Péndulo Vertical

por Daniel Héctor Stolfi Rosso

Contenido

1) Introducción

Este desarrollo consiste en la implementación de un controlador capaz de mantener un péndulo en su posición vertical restringiendo su movimiento a sólo 1 grado de libertad: el ángulo de inclinación. El modelo físico se ha construido mediante una antigua impresora (Figura 1) de la cual se ha aprovechado el sistema de desplazamiento del cabezal de impresión sobre el cual se montó el péndulo y el sensor que devolverá el ángulo de inclinación en forma de un voltaje proporcional.

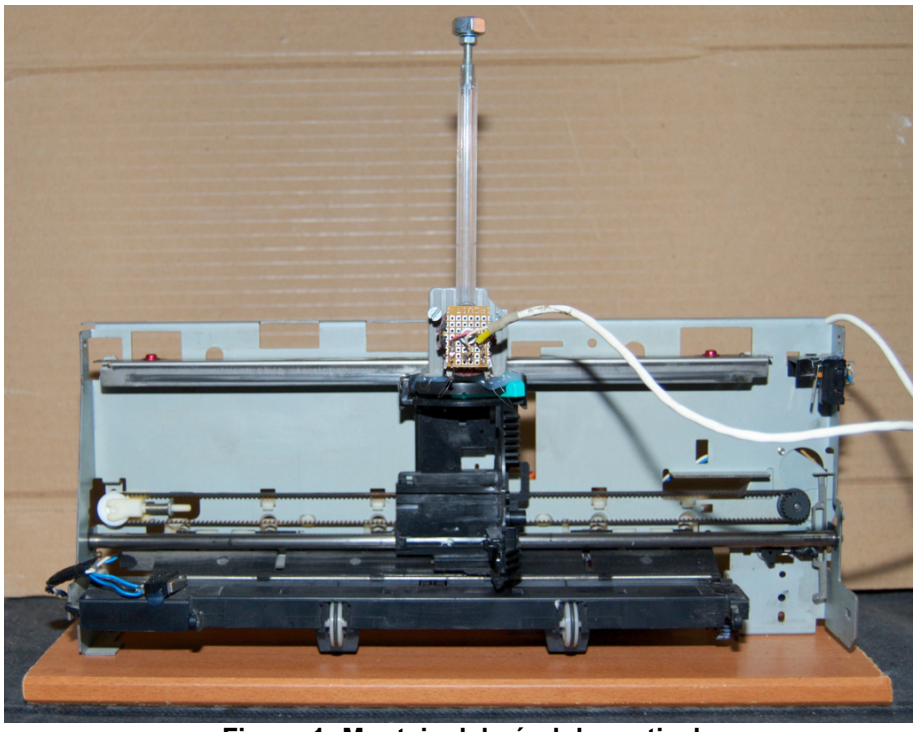

Figura 1: Montaje del péndulo vertical

Luego mediante desplazamientos del cabezal a lo largo del eje **x** el sistema se encarga de evitar

la caída del péndulo corrigiendo el ángulo de inclinación del mismo (Figura 2) mediante movimientos del cabezal de la impresora.

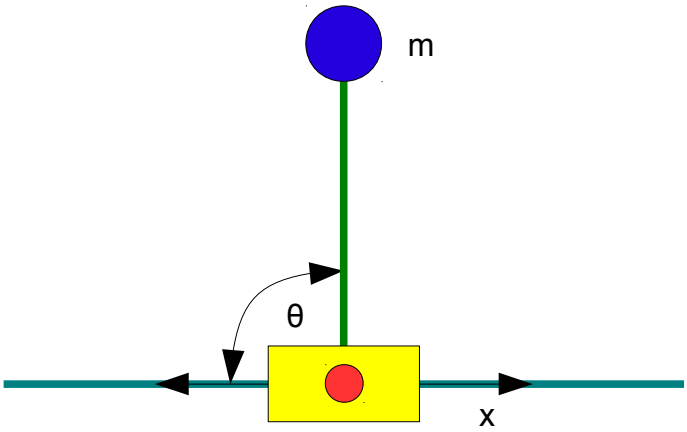

Figura 2: Esquema del péndulo vertical

El sistema tiene como restricción la longitud del raíl de la impresora y la velocidad máxima que puede alcanzar el motor paso a paso que controla la posición del cabezal, sólo siendo capaz de mantener la verticalidad del péndulo para desviaciones muy pequeñas del ángulo **θ**.

El estado inicial del sistema se corresponde con el péndulo en posición vertical para lo cual debe calibrarse el sensor para que el controlador interprete esta posición de forma correcta, luego se iniciará el funcionamiento del sistema y se lo dejará actuar de forma que compense las posibles alteraciones del equilibrio inicial.

Se dispone también de dos sensores de fin de carrera para delimitar la longitud del raíl y evitar las posibles colisiones del carro con los extremos. Por último se ha implementado el volcado de los datos telemétricos correspondientes a la posición del carro, el ángulo de inclinación del péndulo y la acción del controlador para realizar las trazas del comportamiento del sistema.

2) Desarrollo

El desarrollo comprende tanto el diseño y montaje de la electrónica y del software del controlador. Para ello esta memoria se ha dividido en el diseño y montaje del circuito de potencia para manejar el motor paso a paso, la implementación del controlador mediante un microcontrolador **PIC18F4520** y la comunicación con el **PC** a través del puerto serie **RS232** por dónde será transmitida la telemetría del sistema en tiempo real. En la Figura 3 se observa el esquema del controlador desarrollado.

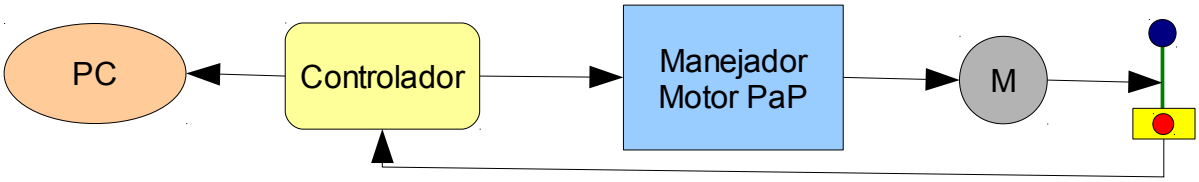

Figura 3: Esquema del controlador desarrollado

2.1.- Manejador Motor Paso a Paso

Este circuito consiste en dotar de la potencia necesaria a las señales recibidas desde el microcontrolador para excitar de forma apropiada los devanados del motor paso a paso responsable del movimiento del carro portador del péndulo.

El motor paso a paso utilizado (Figura 4) es del tipo bipolar ya que contiene dos devanados (Figura 5) que deberán ser alimentados siguiendo una secuencia predefinida para conseguir un giro en sentido horario. Por otro lado si la secuencia de alimentación sigue la secuencia inversa, el giro se producirá en el sentido antihorario tal como se detalla en la Tabla 1.

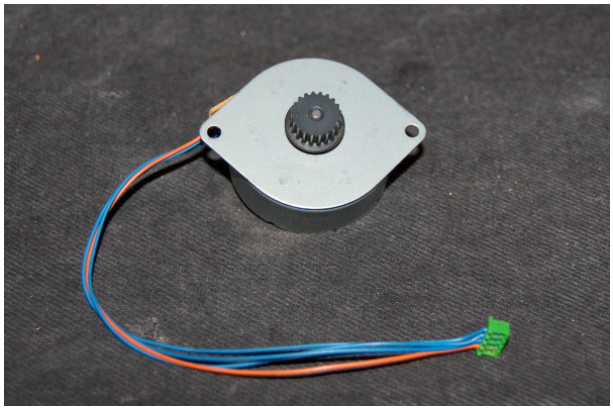

Figura 4: Motor paso a paso utilizado

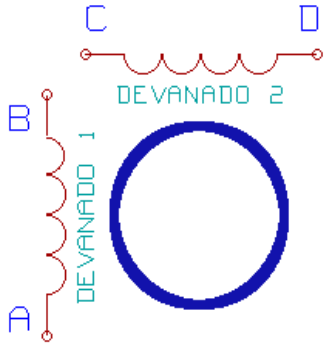

Figura 5: Esquema del motor paso a paso utilizado

Horario					Antihorario				
	Devanado 1		Devanado 2			Devanado 1		Devanado 2	
Secuencia	A	B	C	D	Secuencia	A	B	C	D
1	+	-	+	-	1	+	-	+	-
2	+	-	-	+	4	-	+	+	-
3	-	+	-	+	3	-	+	-	+
4	-	+	+	-	2	+	-	-	+

Tabla 1: Secuencia de excitación del motor paso a paso

El motor avanza por cada secuencia o paso **3.75º**, o lo que es lo mismo, se necesitan **96** pasos para completar una circunferencia. Luego cada paso se traduce en un desplazamiento longitudinal del carro mediante la correa dentada de transmisión (Figura 6).

