

COMPARACIÓN ENTRE DOS ESTRATEGIAS DE RECUPERACIÓN DE EQUILIBRIO PARA UN ROBOT BÍPEDO EN FASE DE BIPEDESTACIÓN ESTÁTICA.

ANEXOS



Daniel Felipe Estrada Carvajal
Yoniver Hoyos Muñoz

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Popayán, 2021

Tabla de contenido

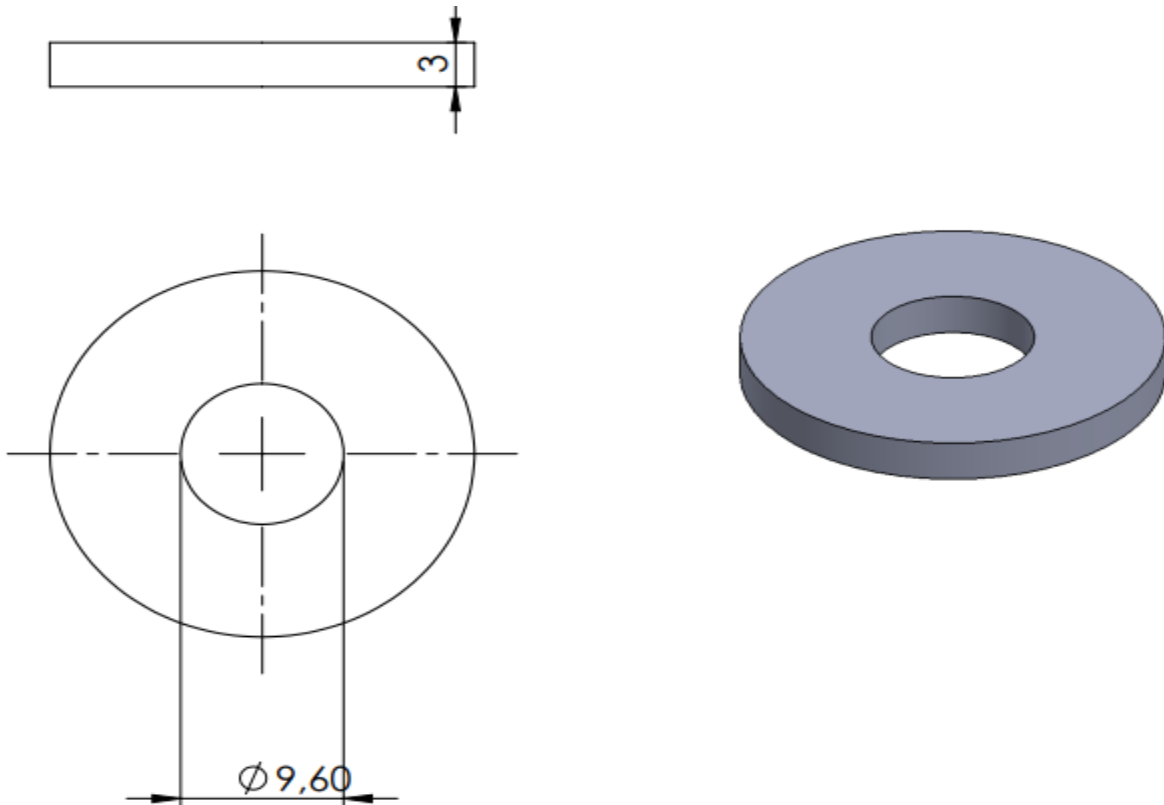
Anexos	1
A. Modelos 3D usados en la base de soporte inclinable	2
B. Código diseñado en la plataforma <i>arduino</i> para la estrategia de tobillo	6
C. Código diseñado en la plataforma <i>arduino</i> para la estrategia de cadera	10
D. Código desarrollado en <i>Matlab</i> para el diseño de la interfaz gráfica utilizada para la recolección de datos	14
E. Código empleado en MatLab para la obtención de los diagramas de cajas y bigotes	20

