Informe de la implementación en un sistema empotrado de un controlador de velocidad

22449 - Sistemas Empotrados

Miró Barceló, Oriol
Seguí Capallonch, Lorenzo
2014-06-20

Índice

1. Introducción

El objetivo de la práctica consiste en la implementación de un controlador de velocidad de un coche, simulado mediante una placa dsPIC30F4011. Se pretende asimilar los conocimientos relacionados con la programación de micro-controladores y sus dispositivos internos, programación de tareas concurrentes en tiempo real y comunicación mediante el bus de campo CAN y el puerto serie.
La arquitectura del sistema general se muestra en la Figura 1. Un conjunto de sensores transmiten información de sus percepciones mediante el bus CAN, y el sistema de control realiza las funciones de actuador y regula la velocidad según los parámetros establecidos.

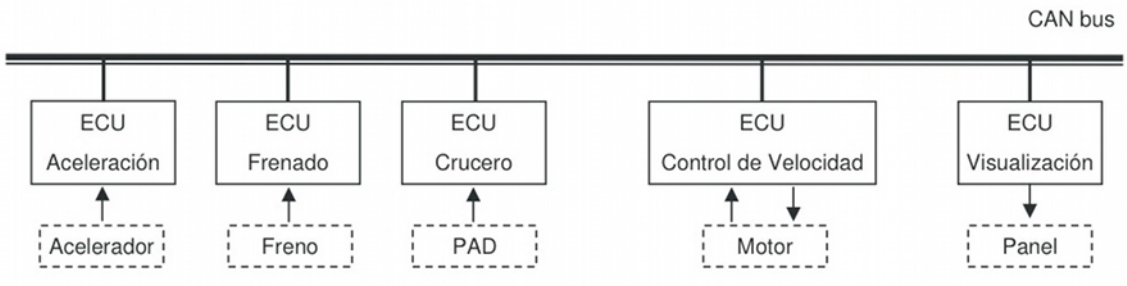

Fig. 1: Arquitectura del sistema general

Para la práctica se usará una simplificación del sistema mencionado arriba. En este sistema encontramos dos sensores, administrados por la tarea Acelerador, que obtiene sus percepciones del potenciómetro, y por la tarea Freno, que obtiene sus percepciones del teclado, y dos actuadores, administrados por la tarea Control, que se encarga del control de velocidad propiamente dicho según los parámetros recibidos, y por la tarea Panel, que muestra los valores de las variables sobre las cuales se actúa (velocidad, aceleración, estado del controlador, etc.).

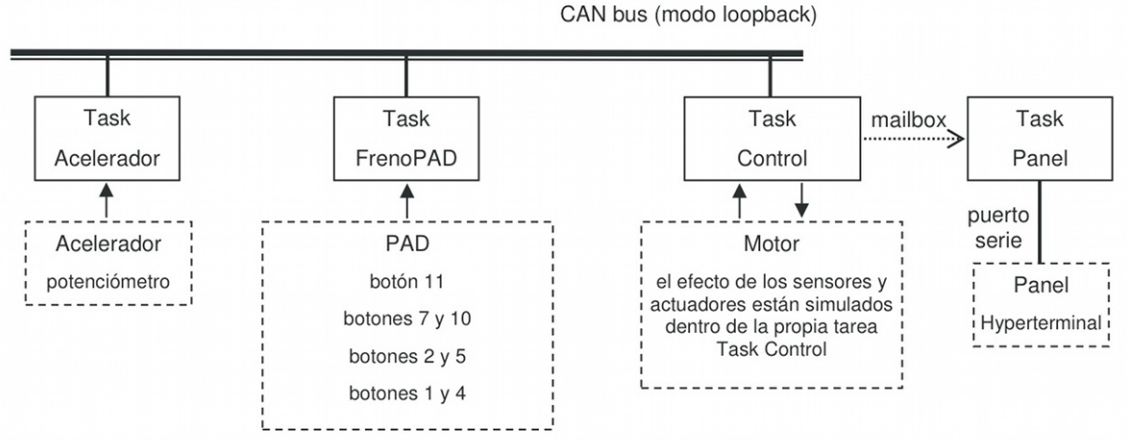

Fig. 2: Arquitectura del sistema simplificado

2. Diseño e implementación:

2.1 Tarea Acelerador

La tarea Acelerador es activada mediante un evento externo, la finalización de una conversión analógica-digital del potenciómetro.
Con el potenciómetro, que actúa como sensor, se obtiene el valor de la aceleración mediante la fórmula:

CADvalue < 512 -> aceleración = (512 - CADvalue) x (-0.005)

CADvalue >= 512 -> aceleración = (512 - CADvalue) x (0.005)

Al finalizar la conversión y la transformación del valor, se activa la tarea usando un *event flag*.