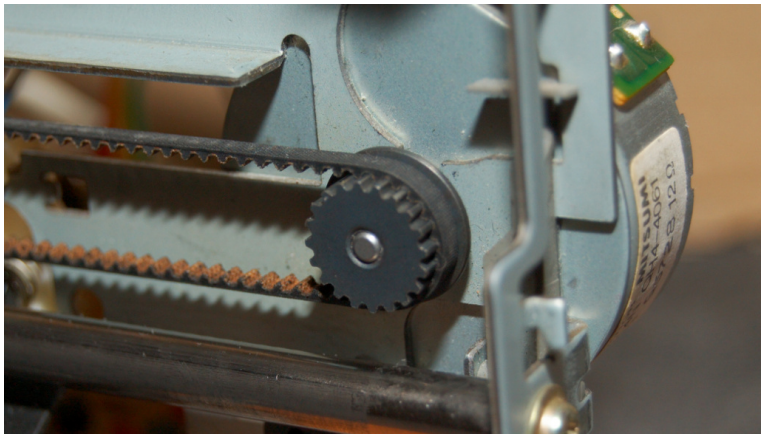

Figura 6: Detalle de la transmisión

La alimentación del motor según la hoja de datos es de **24V**, pero para este desarrollo dónde no

se persigue un alto torque se ha escogido como tensión de alimentación **12V** para prevenir un excesivo calentamiento del motor durante un funcionamiento prolongado. La intensidad de corriente medida en cada devanado cuando se lo alimenta con **12V** es de aproximadamente **850mA**. Para manejar estos niveles de corriente fue necesario diseñar e implementar un circuito de alimentación para dotar de la potencia requerida (**12V/850mA**) a cada salida del microcontrolador (**5V/25mA**).

Para la excitar los devanados se escogió el par de transistores *darlington* complementarios formado por el **TIP120** (**NPN**) y **TIP125** (**PNP**). Estos transistores están especialmente preparados para conmutar cargas inductivas de potencia ya que disponen de un diodo *dumper* entre **C** y **E** para contrarrestar los efectos nocivos de la fuerza contra-electromotriz, además de poseer una baja tensión de saturación ($V_{CE\,(SAT)}$ = **0.5V**) y una alta ganancia de corriente ($h_{FE\,(MIN)}$ = **1000**).

En la Figura 7 se visualiza el método empleado para conmutar la polaridad sobre un devanado y a la vez proveer de la corriente necesaria. Nótese que la conducción de los transistores de potencia se realiza de a pares de modo que saturando **Q3** y **Q6** y manteniendo cortados a **Q5** y **Q4** circula por el devanado la corriente $\mathbf{I_{36}}$ polarizándolo en un sentido (+ -). Por otro lado saturando **Q5** y **Q6** y cortando **Q3** y **Q4** se consigue la circulación de la corriente $\mathbf{I_{54}}$ que polariza el devanado en el sentido contrario (- +).

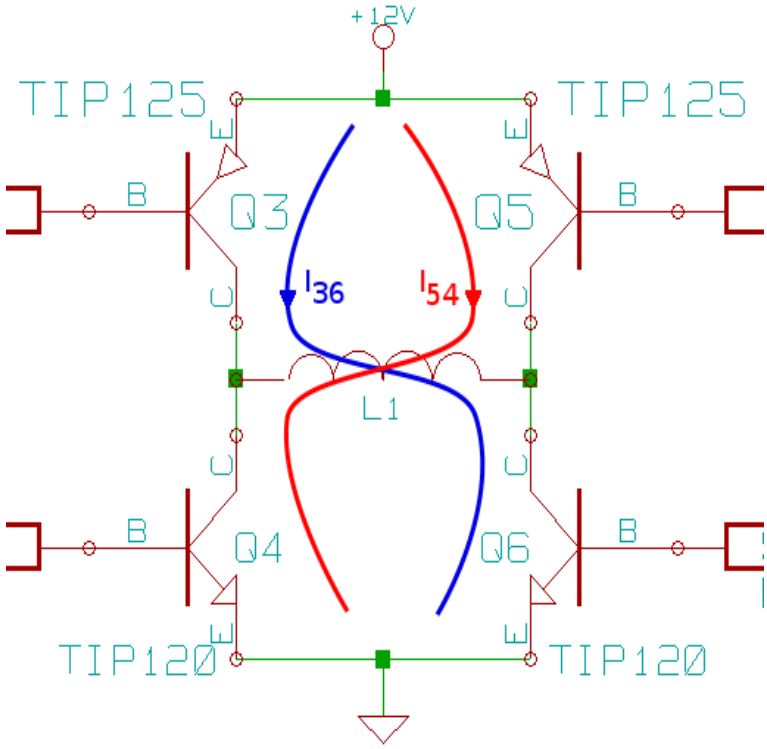

Figura 7: Alimentación de un devanado

Para lograr la conmutación de los transistores de potencia sin drenar demasiada corriente del microcontrolador, se ha diseñado el circuito de excitación para cada par *darlington* que se analiza a continuación.

Cuando la salida del microcontrolador está en nivel alto (**+5V**) los transistores de la etapa se polarizan tal como se observa en la Figura 8. De este modo se consigue la saturación de ambos **BC547** drenando sólo **1.8mA** de la salida del microcontrolador. Luego para saturar el **TIP125** se hace circular por su base una corriente de **9.25mA** lo que garantiza la saturación del mismo mediante la Fórmula 1. Nótese que el cálculo realizado arroja un coeficiente de **91** el cual es mucho menor que **1000**, lo que garantiza la condición de saturación del transistor.

$$\frac{I_c}{I_b} \ll h_{FE}; \frac{850\text{mA}}{9.25\text{mA}} = 91 \ll 1000$$

Fórmula 1: Condición de saturación

El otro *darlington* (**TIP120**) al ser del tipo **NPN** y tener un potencial cercano a **0V** en su base (**BC547** saturado) permanece cortado, por lo que la corriente que circula por el mismo es despreciable.

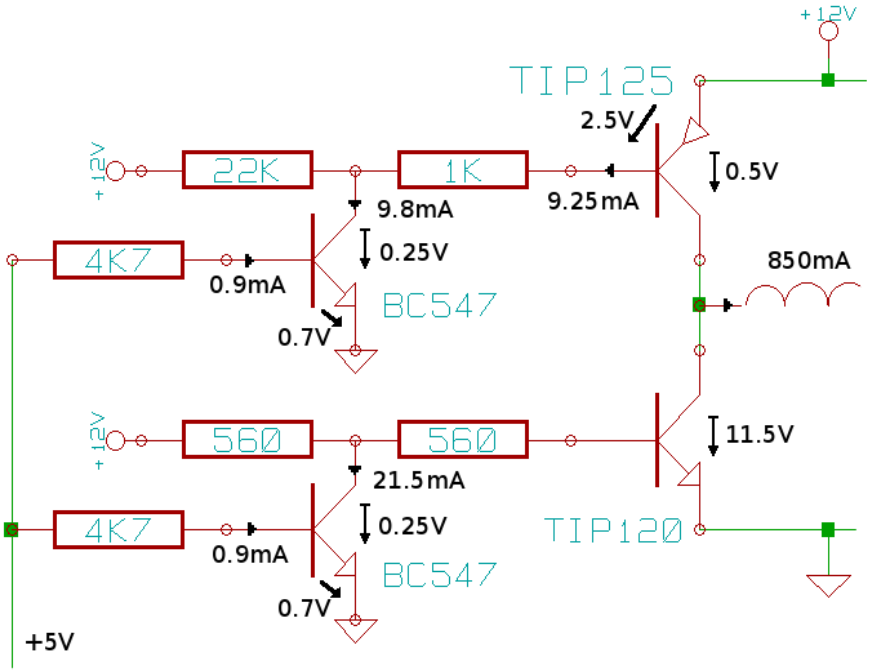

Figura 8: Polarización ante una entrada de +5V

Para que circule corriente por el devanado, el circuito se cierra mediante la etapa complementaria la cual estará polarizada con **0V** en su entrada tal como se analiza en la Figura 9.

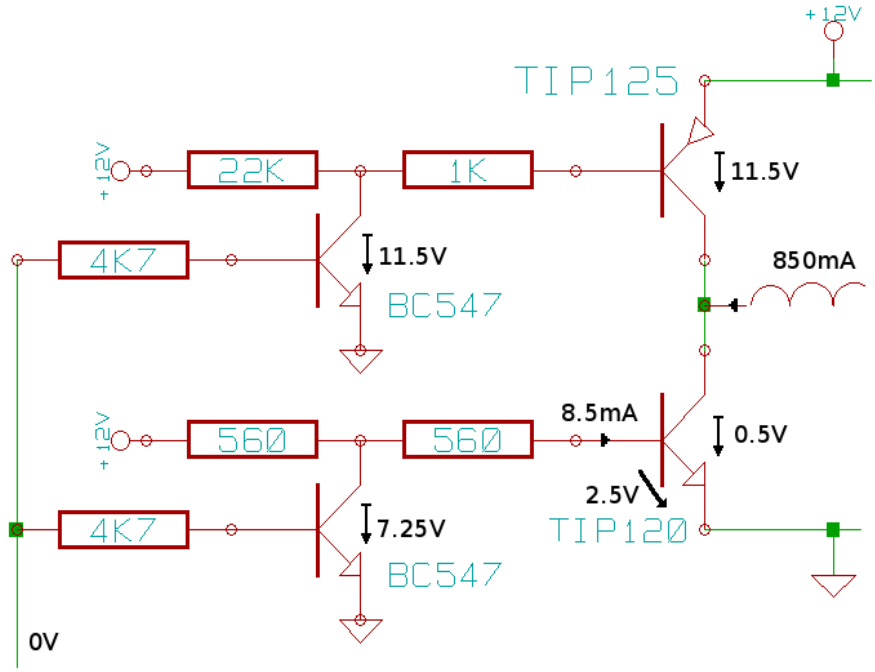

Figura 9: Polarización ante una entrada de 0V

En este caso al encontrarse ambos **BC547** cortados no circula por los mismos ninguna corriente reseñable, sin embargo ahora en cada base de los **TIPs** hay un potencial alto que provoca el corte del **PNP** y la saturación del **NPN**. Nótese que ahora la circulación de la corriente por el devanado se deriva a masa a través de transistor **TIP120**.

La condición de saturación también se cumple en este caso siendo **100** el resultado del cociente que sigue siendo mucho menor que **1000**. La pequeña diferencia en cuanto a corrientes se debe a la necesidad de escoger valores comerciales para las resistencias en vez de los obtenidos a través del cálculo teórico.