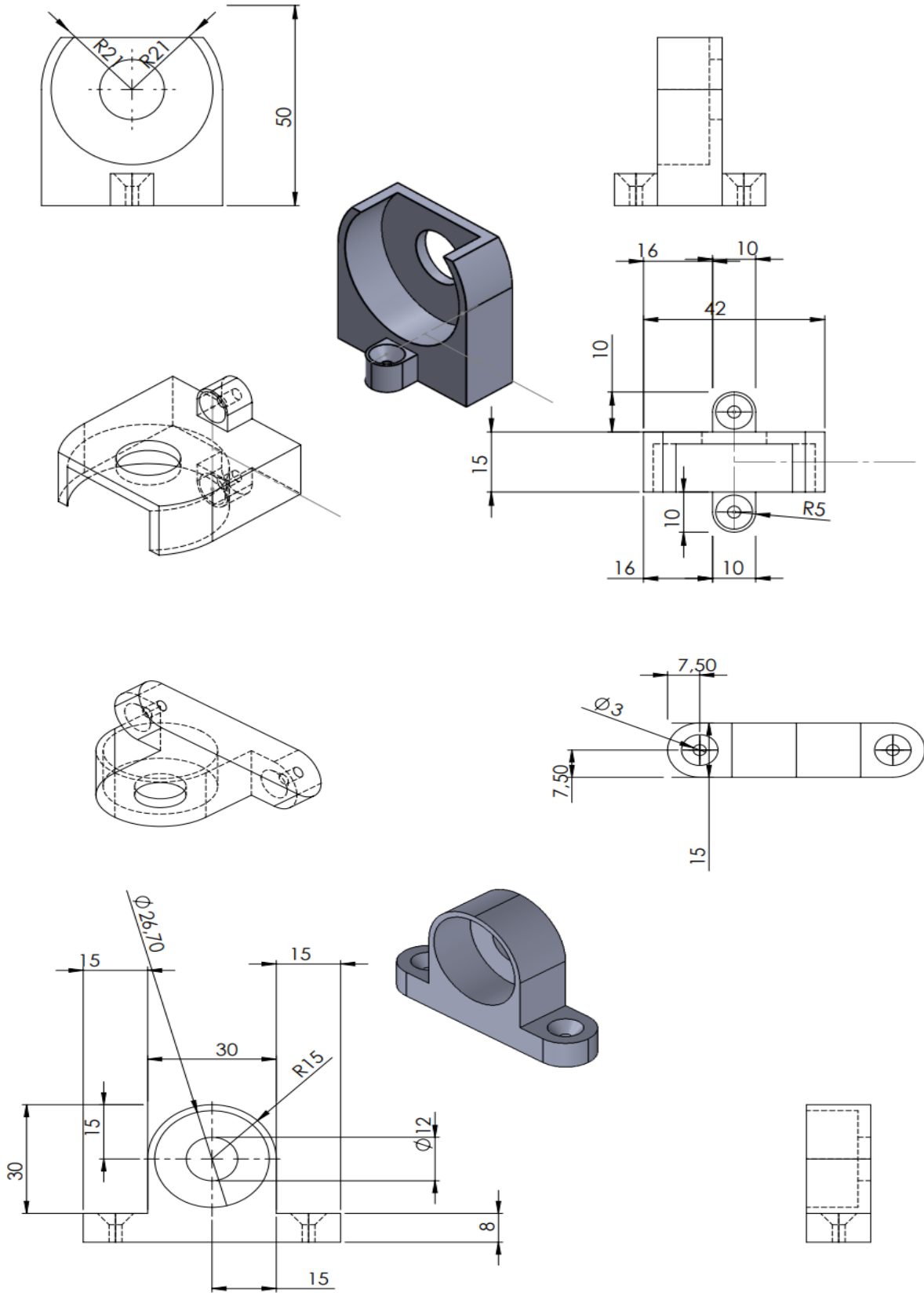
Anexos

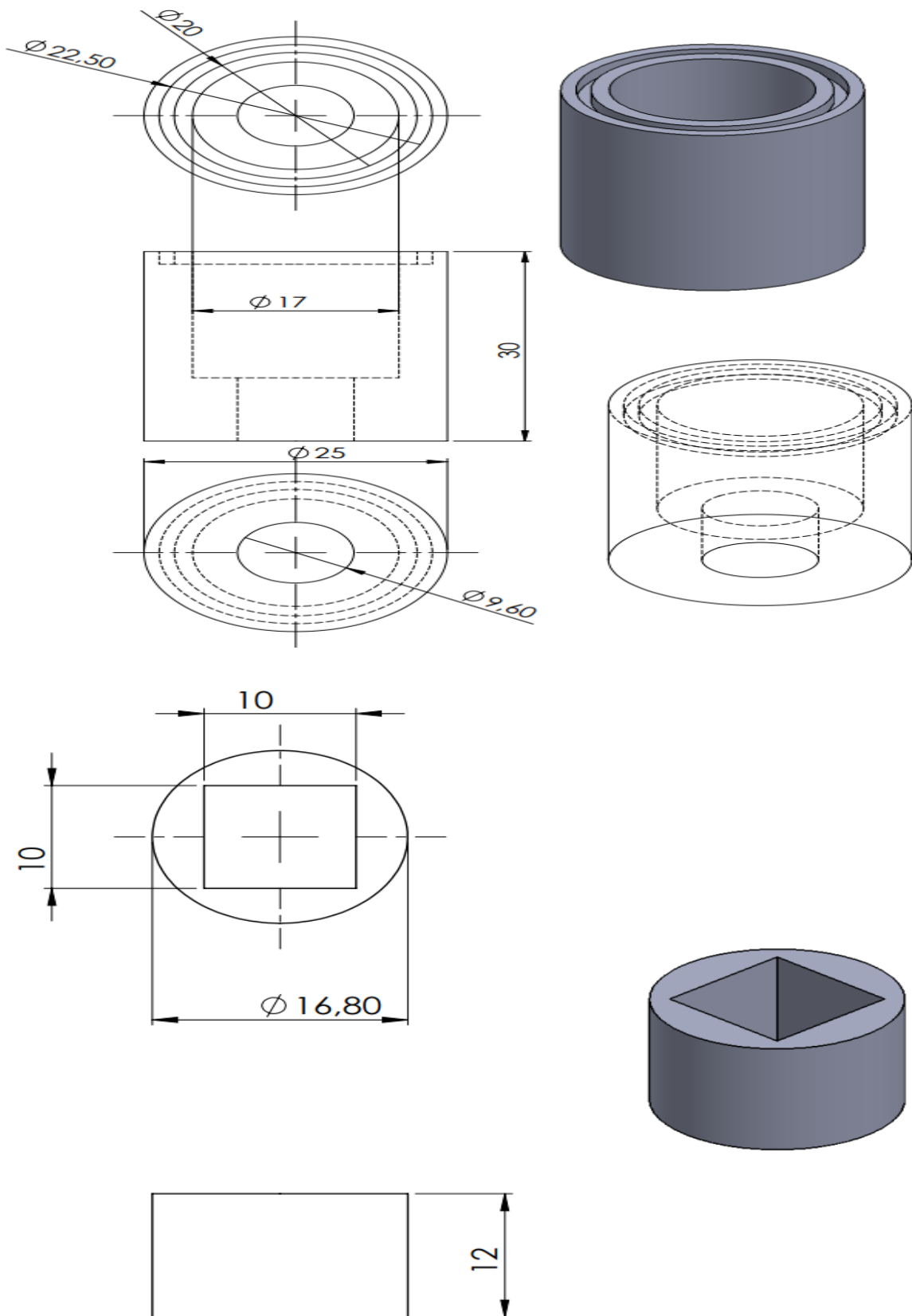
Anexos A

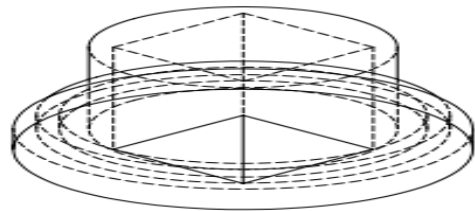
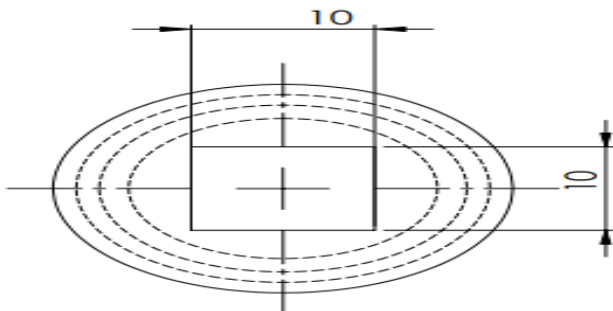
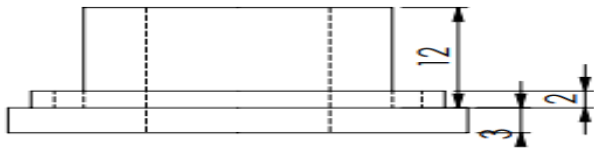
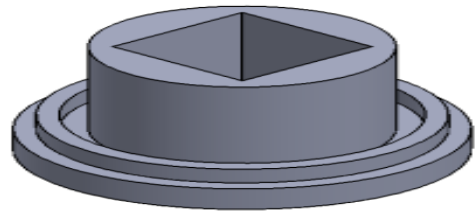
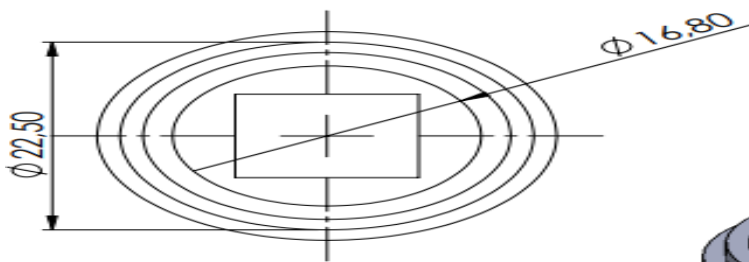
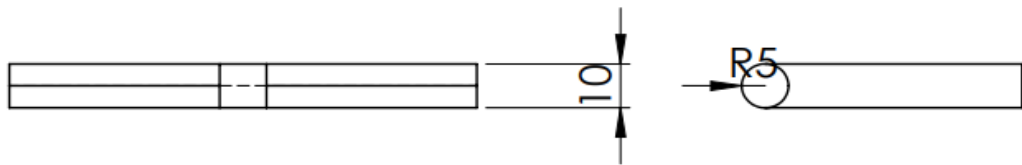
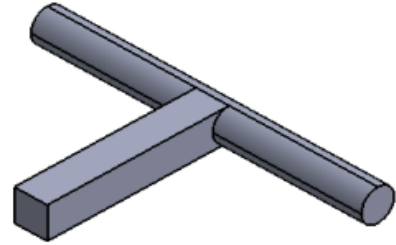
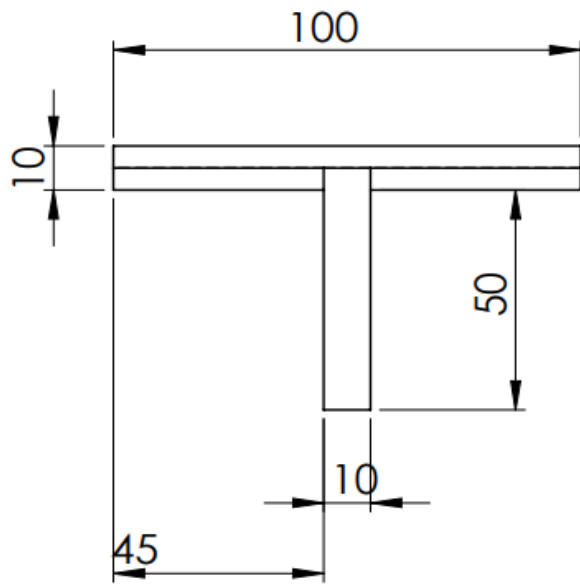
Modelos 3D usados en la base de soporte inclinable

A continuación se presentan diversas vistas de las piezas empleadas para la construcción de la base de soporte inclinable; Para la correcta comprensión de las dimensiones es necesario tener en cuenta que la unidad de medida se encuentran en *mm*.









Anexos B

Código diseñado en la plataforma *arduino* para la estrategia de tobillo

```
1 #include <SoftwareSerial.h>
2 #include <DynamixelSerial.h>
3 SoftwareSerial Blue(9, 10); // RX | TX
4
5 // Funciones
6 float Read_Process_Output(int);
7 float Read_Setpoint(void);
8 float Control_Law_X(float, float);
9 void Write_Control_Law(int);
10 void Wait_Until_Next_Sampling_Time(unsigned long int,unsigned int);
11
12 // Variables
13 float Error = 0.0, ErrorAnt = 0.0, ErrorAcum=0.0, OutputAnt =0.0, Kp=-0.15, Kd= -2.2, Ki= -0.0002, Offset = 606;
14 float ZMP_X, Uproporcional=0.0, Uderivativo = 0.0, Uintegral=0.0;
15 const float Tm = 40, d=10, Y1=d+6, Y2=d+54, X1=44; // Tiempo de muestreo 40 milisegundos, Definicion del tamaño del poligono de soporte
16 unsigned long AnteriorMillis=0.0, ActualMillis=0.0;
17 float data[16], DS[16];
18 const float K[] = {616.8311, 572.4295, 677.6467, 550.4752, 195.0131, 24.2161} ; // Constantes del polinomio de caracterización del sensor FSR
19
20 // Constantes del multiplexor
21 const int muxSIG = A0;
22 const int muxS0 = 12;
23 const int muxS1 = 11;
24 const int muxS2 = 8;
25 const int muxS3 = 7;
26
27 //Configuración multiplexor
28 int SetMuxChannel(byte channel)
