	receptor	INPUT	Analógico	Obtiene el valor del sensor de posición cuando hay un corte del haz de Infrarrojo.
Pin #9	emerg	INPUT	Digital (bool)	Comando físico para el paro emergencia del prototipo
Pin #7	start	INPUT	Digital (bool)	Inicio/paro del sistema automático del prototipo
Pin #4	ledcamara	INPUT	Digital (bool)	Encendido de la iluminación para la captura de la imagen.
Pin #10	pinservo	OUTPUT	PWM (int)	Selección de engranajes bueno y malos.

Pin #11	motorpwm	OUTPUT	PWM (int)	Velocidad del motor para la cinta de transporte.
Pin #12	motordir	OUTPUT	Digital (bool)	Sentido de giro de la cinta de transorte

Tabla 12. Asignación de GPIOs para el ARDUINO UNO

6.5 Guía de laboratorio #2. Calibración del sensor de posición.

En la Guía de laboratorio #1 se logró interactuar con los diferentes actuadores que cuenta el sistema CPS académico. En esta segunda parte de la guía práctica se tratará de comprender sobre los distintos sensores que puede contar un equipo, para nuestro prototipo haremos referencia específicamente al sensor de posición.

La estructura del sensor de posición cuenta con un diodo led emisor y un diodo receptor, el led emisor envía haz de luz a una distancia cierta siendo capturada por el diodo receptor, generando un valor de umbral con relación a la distancia y posición del emisor, este valor debe ser calibrado para que exista la detección de objetos.

Considerando lo aprendido en la guía de Laboratorio #1. El primer paso es añadir el comando que servirá para la calibración del umbral del sensor posición.
En el subsitio de control y monitorización, se debe crear una nueva sección dentro del fichero controlymoni.html. Esta sección puede estar ubicado en cualquier posición del interfaz del subsitio, recomendación la sección debe estar al final de la sección del encabezado. En el espacio de esta sección para nuestra aplicación será crearemos un comando deslizante(slider). Este comando deslizante se crea dentro de una división de la sección y detallando la forma(`<form>`) de este comando, consiguiente de una entrada **input**, especificado su id, el tipo(`range`), su máximo y su mínimo valor de rango.
Para esto utilizaremos la siguiente sintaxis ya desarrollada:

```
<section id="showcase2" style="background-color:#268982"
style="margin-left: 20px" style="margin-right: 20px">
  <div class ="container" style="font-size: 12px">
            <h2>CALIBRACION DEL SENSOR DE POSICIÓN</h2>
  </div>
  <div class ="container">
    <form>
                  <p>  Valor de Umbral </p>
      <input id="umbral" type="range" name="sliderumbral"
             min="800" max="1200" value="800">
    </form>
          <label id="umbralvalor" style="font-size: 15px"
     style="text-align: center" style="color: #000000">
    </label>
  </div>
</section>
```

Creado esta sección destinado a la calibración del umbral de un sensor de posición, en la Figura 46 se muestra el interfaz que se ha generado y que se contará en la aplicación web.

Figura 46. Comando para la calibración del sensor de posición.

Hay que recordar que cada comando cuenta con sus respectivos identificadores, con la final de con el entorno JavaScript del servidor y de la aplicación web. En esta parte de la guía práctica se utilizará los diferentes identificadores presentados en la Tabla 13 tanto para el botón, como para los textos.

Tipo	Nombre del Slider	Identificador del botón (id)	Identificador del texto (id)
input	CALIBRACIÓN DEL SENSOR DE POSICIÓN	id="umbral"	Id="umbralvalor"

Tabla 13. Identificador del comando para la calibración del sensor.

Ahora, se deberá realizar los diferentes eventos dinámicos en el entorno Javascript (main.js), y poder añadir eventos después que se haya interactuado con este comando, enviando el o los identificadores y sus valores para ser enviado al servidor. Para obtener y guardar la información del id, se utiliza la herramienta `var id=document.getElementById("id").`

Con la herramienta `elemento.addEventListener()`, leeremos los eventos del identificador del sitio web, para luego asignarle alguna actividad y enviar al sistema físico una vez que este elemento se haya guardado en un vector `arreglodispositivo.`

```
umbral.addEventListener("click", function () {
    var dispositivo = {};
    dispositivo.dispositivo = 1;
    dispositivo.umbral = Number(umbral.value);
    umbralvalor.innerHTML = umbral.value;
    arreglodispositivo.push(dispositivo);
    console.log(arreglodispositivo);
    socket.emit('datos', arreglodispositivo);
    arreglodispositivo = [];
});
```

Para la obtención del estado del cliente se utilizará la herramienta `socket.on`, este estado guardará el valor de este objeto que viene del sitio web.

```
if (data[i].umbral != null) {
        umbral.value = Number(data[i].umbral);
        umbralvalor = Number(data[i].umbral);
    }
```

En el fichero **app.js,** fichero correspondiente al servidor. En el servidor también debe incluirse el nuevo objeto de esta práctica. Cabe recordar que el nuevo objeto debe de estar dentro del arreglo Json **dispositivo.**

Para establecer la conexión y leer las variables del arreglo Json se utiliza la herramienta **io.on,** estas variables deberán ser agregados en un vector (`arreglodispositivo`) mediante el método **push**, y al final emitiendo todo este vector a la conexión del servidor.

