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 *arduino* para la estrategia de tobillo 6

C. Código diseñado en la plataforma *arduino* para la estrategia de cadera 10

D. Código desarrollado en *Matlab* para el diseño de la interfaz gráfica utilizada para la recolección de datos 14

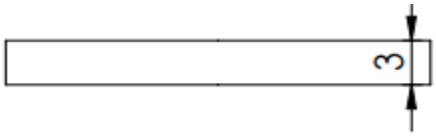
E. Código empleado en MatLab para la obtención de los diagramas de cajas y bigotes 20

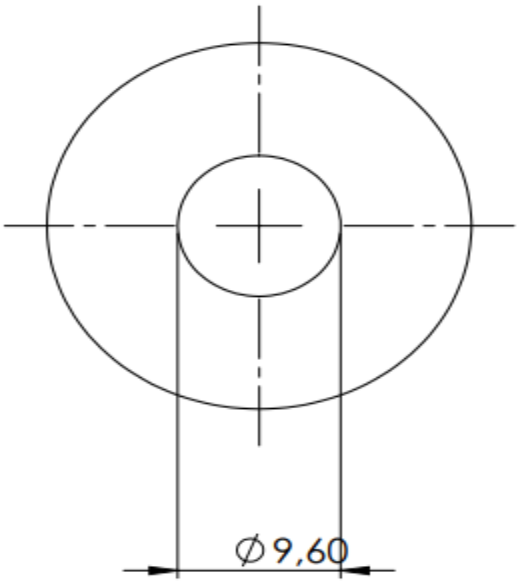
Anexos

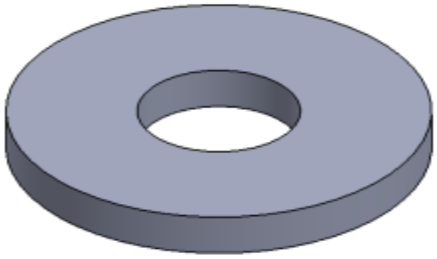
Anexos A

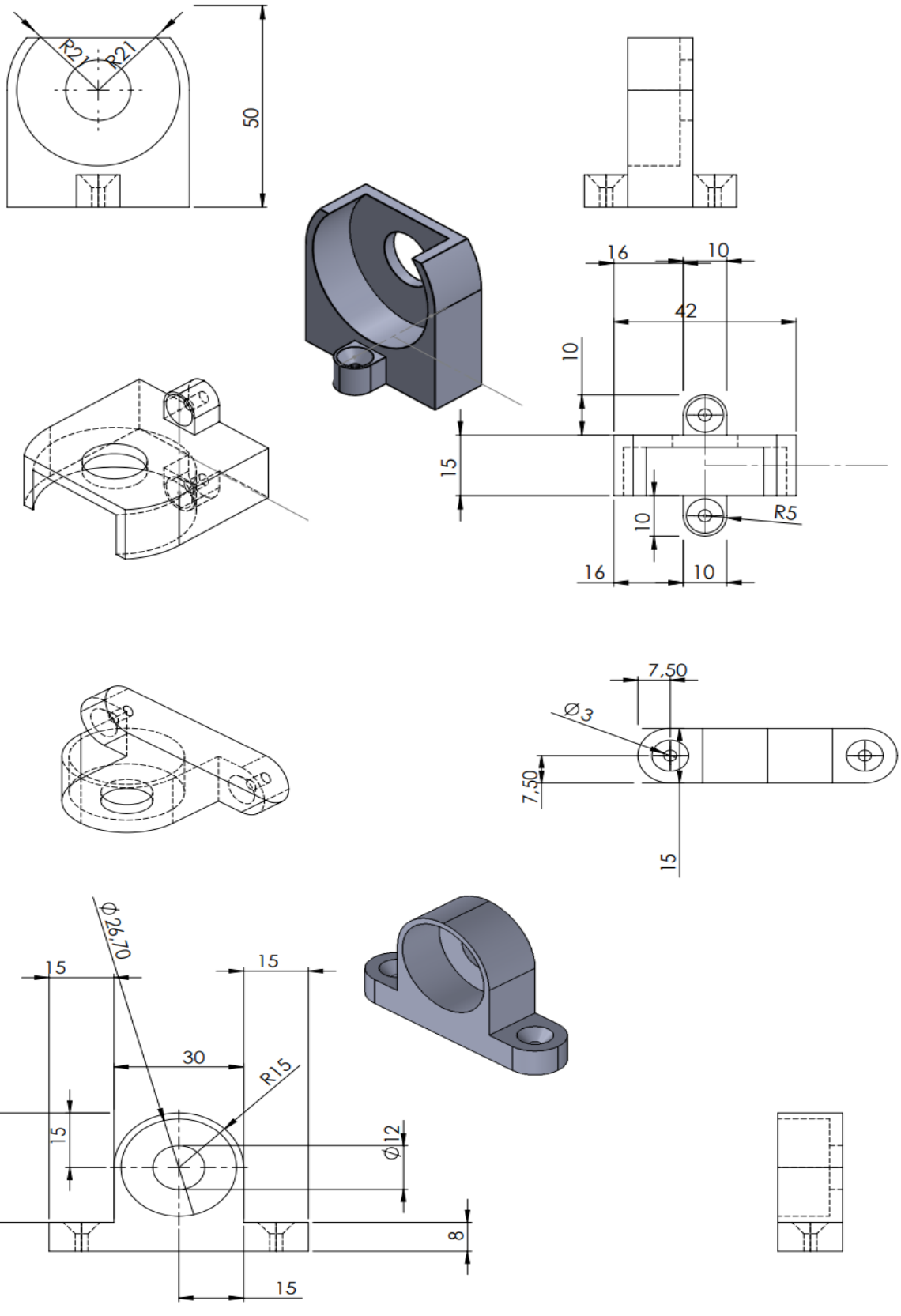
Modelos 3D usados en la base de soporte inclinable