29 {
30     digitalWrite(muxS0, bitRead(channel, 0));
31     digitalWrite(muxS1, bitRead(channel, 1));
```

```

32  digitalWrite(muxS2, bitRead(channel, 2));
33  digitalWrite(muxS3, bitRead(channel, 3));
34 }
35
36 void setup() {
37  Blue.begin(115200);
38  delay(1000);
39  Blue.println("I");
40
41  //Configuración multiplexor
42  pinMode(muxS0, OUTPUT);
43  pinMode(muxS1, OUTPUT);
44  pinMode(muxS2, OUTPUT);
45  pinMode(muxS3, OUTPUT);
46
47  Dynamixel.setSerial(¿Serial);    // ¿Serial - Arduino UNO/NANO/MICRO, ¿Serial1, ¿Serial2, ¿Serial3 - Arduino Mega
48  Dynamixel.begin(1000000,2);     // Inicializar el servo a 1 Mbps y un pin de control en la salida digital 2
49
50  //Configuración de torque para los motores del robot
51  Dynamixel.setMaxTorque(15,1023);
52  Dynamixel.setMaxTorque(16,1023);
53  Dynamixel.setMaxTorque(1,200);
54  Dynamixel.setMaxTorque(2,200);
55  Dynamixel.setMaxTorque(3,200);
56  Dynamixel.setMaxTorque(4,200);
57  Dynamixel.setMaxTorque(5,200);
58  Dynamixel.setMaxTorque(6,200);
59  delay(100);
60
61
62  //Posición inicial de los motores del robot
63  //Motores del tren superior (Brazos, hombros, codos)
64  Dynamixel.move(1,200);
65  Dynamixel.move(2,824);
66  Dynamixel.move(3,324);
67  Dynamixel.move(4,790);
68  Dynamixel.move(5,490);
69  Dynamixel.move(6,534);
70
71  //Motores del tren inferior (Tobillos, cadera, rodillas)
72  Dynamixel.move(7,360);
73  Dynamixel.move(8,664);
74  Dynamixel.move(9,500);
75  Dynamixel.move(10,512);
76  Dynamixel.move(11,418);
77  Dynamixel.move(12,606);
78  Dynamixel.move(13,324);
79  Dynamixel.move(14,700);
80  Dynamixel.move(15,606);
81  Dynamixel.move(16,418);
82  Dynamixel.move(17,512);
83  Dynamixel.move(18,512);
84  delay(5000);
85 }
86 void loop() {
87
88  float Reference, Control_X,ZMP;
89  ActualMillis = millis();
90
91  ZMP=Read_Process_Output();           // Función de lectura de la variable salida del proceso

