Anexo B. Programas de Control

El objetivo de este Anexo es el de explicar brevemente la programación realizada para ejecutar las rutinas de control del sistema real. Lo primero que hay que decir es que todos los programas, en lenguaje *C*, han sido insertados en el archivo fuente *function*, contenido este último en un proyecto de *Visual C++*. Dicho proyecto genera una interfaz con el usuario que permite ejecutar un hilo llamado *hilocapture* de forma que mientras este activado hará sucesivas llamadas al programa de control contenido en *function*.

B.1 Consideraciones prácticas

Antes de desarrollar los programas es conveniente efectuar varias aclaraciones acerca de las modificaciones que se han tenido que introducir en dichos códigos. Estas modificaciones son consecuencia de algunas de las dificultades prácticas que se expusieron en el Apartado 2.6. La primera es que el sistema funciona bien en pocas ocasiones si se le envía una nueva orden de movimiento antes de que se haya detectado un cambio en el ángulo. Esto se debe a que se provoca una saturación de la cola de mensajes recibidos y el sistema deja de interpretarlos correctamente.

Por este motivo aunque la señal de actuación se actualizan en cada ciclo, sólo se envía en caso de que se haya detectado el desplazamiento, hecho que se marca poniendo una variable que se llamará *x* a 1. De esta forma la probabilidad de fallo del programa es mínima. En cualquier caso este problema sólo se presenta en el retardo inicial y para señales pequeñas (inferiores a 250), ya que en el resto la velocidad es tal que el ángulo siempre ha cambiado a los 200 ms.

Otro detalle que ya se comentó y que se ha tenido que corregir debido a que fallaba mucho es que al hacer un cambio de sentido, aunque sigue moviéndose, deja de detectar ángulos. Tras muchas pruebas la única solución que se ha encontrado es enviar una orden de parada justo antes del cambio. De la experiencia se ha comprobado que esta modificación apenas afecta al control ya que en la mayoría de las ocasiones el sistema no se detiene en el cambio de sentido.

B.2 Programas

La estructura de los programas es idéntica en todos los tipos de reguladores y se puede dividir en cuatro partes. La primera es la declaración e inicialización de las variables que se van a emplear. El nombre y número de variables dependerá del tipo de control, pero en general aparecerán las constantes de los reguladores así como la actuación, el error y la salida en cada instante y en instantes precedentes. También se incluyen variables auxiliares.

En segundo lugar aparece el bloque de control propiamente dicho, en el que una vez medidas y actualizadas todas las señales (incluido el tiempo), se calcula la señal de control para ese ciclo. Esta actuación se envía al sistema teniendo presente que no puede ser superior a 2500 y asignando un sentido de movimiento u otro según su signo. Como ya se mencionó en el Capítulo 3 se considera sentido positivo ó creciente un giro hacia la derecha (visto desde la cámara) en azimut y hacia arriba en elevación.

Un tercer bloque del programa tiene la misión de crear un fichero de *Matlab* en el que se guardan todos los datos deseados sobre el experimento de control (tiempo, ángulos, señales de actuación) y que será imprescindible para su representación gráfica y la posterior comparación con el modelo teórico. La cuarta y última parte de la función

tiene como finalidad realizar un bucle de espera hasta que haya transcurrido el tiempo fijado en el período de control, manteniendo así el sincronismo de la señal.

Con el fin de evitar contenidos innecesarios sólo se va a incluir el desarrollo completo del programa para el control PID en el eje de azimut. Para los demás reguladores únicamente se presenta la ley de control, dado que el resto es siempre igual.