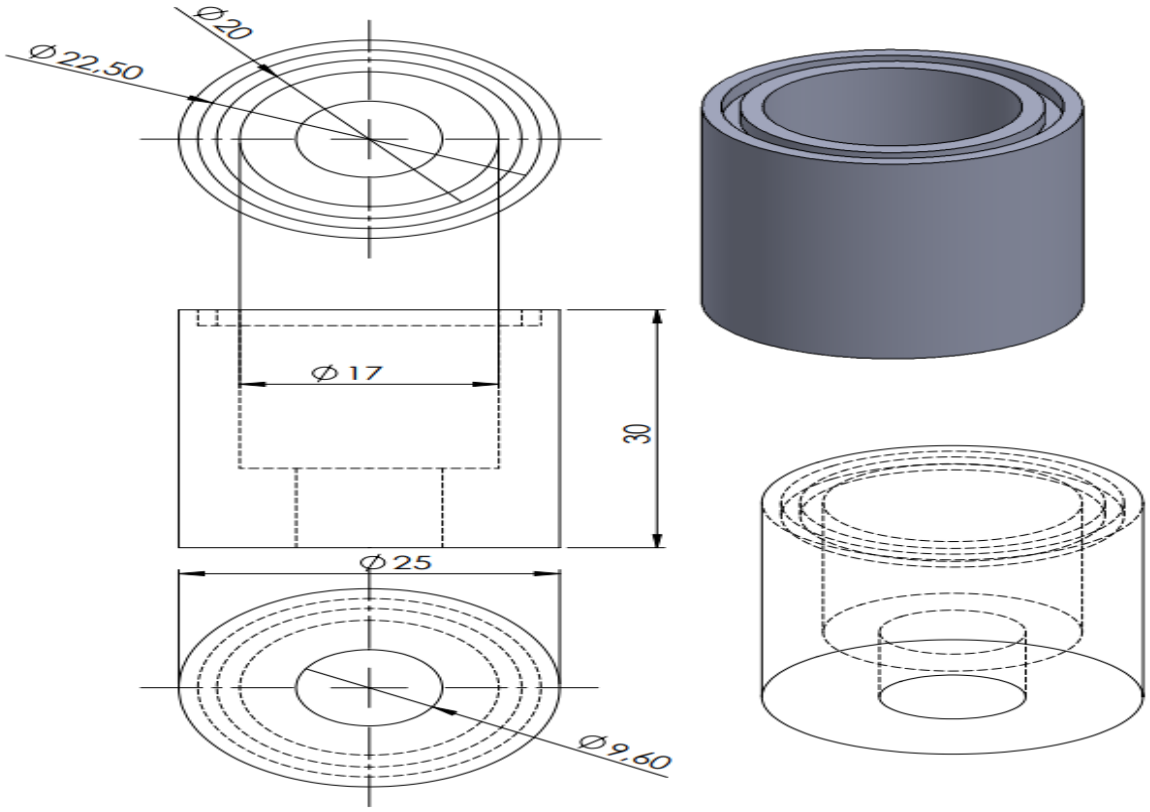
A continuación se presentan diversas vistas de las piezas empleadas para la construcción de la base de soporte inclinable; Para la correcta comprensión de las dimensiones es necesario tener en cuenta que la unidad de medida se encuentran en mm.