```

8ANEXOS B. CÓDIGO DISEÑADO EN LA PLATAFORMA ARDUINO PARA LA ESTRATEGIA DE TOBILLO

```
92 Reference = Read_Setpoint();           // Función que establece el valor de referencia
93 Control_X = Control_Law_X(Reference,ZMP); // Función de la ley de control
94 Write_Control_Law(Control_X);         // Función encargada de enviar el esfuerzo de control a los actuadores
95
96 //Envío de los valores para cada indicador de comparación
97 Blue.println(data[6]);
98 Blue.println(data[15]);
99 Blue.println(ErrorAcum);
100 Blue.println(Control_X);
101
102 Wait_Until_Next_Sampling_Time(ActualMillis,Tm); // Función que garantiza el cumplimiento del tiempo de muestreo en cada ciclo de control
103 }
104
105 //Polinomio de caracterización del sensor FSR
106 float pol5(float x){
107 x=(x-492.292682926829)/152.200072914312;
108 float masa = K[5]^pow(x,5)+K[4]^pow(x,4)+K[3]^pow(x,3)+K[2]^pow(x,2)+K[1]^pow(x,1)+K[0];
109 return masa;
110 }
111
112 //Lectura de la variable de proceso
113 float Read_Process_Output() {
114
115 //Lectura de las entradas del multiplexor
116 for (byte i = 6; i < 16; i++)
117 {
118     SetMuxChannel(i);
119     DS[i] = analogRead(muxSIG);
120
121
122
121 switch (i)
122 {
123 //Cálculo del voltaje entregado por el sensor
124 case 6:
125     {
126         float voltajeSensor = DS[6] * (5.0 / 1023.0);
127         data[6]=voltajeSensor;
128     }
129 break;
130 //Cálculo de la corriente entregada por el sensor
131 case 15:
132     {
133         float total = 0;
134         for(int j=0; j <= 10; j++){
135             float corriente=0;
136             float voltaje=DS[15] * (5.0 / 1023.0);
137             corriente = -5.3094*voltaje + 13.1731;
138             total = total + corriente;
139             DS[15] = analogRead(muxSIG);
140         }
141         float media = total / 10;
142         data[15] = media;
143     }
144 break;
145 // Calculo de masa para cada sensor FSR
146 default:|
147     {
148         data[i] = pol5(DS[i]);
149         if(data[i]<50){
150             data[i] = 0;
151         }
152     }
```

```

152     }
153     break;
154 }
155 }
156
157 //Cálculo del ZMP
158 return ZMP_X=X1*(data[11]+data[12]+data[7]+data[8]-data[9]-data[10]-data[13]-data[14])/(data[7]+data[8]+data[9]+data[10]+data[11]+data[12]+data[13]+data[14]);
159 }
160
161 //Asignación del valor para la señal de referencia
162 float Read_Setpoint(void) {
163     static float Reference = 0;
164     return Reference;
165 }
166
167 // Funcion para la ley de control
168 float Control_Law_X(float Reference, float Output){
169
170     int Control ;
171     Error = Reference - Output;
172     Uproporcional = Offset + ( Kp * Error);
173     Uderivativo   = Kd*((Output - OutputAnt)/Tm);
174     ErrorAcum = ErrorAcum + (Ki*((Tm/2)*(ErrorAnt + Error)));
175     Uintegral = ErrorAcum;
176     Control = Uproporcional - Uderivativo + Uintegral;
177     Control = constrain(Control,0,1023);
178     ErrorAnt = Error;
179     OutputAnt = Output;
180     return Control;
181 }
182 }

```

```

183
184 // Envio del esfuerzo de control a los actuadores
185 void Write_Control_Law(int Control) {
186
187     int Control_15 = Control;
188     int Control_16 = 1023-Control;
189     Dynamixel.moveSpeed(15,Control_15,100);
190     Dynamixel.moveSpeed(16,Control_16,100);
191 }
192
193 // Funcion que asegura que el tiempo de muestreo siempre sea de 40ms
194 void Wait_Until_Next_Sampling_Time(unsigned long int TBegin, unsigned int Ts)
195 {
196     while ((millis() - ActualMillis) < Tm);
197 }