Seguidamente, en la tarea se transmite este valor a través del bus CAN.

```
void _ISR _ADCInterrupt(void){
     unsigned int valueCAD = CADGetValue();
     CADClearInt();

     // Convertion
     if(valueCAD < 512) {
          aceleracion = (512 - valueCAD) * (-0.005);
     } else {
          aceleracion = (valueCAD - 512) * 0.005;
     }

     //Set of Flags.
     OSSetEFlag(EFLAG_FOR_SPEED, 0xff);
}

void TaskAcelerador (void)
{
     while(1) {
          OS_WaitEFlag(EFLAG_FOR_SPEED, maskEventForSpeed, OSANY_BITS,
OSNO_TIMEOUT);

          OS_WaitBinSem(SEM_CAN, OSNO_TIMEOUT);
          if (CANtransmissionCompleted()) {
               CANsendMessage(ID_SPEED, &aceleracion, sizeof(aceleracion));
          }
          OSSignalBinSem(SEM_CAN);

          OSClrEFlag(EFLAG_FOR_SPEED, maskEventForSpeed);
     }
}
```

2.2 Tarea Freno

La tarea Freno es activada mediante el temporizador, en un período de 62.5ms. Administra el sensor del panel de control y el freno, simulado con el teclado del microcontrolador, enviando la percepción usando el bus CAN. Para simplificar esta transmisión hacemos uso de los identificadores de los mensajes, que se corresponde con la tecla pulsada, ahorrándonos la transmisión de mensaje.

Las funciones simuladas son la activación del freno, el encendido o apagado del control de crucero, la activación o desactivación de dicho controlador, y el aumento o reducción de la velocidad objetivo.

```
void TaskFreno (void) {
     char c;
     while (1) {
          c = getKeyNotBlocking();
          // If key is not valid or it has no function
          if (c != NO_BUTTON_PRESSED && c!= 0 && c!= 3 && c != 9 && c != 6
&& c != 8) {
               OS_WaitBinSem(SEM_CAN, OSNO_TIMEOUT);
               if(CANtransmissionCompleted()) {
                    CANsendMessage((unsigned int)c, &c, 0);
               }
               OSSignalBinSem(SEM_CAN);
          }
          OS_Delay(1);
```

```
    }
}
```