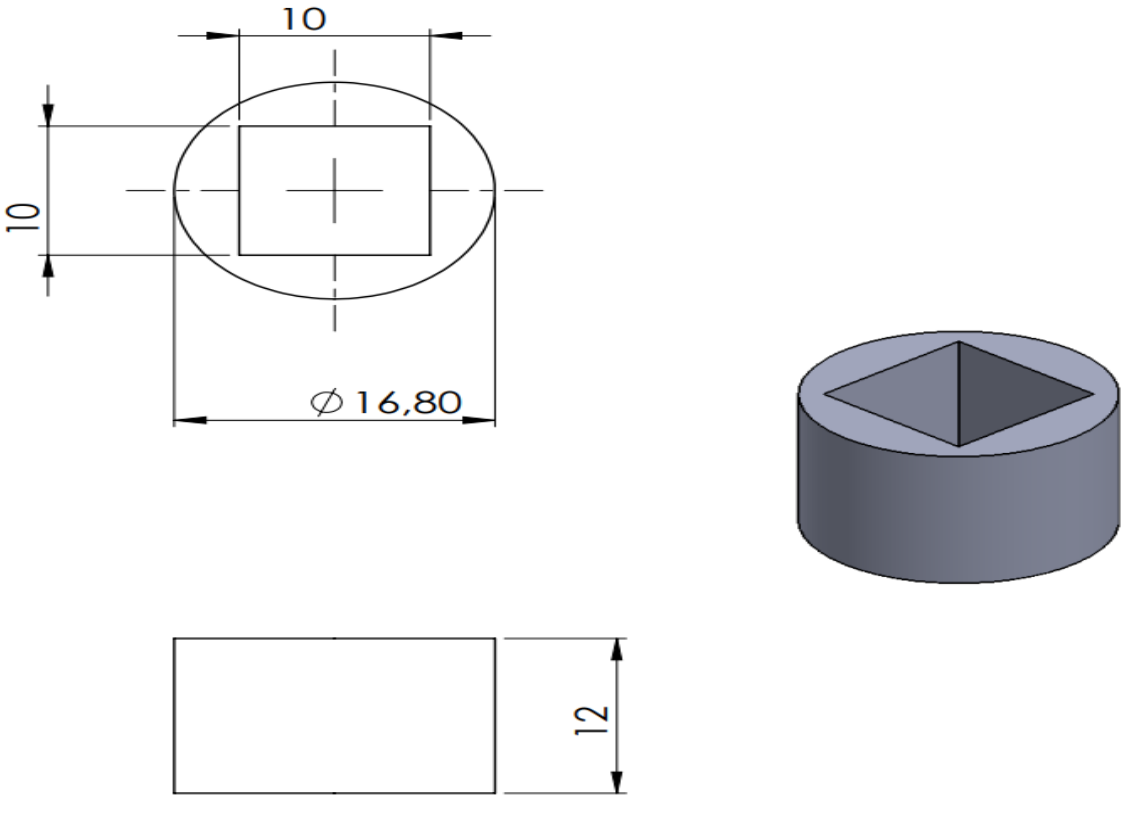


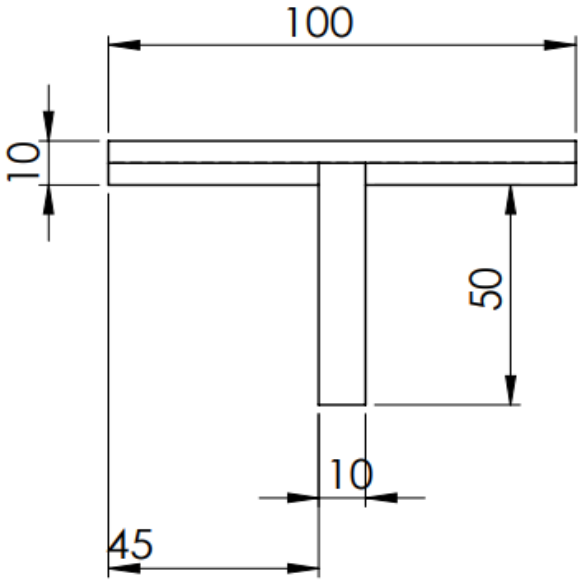


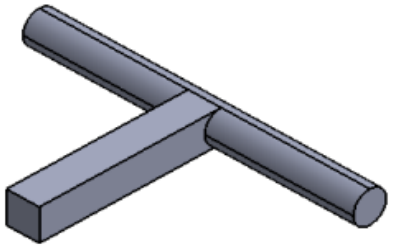


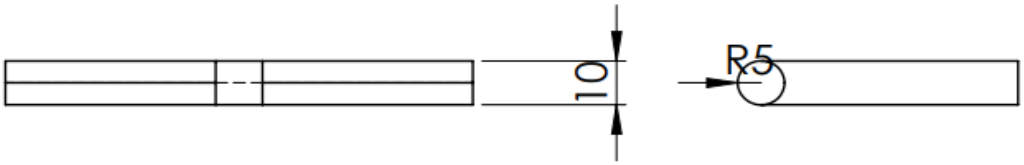

R21
R21
50
16
10
42
10
15
10
R5
16
10
7,50
Ø3
7,50
15
Ø26,70
15
15
30
R15
15
30
Ø12
8
15

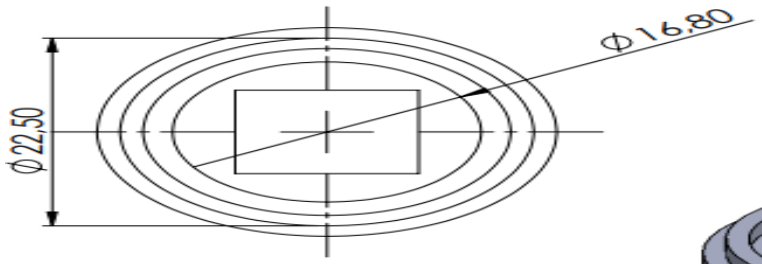

Ø20
Ø22,50
Ø17
30
Ø25
Ø9,60

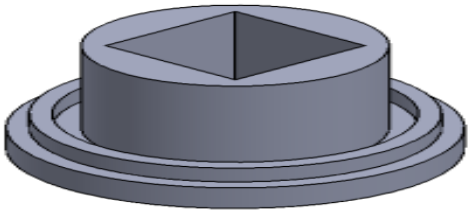

10
10
Ø16,80
12

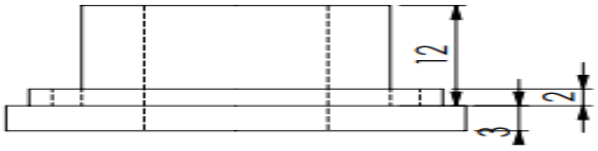

100
10
50
10
45

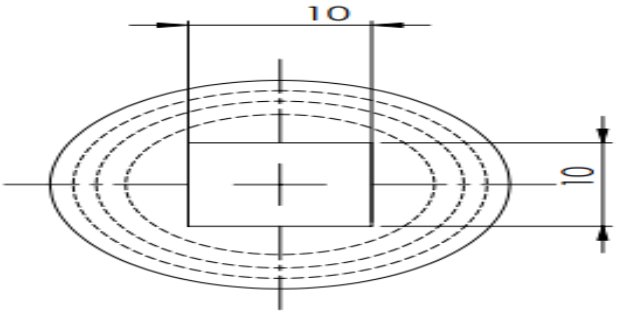


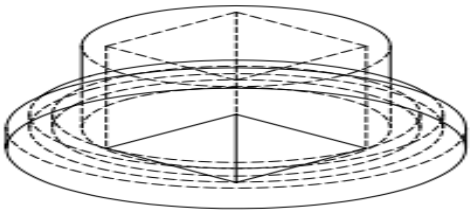

10
R5


Ø16,80
Ø22,50




12
2
3


10
10



Anexos B

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

```
#include <SoftwareSerial.h>
#include <DynamixelSerial.h>
SoftwareSerial Blue(9, 10); // RX | TX

// Funciones
float  Read_Process_Output(int);
float  Read_Setpoint(void);
float  Control_Law_X(float, float);
void   Write_Control_Law(int);
void   Wait_Until_Next_Sampling_Time(unsigned long int,unsigned int);

// Variables
float Error = 0.0, ErrorAnt = 0.0, ErrorAcum=0.0, OutputAnt =0.0, Kp=-0.15, Kd= -2.2, Ki= -0.0002, Offset = 606;
float ZMP_X, Uproporcional=0.0, Uderivativo = 0.0, Uintegral=0.0;
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
unsigned long AnteriorMillis=0.0, ActualMillis=0.0;
float data[16], DS[16];
const float K[] =  {616.8311, 572.4295, 677.6467, 550.4752, 195.0131, 24.2161} ; // Constantes del polinomio de caracterización del sensor FSR

// Constantes del multiplexor
const int muxSIG = A0;
const int muxS0 = 12;
const int muxS1 = 11;
const int muxS2 = 8;
const int muxS3 = 7;

//Configuración multiplexor
int SetMuxChannel(byte channel)
{
    digitalWrite(muxS0, bitRead(channel, 0));