Hay que aclarar que el programa necesario para obtener el modelo del sistema es más simple, por lo que no se incluye. La estructura es la misma, con la diferencia de que en el instante inicial el tercer bloque se envía una única consigna de velocidad y se va registrando la evolución en cada ciclo.

```
int im=0;                              //INDICE PARA CONTAR CICLOS DE EJECUCION
struct _timeb ts0;                     //ESTRUCTURAS QUE GUARDAN EL TIEMPO
struct _timeb ts1;
struct _timeb ts2;
double resultadosm[10000][4];          //ALMACENA TIEMPO Y ÁNGULOS


#define DATOS_CONTROLPID1d     "DATOS_CONTROLPID1d.m"  //ARCHIVO
void controlPID1d(void)
{
static double valorinic=0.0;           //VARIABLES NECESARIAS PARA EL CALCULO
static double valorfin=0.0;            //DE LA SEÑAL DE ACTUACION
static double desplazado=0.0;
static double error=0.0;
static double error_ant=0.0;
static double error_ant2=0.0;
static float actuacion=0.0;
static float actuacion_real=0.0;
static float actuacion_antreal=0.0;
static float actuacion_ant=0.0;
static float actuacion_aux=0.0;

static float K=400;                    //PARAMETROS DEL CONTROLADOR
static double Ti=2;
static double Td=0.1;
static double Tc=0.2;
static double Kp=0.0;
static double Ki=0.0;
static double Kd=0.0;

static float x=0.0;                    //PARA MARCAR EL CAMBIO DE ANGULO
static int flag=0;                     //PARA SABER EL SENTIDO EN CADA MOMENTO
                                       //1-derecha;2-parado;3-izquierda
static int paso=0;                     //AUXILIAR PARA DETECTAR PASO DE 360
                                       //GRADOS A CERO Y VICEVERSA


double referencia=10;                  //GRADOS A DESPLAZARSE, CON SIGNO
double Tcontrol=200.0;                 //PERIODO DE CONTROL

if(im==0)
{
```

```
   pclase->sck->EnviarDatos(VELOCIDAD_AZIMUT,ACCSET,POSICIONADOR,0);

   flag=0;                              //POR SEGURIDAD PORQUE SE HA VISTO QUE
   error=0.0;                           //A VECES NO INICIALIZA BIEN
   error_ant=0.0;
   error_ant2=0.0;
   actuacion=0.0;
   actuacion_real=0.0;
   actuacion_antreal=0.0;
   actuacion_ant=0.0;
   actuacion_aux=0.0;
   x=0;

   Sleep(1500);
   _ftime(&ts0);                        //TIEMPO INICIAL
   valorinic=pclase->AngOrientPosic;    //VALOR INICIAL
   valorfin=valorinic+referencia;       //VALOR FINAL ABSOLUTO

   Kp=K*(1-(Tc/(2*Ti)));                //OBTENCIÓN DE LAS CONSTANTES A PARTIR
   Ki=K*(Tc/Ti);                        //DE LOS PARÁMETROS DEL CONTROLADOR
   Kd=K*(Td/Tc);
}

_ftime(&ts1);                           //TIEMPO ACTUAL
if((ts1.millitm-ts0.millitm)>=0)        //TIEMPO TRANSCURRIDO DESDE INICIO
{                                       //DEL HILO DE CAPTURA
   resultadosm[im][0]=(ts1.time-ts0.time);
   resultadosm[im][1]=(ts1.millitm-ts0.millitm);
}
else
{
   resultadosm[im][0]=((ts1.time-ts0.time)-1);           //TIEMPO EN SEGUNDOS
   resultadosm[im][1]=(ts1.millitm+1000-ts0.millitm);    //TIEMPO EN MILÉSIMAS
}

resultadosm[im][2]=pclase->AngOrientPosic;               //POSICION ACTUAL
if(((resultadosm[im][2]-resultadosm[im-1][2])!=0)||((resultadosm[im][3]-
-resultadosm[im-1][3])!=0))
{
   x=1;                                 //SOLO SE HACE CONTROL A LOS 200 ms.
}                                       //SI HAY CAMBIO

if(im!=0)
{
   error_ant2=error_ant;
   error_ant=valorfin-resultadosm[im-1][2];
}

if((resultadosm[im][2]<10)&&(resultadosm[im-1][2]>350))  //PARA QUE PUEDA
{                                                        //PASAR POR 360
   paso=1;
   valorinic=valorinic-360;
   valorfin=valorfin-360;
}
if((resultadosm[im][2]>350)&&(resultadosm[im-1][2]<10)&&(im!=0))
{
   paso=-1;
   valorinic=360+valorinic;
   valorfin=360+valorfin;
}

error=valorfin-resultadosm[im][2];                       //CALCULO DE SEÑAL
desplazado=resultadosm[im][2]-valorinic;                 //DE ACTUACION
actuacion_ant=actuacion;
actuacion_antreal=actuacion_real;
actuacion=(actuacion_ant+((Kp+Ki+Kd)*error-(Kp+2.0*Kd)*error_ant+
+Kd*error_ant2));
```

```
if(x==1)
{
   actuacion_real=actuación;

   if(((actuacion_real>0)&&(actuacion_antreal<0))||((actuacion_real<0)&&
   (actuacion_antreal>0)))