```

Anexos C

Código diseñado en la plataforma *arduino* para la estrategia de cadera

```
1 #include <SoftwareSerial.h>
2 #include <DynamicSerial.h>
3 SoftwareSerial Blue(9, 10); // RX | TX
4
5 // Funciones
6 float Read_Process_Output(int);
7 float Read_Setpoint(void);
8 float Control_Law_X(float, float);
9 void Write_Control_Law(int);
10 void Wait_Until_Next_Sampling_Time(unsigned long int,unsigned int);
11
12 // Variables
13 float Error = 0.0, ErrorAnt = 0.0, ErrorAcum=0.0, OutputAnt =0.0, Kp=-0.15, Kd= -2.2, Ki= -0.0002, Offset1 = 606,Offset2 = 606;
14 float ZMP_X, Uproporcional = 0.0, Uderivativo = 0.0, Uintegral = 0.0;
15 int Control1 = 0, Control2 = 0, ControlHip1, ControlHip2;
16 const float Tm = 60, d=10, Y1=d+6, Y2=d+54, X1=44; // Tiempo de muestreo 60 milisegundos, Definicion del tamaño del poligono de soporte
17 unsigned long AnteriorMillis=0.0, ActualMillis=0.0;
18 float data[16], DS[16];
19 const float K[] = {616.8311, 572.4295, 677.6467, 550.4752, 195.0131, 24.2161} ; // Constantes del polinomio de caracterización del sensor FSR
20
21 // Constantes del multiplexor
22 const int muxSIG = A0;
23 const int muxS0 = 12;
24 const int muxS1 = 11;
25 const int muxS2 = 8;
26 const int muxS3 = 7;
27
28 //Configuración multiplexor
29 int SetMuxChannel(byte channel)
30 {
31     digitalWrite(muxS0, bitRead(channel, 0));
```

```

32  digitalWrite(muxS1, bitRead(channel, 1));
33  digitalWrite(muxS2, bitRead(channel, 2));
34  digitalWrite(muxS3, bitRead(channel, 3));
35  }
36
37  void setup() {
38  Blue.begin(115200);
39  delay(1000);
40  Blue.println("I");
41
42  //Configuración multiplexor
43  pinMode(muxS0, OUTPUT);
44  pinMode(muxS1, OUTPUT);
45  pinMode(muxS2, OUTPUT);
46  pinMode(muxS3, OUTPUT);
47
48  Dynamixel.setSerial(Blue);          // Blue - Arduino UNO/NANO/MICRO, Serial1, Serial2, Serial3 - Arduino Mega
49  Dynamixel.begin(1000000, 2);       // Inicializar el servo a 1 Mbps y un pin de control en la salida digital 2
50
51  //Configuración de torque para los motores del robot
52  Dynamixel.setMaxTorque(15,1023);
53  Dynamixel.setMaxTorque(16,1023);
54  Dynamixel.setMaxTorque(11,1023);
55  Dynamixel.setMaxTorque(12,1023);
56  Dynamixel.setMaxTorque(1,200);
57  Dynamixel.setMaxTorque(2,200);
58  Dynamixel.setMaxTorque(3,200);
59  Dynamixel.setMaxTorque(4,200);
60  Dynamixel.setMaxTorque(5,200);
61  Dynamixel.setMaxTorque(6,200);
62  delay(100);

64  //Posición inicial de los motores del robot
65  //Motores del tren superior (Brazos, hombros, codos)
66  Dynamixel.move(1,200);
67  Dynamixel.move(2,824);
68  Dynamixel.move(3,324);
69  Dynamixel.move(4,790);
70  Dynamixel.move(5,490);
71  Dynamixel.move(6,534);
72
73  //Motores del tren inferior (Tobillos, cadera, rodillas)
74  Dynamixel.move(7,360);
75  Dynamixel.move(8,664);
76  Dynamixel.move(9,500);
77  Dynamixel.move(10,512);
78  Dynamixel.move(11,418);
79  Dynamixel.move(12,606);
80  Dynamixel.move(13,324);
81  Dynamixel.move(14,700);
82  Dynamixel.move(15,614);
83  Dynamixel.move(16,410);
84  Dynamixel.move(17,512);
85  Dynamixel.move(18,512);
86  delay(5000);
87  }
88
89  void loop() {
90  float Reference, Control_X,ZMP;
91
92  ActualMillis = millis();
93
94  ZMP=Read_Process_Output();          // Función de lectura de la variable salida del proceso

```

12 ANEXOS C. CÓDIGO DISEÑADO EN LA PLATAFORMA ARDUINO PARA LA ESTRATEGIA DE CADERA

```
95 Reference = Read_Setpoint(); // Función que establece el valor de referencia
96 Control_X = Control_Law_X(Reference,ZMP); // Función de la ley de control
97 Write_Control_Law(Control_X); // Función encargada de enviar el esfuerzo de control a los actuadores
98
99 //Envío de los valores para cada indicador de comparación
100 Blue.println(data[6]);
101 Blue.println(data[15]);
102 Blue.println(ErrorAcum);
103 Blue.println(Control1);
104
105 Wait_Until_Next_Sampling_Time(ActualMillis,Tm); // Función que garantiza el cumplimiento del tiempo de muestreo en cada ciclo de control
106 }
107
108 //Polinomio de caracterización del sensor FSR
109 float pol5(float x){
110 x=(x-492.292682926829)/152.200072914312;
111 float masa = K[5]^pow(x,5)+K[4]^pow(x,4)+K[3]^pow(x,3)+K[2]^pow(x,2)+K[1]^pow(x,1)+K[0];
112 return masa;
113 }
114
115 //Lectura de la variable de proceso
116 float Read_Process_Output() {
117
118 //Lectura de las entradas del multiplexor
119 for (byte i = 6; i < 16; i++)
120 {
121 SetMuxChannel(i);
122 DS[i] = analogRead(muxSIG);
123
124
125 switch (i)
126 {
127 //Cálculo del voltaje entregado por el sensor
128 case 6:
129 {
130 float voltajeSensor = DS[6] * (5.0 / 1023.0);
131 data[6]=voltajeSensor;
132 }
133 break;
134 //Cálculo de la corriente entregada por el sensor
135 case 15:
136 {
137 float total = 0;
138 for(int j=0; j <= 10; j++){
139 float corriente=0;
140 float voltaje=DS[15] * (5.0 / 1023.0);
141 corriente = -5.3094*voltaje + 13.1731;
142 total = total + corriente;
143 DS[15] = analogRead(muxSIG);
144 }
145 float media = total / 10;
146 data[15] = media;
147 }
148 break;
149 // Calculo de masa para cada sensor FSR
150 default:
151 {
152 data[i] = pol5(DS[i]);
153 if(data[i]<50){
154 data[i] = 0;
155 }
```

```

155     }
156     break;
157 }
158 }
159
160 //Cálculo del ZMP
161 return ZMP_X=X1*(data[11]+data[12]+data[7]+data[8]-data[9]-data[10]-data[13]-data[14])/(data[7]+data[8]+data[9]+data[10]+data[11]+data[12]+data[13]+data[14]);
162 }
163
164 //Asignación del valor para la señal de referencia
165 float Read_Setpoint(void) { //retorna el valor de la referencia o set point decontrol
166     static float Reference = 0;
167     return Reference;
168 }
169
170 // Funcion para la ley de control
171 float Control_Law_X(float Reference, float Output){
172
173     int Control ;
174     Error = Reference - Output;
175     Uproporcional = ( Kp * Error);
176     Uderivativo = Kd*((Output - OutputAnt)/Tm);
177     ErrorAcum = ErrorAcum + (Ki*((Tm/2)*(ErrorAnt + Error)));
178     Uintegral = ErrorAcum;
179     Control = Uproporcional - Uderivativo + Uintegral;
180     ErrorAnt = Error;
181     OutputAnt = Output;
182     return Control;
183 }
184 }
185
186 // Envio del esfuerzo de control a los actuadores
187 void Write_Control_Law(float Control) {
188     float Khip=- (5.3/17.7);
189     Control1 = Offset1+Control;
190     Control1 = constrain(Control1,0,1023);
191     Control2 = 1023-(Offset1+Control);
192     Control2 = constrain(Control2,0,1023);
193     ControlHip1 = Offset2+(Khip*Control);
194     ControlHip1 = constrain(ControlHip1,0,1023);
195     ControlHip2 = 1023-(Offset2+(Khip*Control));
196     ControlHip2 = constrain(ControlHip2,0,1023);
197     Dynamixel.moveSpeed(15,Control1,100);
198     Dynamixel.moveSpeed(16,Control2,100);
199     Dynamixel.moveSpeed(11,ControlHip2,100);
200     Dynamixel.moveSpeed(12,ControlHip1,100);
201 }
202
203 // Funcion que asegura que el tiempo de muestreo siempre sea de 60ms
204 void Wait_Until_Next_Sampling_Time(unsigned long int TBegin, unsigned int Ts)
205 {
206     while ((millis() - ActualMillis) < Tm);
207 }