    digitalWrite(muxS1, bitRead(channel, 1));
```

```
    digitalWrite(muxS2, bitRead(channel, 2));
    digitalWrite(muxS3, bitRead(channel, 3));
}

void setup() {
  Blue.begin(115200);
  delay(1000);
  Blue.println("I");

  //Configuración multiplexor
  pinMode(muxS0, OUTPUT);
  pinMode(muxS1, OUTPUT);
  pinMode(muxS2, OUTPUT);
  pinMode(muxS3, OUTPUT);

  Dynamixel.setSerial(&Serial);         // &Serial - Arduino UNO/NANO/MICRO, &Serial1, &Serial2, &Serial3 - Arduino Mega
  Dynamixel.begin(1000000,2);           // Inicializar el servo a 1 Mbps y un pin de control en la salida digital 2

  //Configuracion de torque para los motores del robot
  Dynamixel.setMaxTorque(15,1023);
  Dynamixel.setMaxTorque(16,1023);
  Dynamixel.setMaxTorque(1,200);
  Dynamixel.setMaxTorque(2,200);
  Dynamixel.setMaxTorque(3,200);
  Dynamixel.setMaxTorque(4,200);
  Dynamixel.setMaxTorque(5,200);
  Dynamixel.setMaxTorque(6,200);
  delay(100);


  //Posición inicial de los motores del robot
  //Motores del tren superior (Brazos, hombros, codos)
  Dynamixel.move(1,200);
  Dynamixel.move(2,824);
  Dynamixel.move(3,324);
  Dynamixel.move(4,790);
  Dynamixel.move(5,490);
  Dynamixel.move(6,534);

  //Motores del tren inferior (Tobillos, cadera, rodillas)
  Dynamixel.move(7,360);
  Dynamixel.move(8,664);
  Dynamixel.move(9,500);
  Dynamixel.move(10,512);
  Dynamixel.move(11,418);
  Dynamixel.move(12,606);
  Dynamixel.move(13,324);
  Dynamixel.move(14,700);
  Dynamixel.move(15,606);
  Dynamixel.move(16,418);
  Dynamixel.move(17,512);
  Dynamixel.move(18,512);
  delay(5000);
}

void loop() {

  float Reference, Control_X,ZMP;
  ActualMillis = millis();

  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