2.3 Tarea Control

La tarea Control es activada mediante un evento externo, la recepción de un mensaje mediante el bus CAN.
Realiza la función de actuador sobre el motor, es decir, sobre la velocidad, según las percepciones recibidas por los sensores simulados por las anteriores tareas.
Al recibir un mensaje, obtiene la información del mensaje transmitido por CAN (identificador y *buffer* con los datos) y activa la tarea usando un *event flag*.
La tarea, según las percepciones, actuará de distintos modos. Actualizará la velocidad según si el control de crucero está activado, en cuyo caso se limitará a la velocidad objetivo, o si el freno está pulsado, reduciéndose así la velocidad con una desaceleración constante. También se puede activar, desactivar, encender o apagar el control del crucero y aumentar o disminuir la velocidad objetivo.

```
void _ISR _C1Interrupt(void) {
     CANclearGlobalInt ();

     if (CANrxInt ()) {
          CANclearRxInt () ;
          // Read SID, DLC and DATA
          rxMsgSID = CANreadRxMessageSID();
          rxMsgDLC = CANreadRxMessageDLC();
          CANreadRxMessageDATA (rxMsgData);

          CANclearRxBuffer();
          OSSetEFlag(EFLAG_FOR_CONTROL, 0xff);
     }
}

void TaskControl (void) {
     static struct StructMessage mensaje;

     estado.velAct = 0;
     estado.velObj = -1;
     estado.cruceroEncendido = OFF;
     estado.cruceroActivado = OFF;
     estado.acelAct = 0;

     while (1) {
          OS_WaitEFlag(EFLAG_FOR_CONTROL,  maskEventForControl,  OSANY_BITS,
OSNO_TIMEOUT);

          // Return freno to OFF if TaskFreno message for forward checking
          if (rxMsgSID != ID_SPEED) {
               estado.freno = OFF;
          }

          switch(rxMsgSID) {
               case ID_SPEED: {
                    if (estado.freno == OFF) {
                         float *aux = rxMsgData;
                         estado.acelAct = *aux;
                    }
                    break;
               }
               case ID_BRAKE_11: {
                    estado.freno = ON;
                    estado.acelAct = -6.0;
```

```
                estado.cruceroActivado = OFF;
                break;
            }
            case ID_BRAKE_7: {
                estado.cruceroEncendido = ON;
                estado.velObj = estado.velAct;
                break;
            }
            case ID_BRAKE_10: {
                estado.cruceroEncendido = OFF;
                estado.velObj = -1; //Null
                break;
            }
            case ID_BRAKE_2: {
                if (estado.cruceroEncendido == ON) {
                    estado.velObj = estado.velObj + 0.005;
                }
                break;
            }
            case ID_BRAKE_5: {
                if (estado.cruceroEncendido == ON) {
                    estado.velObj = estado.velObj - 0.005;
                }
                break;
            }
            case ID_BRAKE_1: {
                if (estado.cruceroEncendido == ON) {
                    estado.cruceroActivado = ON;
                }
                break;
            }
            case ID_BRAKE_4: {
                if (estado.cruceroEncendido == ON) {
                    estado.cruceroActivado = OFF;
                }
                break;
            }
        }

        estado.velAct = estado.velAct + estado.acelAct*(TIEMPO/1000);
        if (estado.velAct < 0) {
            estado.velAct = 0;
        }
        if (estado.cruceroEncendido == ON && estado.cruceroActivado ==
ON) {
            if (estado.velAct > estado.velObj) {
                estado.velAct = estado.velObj;
            }
        }

        mensaje.velAct = estado.velAct;
        mensaje.velObj = estado.velObj;
        mensaje.cruceroEncendido = estado.cruceroEncendido;
        mensaje.cruceroActivado = estado.cruceroActivado;
        OSSignalMsg(MSG_FOR_PANEL, (OStypeMsgP)&mensaje);

        char buffer[20];
        LCDMoveFirstLine();
        sprintf(buffer, "V: %f", estado.velAct);
        LCDPrint(buffer);
        LCDMoveSecondLine();
        sprintf(buffer, "A: %f", estado.acelAct);
        LCDPrint(buffer);
```

```
          OSClrEFlag(EFLAG_FOR_CONTROL, maskEventForControl);
     }
}
```