```


Anexos D

Código desarrollado en *Matlab* para el diseño de la interfaz gráfica utilizada para la recolección de datos

```

1  function varargout = Interfaz_tesis(varargin)
2  %% Código de inicialización - NO EDITAR
3  gui_Singleton = 1;
4  gui_State = struct('gui_Name',       mfilename, ...
5                    'gui_Singleton',  gui_Singleton, ...
6                    'gui_OpeningFcn', @Interfaz_tesis_OpeningFcn, ...
7                    'gui_OutputFcn',  @Interfaz_tesis_OutputFcn, ...
8                    'gui_LayoutFcn',  [], ...
9                    'gui_Callback',   []);
10
11  if nargin && ischar(varargin{1})
12      gui_State.gui_Callback = str2func(varargin{1});
13  end
14
15  if nargin
16      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17  else
18      gui_mainfcn(gui_State, varargin{:});
19  end
20  end
21
22  %% Funcion de inicialización
23  function Interfaz_tesis_OpeningFcn(hObject, eventdata, handles, varargin)
24
25      escudo = imread('EscudoUnicaucal.jpg');
26      axes(handles.axes5);
27      image(escudo);
28      axis off;
29
30      clear instrfind;
31      delete(instrfind);           % Elimina objetos de comunicacion pasados
32      delete(timerfind);          % Elimina objetos temporizaodres usados antes

```

16 ANEXOS D. CÓDIGO DESARROLLADO EN MATLAB PARA EL DISEÑO DE LA INTERFAZ GRÁFICA UTILIZADA

```

34 - handles.Timer1 = timer(...) %Definición del temporizador utilizado
35 -     'ExecutionMode', 'fixedrate',...
36 -     'Period',.2,...
37 -     'TimerFcn',{@timer_callback,hObject});
38
39 - handles.output = hObject; % Elija la salida de línea de comando predeterminada
40
41 - guidata(hObject, handles); % Actualiza la estructura handles
42 - end
43
44 - %% Las salidas de esta función son retornadas a la línea de comando
45 - function varargout = Interfaz_tesis_OutputFcn(hObject, eventdata, handles)
46
47 - varargout{1} = handles.output; % Obtener la salida de la línea de comando predeterminada
48 - % desde la estructura handles
49 - end
50
51 - %% Función ejecutada al presionar el botón "Conectar" en la interfaz
52 - function pushbutton1_Callback(hObject, eventdata, handles)
53
54 - global Serial_Com; % Define variable global para ser utilizada por cualquier subfuncion
55 - handles.Serial_Port = 'COM6'; % Cambiar segun el puerto serial utilizado para la comunicación serial
56 - Serial_Com = serial('COM6','BaudRate',115200,'Terminator', 'CR/LF');
57
58 - fopen(Serial_Com); % Inicia la comunicacion serial por el puerto definido anteriormente
59
60 - [y, Fs] = audioread('Avisol.wav'); % Tono de aviso para informar de la conexión serial exitosa
61 - sound(y,Fs) ;
62
63 - start(handles.Timer1); % Inicializacion del temporizador utilizado
64

```

```

65 - % Funcion que espera a recibir una 'I' para iniciar la transmision
66 - fprintf('Esperando para recibir datos %s\n',handles.Serial_Port);
67 - Start = ' ';
68 - while Start ~= 'I'
69 -     while Serial_Com.BytesAvailable == 0
70 -     end
71 -     Start = fscanf(Serial_Com);
72 - end
73
74 - guidata(hObject, handles);
75 - end
76
77 - %% Función ejecutada al presionar el botón "Desconectar" en la interfaz
78 - function pushbutton3_Callback(hObject, eventdata, handles)
79
80 - global Serial_Com;
81 - stop(handles.Timer1); % Finalizar el funcionamiento del temporizador
82 - fclose(Serial_Com); % Finalizar la comunicación serial
83
84 - % Almacenado de los vectores de datos para cada indicador de comparación
85 - global dato1;
86 - global dato2;
87 - global dato3;
88 - global dato4;
89 - datos = [dato1, dato2,dato3,dato4];
90 - save('datos.mat','datos');
91
92 - [y, Fs] = audioread('Aviso2.wav'); % Tono de aviso para indicar la finalización de la comunicación serial
93 - sound(y,Fs) ;
94
95 - disp('Programa se ha detenido');
96 - end