```
  Reference = Read_Setpoint();                          // Función que establece el valor de referencia
  Control_X   = Control_Law_X(Reference,ZMP);           // Función de la ley de control
  Write_Control_Law(Control_X);                         // Función encargada de enviar el esfuerzo de control a los actuadores

  //Envio de los valores para cada indicador de comparación
  Blue.println(data[6]);
  Blue.println(data[15]);
  Blue.println(ErrorAcum);
  Blue.println(Control_X);

  Wait_Until_Next_Sampling_Time(ActualMillis,Tm);       // Función que garantiza el cumplimiento del tiempo de muestreo en cada ciclo de control
}

//Polinomio de caractrizacion del sensor FSR
float pol5(float x){
  x=(x-492.292682926829)/152.200072914312;
  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];
  return masa;
}

//Lectura de la variable de proceso
float Read_Process_Output() {

  //Lectura de las entradas del multiplexor
  for (byte i = 6; i < 16; i++)
  {
      SetMuxChannel(i);
      DS[i] = analogRead(muxSIG);


  switch (i)
  {
  //Cálculo del voltaje entregado por el sensor
  case 6:
      {
      float voltajeSensor = DS[6] * (5.0 / 1023.0);
      data[6]=voltajeSensor;
      }
  break;
  //Cálculo de la corriente entregada por el sensor
  case 15:
      {
      float total = 0;
        for(int j=0; j <= 10; j++){
          float corriente=0;
          float voltaje=DS[15] * (5.0 / 1023.0);
          corriente = -5.3094*voltaje + 13.1731;
          total = total + corriente;
          DS[15] = analogRead(muxSIG);
          }
        float   media = total / 10;
        data[15] = media;
      }
  break;
  // Calculo de masa para cada sensor FSR
  default:
       {
       data[i] = pol5(DS[i]);
         if(data[i]<50){
          data[i] = 0;
          }
```

```
      }
  break;
  }
}

 //Cálculo del ZMP
 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]);
}

//Asignación del valor para la señal de referencia
float Read_Setpoint(void) {
  static float Reference = 0;
  return Reference;
}

// Funcion para la ley de control
float Control_Law_X(float Reference, float Output){

  int Control ;
  Error = Reference - Output;
  Uproporcional = Offset + ( Kp * Error);
  Uderivativo   = Kd*((Output - OutputAnt)/Tm);
  ErrorAcum = ErrorAcum + (Ki*((Tm/2)*(ErrorAnt + Error)));
  Uintegral = ErrorAcum;
  Control = Uproporcional - Uderivativo + Uintegral;
  Control = constrain(Control,0,1023);
  ErrorAnt  = Error;
  OutputAnt = Output;
  return Control;

}
```

```

// Envio del esfuerzo de control a los actuadores
void Write_Control_Law(int Control) {

  int Control_15 = Control;
  int Control_16 = 1023-Control;
  Dynamixel.moveSpeed(15,Control_15,100);
  Dynamixel.moveSpeed(16,Control_16,100);
}

// Funcion que asegura que el tiempo de muestreo siempre sea de 40ms
void Wait_Until_Next_Sampling_Time(unsigned long int TBegin, unsigned int Ts)
{
  while ((millis() - ActualMillis) < Tm);
}
```

Anexos C

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

```
#include <SoftwareSerial.h>
#include <DynamixelSerial.h>
SoftwareSerial Blue(9, 10); // RX | TX

// Funciones
float  Read_Process_Output(int);
float  Read_Setpoint(void);
float  Control_Law_X(float, float);
void   Write_Control_Law(int);
void   Wait_Until_Next_Sampling_Time(unsigned long int,unsigned int);

// Variables
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;
float ZMP_X, Uproporcional = 0.0, Uderivativo = 0.0, Uintegral = 0.0;
int Control1 = 0, Control2 = 0, ControlHip1, ControlHip2;
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
unsigned long AnteriorMillis=0.0, ActualMillis=0.0;
float data[16], DS[16];
const float K[] =  {616.8311, 572.4295, 677.6467, 550.4752, 195.0131, 24.2161} ; // Constantes del polinomio de caracterización del sensor FSR

// Constantes del multiplexor
const int muxSIG = A0;
const int muxS0 = 12;
const int muxS1 = 11;
const int muxS2 = 8;
const int muxS3 = 7;

//Configuración multiplexor
int SetMuxChannel(byte channel)
{
    digitalWrite(muxS0, bitRead(channel, 0));
```

```
    digitalWrite(muxS1, bitRead(channel, 1));
    digitalWrite(muxS2, bitRead(channel, 2));
    digitalWrite(muxS3, bitRead(channel, 3));
}

void setup() {
  Blue.begin(115200);
  delay(1000);
  Blue.println("I");