   {
      pclase->sck->EnviarDatos(PARAR_TODO,ACCSET,POSICIONADOR,AUTO);
   }


   if(actuacion>=0)
   {
      if(actuacion>2500)
      {
         actuacion_real=2500;         //ACTUACION_REAL, UNA VEZ SATURADA
      }
      pclase->sck->EnviarDatos(VELOCIDAD_AZIMUT,ACCSET,POSICIONADOR,
      actuacion_real);
      if(flag!=1)                     //SI PERMANECE CON MISMO SENTIDO NO SE
      {                               //MANDA PARA NO SATURAR COLA DE MENSAJES
         pclase->sck->EnviarDatos(DERECHA,ACCSET,POSICIONADOR,AUTO);
         flag=1;
      }
   }
   else
   {
      actuacion_aux=-1*actuacion;     //LA CONSIGNA DE VELOCIDAD HA
      if(actuacion<-2500)             //DE SER SIEMPRE POSITIVA
      {
         actuacion_real=-2500;
         actuacion_aux=2500;
      }
      pclase->sck->EnviarDatos(VELOCIDAD_AZIMUT,ACCSET,POSICIONADOR,
      actuacion_aux);
      if(flag!=3)
      {
         pclase->sck>EnviarDatos(IZQUIERDA,ACCSET,POSICIONADOR,AUTO);
         flag=3;
      }
   }
   x=0;
}

FILE *fout;
fout=fopen(DATOS_CONTROLPID1d,"a");
if (fout==NULL)
{
   sprintf(cadena,"error fichero datos_controlpid1d");
   muestra_en_pantalla(cadena);
}
else
{
   if(im==0)
   {
      fprintf(fout,"\n");
      fprintf(fout,"%%------NUEVO EXPERIMENTO CONTROL PID1d------\n");
      fprintf(fout,"%%----------REFERENCIA  %f  K=%f  Td=%f  Ti=%f----------
      \n",referencia,K,Td,Ti);
      fprintf(fout,"%%Tiempo  Real  Desplazado  Error  Actuación
      Actuacion_real  \n");
      fprintf(fout,"%.0f.%03.0f  %f  %f  %f  %f  %f \n",resultadosm[im][0],
      resultadosm[im][1],resultadosm[im][2],desplazado,error,actuacion,
      actuacion_real);
      fclose(fout);
   }
   else
   {
```

```
      if(((resultadosm[im][2]-resultadosm[im-1][2])==0)&&((resultadosm[im][3]-
      resultadosm[im-1][3])==0))
      {
         fprintf(fout,"%.0f.%03.0f  %f  %f  %f  %f  %f  \n",
         resultadosm[im][0],resultadosm[im][1],resultadosm[im][2],
         desplazado,error,actuacion,actuacion_real);
         fclose(fout);
      }
      else
      {
         if(paso!=0)
         {
            if(paso==1)
            {
               fprintf(fout,"%.0f.%03.0f  %f  %f  %f  %f  %f  \n",
               resultadosm[im][0],resultadosm[im][1],resultadosm[im-1][2],
               resultadosm[im-1][2]-valorinic-360,error_ant,actuacion_ant,
               actuacion_antreal);
               paso=0;
            }
            else
            {
               fprintf(fout,"%.0f.%03.0f  %f  %f  %f  %f  %f  \n",
               resultadosm[im][0],resultadosm[im][1],resultadosm[im-1][2],
               resultadosm[im-1][2]-valorinic+360,error_ant,actuacion_ant,
               actuacion_antreal);
               paso=0;
            }
         }
         else
         {
            fprintf(fout,"%.0f.%03.0f  %f  %f  %f  %f  %f  \n",
            resultadosm[im][0],resultadosm[im][1],resultadosm[im-1][2],
            resultadosm[im-1][2]-valorinic,error_ant,actuacion_ant,
            actuacion_antreal);
         }
         fprintf(fout,"%.0f.%03.0f  %f  %f  %f  %f  %f  \n",
         resultadosm[im][0],resultadosm[im][1],resultadosm[im][2],desplazado,
         error,actuacion,actuacion_real);
         fclose(fout);
      }
   }
}

if(resultadosm[im][0]>=90.0)          //SE LIMITA EL TIEMPO DE MOVIMIENTO
{                                     //DE LA CAMARA POR SEGURIDAD
   pclase->sck->EnviarDatos(PARAR_TODO,ACCSET,POSICIONADOR,AUTO);
}

im++;                                 //CICLO SIGUIENTE

int fin=1;                            //ESPERA A SIGUIENTE PERIODO DE MUESTREO
double retardo;

_ftime(&ts2);
retardo=1000*(ts2.time-ts1.time);
retardo=retardo+(ts2.millitm-ts1.millitm);
double tt;
tt=Tcontrol;

if(retardo>=tt)                       //VALIDACIÓN DEL PERIODO DE MUESTREO
{                                     //PARA VER SI ES SUFICIENTE PARA
   fin=0;                             //EJECUTAR LAS INSTRUCCIONES
   printf("Periodo de muestreo insuficiente");
}
while(fin)
{
   _ftime(&ts2);
```

```
   retardo=1000*(ts2.time-ts1.time);
   retardo=retardo+(ts2.millitm-ts1.millitm);
   if(retardo>=tt)
   {
      fin=0;
   }
}

}
```