2.4 Tarea Panel

La tarea Panel es activada mediante el temporizador, en un período de 125ms. Realiza la función de pantalla del control de crucero, mostrando su estado actual. Recibe la información mediante un mensaje enviado por la tarea Control usando *mailbox* y la imprime en un monitor comunicándose a través de su puerto serie. La información mostrado es la comprobación de la superación de una velocidad de seguridad, la velocidad actual, la velocidad objetivo, y la activación y encendido del control de crucero.

```
void TaskPanel (void)
{
     OStypeMsgP msgP;
     struct StructMessage *mensaje;

     while (1) {
               OS_WaitMsg(MSG_FOR_PANEL, &msgP, OSNO_TIMEOUT);
               mensaje = (struct StructMessage *)msgP;


               float velActKMH, velObjKMH;
               velActKMH = (*mensaje).velAct*3.6;
               velObjKMH = (*mensaje).velObj*3.6;

               char buffer[40];
               if (velActKMH > 120) {
                    sprintf(buffer, "Velocidad excesiva (> 120): WARNING\r
\n");
               } else {
                    sprintf(buffer, "Velocidad excesiva (> 120): OK\r\n");

               }
               putsUART1((unsigned int *)buffer);
               while (BusyUART1()) {} ;

               sprintf(buffer, "Velocidad actual: %f\r\n", velActKMH);
               putsUART1((unsigned int *)buffer);
               while (BusyUART1()) {} ;

               if (velObjKMH < 0) {
                    sprintf(buffer, "Velocidad objetivo: DESACTIVADA\r\n");
               } else {
                    sprintf(buffer, "Velocidad objetivo:%f\r\n", velObjKMH);
               }
               putsUART1((unsigned int *)buffer);
               while (BusyUART1()) {} ;

               int cruceroEncendido = (*mensaje).cruceroEncendido;
               int cruceroActivado = (*mensaje).cruceroActivado;

               if (cruceroEncendido == ON) {
                    sprintf(buffer, "Crucero encendido: SI\r\n");
               } else {
                    sprintf(buffer, "Crucero encendido: NO\r\n");
               }
               putsUART1((unsigned int *)buffer);
               while (BusyUART1()) {} ;
```

```
                if (cruceroActivado == ON) {
                    sprintf(buffer, "Crucero activado: SI\r\n");
                } else {
                    sprintf(buffer, "Crucero activado: NO\r\n");
                }
                putsUART1((unsigned int *)buffer);
                while (BusyUART1()) {} ;

                OS_Delay(2);
    }
}
```

3. Funcionamiento:

La aplicación, al ejecutarse, mostrará en la pantalla LCD del microcontrolador la velocidad actual y la aceleración, obtenida mediante el potenciómetro y transmitida por la tarea Acelerador.
A la vez, en el monitor, mediante el uso de Hyperterminal, se irán actualizando los datos del control del crucero.
Se puede comprobar el correcto funcionamiento variando el valor del potenciómetro o pulsando la tecla de frenado, que debería modificar el valor de la aceleración mostrada por la pantalla LCD y actualizar el valor de la velocidad según la nueva velocidad.
Al mismo tiempo, se puede observar el estado del control del crucero en la pantalla del monitor, y si este está apagado la velocidad, ante una aceleración positiva, debería aumentar sin límites, y al activarse debería reducir, en caso de superarla, a la velocidad objetivo.
Finalmente, la velocidad objetivo deberá mostrarse desactivada si el control de crucero se encuentra apagado, o deberá aumentar o disminuir si las teclas relacionadas son pulsadas.

4. Librerías:

Las librerías usadas nos han sido dadas. Contenían las constantes y las funciones necesarias para usar los elementos internos necesarios del microcontrolador, hacer uso de los protocolos de comunicación CAN y UART y usar el sistema operativo en tiempo real SalvoOS.
La única librería creada contiene las constantes e identificadores usados por Salvo OS y el bus CAN, con el objetivo de mantener la consistencia en caso de hacer uso de un sistema distribuido.

```
// Tasks TCBs
#define TASK_SPEED_P    OSTCBP(1)
#define TASK_CONTROL_P  OSTCBP(2)
#define TASK_BRAKE_P    OSTCBP(3)
#define TASK_PANEL_P    OSTCBP(4)

// Control structures
#define SEM_CAN                 OSECBP(1)

#define EFLAG_FOR_SPEED         OSECBP(2)
#define EFLAG_FOR_SPEED_EFCB    OSEFCBP(1)
```

```
#define iniValueEventForSpeed   0x00
#define maskEventForSpeed       0xff

#define EFLAG_FOR_CONTROL       OSECBP(3)
#define EFLAG_FOR_CONTROL_EFCB  OSEFCBP(2)

#define iniValueEventForControl 0x00
#define maskEventForControl     0xff

#define MSG_FOR_PANEL           OSECBP(4)

// Message structure
struct StructMessage {
    float velAct;
    float velObj;
    unsigned char cruceroEncendido;
    unsigned char cruceroActivado;
};

// CAN identifiers
#define ID_SPEED    3

#define ID_BRAKE_1  1
#define ID_BRAKE_2  2
#define ID_BRAKE_4  4
#define ID_BRAKE_5  5
#define ID_BRAKE_7  7
#define ID_BRAKE_10 10
#define ID_BRAKE_11 11
```