  //Configuración multiplexor
  pinMode(muxS0, OUTPUT);
  pinMode(muxS1, OUTPUT);
  pinMode(muxS2, OUTPUT);
  pinMode(muxS3, OUTPUT);

  Dynamixel.setSerial(&Serial);           // &Serial - Arduino UNO/NANO/MICRO, &Serial1, &Serial2, &Serial3 - Arduino Mega
  Dynamixel.begin(1000000, 2);            // Inicializar el servo a 1 Mbps y un pin de control en la salida digital 2

  //Configuracion de torque para los motores del robot
  Dynamixel.setMaxTorque(15,1023);
  Dynamixel.setMaxTorque(16,1023);
  Dynamixel.setMaxTorque(11,1023);
  Dynamixel.setMaxTorque(12,1023);
  Dynamixel.setMaxTorque(1,200);
  Dynamixel.setMaxTorque(2,200);
  Dynamixel.setMaxTorque(3,200);
  Dynamixel.setMaxTorque(4,200);
  Dynamixel.setMaxTorque(5,200);
  Dynamixel.setMaxTorque(6,200);
  delay(100);


  //Posición inicial de los motores del robot
  //Motores del tren superior (Brazos, hombros, codos)
  Dynamixel.move(1,200);
  Dynamixel.move(2,824);
  Dynamixel.move(3,324);
  Dynamixel.move(4,790);
  Dynamixel.move(5,490);
  Dynamixel.move(6,534);

  //Motores del tren inferior (Tobillos, cadera, rodillas)
  Dynamixel.move(7,360);
  Dynamixel.move(8,664);
  Dynamixel.move(9,500);
  Dynamixel.move(10,512);
  Dynamixel.move(11,418);
  Dynamixel.move(12,606);
  Dynamixel.move(13,324);
  Dynamixel.move(14,700);
  Dynamixel.move(15,614);
  Dynamixel.move(16,410);
  Dynamixel.move(17,512);
  Dynamixel.move(18,512);
  delay(5000);
}

void loop() {
  float Reference, Control_X,ZMP;

  ActualMillis = millis();

  ZMP=Read_Process_Output();                          // Función de lectura de la variable salida del  proceso
```

```
  Reference = Read_Setpoint();                          // Función que establece el valor de referencia
  Control_X   = Control_Law_X(Reference,ZMP);           // Función de la ley de control
  Write_Control_Law(Control_X);                         // Función encargada de enviar el esfuerzo de control a los actuadores

  //Envio de los valores para cada indicador de comparación
  Blue.println(data[6]);
  Blue.println(data[15]);
  Blue.println(ErrorAcum);
  Blue.println(Control1);

  Wait_Until_Next_Sampling_Time(ActualMillis,Tm);       // Función que garantiza el cumplimiento del tiempo de muestreo en cada ciclo de control
}

//Polinomio de caractrizacion del sensor FSR
float pol5(float x){
  x=(x-492.292682926829)/152.200072914312;
  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];
  return masa;
}

//Lectura de la variable de proceso
float Read_Process_Output() {

  //Lectura de las entradas del multiplexor
  for (byte i = 6; i < 16; i++)
  {
      SetMuxChannel(i);
      DS[i] = analogRead(muxSIG);


  switch (i)
  {
  //Cálculo del voltaje entregado por el sensor
  case 6:
      {
      float voltajeSensor = DS[6] * (5.0 / 1023.0);
      data[6]=voltajeSensor;
      }
  break;
  //Cálculo de la corriente entregada por el sensor
  case 15:
      {
      float total = 0;
        for(int j=0; j <= 10; j++){
          float corriente=0;
          float voltaje=DS[15] * (5.0 / 1023.0);
          corriente = -5.3094*voltaje + 13.1731;
          total = total + corriente;
          DS[15] = analogRead(muxSIG);
          }
        float   media = total / 10;
        data[15] = media;
      }
  break;
  // Calculo de masa para cada sensor FSR
  default:
       {
       data[i] = pol5(DS[i]);
         if(data[i]<50){
          data[i] = 0;
          }
```

```
      }
  break;
  }
}

//Cálculo del ZMP
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]);
}

//Asignación del valor para la señal de referencia
float Read_Setpoint(void) {                                  //retorna el valor de la referencia o set point decontrol
  static float Reference = 0;
  return Reference;
}

// Funcion para la ley de control
float Control_Law_X(float Reference, float Output){

  int Control ;
  Error = Reference - Output;
  Uproporcional = ( Kp * Error);
  Uderivativo   = Kd*((Output - OutputAnt)/Tm);
  ErrorAcum = ErrorAcum + (Ki*((Tm/2)*(ErrorAnt + Error)));
  Uintegral = ErrorAcum;
  Control = Uproporcional - Uderivativo + Uintegral;
  ErrorAnt  = Error;
  OutputAnt = Output;
  return Control;