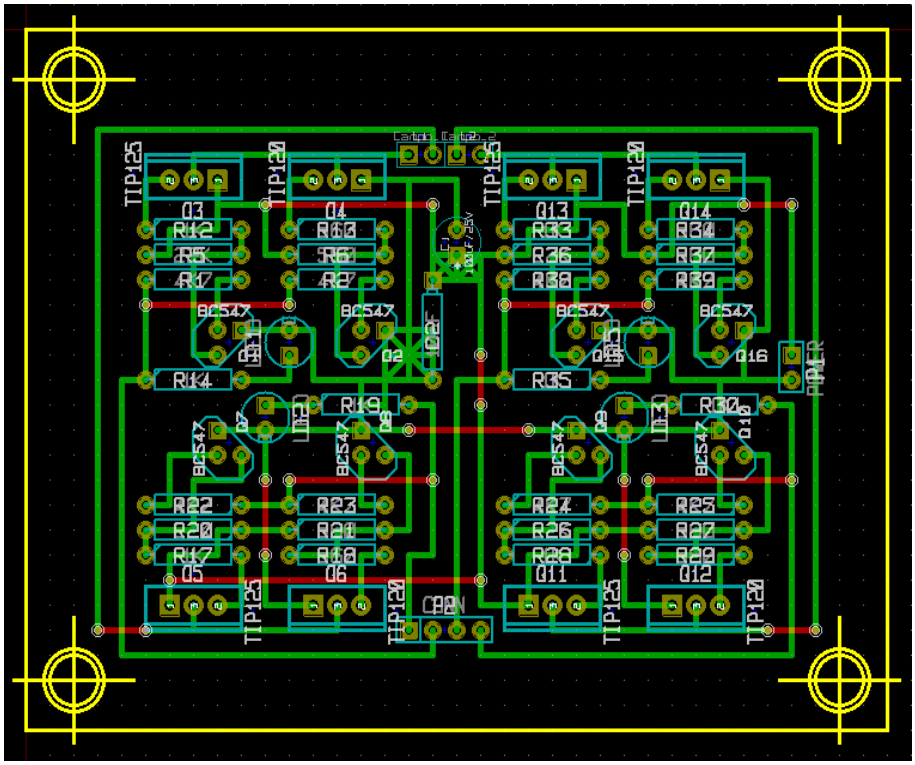

Figura 10: Captura del diseño de la placa de circuito impreso del manejador

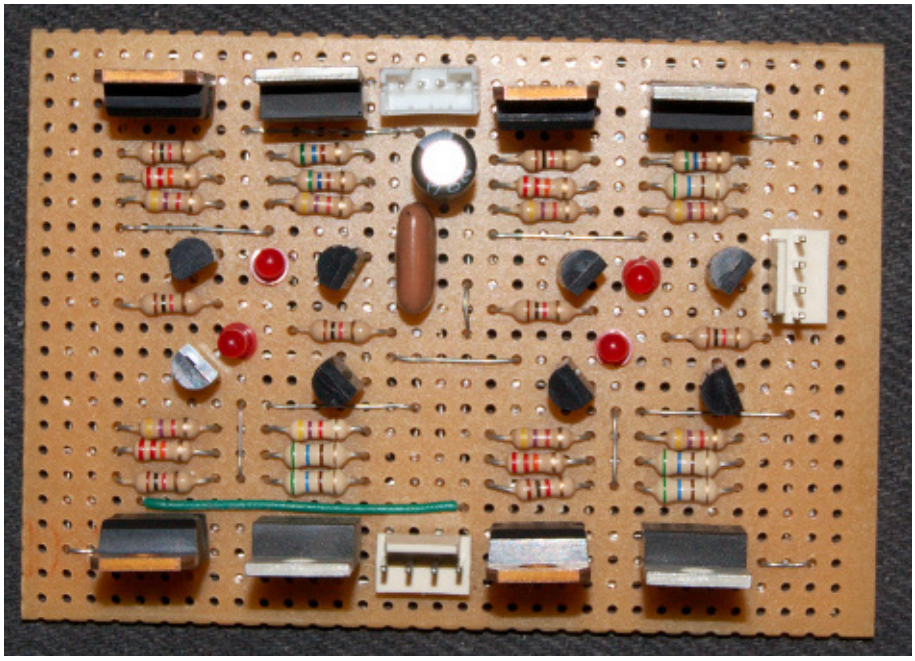

Figura 11: Montaje del circuito impreso del manejador

Nótese que los valores de tensiones y corrientes en las figuras son aproximados y que dependen

de la tolerancia de los componentes empleados, pero de todas formas las condiciones de corte y saturación se encuentran garantizadas.

El circuito al completo se puede consultar en el fichero del esquemático que acompaña a este documento, así como los ficheros *Gerber* para la construcción del circuito impreso. Con fines ilustrativos se presenta en la Figura 10 una captura del diseño de la placa y una foto del montaje realizado en la Figura 11.

2.2.- Controlador

El controlador está implementado con el microcontrolador **PIC18F4520** de *Microchip*, del cual se ha utilizado su **gestor de interrupciones**, uno de sus **temporizadores**, el **conversor A/D** y la **EUSART** (*Enhanced Universal Synchronous Asynchronous Receiver Transmitter*).

En la Figura 12 se puede visualizar la porción del esquemático correspondiente al controlador y en la Figura 13 el montaje realizado en la *protoboard* con fines de realizar los ensayos y ajustes necesarios antes de dar el diseño por válido.

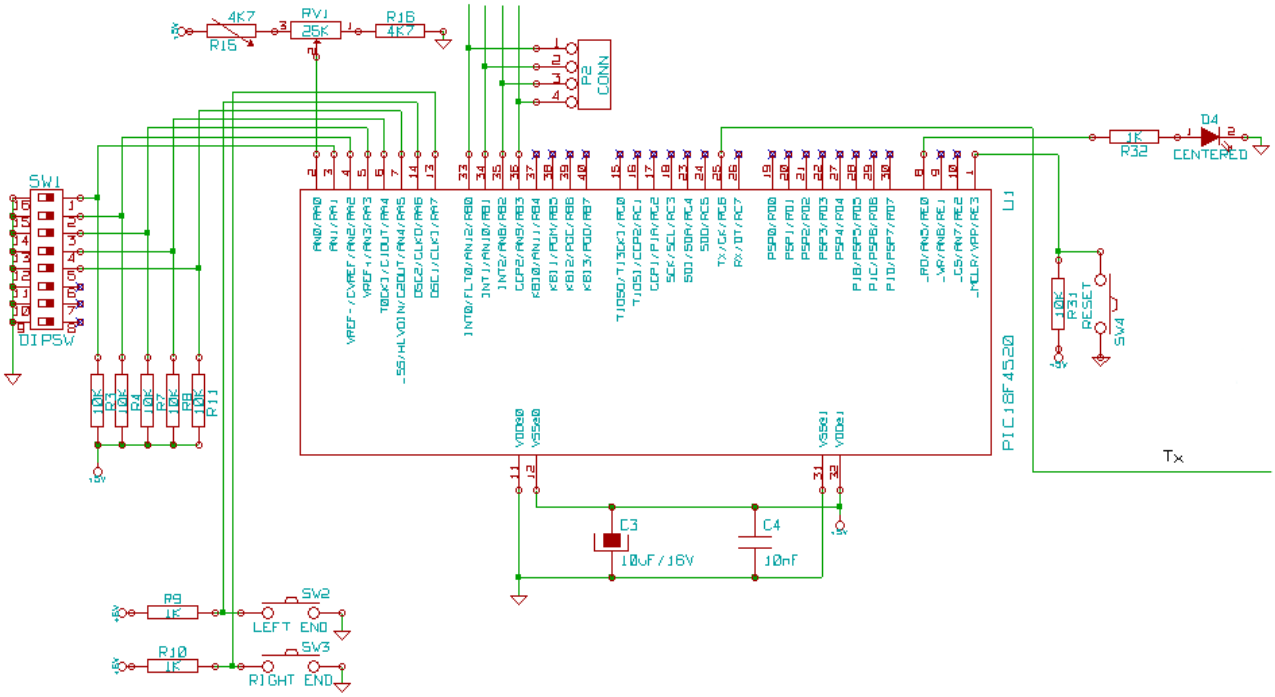

Figura 12: Esquemático del controlador

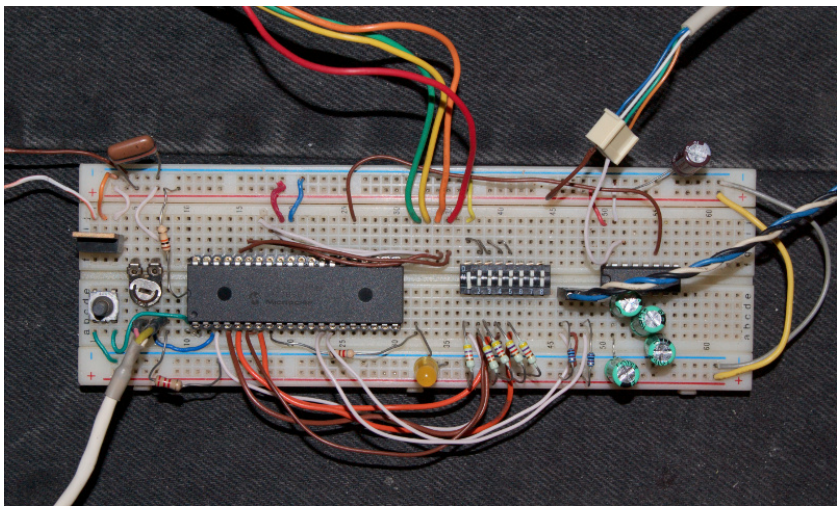

Figura 13: Montaje para los ensayos en la *protoboard*

En la Figura 14 se presenta el esquema de conexionado de los componentes que constituyen el controlador del péndulo vertical.