```

```

98 %% Función realizada cada tiempo de muestreo
99 function timer_callback(Obj, handles, hObject, ~)
100 global Serial_Com ;
101 persistent Volta_Buffer Corrien_Buffer Pot_Buffer Error_Buffer Control_Buffer HPlot1 HPlot3 HPlot2 HPlot4 HPlot5 ...
102 Time_Data Sampling_Time Buffer_Size;
103
104 handles = guidata(hObject);
105
106 if isempty(Time_Data)
107     % Definición de las variables utilizadas dentro de la función
108     Sampling_Time = 60E-3;
109     Buffer_Size = 250;
110     Time_Data = (0:1:Buffer_Size-1)';
111
112     % Definición de vectores para los indicadores de comparación
113     Volta_Buffer = NaN(Buffer_Size,1);
114     Corrien_Buffer = NaN(Buffer_Size,1);
115     Pot_Buffer = NaN(Buffer_Size,1);
116     Error_Buffer = NaN(Buffer_Size,1);
117     Control_Buffer = NaN(Buffer_Size,1);
118
119     % Grafica de las señales obtenidas para cada indicador
120     % Señales de corriente y voltaje
121     axes(handles.axes1);
122     hold on
123     HPlot2 = plot(Time_Data * Sampling_Time, Volta_Buffer, 'b-', 'LineWidth', 2);
124     HPlot5 = plot(Time_Data * Sampling_Time, Corrien_Buffer, 'c-', 'LineWidth', 2);
125     axis([-inf, inf, 0, 7]);
126     xlabel('Tiempo [s]');
127     ylabel('Voltaje [V] - Corriente [A]');
128     grid on;
129     hold off
130
131     % Señal de potencia electrica
132     HPlot1 = plot(handles.axes2, Time_Data * Sampling_Time, Pot_Buffer, 'r-', 'LineWidth', 2);
133     axes(handles.axes2);
134     axis([-inf, inf, 0, 8]);
135     xlabel('Tiempo [s]');
136     ylabel('Potencia [Watts]');
137     grid on;
138
139     % Señal de Error Acumulado
140     HPlot3 = plot(handles.axes3, Time_Data * Sampling_Time, Error_Buffer, 'g-', 'LineWidth', 2);
141     axes(handles.axes3);
142     axis([-inf, inf, -250, 250]);
143     xlabel('Tiempo [s]');
144     ylabel('Error acumulado');
145     grid on;
146
147     % Señal de Esfuerzo de Control
148     HPlot4 = plot(handles.axes4, Time_Data * Sampling_Time, Control_Buffer, 'k-', 'LineWidth', 2);
149     axes(handles.axes4);
150     xlabel('Tiempo [s]');
151     axis([-inf, inf, 162, 674]);
152     ylabel('Esfuerzo de Control');
153     grid on;
154
155 else
156     if Serial_Com.BytesAvailable > 0
157         % Llamado a la función encargada de la lectura de datos por medio del puerto serial
158         [Vector1, Vector2, Vector3, Vector4] = reads_n_integers_from_serial_port(Serial_Com);
159
160         % Llenado de los vectores de datos definidos para cada indicador
161         N = length(Vector1);
162         N = min(N, Buffer_Size);
163         Vector1 = Vector1(1:N);

```

18ANEXOS D. CÓDIGO DESARROLLADO EN MATLAB PARA EL DISEÑO DE LA INTERFAZ GRÁFICA UTILIZ

```

164 -         Vector2      = -Vector2(1:N);
165 -         Vector3      = Vector3(1:N);
166 -         Vector4      = Vector4(1:N);
167 -         Time_Data    = Time_Data + N;
168 -         Volta_Buffer  = [Volta_Buffer(N+1:end); Vector1];
169 -         Corrien_Buffer = [Corrien_Buffer(N+1:end); Vector2];
170 -         Pot_New      = Vector1 .* Vector2; % Obtención de la señal de potencia
171 -         Pot_Buffer   = [Pot_Buffer(N+1:end); Pot_New];
172 -         Error_Buffer  = [Error_Buffer(N+1:end); Vector3];
173 -         Control_Buffer = [Control_Buffer(N+1:end); Vector4];
174
175 -         % Actualización de las señales de datos para cada uno de los gráficos
176 -         set(HPlot2,'XData',Time_Data * Sampling_Time);
177 -         set(HPlot2,'YData',Volta_Buffer);
178
179 -         set(HPlot5,'XData',Time_Data * Sampling_Time);
180 -         set(HPlot5,'YData',Corrien_Buffer);
181
182 -         set(HPlot1,'XData',Time_Data * Sampling_Time);
183 -         set(HPlot1,'YData',Pot_Buffer);
184
185 -         set(HPlot3,'XData',Time_Data * Sampling_Time);
186 -         set(HPlot3,'YData',Error_Buffer);
187
188 -         set(HPlot4,'XData',Time_Data * Sampling_Time);
189 -         set(HPlot4,'YData',Control_Buffer);
190
191 -     end
192 - end
193 - guidata(hObject, handles);
194 - end
195

```

```

196 - %% Funcion encargada de la lectura de datos recibidos por el puerto serial
197 - function [Vector1, Vector2, Vector3, Vector4] = reads_n_integers_from_serial_port(Serial_Com)
198
199 -     % Funciones utilizadas para el almacenamiento de los vectores de datos
200 -     global dato1; % Vector de Corriente
201 -     global dato2; % Vector de Voltaje
202 -     global dato3; % Vector de Error Acumulado
203 -     global dato4; % Vector de Esfuerzo de Control
204
205 -     % Vectores utilizados para almacenar los datos recibidos por el puerto serial
206 -     Vector1 = zeros(1000,1); % Vector de Corriente
207 -     Vector2 = zeros(1000,1); % Vector de Voltaje
208 -     Vector3 = zeros(1000,1); % Vector de Error Acumulado
209 -     Vector4 = zeros(1000,1); % Vector de Esfuerzo de Control
210 -     Count = 0;
211
212 -     % Lectura de puerto serial
213 -     while Serial_Com.BytesAvailable > 0
214 -         Count = Count + 1;
215 -         String = fscanf(Serial_Com);
216 -         Vector1(Count) = (str2double(String));
217 -         String = fscanf(Serial_Com);
218 -         Vector2(Count) = (str2double(String));
219 -         String = fscanf(Serial_Com);
220 -         Vector3(Count) = (str2double(String));
221 -         String = fscanf(Serial_Com);
222 -         Vector4(Count) = (str2double(String));
223
224 -     end
225
226 -     %Llenado de los vectores de datos recibidos
227 -     Vector1 = Vector1(1:Count);
228 -     Vector2 = Vector2(1:Count);

```

```
229 - Vector3 = Vector3(1:Count);
230 - Vector4 = Vector4(1:Count);
231
232 % Llenado de los vectores de almacenamiento para las señales de cada indicador
233 - dato1 = [dato1 ; Vector1];
234 - dato2 = [dato2 ; Vector2];
235 - dato3 = [dato3 ; Vector3];
236 - dato4 = [dato4 ; Vector4];
237 - end
```