Como ya se ha justificado sólo se van a incluir las leyes de control. En las siguientes líneas de código aparece el cálculo de la señal de actuación para los controladores proporcionales P, PD, PI y PID.

```
actuacion=K*error;                                                    //CONTROL P

actuacion=K*((1+(Td/Tc))*error-(Td/Tc)*error_ant);                    //CONTROL PD

actuacion=(actuacion_ant+((K/(2.0*Ti))*((2.0*Ti+Tc)*error+(Tc-        //CONTROL PI
-2.0*Ti)*error_ant)));

actuacion=(actuacion_ant+((Kp+Ki+Kd)*error-                          //CONTROL PID
-1*(Kp+2.0*Kd)*error_ant+Kd*error_ant2));
```

Para la aplicación del método *Anti-Windup* de limitación del término integral tanto en el control PI como en el PID las instrucciones pertinentes se muestran bajo estas líneas, en las que se denomina "*actuacion_int*" a la parte integral de la actuación y "*tope*" al límite establecido para dicha actuación.

```
actuacion_int=(((K*Tc)/(2.0*Ti))*(error+error_ant)+actuacion_int_ant);//PI,PID

if(actuacion_int>tope)||(actuacion_int<-1*tope))       //TECNICA DE LIMITACION
{                                                       //INTEGRAL
   if(actuacion_int>tope)
   {
      actuacion_int_real=tope;
   }
   else
   {
      actuacion_int_real=-1*tope;
   }
}
actuacion=(actuacion_int_real+K*error);                                  //PI

actuacion=actuacion_int_real+K*error+(K*(Td/Tc))*(error-error_ant);   //PID
```

En cuanto al Predictor de Smith hay que decir que el índice "*dm*" es un entero resultado del cociente entre el retraso teórico del modelo y el período de control. Las constantes *a*, *b*, *c* y *d* se obtienen a partir de los parámetros de control y se han empleado para simplificar las instrucciones. Ahora el error que le llega al regulador no

es la diferencia entre la referencia y la salida y la actuación que se emplea para conocer la salida teórica es la real, es decir, la que se transmite al sistema una vez acotada.

```
ym=ym_ant+k*actuacion_antreal;                          //PREDICCION DE LA SALIDA
ym_aux[im]=ym;                                          //VECTOR AUXILIAR
if(im>=dm)
{
    ym_dm=ym_aux[im-dm];
}
error=valorfin-resultadosm[im][2]-ym+ym_dm;

actuacion=((d/c)*actuacion_ant+((a/c)*error+(b/c)*error_ant));        //PI

actuacion=(actuacion_ant+(a*error-b*error_ant+c*error_ant2));         //PID
```

En el control por inversión la ley de control posee unos parámetros distintos en función de la dinámica deseada. Se presenta a continuación el caso de respuesta críticamente amortiguada en el eje de azimut.

```
actuacion_ant3=actuacion_ant2;
actuacion_ant2=actuacion_ant;
actuacion_ant=actuacion;
actuacion_antreal=actuacion_real;
actuacion=1.01*actuacion_ant-0.26*actuacion_ant2+0.25*actuacion_ant3+
+550*(error-error_ant);
```

Por último se escriben los pasos necesarios para calcular las consignas de velocidad en el espacio de estados, tanto con observador de orden completo como con observador de orden mínimo.

```
x1ant=x1;                                                   //ORDEN COMPLETO
x2ant=x2;
x3ant=x3;
salida_ant=salida;
actuacion_antreal=actuacion_real;

x1=x1ant+Ks*actuacion_antreal+l1*(salida_ant-x3ant);    //ESTIMACION DE
x2=x1ant+l2*(salida_ant-x3ant);                         //ESTADOS
x3=x2ant+l3*(salida_ant-x3ant);

actuacion=K*referencia-K1*x1-K2*x2-K3*x3;


x1bant=x1b;                                                 //ORDEN MINIMO
x2bant=x2b;
salida_ant=salida;
actuacion_antreal=actuacion_real;

x1b=-0.01*x2bant-0.008*salida_ant+Ks*actuacion_antreal; //ESTIMACION DE
```

```
x2b=x1bant+0.2*x2bant+0.17*salida_ant;                    //ESTADOS AUXILIARES

x1=x1b+x2b+0.81*salida;                                   //ESTADOS EN BASE
x2=x2b+0.8*salida;                                        //ELEGIDA
x3=salida;

actuacion=K*referencia-K1*x1-K2*x2-K3*x3
```