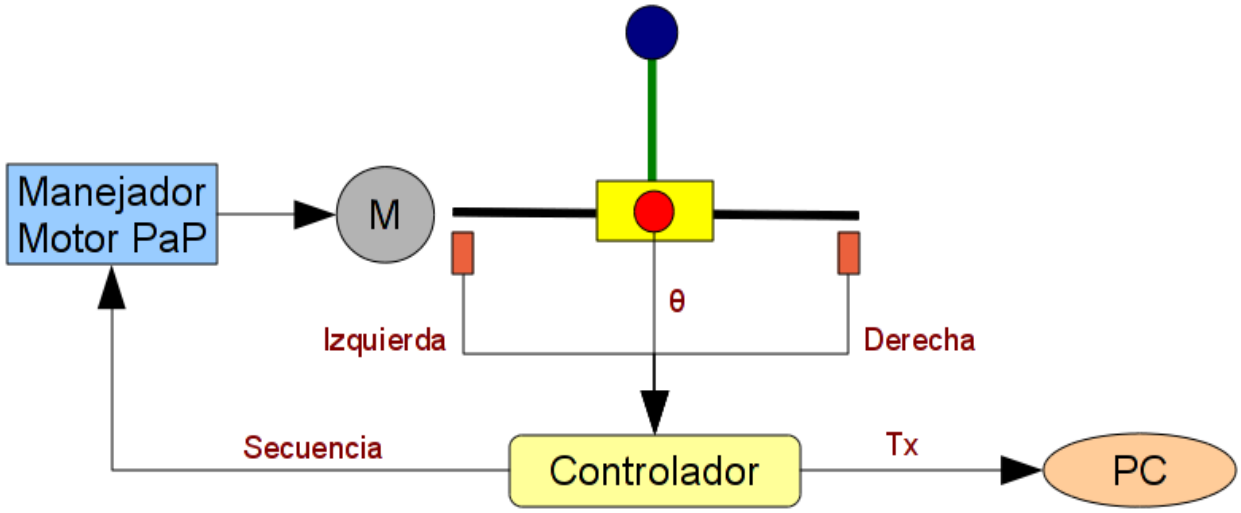

Figura 14: Esquema de conexionado del controlador

La posición del péndulo es recogida por el **conversor A/D** desde un montaje realizado en base a un potenciómetro de **25KΩ** que sirve tanto de eje como de sensor de inclinación (ver Figura 15). El objeto de montar de esta forma el eje es para minimizar el rozamiento propio de un potenciómetro limitando el punto de contacto a solamente el cursor que se desplaza sobre la zona resistiva y con una presión mínima.

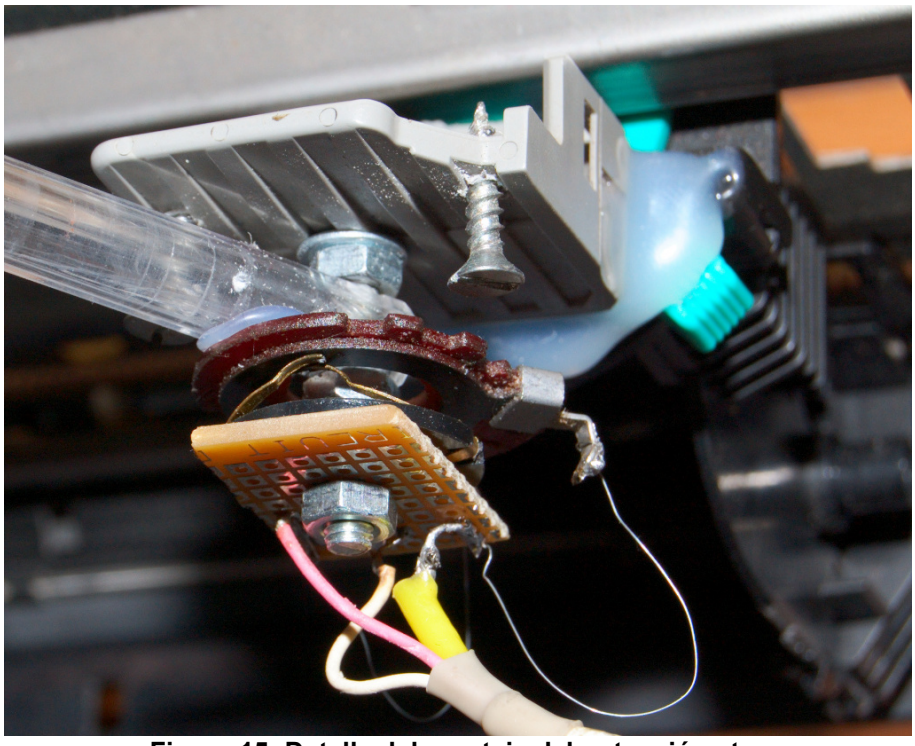

Figura 15: Detalle del montaje del potenciómetro

Además en los extremos del carro de la impresora se han dispuesto dos sensores de final de carrera para evitar la colisión del cabezal con los extremos (Figura 16). Cuando el cabezal acciona uno de estos interruptores el controlador detiene el funcionamiento del motor, siendo necesario volver a posicionarlo manualmente en el centro de su recorrido y *resetear* el sistema.

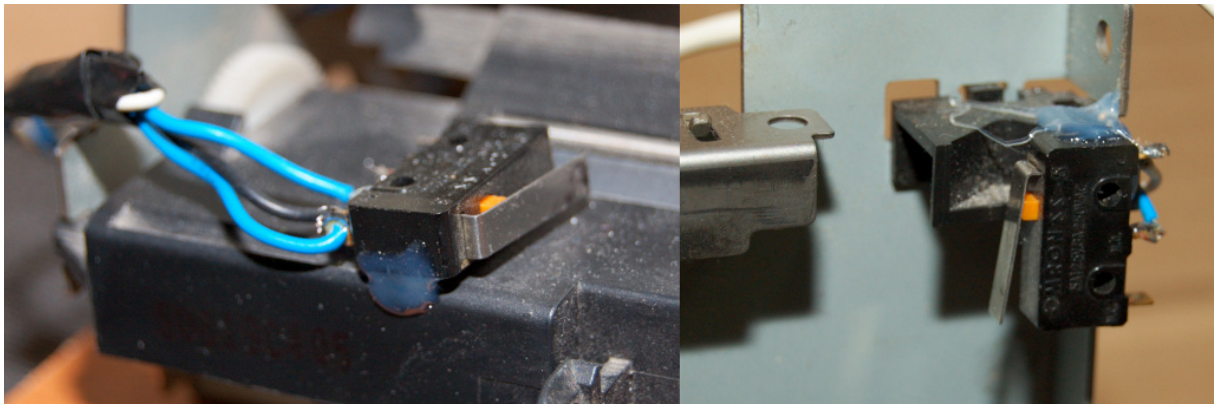

Figura 16: Detalle de los interruptores de final de carrera

La estructura del programa es la mostrada en la Figura 17, en donde el flujo del programa comienza con la inicialización del microcontrolador y de las variables para luego entrar en un bucle infinito que comprende la lectura de la señal analógica del error y el acondicionado y recorte de los niveles de la misma.

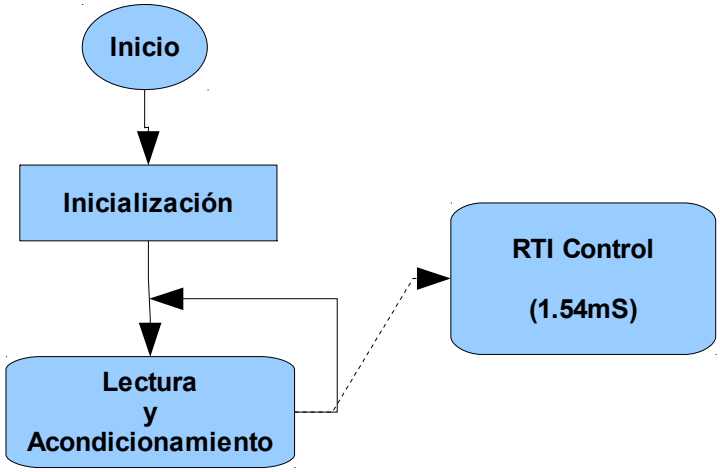

Figura 17: Estructura del programa del controlador

Cada **1.54mS** se produce una interrupción originada por el **temporizador** que ejecuta la rutina de control utilizando el último de los valores obtenido desde el **conversor A/D**.

A continuación se analizará el código de cada una de las secciones del programa con las que se implementa el controlador.

2.2.1 Inicialización

La inicialización comprende la configuración del microcontrolador y de los dispositivos que se van a utilizar y la asignación de los valores iniciales a las variables globales del programa. El código de esta sección se puede consultar en el Listado 1 en dónde realiza primero la configuración del reloj interno de microcontrolador escogiendo un valor de **8MHz** que es el mayor admitido por el dispositivo. La utilidad de una frecuencia alta es la de ejecutar toda la rutina controladora antes del siguiente ciclo de control para evitar solapamientos ya que la transmisión por el puerto serie prolonga su duración al formar parte de la rutina de control y nunca el tiempo de espera para la transmisión debe penalizar a la rutina de control. Véase el punto 2.3 para más información sobre la comunicación por el puerto serie **RS232**.