Anexos E

Código empleado en MatLab para la obtención de los diagramas de cajas y bigotes

```
1- clear all;
2- close all;
3- clc;
4- %% INCLINACION POSITIVA
5- % DATOS ERROR ACUMULADO
6- TEP=[2.8768,2.8157,3.0133,3.0085, 3.0600, 3.0471,2.9490,3.0915, 3.1028,2.5120,2.8181, 2.6465,2.8682,3.0939,2.6928]';
7- CEP = -[-3.2684 -3.0920 -2.9279 -3.2237 -2.8194 -3.1027 -3.1219 -3.0893 -3.5064 -3.0060 -3.4122 -2.7732 -3.1225 -3.1409 -3.2365]';
8- % DATOS POTENCIA
9- TPP = [5.6887, 6.0797, 5.6353, 5.4518, 6.1587, 5.9116, 5.8893, 5.8725, 5.6674, 5.9849, 5.6515, 5.1656, 5.3752, 5.2763 5.6502]';
10- CPP = [5.4847 6.0412 5.3663 5.4025 5.1697 5.6231 5.7000 5.4286 4.9670 5.4434 5.3191 5.4882 5.5107 5.4704 5.5331]';
11- % DATOS ESFUERZO DE CONTROL
12- TECP = [2.9035 2.8264 3.0255 2.9970 3.0943 3.0945 2.9742 3.1027 3.0884 2.5605 2.8570 2.6724 2.8847 3.0503 2.6761]';
13- CECP = -[-3.2520 -3.1134 -2.9384 -3.2297 -2.8208 -3.0717 -3.1139 -3.0881 -3.5017 -3.0028 -3.4120 -2.7934 -3.1205 -3.1429 -3.2244]';
14- %% INCLINACION NEGATIVA
15- % DATOS ERROR ACUMULADO
16- TEN = [-2.8375,-3.0302, -2.6849,-2.4986,-2.6028,-2.4501,-2.3885,-2.3922, -2.4351,-2.6660,-2.4135,-2.2456,-2.3884,-2.2245, -2.3939]';
17- CEN = [3.0691 2.7460 3.0465 2.8962 2.8462 2.9221 2.6916 3.0941 2.5722 2.8695 3.0969 2.7388 2.8952 2.8628 2.9773]';
18- % DATOS POTENCIA
19- TPN = [4.9949 5.2454 4.5752 4.3705 4.7493 4.7641 4.5304 4.6297 4.8647 5.1029 5.2826 5.2019 4.861 5.2924 5.3345]';
20- CPN = [5.0544 5.1984 4.9670 5.1605 5.0507 5.1594 5.1123 5.2187 5.4478 5.1635 5.4533 4.8522 5.3650 5.4862 5.0269]';
21- % DATOS ESFUERZO DE CONTROL
22- TECN = [-2.8668 -2.9948 -2.7278 -2.5330 -2.6446 -2.4737 -2.4149 -2.4180 -2.4898 -2.6866 -2.4438 -2.2720 -2.4150 -2.2400 -2.4248]';
23- CECN = [3.1072 2.7711 3.0106 2.9066 2.8397 2.9143 2.7033 3.1018 2.5702 2.8637 3.0837 2.7631 2.8924 2.8555 2.9753]';
24- %% DATOS MÁXIMO ÁNGULO DE INCLINACIÓN
25- MAIP = [44 45 44 43 46 44 44 44 43 46;43 41 45 44 41 41 43 41 41 41]';
26- MAIN = [49 49 47 47 48 47 46 47 46 45;48 46 45 45 46 46 44 45 43 44]';
27- %% TOTAL POSITIVO
28- ETP = [TEP,CEP];
29- PTP = [TPP,CPP];
30- ECTP = [TECP,CECP];
```

```

31     %% TOTAL NEGATIVO
32     ETN = [TEN,CEN];
33     PTN = [TPN,CPN];
34     ECTN = [TECN,CECN];
35     M = [TEP,TECP];
36     % boxplot(x,y)
37     %% Comparación de estrategias en Error Acumulado con inclinación positiva
38     boxplot(ETP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
39     xlabel('Estrategias')
40     ylabel('Pendiente')
41     title('COMPARACIÓN DEL ERROR ACUMULADO CON INCLINACIÓN POSITIVA')
42     grid on
43     %% Comparación de estrategias en Error Acumulado con inclinación Negativa
44     boxplot(ETN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
45     xlabel('Estrategias')
46     ylabel('Pendiente')
47     title('COMPARACIÓN DEL ERROR ACUMULADO CON INCLINACIÓN NEGATIVA')
48     grid on
49     %% Comparación de estrategias en consumo de energía con inclinación positiva
50     boxplot(PTP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
51     xlabel('Estrategias')
52     ylabel('Pendiente')
53     title('COMPARACIÓN DEL CONSUMO DE ENERGIA CON INCLINACIÓN POSITIVA')
54     grid on
55     %% Comparación de estrategias en consumo de energía con inclinación Negativa
56     boxplot(PTN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
57     xlabel('Estrategias')
58     ylabel('Pendiente')
59     title('COMPARACIÓN DEL CONSUMO DE ENERGIA CON INCLINACIÓN NEGATIVA')
60     grid on

61     %% Comparación de estrategias esfuerzo de control con inclinación positiva
62     boxplot(ECTP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
63     xlabel('Estrategias')
64     ylabel('Pendiente')
65     title('COMPARACIÓN DEL ESFUERZO DE CONTROL CON INCLINACIÓN POSITIVA')
66     grid on
67     %% Comparación de estrategias esfuerzo de control con inclinación Negativa
68     boxplot(ECTN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
69     xlabel('Estrategias')
70     ylabel('Pendiente')
71     title('COMPARACIÓN DEL ESFUERZO DE CONTROL CON INCLINACIÓN NEGATIVA')
72     grid on
73     %% Comparación de máximo ángulo de inclinación con inclinación Positiva
74     boxplot(MAIP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
75     xlabel('Estrategias')
76     ylabel('Ángulo')
77     title('COMPARACIÓN DEL MÁXIMO ÁNGULO DE INCLINACIÓN CON INCLINACIÓN POSITIVA')
78     grid on
79     %% Comparación de máximo ángulo de inclinación con inclinación Negativa
80     boxplot(MAIN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
81     xlabel('Estrategias')
82     ylabel('Ángulo')
83     title('COMPARACIÓN DEL MÁXIMO ÁNGULO DE INCLINACIÓN CON INCLINACIÓN NEGATIVA')
84     grid on

```