}
```

```
// Envio del esfuerzo de control a los actuadores
void Write_Control_Law(float Control) {
  float Khip=-(5.3/17.7);
  Control1 = Offset1+Control;
  Control1 = constrain(Control1,0,1023);
  Control2 = 1023-(Offset1+Control);
  Control2 = constrain(Control2,0,1023);
  ControlHip1 = Offset2+(Khip*Control);
  ControlHip1 = constrain(ControlHip1,0,1023);
  ControlHip2 = 1023-(Offset2+(Khip*Control));
  ControlHip2 = constrain(ControlHip2,0,1023);
  Dynamixel.moveSpeed(15,Control1,100);
  Dynamixel.moveSpeed(16,Control2,100);
  Dynamixel.moveSpeed(11,ControlHip2,100);
  Dynamixel.moveSpeed(12,ControlHip1,100);
}

// Funcion que asegura que el tiempo de muestreo siempre sea de 60ms
void Wait_Until_Next_Sampling_Time(unsigned long int TBegin, unsigned int Ts)
{
  while ((millis() - ActualMillis) < Tm);
}
```

Anexos D

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

```
function varargout = Interfaz_tesis(varargin)
%% Codigo de inicialización - NO EDITAR
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_tesis_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_tesis_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

end

%% Funcion de inicialización
function Interfaz_tesis_OpeningFcn(hObject, eventdata, handles, varargin)

escudo = imread('EscudoUnicaucal.jpg');
axes(handles.axes5);
image(escudo);
axis off;

clear instrfind;
delete(instrfind);                              % Elimina objetos de comunicacion pasados
delete(timerfind);                              % Elimina objetos temporizaodres usados antes
```

```
handles.Timer1      = timer(...                         %Definición del temporizador utilizado
                'ExecutionMode', 'fixedrate',...
                'Period',.2,...
                'TimerFcn',{@timer_callback,hObject});

handles.output = hObject;                               % Elija la salida de línea de comando predeterminada

guidata(hObject, handles);                              % Actualiza la estructura handles
end

%% Las salidas de esta función son retornadas a la linea de comando
function varargout = Interfaz_tesis_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;                          % Obtener la salida de la línea de comando predeterminada
                                                        % desde la estructura handles
end

%% Función ejecutada al presionar el botón "Conectar" en la interfaz
function pushbutton1_Callback(hObject, eventdata, handles)

global Serial_Com;                                      % Define variable global para ser utilizada por cualquier subfuncion
handles.Serial_Port = 'COM6';                           % Cambiar segun el puerto serial utilizado para la comunicación serial
Serial_Com = serial('COM6','BaudRate',115200,'Terminator', 'CR/LF');

fopen(Serial_Com);                                      % Inicia la comunicacion serial por el puerto definido anteriormente

[y, Fs] = audioread('Aviso1.wav');                      % Tono de aviso para informar de la conexion serial exitosa
sound(y,Fs) ;

start(handles.Timer1);                                  % Inicializacion del temporizador utilizado

```

```
% Funcion que espera a recibir una 'I' para iniciar la trasmision
fprintf('Esperando para recibir datos %s\n',handles.Serial_Port);
    Start       = ' ';
    while Start ~= 'I',
        while Serial_Com.BytesAvailable == 0,
        end
        Start = fscanf(Serial_Com);
    end

guidata(hObject, handles);
end

%% Función ejectuada al presionar el botón "Desconectar" en la interfaz
function pushbutton3_Callback(hObject, eventdata, handles)

global Serial_Com;
stop(handles.Timer1);                               % Finalizar el funcionamiento del temporizador
fclose(Serial_Com);                                 % Finalizar la comunicación serial

% Almacenado de los vectores de datos para cada indicador de comparación
global dato1;
global dato2;
global dato3;
global dato4;
datos = [dato1, dato2,dato3,dato4];
save('datos.mat','datos');

[y, Fs] = audioread('Aviso2.wav');                  % Tono de aviso para indicar la finalización de la comunicación serial
sound(y,Fs) ;

disp('Programa se ha detenido');
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       % Definicion 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
```

```
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 funcion 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);
```

```
            Vector2         = -Vector2(1:N);
            Vector3         = Vector3(1:N);
            Vector4         = Vector4(1:N);
            Time_Data       = Time_Data + N;
            Volta_Buffer    = [Volta_Buffer(N+1:end); Vector1];
            Corrien_Buffer  = [Corrien_Buffer(N+1:end); Vector2];
            Pot_New         = Vector1 .* Vector2;  % Obtención de la señal de potencia
            Pot_Buffer      = [Pot_Buffer(N+1:end); Pot_New];
            Error_Buffer    = [Error_Buffer(N+1:end); Vector3];
            Control_Buffer  = [Control_Buffer(N+1:end); Vector4];

            % Actualización de las señales de datos para cada uno de los graficos
            set(HPlot2,'XData',Time_Data * Sampling_Time);
            set(HPlot2,'YData',Volta_Buffer);

            set(HPlot5,'XData',Time_Data * Sampling_Time);
            set(HPlot5,'YData',Corrien_Buffer);

            set(HPlot1,'XData',Time_Data * Sampling_Time);
            set(HPlot1,'YData',Pot_Buffer);

            set(HPlot3,'XData',Time_Data * Sampling_Time);
            set(HPlot3,'YData',Error_Buffer);

            set(HPlot4,'XData',Time_Data * Sampling_Time);
            set(HPlot4,'YData',Control_Buffer);

    end
end
guidata(hObject, handles);
end

```

```
%% Funcion encargada de la lectura de datos recibidos por el puerto serial
function [Vector1, Vector2, Vector3, Vector4] = reads_n_integers_from_serial_port(Serial_Com)