Posteriormente se inicializa el **puerto B** configurándolo como salida para ser utilizado (sus **4 bits** menos significativos) para enviar las señales de control del motor paso a paso al manejador. El **puerto E** también se configura como salida para excitar el **LED** de calibración a través de su **bit 0**. Luego le toca el turno al **puerto A** pero en este caso la configuración será de todas sus líneas como entradas y por último el **puerto C** se configurará también como entrada ya que es un requisito para

utilizar la **EUSART** con la que comparte algunos de sus pines.

Además del puerto, la **EUSART** requiere configurar los registros que se utilizarán para generar la señal de transmisión de **57.600 Baudios**. Una vez configurada se procede a habilitar este dispositivo.

La sección siguiente se encarga de configurar el **conversor A/D** seleccionando como entrada la menos significativa del **puerto A** (**AN0**) y configurando el modo en que se volcará el valor leído en los registros **ADRESH:ADRESL**.

Otro paso necesario es la inhabilitación del **comparador** ya que este comparte el **puerto A** con el **conversor A/D** y no se pueden utilizar al mismo tiempo.

Por último se inicializa el gestor de **interrupciones** habilitando sólo la correspondiente al **temporizador** que será la encargada de iniciar la ejecución cíclica de la rutina de control.

Sólo resta asignar los valores iniciales a las variables que se utilizarán dentro del programa para que el microcontrolador esté listo para ejecutar el bucle principal.

```
; Clock
bsf    OSCCON,IRCF0
bsf    OSCCON,IRCF1
bsf    OSCCON,IRCF2        ; 8 MHz
bsf    OSCCON,SCS1         ; Internal Oscilator

; Initialize PORT B: Drive the motor
clrf   PORTB
clrf   LATB
clrf   TRISB               ; all outputs

; Initialize PORT E: Calibrating LED
clrf   PORTE
clrf   LATE
clrf   TRISE               ; all outputs

; Initialize PORT A: Error, Cal, Kp & Kd
clrf   PORTA
clrf   LATA
setf   TRISA               ; all inputs

; Initialize EUSART (Port C): Telemetry
setf   TRISC               ; all inputs

bcf    TXSTA,TX9           ; 8-bit transmission
bcf    TXSTA,SYNC          ; Asynchronous mode
bsf    TXSTA,BRGH          ; High speed

bsf    BAUDCON,BRG16       ; 8-bit Baud Rate Generator

movlw  .34                 ; Internal counter
movwf  SPBRG               ; 57600 Bauds

bsf    RCSTA,SPEN          ; Enable EUSART
bsf    TXSTA,TXEN          ; Enable the transmission

; Initialize ADC
movlw  b'00000000'
movwf  ADCON0              ; select channel 0 (AN0)

movlw  b'00001110'
movwf  ADCON1              ; Only input AN0 is analog

movlw  b'10000110'         ; Read Right justified
movwf  ADCON2              ; Clock: FOSC/64

bsf    ADCON0,ADON         ; converter enabled
```

```
; Initialize Comparator: Turn off
movlw  0x07
movwf  CMCON               ; comparator disabled

; Initialize timer0: Control cycle
                                    ; 8 bits mode
movlw  b'01010011'         ; Internal clock
movwf  T0CON               ; prescaler (divides by 16)

; Interruption
bcf    RCON,IPEN           ; disable priority levels
bsf    INTCON,GIE          ; enable all unmasked interrupts
bcf    INTCON,PEIE         ; disable all unmasked peripheral interrupts
bcf    INTCON,TMR0IF       ; clear flag timer int.
bsf    INTCON,TMR0IE       ; activate timer overflow interruption

; Initialize vars
movlw  _SEQ_1              ; Motor sequence Literal
movwf  SEQ                 ; Initialize Sequence
movlw  _DIR_STOP           ; Direction Literal
movwf  DIRECTION           ; Stopped
clrf   ACTION              ; Controler Action
movlw  0xFF
movwf  DELAY               ; Initial Delay
movlw  0x01
movwf  COUNT               ; Initial Count
clrf   ENDS                ; not in left end, not in right end
clrf   ERR_1               ; Initial E(k-1)
clrf   ERR                 ; Initial E(k)

;;;;;;;;;;;;;;;;;;;;;;;;; Telemetry
clrf   TEL_ACT             ; Control
movlw  'S'
movwf  HEAD                ; Head stopped
clrf   ANGLE               ; Initial angle
;;;;;;;;;;;;;;;;;;;;;;;;;
```

Listado 1: Código de inicialización

2.2.2 Lectura y acondicionamiento del error

La lectura del error o desviación del péndulo de la vertical es realizado por el **conversor A/D** al cual le llega a través del pin **AN0** del microcontrolador la señal desde el potenciómetro ubicado sobre el cabezal de la impresora. Además se ha añadido un ***preset*** de ajuste para realizar la puesta a **0** en el modo de calibración, indicando que la posición en la que el péndulo ha sido colocado se corresponde con la que debe ser interpretada como vertical por el controlador (Figura 18).

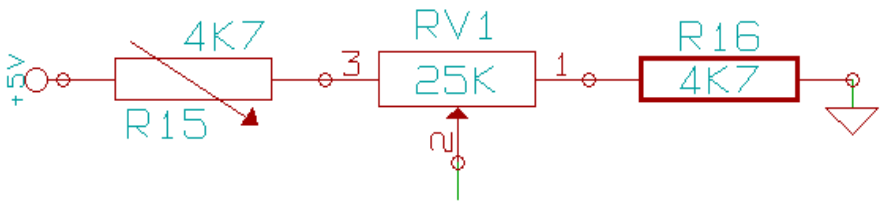

Figura 18: Ajuste fino para la puesta a 0

La tensión presente en le pin **AN0** es procesada por el **conversor A/D** y puede ser leída en los registros **ADRESH:ADRESL** como un número de **10 bits** y ser volcada en las variables **ADH** y **AHL**.

Luego se llama a la rutina que se encargará de comprobar de que el valor se encuentra dentro del rango deseado y si es necesario realizar el recorte del valor. Debido a que se va a actuar frente a desviaciones muy pequeñas, el error se restringirá al rango **±4**, valor que experimentalmente ha demostrado ser el más adecuado dando los mejores resultados.

Además dentro de esta rutina se comprobará el rango para el valor de la variable **ANGLE**, la que será utilizada para enviar le información telemétrica al **PC** indicando la inclinación del péndulo. En este caso se ha tomando un rango de valores de **±128** permitiendo enviar en **1 byte** tanto el ángulo como el signo.

En la Figura 19 se observa el diagrama de la rutina que lee y procesa el error y en el Listado 2 se puede consultar el código ensamblador.

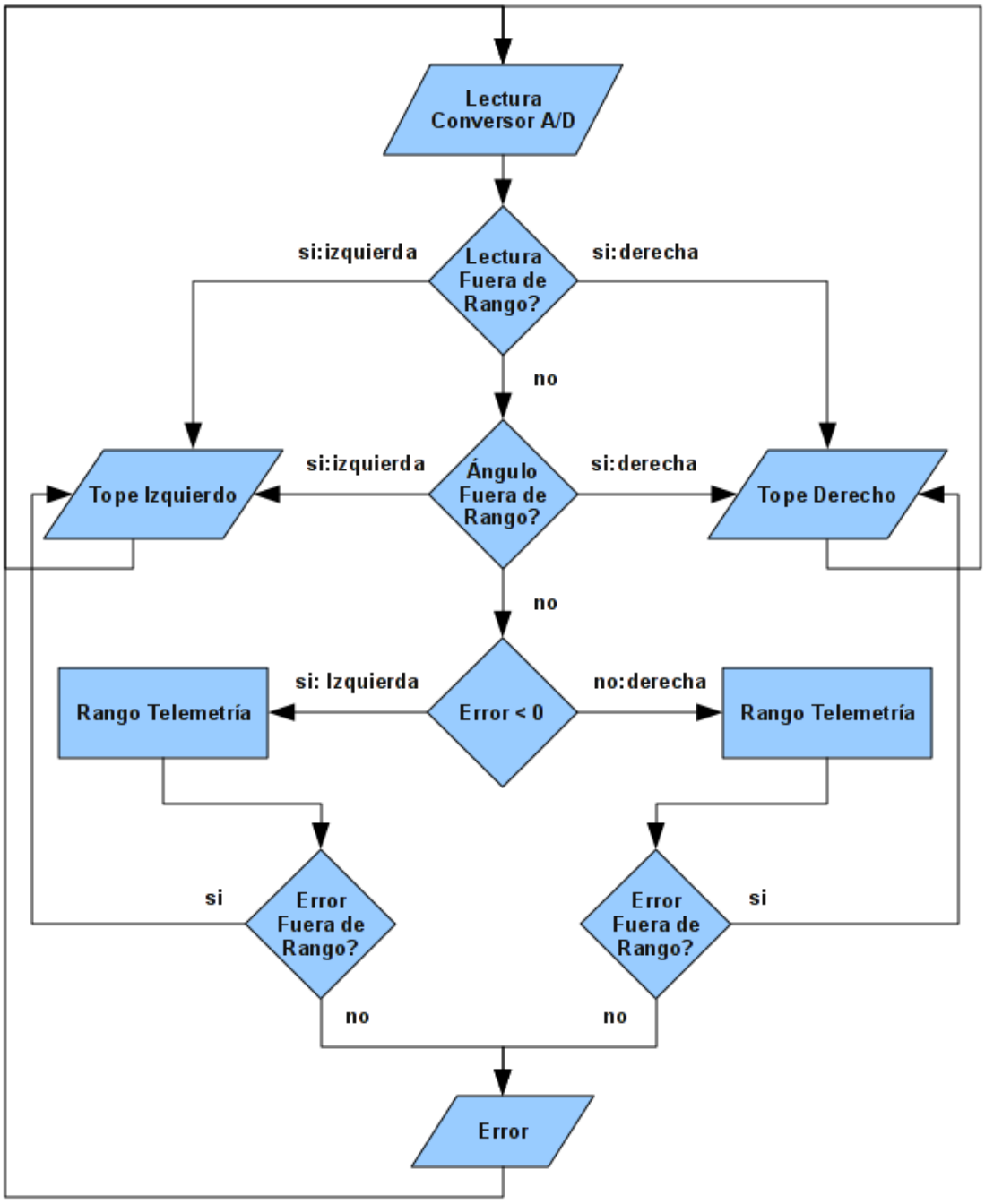

Figura 19: Estructura de la rutina de lectura y acondicionamiento del error

Al finalizar la rutina el error devuelto se guarda en la variable **ERR** ser procesado por la rutina de control la que se ejecuta periódicamente como ya se ha mencionado anteriormente. Además si el error es nulo se enciende el **LED** de centrado (salida **RE0** del **puerto E**) para indicar esta situación, dejándolo apagado en caso contrario. Por último para finalizar el bucle de lectura se introduce el retardo necesario para que el **conversor A/D** inicie una nueva conversión.