Finalizaremos con la plataforma de Arduino, aquí se debe incrementar el tamaño del buffer a 11 objetos: `JSON_OBJECT_SIZE(11)`, y se aplicará los mismos procedimientos de programación de la guía práctica #1 para su correspondiente funcionamiento.

El funcionamiento se muestra mediante el algoritmo presentado en la Figura 47, el mismo que servirá solo para habilitar el sistema del mando manual, sistema que fue implementado en la guía práctica # 1.

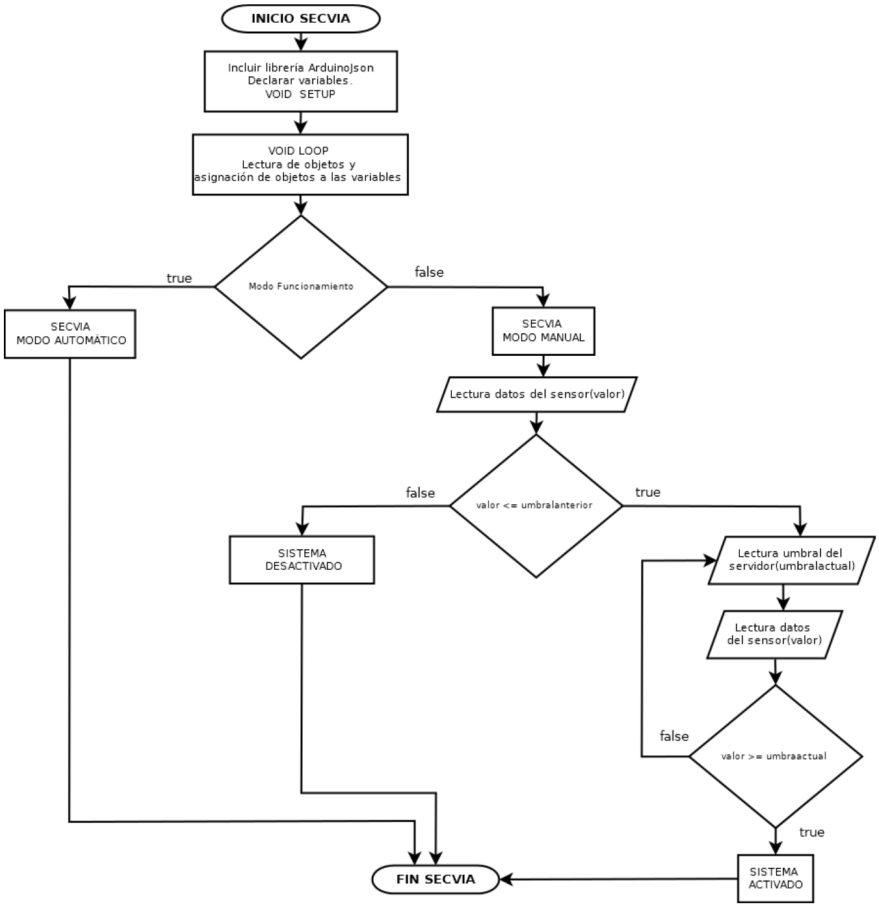

Figura 47. **Algoritmo Sistema manual y calibración sensor**

6.6 Recomendaciones para las comprobaciones del estado del servidor web y del clientesocket

En este apartado se presentará algunos comandos necesarios que servirá para comprobar, inicializar o paralizar los diferentes motores de ejecución tanto el servidor web como el cliente correspondiente al prototipo. Comandos que ayudará al estudiante a verificar si el sistema se encuentra en error.
El principal comando `systemctl` es un ejecutador que ayuda al servidor web como al cliente funcionen automáticamente, sin la necesidad de que el usuario ejecute manualmente los correspondientes interfaces mencionados.

6.6.1 Servidor

- **Iniciar un servidor.**

La siguiente línea de comando presentando a continuación permitirá al sistema ejecutar.

```
sudo systemctl start servidorweb
```

- **Parar un servidor.**

Ahora bien si el estudiante desea apagar o parar al servidor, para realizar cualquier cambio o modificación en los ficheros correspondientes al servidor.

```
sudo systemctl stop servidorweb
```

- **Reiniciar un servidor.**

Si el usuario o estudiante observa que el servidor presenta alguna anomalía, con el comando siguiente permitirá reiniciar al servidor web y por ende reinicia a los valores de los clientes conecatados a este servidor.

```
sudo systemctl restart servidorweb
```

- **Estado del servidor.**

El último comando presentado permitirá al usuario ver en que estado se encuentra el servidor web, es decir si el servidor esta en funcionamineto o presente algún problema.

```
sudo systemctl status servidorweb
```

6.6.2 Cliente Socket

De la misma manera que el servidor web, el motor ejecutador del cliente del prototipo, se utilizará los mismos comandos, haciendo referencia al fichero ejecutador del cliente. Tal como se muestra acontinuación:

```
sudo systemctl start programabanda
sudo systemctl stop programabanda
sudo systemctl restart programabanda
sudo systemctl status programabanda
```

Por último, si el estudiante desea ejecutar manualmente cada vez que ponga en funcionamiento el prototipo puede utilizar los siguientes comandos.

- **Motor ejecutador Servidor Web.**

```
cd Documentos/ UPC_TFM_Fernando_Jacome/ServidorWeb/bin
./www
```

- **Motor ejecutador Cliente Socket SECVIA.**

```
cd Documentos/ UPC_TFM_Fernando_Jacome/ClienteSocket/dist/Debug/GNU-Linux
./clientesocket
```