    % Funciones utilizadas para el almacenamiento de los vectores de datos
    global dato1;                               % Vector de Corriente
    global dato2;                               % Vector de Voltaje
    global dato3;                               % Vector de Error Acumulado
    global dato4;                               % Vector de Esfuerzo de Control

    % Vectores utilizados para almacenar los datos recibidos por el puerto serial
    Vector1  = zeros(1000,1);                   % Vector de Corriente
    Vector2  = zeros(1000,1);                   % Vector de Voltaje
    Vector3  = zeros(1000,1);                   % Vector de Error Acumulado
    Vector4  = zeros(1000,1);                   % Vector de Esfuerzo de Control
    Count    = 0;

    % Lectura de puerto serial
    while Serial_Com.BytesAvailable > 0,
        Count           = Count + 1;
        String          = fscanf(Serial_Com);
        Vector1(Count)  = (str2double(String));
        String          = fscanf(Serial_Com);
        Vector2(Count)  = (str2double(String));
        String          = fscanf(Serial_Com);
        Vector3(Count)  = (str2double(String));
        String          = fscanf(Serial_Com);
        Vector4(Count)  = (str2double(String));

    end

    %Llenado de los vectores de datos recibidos
    Vector1 = Vector1(1:Count);
    Vector2 = Vector2(1:Count);
```

```
229 -       Vector3 = Vector3(1:Count);
230 -       Vector4 = Vector4(1:Count);
231
232         % Llenado de los vectores de almanecamiento 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

```
clear all;
close all;
clc;
%% INCLINACION POSITIVA
% DATOS  ERROR ACUMULADO
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]';
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]';
% DATOS POTENCIA
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]';
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]';
% DATOS ESFUERZO DE CONTROL
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]';
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]';
%% INCLINACION NEGATIVA
% DATOS ERROR ACUMULADO
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]';
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]';
% DATOS POTENCIA
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]';
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]';
% DATOS ESFUERZO DE CONTROL
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]';
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]';
%% DATOS MÁXIMO ÁNGULO DE INCLINACIÓN
MAIP = [44 45 44 43 46 44 44 44 43 46;43 41 45 44 41 41 43 41 41 41]';
MAIN = [49 49 47 47 48 47 46 47 46 45;48 46 45 45 46 46 44 45 43 44]';
%% TOTAL POSITIVO
ETP = [TEP,CEP];
PTP = [TPP,CPP];
ECTP = [TECP,CECP];
```

```
%% TOTAL NEGATIVO
ETN = [TEN,CEN];
PTN = [TPN,CPN];
ECTN = [TECN,CECN];
M = [TEP,TECP];
% boxplot(x,y)
%% Comparación de estrategias en Error Acumulado con inclinación positiva
boxplot(ETP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Pendiente')
title('COMPARACIÓN DEL ERROR ACUMULADO CON INCLINACIÓN POSITIVA')
grid on
%% Comparación de estrategias en Error Acumulado con inclinación Negativa
boxplot(ETN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Pendiente')
title('COMPARACIÓN DEL ERROR ACUMULADO CON INCLINACIÓN NEGATIVA')
grid on
%% Comparación de estrategias en consumo de energia con inclinación positiva
boxplot(PTP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Pendiente')
title('COMPARACIÓN DEL CONSUMO DE ENERGIA CON INCLINACIÓN POSITIVA')
grid on
%% Comparación de estrategias en consumo de energia con inclinación Negativa
boxplot(PTN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Pendiente')
title('COMPARACIÓN DEL CONSUMO DE ENERGIA CON INCLINACIÓN NEGATIVA')
grid on
```

```
%% Comparación de estrategias esfuerzo de control con inclinación positiva
boxplot(ECTP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Pendiente')
title('COMPARACIÓN DEL ESFUERZO DE CONTROL CON INCLINACIÓN POSITIVA')
grid on
%% Comparación de estrategias esfuerzo de control con inclinación Negativa
boxplot(ECTN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Pendiente')
title('COMPARACIÓN DEL ESFUERZO DE CONTROL CON INCLINACIÓN NEGATIVA')
grid on
%% Comparación de máximo ángulo de inclinación con inclinación Positiva
boxplot(MAIP,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Ángulo')
title('COMPARACIÓN DEL MÁXIMO ÁNGULO DE INCLINACIÓN CON INCLINACIÓN POSITIVA')
grid on
%% Comparación de máximo ángulo de inclinación con inclinación Negativa
boxplot(MAIN,'Labels',{'Estrategia de Tobillo','Estrategia de Cadera'})
xlabel('Estrategias')
ylabel('Ángulo')
title('COMPARACIÓN DEL MÁXIMO ÁNGULO DE INCLINACIÓN CON INCLINACIÓN NEGATIVA')
grid on
```