Nótese que esta rutina será interrumpida cada **1.54mS** por la rutina de control mediante la interrupción provocada por el **temporizador** del microcontrolador.

```
      bsf   T0CON,TMR0ON ; activate timer

Loop
      bsf   ADCON0,NOT_DONE     ; Begins ADC conversion

Read
      btfsc ADCON0,NOT_DONE
      goto  Read                ; Polling for ADC value

      movff ADRESH,ADH          ; ADRESH -> ADH
      movff ADRESL,ADL          ; ADRESL -> ADL

      call  Calc_E              ; Convert ADC value in SPEED and DIRECTION
                                ; returns error in WREG [-4 - 4]

      movwf ERR                 ; Updates ERR

      movlw 0x02
      movwf LOOP                ; Counter

      movlw 0x00
      bcf   PORTE,RE0           ; Turn off Center LED (calibrating)
      cpfseq ERR
      goto  ADCDelay            ; if ERR = 0
      bsf   PORTE,RE0           ; Turn on Center LED (calibrating)

ADCDelay
      decfsz LOOP
      goto  ADCDelay            ; Wait a minimum of 2 TADs before another conversion

      goto  Loop

;*********************************************************
; Transform Error
;*********************************************************
Calc_E
      ;;;;;;;;;;;;;;;;;;;;;;;; Range test (ADH value)
      movlw B'00000011'         ; Mask b0 & b1
      andwf ADH
      movlw .3
      cpfslt ADH                ; if ERROR > 1536 (11 00000000)
      goto  OutGreater          ; Out of Range (Greater)
      movlw .0
      cpfsgt ADH                ; if ERROR < 512 (01 00000000)
      goto  OutLower            ; Out of Range (Lower)
      movlw .1
      cpfseq ADH                ; if ERROR is positive (0x0200 - 0x02FF)
      goto  Greater             ; move to the right (greater)
                                ; else move to the left (lower)
                                ; (0x01FF - 0x0100)

Lower
      ;;;;;;;;;;;;;;;;;;;;;;;; Telemetry
      movff ADL,ANGLE
      movlw .128
      cpfsgt ANGLE              ; if ANGLE < -128
      movwf ANGLE               ; ANGLE = -128
      ;;;;;;;;;;;;;;;;;;;;;;;;
      movlw .252                ; Saturation Check (-4)
      cpfsgt ADL                ; if ADL <= -4
      retlw .252                ; Saturation: return -4
      movf  ADL,W               ; else
      return                    ; return Error

Greater
      ;;;;;;;;;;;;;;;;;;;;;;;; Telemetry
```

```
        movff  ADL,ANGLE
        movlw  .127
        cpfslt ANGLE                ; if ANGLE > 127
        movwf  ANGLE                ; ANGLE = 127
        ;;;;;;;;;;;;;;;;;;;;;;;;;
        movlw  .4                   ; Saturation Check
        cpfslt ADL                  ; if ADL >= 4
        retlw  .4                   ; Saturation: return 4
        movf   ADL,W                ; else
        return                      ; return error

OutGreater
        ;;;;;;;;;;;;;;;;;;;;;;;;; Telemetry
        movlw  .127
        movwf  ANGLE                ; ANGLE = 127
        ;;;;;;;;;;;;;;;;;;;;;;;;;
        retlw  .4                   ; Saturation: return 4

OutLower
        ;;;;;;;;;;;;;;;;;;;;;;;;; Telemetry
        movlw  .128
        movwf  ANGLE                ; ANGLE = -128
        ;;;;;;;;;;;;;;;;;;;;;;;;;
        retlw  .252                 ; Saturation: return -4
```

Listado 2: Lectura y acondicionamiento del error

2.2.3 Rutina de control

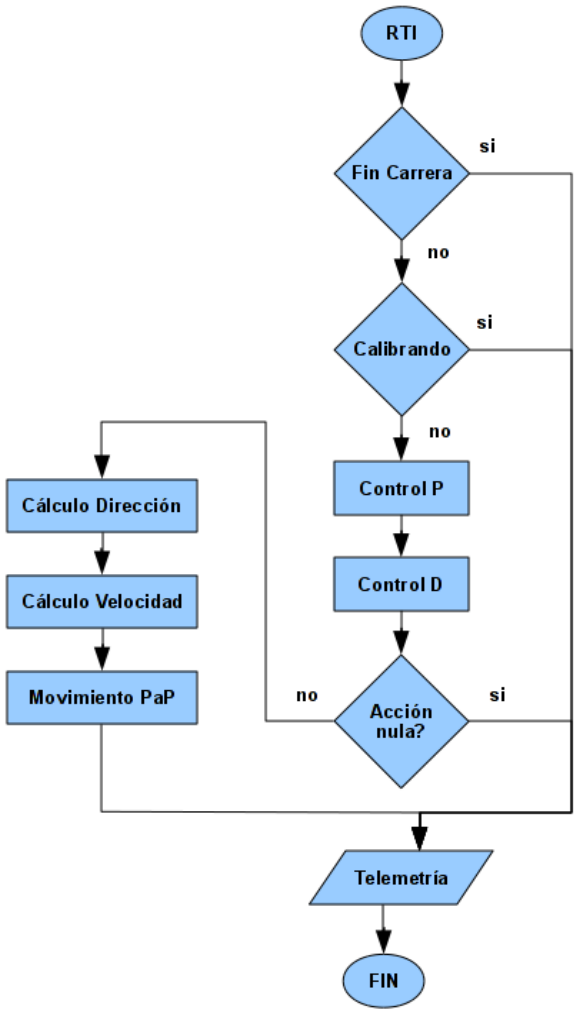

Figura 20: Rutina de control

La rutina comienza comprobando el estado del sistema ya que si se ha alcanzado alguno de los extremos del desplazamiento o se está calibrando la vertical no se realizará movimiento alguno a través del motor paso a paso aunque sí se enviará la información del estado a través de la telemetría.

El esquema de funcionamiento de la rutina de control se puede observar en la Figura 20 y para una mejor comprensión se analizará cada grupo de acciones individualmente.

2.2.3.1 Comprobaciones iniciales

En el Listado 3 se detallan las instrucciones que ejecutan las comprobaciones iniciales. Primero se comprueba que no se hayan activado los finales de carrera (entradas **RA6** y **RA7** del **puerto A**), si es el caso, de registra este estado en la variable **ENDS** y se salta hacia la etiqueta **Move0** para evitar que el cabezal se mueva. En caso contrario se comprueba el modo calibración, ya que si se está ajustando la verticalidad del péndulo, tampoco debe moverse el cabezal por motivos obvios.

Si no se da ninguna de estas condiciones la rutina continua su ejecución normal, pasando al cálculo del control proporcional derivativo.

```
;;; Test Ends
btfss  PORTA,RA6          ; if left end is reached (RA6=0)
bsf    ENDS,0             ; turn on ENDS<0>
btfss  PORTA,RA7          ; if right end is reached (RA7=0)
bsf    ENDS,1             ; turn on ENDS<1>
btfsc  ENDS,0             ; if Left End
goto   Move0              ; don't move head
btfsc  ENDS,1             ; if Right End
goto   Move0              ; don't move head

btfss  PORTA,RA1          ; if calibrating (RA1=0)
goto   Move0              ; don't move head
                          ; else calculate movement
```

Listado 3: Comprobaciones Iniciales

2.2.3.2 Control proporcional derivativo

El bloque de control comienza calculando el diferencial entre el error actual y el de la iteración anterior guardándolo en la variable **ERR_DIF** que será utilizada luego para calcular la componente diferencial del control. Seguidamente se guarda el error actual en la variable **ERR_1** para ser utilizado en la próxima iteración y se comienza el cálculo de la acción proporcional del control, véase el Listado 4.

El control proporcional obtiene la constante de proporcionalidad **KP** desde las entradas **RA2** y **RA3** del **puerto A**, valor que puede ser modificado durante la ejecución desde el *dip switch* dispuesto para tal fin. El valor de **KP**, que podrá ser de **1**, **2**, **3** ó **4**, se multiplica por el error, el cual al variar entre **±4** dará como resultado una excursión máxima del valor de la acción proporcional de **±16**.

El control derivativo obtiene el valor de su constante **KD** desde las entradas **RA4** y **RA5** pertenecientes al **puerto A** también controladas por el *dip switch*. En este caso **KD** puede tomar el valor de **0**, **1**, **2** ó **3** permitiendo anularse la acción derivativa si **KD** toma el valor **0**. Con el valor obtenido para **KD** se realiza la multiplicación por el error diferencial calculado anteriormente (guardado en **ERR_DIF**), se lo divide por dos para tener un control más suave sobre **KD** y finalmente se lo suma a la acción proporcional, obteniéndose así en la variable **ACTION** la suma de ambas acciones calculadas.

Teniendo en cuenta el valor máximo para el diferencial del error y el de la constante **KD**, la acción diferencial se encontrará entre **±24**. Con esto el rango de valores que puede tomar **ACTION** será de **±40**.

Para finalizar se guarda el valor de la acción calculada en la variable **TEL_ACT** para enviarla como telemetría y se calcula la dirección en la que se moverá el cabezal dependiendo del signo de la variable **ACTION**. Cabe destacar que para el siguiente apartado en dónde se calcula la velocidad con la que se moverá el motor paso a paso, **ACTION** debe ser siempre positivo por lo que si es necesario se le aplica el valor absoluto también dentro de este bloque de instrucciones.

```
      ;;; Calculate ERR_DIF
      movf   ERR_1,W
      subwf  ERR,W
      movwf  ERR_DIF            ; ERR(k) - ERR(k-1) -> ERR_DIF
      ;;; Save current error
      movff  ERR,ERR_1          ; ERR(k) -> ERR(k-1)

Control
      ;;;;;;;;;;;;;;;;;;;;;;;;; Proportional Control
      movlw  B'00001100'        ; Mask b2 and b3
      andwf  PORTA,W            ; KP
      rrncf  WREG
      rrncf  WREG
      incf   WREG               ; KP (1,2,3 or 4)

      ;;;;;;;;;;;;;;;;;;;;;;;;; Action Proportional
      mulwf  ERR                ; KP * ERR(k) -> PRODH:PRODL
      movff  PRODL,ACTION       ; Action P in ACTION [-16 - +16]

      ;;;;;;;;;;;;;;;;;;;;;;;;; Derivative Control
      movlw  B'00110000'        ; Mask b4 and b5
      andwf  PORTA,W            ; KD
      rrncf  WREG
      rrncf  WREG
      rrncf  WREG
      rrncf  WREG               ; KD (0,1,2 or 3)

      ;;;;;;;;;;;;;;;;;;;;;;;;; Action Derivative
      mulwf  ERR_DIF            ; KD * [ERR(k) - ERR(k-1)] -> PRODH:PRODL
      btfsc  ERR_DIF,7
      subwf  PRODH
      movf   PRODL,W
      rlcf   PRODH              ; sign PRODH:PRODL -> carry
      rrcf   WREG               ; Divides by 2 (KD = 0, 0.5, 1, 1.5)
      addwf  ACTION

      movff  ACTION,TEL_ACT     ; Telemetry

      ;;;;;;;;;;;;;;;;;;;;;;;;; Resolve direction

      movlw  0x00
      cpfseq ACTION             ; if ACTION <> 0
      goto   Direction          ; goto Direction: move head
      goto   Move0              ; else No Error: Don't move head

Direction
      movlw  _DIR_RIGHT
      movwf  DIRECTION          ; Move to the right by default
      btfss  ACTION,7           ; if ACTION >= 0
      goto   Quantizer          ; goto Quantizer

      ;;; Positivate
      negf   ACTION             ; -ACTION -> ACTION
      movlw  _DIR_LEFT
      movwf  DIRECTION          ; _DIR_LEFT -> DIRECTION
```

Listado 4: Control proporcional derivativo

2.2.3.3 Cálculo de la velocidad

Con el valor absoluto de la acción proporcional derivativa en la variable **ACTION** toca calcular

a que velocidad se desplazará el motor para corregir la inclinación del péndulo. Experimentalmente se ha comprobado que para este tipo de motor existen **4** niveles útiles de velocidad a partir de los cuales el movimiento no provoca efecto correctivo alguno sobre el péndulo.

Teniendo en cuenta esta restricción se calcula la velocidad para el motor como ciclos de retardo y se guarda este valor en la variable **DELAY** como se puede comprobar en el Listado 5. Luego este retardo será utilizado por el bloque de movimiento en el siguiente apartado para incrementar la fase de alimentación del motor paso a paso.

```
Quantizer
      ;;;;;;;;;;;;;;;;;;;;;;;; Quantize in 4 levels
      movff  ACTION,DELAY
      bsf    STATUS,C           ; Set Carry
      movlw  .5
      subfwb DELAY              ; 5 - DELAY -> DELAY
      btfsc  DELAY,7            ; if DELAY < 0
      clrf   DELAY              ; 0 -> DELAY
      movlw  .0
      cpfseq DELAY
      goto   Dir                ; if DELAY = 0
      movlw  .1
      movwf  DELAY              ; 1 -> DELAY
```

Listado 5: Cálculo de la velocidad

2.2.3.4 Movimiento motor paso a paso

Este bloque consta de dos secciones, la primera se encarga de llevar la cuenta de los ciclos de ejecución de la rutina de control en la variable **COUNT** que al ser comparada con el valor de **DELAY** determinará si el motor avanza un paso en este ciclo o no, controlando así la velocidad de giro que variará entre **6.76giros/seg** y **1.69giros/seg**.

La segunda sección comprueba el valor actual para la secuencia de movimiento del motor paso a paso (ver Tabla 1) almacenada en la variable **SEQ** pasando a la siguiente o anterior para provocar el avance o retroceso del mismo una vez que **SEQ** sea volcada en el **puerto B**.

En el Listado 6 se observa la porción de código en dónde se ha implementado este código, en dónde también se guarda el sentido de movimiento en la variable **HEAD** para ser transmitida luego como telemetría vía el **puerto serie RS232**.

```
Dir
      incf   COUNT              ; Inc COUNT
      movf   DELAY,W
      cpfsgt COUNT              ; if COUNT <= DELAY
      goto   Move0              ; do nothing (Stop & Wait)
      movlw  0x01
      movwf  COUNT              ; else begin count again
                                ; Resolves direction of movement
      movlw  _DIR_LEFT
      cpfseq DIRECTION          ; if DIRECTION equals DIR_LEFT
      goto   Right              ; jump to Right
      goto   Left               ; else jump to Left

Left                            ; Moves to the Left
      movlw  'L'
      movwf  HEAD               ; Telemetry: Move Head to the Left

      incf   SEQ                ; Inc SEQ
      movlw  0x04
      cpfsgt SEQ                ; if SEQ <= 4
      goto   Move               ; jump to Move
      movlw  0x01
      movwf  SEQ                ; else SEQ = 1
      goto   Move               ; jump to Move
```

```
Right                               ; Moves to the Right
       movlw  'R'
       movwf  HEAD                  ; Telemetry: Move Head to the Right

       decfsz SEQ                   ; Dec SEQ, if SEQ <> 0
       goto   Move                  ; Jump to Move
       movlw  0x04
       movwf  SEQ                   ; else SEQ = 4

Move                                ; moves head
       movlw  0x01
       cpfseq SEQ                   ; if SEQ <> 1
       goto   Move2                 ; go to test for _SEQ_2
       movlw  _SEQ_1                ; else w = _SEQ_1
       goto   EndMove
Move2
       movlw  0x02
       cpfseq SEQ                   ; if SEQ <> 2
       goto   Move3                 ; go to test for _SEQ_3
       movlw  _SEQ_2                ; else w = _SEQ_2
       goto   EndMove
Move3
       movlw  0x03
       cpfseq SEQ                   ; if SEQ <> 3
       goto   Move4                 ; go to test for _SEQ_3
       movlw  _SEQ_3                ; else w = _SEQ_3
       goto   EndMove
Move4
       movlw  _SEQ_4                ; w = _SEQ_4
       goto   EndMove

Move0
       movlw  'S'
       movwf  HEAD                  ; Telemetry: Stop Head

       movlw  0x00                  ; remove voltage of motor

EndMove
       movwf  PORTB                 ; puts W in PORTB
```

Listado 6: Movimiento motor paso a paso

2.3.- Comunicación Serie

La telemetría recogida en cada ejecución de la rutina de control se vuelca en el puerto serie del microcontrolador para ser enviada al **PC** al finalizar la ejecución de cada iteración. Como un puerto **RS232** trabaja con niveles de tensión diferentes a los del microcontrolador (**±9V** frente a **5V**) se hace necesario utilizar un circuito adaptador de tensiones el cual se presenta en la Figura 21, que como se puede contemplar está realizado con el circuito integrado **MAX232**.

Ya que solo se va a transmitir información desde el controlador hacia el **PC** se hace necesario conectar únicamente la señal de transmisión. Nótese que el circuito adaptador con **MAX232** se corresponde con el circuito de aplicación estándar sugerida por el fabricante.

La transmisión de datos se realizará a **57.600 Baudios** tal como se ha comentado anteriormente al configurar la **EUSART**. Esta transmisión demanda un tiempo que puede paralelizarse con la ejecución del código utilizando una interrupción extra y una cola de mensajes para tal fin, o bien si el período de ejecución de la rutina lo permite (como es el caso) esperar a que los datos hayan sido transmitidos antes de volcar un nuevo valor en el registro **TXREG** que es el destinado a esta operación.

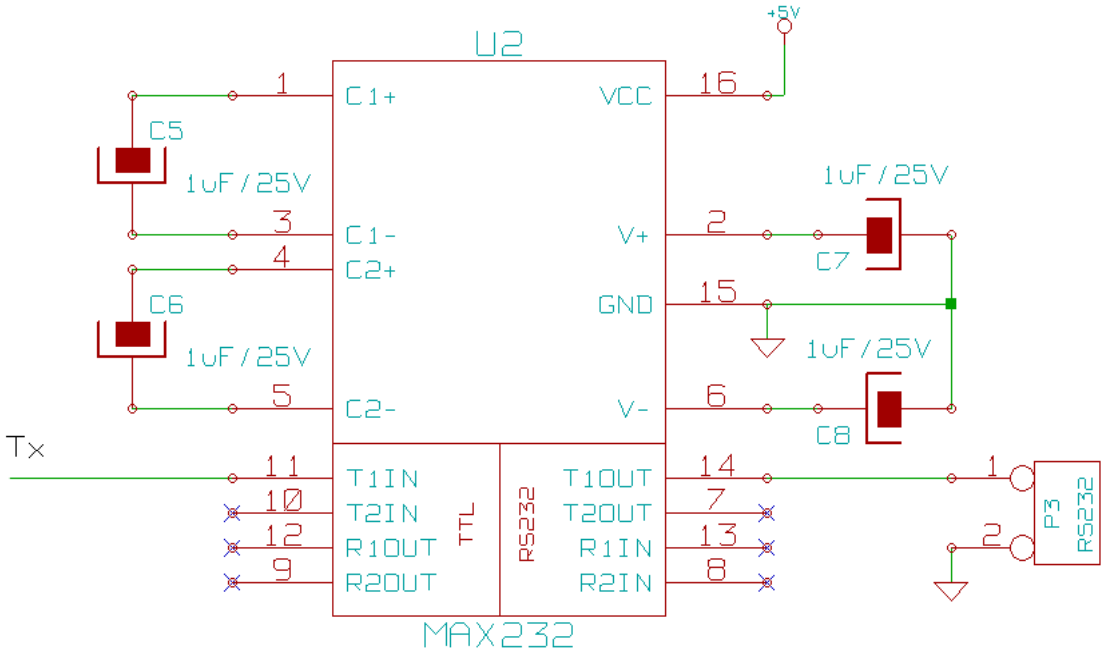

Figura 21: Adaptador señal de transmisión

Para optimizar la ejecución se transmite en primer lugar el ángulo de inclinación del péndulo (variable **ANGLE**) al comienzo de la rutina de control (ver el Listado 7). Luego mientras los datos son transmitidos se ejecutan los cálculos de control y al finalizar se transmiten los datos restantes correspondientes a la acción calculada (**TEL_ACT**) y la dirección de movimiento de la cabeza resuelta (**HEAD**).

Nótese que en la llamada a la rutina de transmisión (**Tx**) se comprueba el bit **TRMT** del registro **TXSTA** de modo que se encuentre activo para indicar que es posible transmitir el dato siguiente, esta condición garantiza que los datos previamente volcados en **TXREG**, han sido transmitidos en su totalidad.

```
      movf   ANGLE,W             ; Signed Byte [-128 - +127]
      call   Tx                  ; Angle -> RS232
( . . . )
      movf   TEL_ACT,W
      call   Tx                  ; Signed Action -> RS232
      movf   HEAD,W
      call   Tx                  ; Head -> RS232
( . . . )
;*****************************************************
Tx                               ; RS232
      btfss  TXSTA,TRMT          ; 1 = TSR Empty
      goto   Tx                  ; Wait to transmit
      movwf  TXREG               ; Transmits WREG
      return
```

Listado 7: Envío de telemetría

La forma más sencilla de recoger los datos enviados al **PC** es utilizar un programa emulador de terminal que permita volcar los datos recibidos por el puerto **RS232** a un fichero binario en disco. Luego con el programa del Listado 8 se procesa este fichero, convirtiéndolo al formato *csv* o "delimitado" y así ser luego importado por cualquier programa de hoja de cálculo como por ejemplo el que forma parte de la suite **OpenOffice.org** de **Sun Microsystems Inc**. Una vez leídos los valores de **tiempo**, **ángulo**, **acción** y **posición** pueden ser interpretados, trazar las gráficas, calcular máximos y mínimos, etc.

```
/**********************
 * Pendulo.java
 * Converts telemetry into .csv file
 *
 * @author Daniel Héctor Stolfi Rosso
 */

import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Pendulo {

	public static void main(String[] args) {

		DataInputStream in = null;

		if (args.length < 1) {
			System.out.println("Uso: Pendulo fichero.log");
		} else {
			/* Open file */
			try {
			in = new DataInputStream(new BufferedInputStream(
								new FileInputStream(args[0])));
			} catch (FileNotFoundException e1) {
				e1.printStackTrace();
			}
			/* Initialize vars */
			int ch;
			int time = 1;
			int pos = 0;
			try {
				/* Read and process data */
				while (true) {
					System.out.print(time); // Time
					System.out.print(",");
					ch = in.readByte(); // Angle
					System.out.print(ch);
					System.out.print(",");
					ch = in.readByte(); // Action
					System.out.print(ch);
					System.out.print(",");
					ch = in.readByte(); // Move
					if (ch=='L') {
						pos--; // Left
					} else if (ch=='R') {
						pos++; // Right
					}
					System.out.println(pos);
					time++;
				}
			} catch (IOException e) {
			}
			/* Close and quit */
			try {
				in.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}
}
```

Listado 8: Programa para procesar la telemetría

En el Listado 9 se indica la forma de procesar el fichero de telemetría para convertirlo en **.csv** y en las Figuras 22, 23 y 24 se pueden visualizar ejemplos de las gráficas de pueden confeccionarse con el programa **OpenOffice.org** y los datos generados por el sistema.

```
java Pendulo fichero.log > fichero.csv
```

Listado 9: Línea de comando para procesar la telemetría

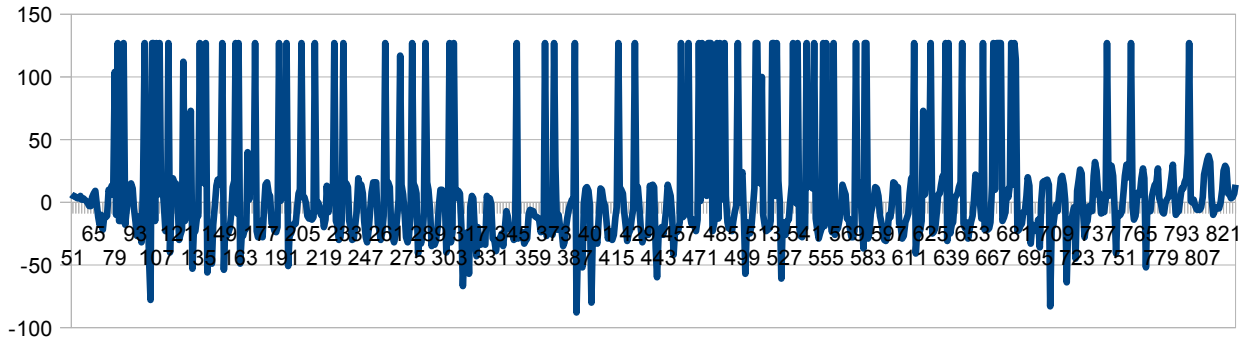

Figura 22: Telemetría: Ángulo de inclinación del péndulo

Figura 23: Telemetría: Acción del controlador

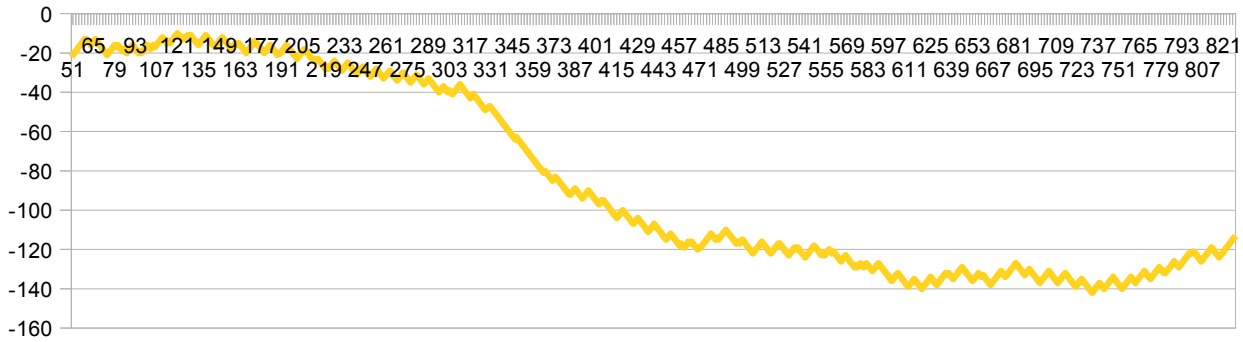

Figura 24: Telemetría: Posición del cabezal

3) Conclusiones

Con este desarrollo se ha implementado un controlador proporcional derivativo real, basado en los conocimientos teóricos adquiridos en la asignatura **control por computador**. Mediante el mismo se ha podido comprobar los problemas que surgen cuando los sistemas son reales y no teóricos como son los rozamientos, alinealidades en los sensores, etc.

Si bien el péndulo no mantiene su verticalidad por mucho tiempo, si lo hace por un lapso más prolongado del que lo haría si no existiese el control.

Las mejoras posibles al sistema son principalmente el reemplazo del motor paso a paso por uno

de corriente continua controlando su velocidad de giro mediante modulación de ancho de pulso (*PWM*) de modo minimizar las perturbaciones que provoca el diseño actual sobre la estabilidad del péndulo. Esto permitiría también un control más preciso de la velocidad de giro. Otra mejora de cara a minimizar el rozamiento sería el reemplazo del potenciómetro por un codificador óptico para obtener la inclinación del péndulo respecto a la vertical.

Por último, una mejora del tipo *software*, sería la recepción de telemetría en tiempo real, para lo cual sólo es necesario modificar el programa **Java** para que tome los datos directamente desde el puerto serie y los grafique en pantalla a medida que son recibidos.

Índice de Figuras

Índice de Listados