Escuela Politécnica Superior de Elche

CONTROL DE ROBOTS Y SISTEMAS SENSORIALES

PRÁCTICAS DE CONTROL VISUAL DE ROBOTS
Taller 3. Control de los Motores del Robot

Prof. Luis Miguel Jiménez

Departamento de Ingeniería
Área de Ingeniería de Sistemas y Automática

1 OBJETIVO

El objetivo de este documento es plantear los fundamentos del control de los motores del robot EyeBot. Se recomienda consultar el taller1 de introducción al Eyebot y el taller 2 de captura de imágenes, así como el servidor web (http://lorca.umh.es/isa/es/temas/eyebot) donde se encuentra documentación adicional.

Se describirán brevemente el manejo de los motores mostrando varios ejemplos.

2 MATERIAL EMPLEADO

La práctica se realizará en grupos de tres personas, disponiendo de un PC con S.O. Windows 95/98/2000/XP, el entorno de desarrollo Robios 6.0, y un Robot SoccerBot. La documentación y el software se puede encontrar en el servidor http://lorca.umh.es/isa/es/temas/eyebot

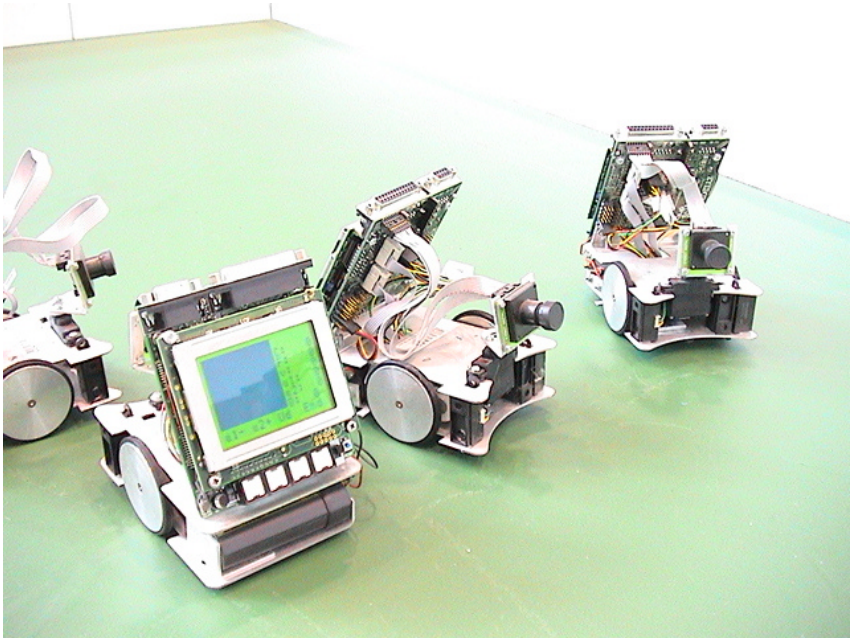

Figura 1. Robots SoccerBot disponibles en el departamento

3 CONTROL DEL SERVO DEL ROBOT

3.1 **¿Qué es un servomotor?**

Un servomotor no es más que un motor de corriente continua en el que la posición angular (medida mediante un potenciómetro) está realimentada a la entrada del mismo, generando un comportamiento estable y anulando el error de posición respecto a una referencia de entrada. Esta señal de referencia es generalmente una señal modulada PWM (pulsos periódicos de anchura variable). La anchura del pulso indica la amplitud de la señal de referencia que expresa la posición angular deseada para el eje del motor.

Al tratarse de un motor controlado en posición, no pueden girar de forma continua por lo que solo es útil cuando el recorrido del movimiento está limitado.

El controlador Eyebot permite controlar directamente este tipo de motores mediante la TPU del 68332, por lo que su conexión es muy sencilla. No requiere de drivers de potencia ya que los incorpora el propio motor.

Los servomotores comerciales suelen tener diferentes especificaciones para el periodo de la modulación PWM y los límites máximo y mínimo de giro del motor. Estos detalles se configuran en el fichero HDT. El conector estándar está formado por tres cables: alimentación del motor (rojo), masa (negro) y señal de control PWM (cable amarillo) (figura 2).

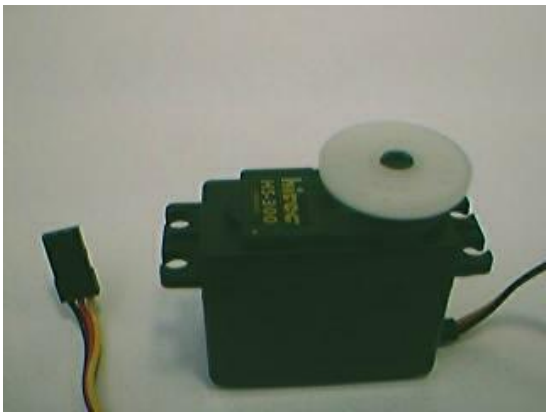

Figura 2. Servomotor

3.2 **Control de la cámara**

La cámara EyeCam está fijada al eje de un servomotor HITEC HS-81MG. Que nos permite orientar la cámara. Para fijar la orientación basta con mandar una nueva referencia al servomotor. Esta referencia, que varía entre 0 y 255, no está calibrada. En el programa de test visto en el taller se pueden comprobar los límites en el desplazamiento angular:

Referencia: 0 -> cámara a la derecha (límite)
Referencia: 128 -> cámara al centro
Referencia: 255 -> cámara a la izquierda (límite)

En la tabla 1 se describen las funciones asociadas al manejo del servo:

```
Servos
ServoHandle SERVOInit(DeviceSemantics semantics);
        Input:          (semantics) semantic (see hdt.h)
        Output:         (returncode) ServoHandle
        Semantics:      Initialize given servo

int SERVORelease (ServoHandle handle)
        Input:    (handle) sum of all ServoHandles which should be released
        Output:    (returncode)
                0 = ok
                errors (nothing is released):
                0x11110000 = totally wrong handle
               0x0000xxxx = the handle parameter in which only those bits remained
                                set that are connected to a releasable TPU-channel
        Semantics:      Release given servos

int SERVOSet (ServoHandle handle,int angle);
        Input:    (handle) sum of all ServoHandles which should be set parallel
                      (angle) servo angle
                       Valid values: 0-255
        Output:         (returncode)
                     0 = ok
                    -1 = error wrong handle
        Semantics:      Set the given servos to the same given angle
```

Tabla 1. Librería ROBIOS para manejo del servo de la cámara

Ejemplo: compilar y transferir este programa de ejemplo (*servo.c*) al robot**.**

```
/* servo.c */
#include "eyebot.h"

ServoHandle     servo_camara;

int main ()
{
       LCDClear();
       LCDMode(SCROLLING|NOCURSOR);
       LCDMenu("","","","END");

       LCDPutString("Camara ...\n");

       servo_camara = SERVOInit(SERVO11);
       if(!servo_camara)
       {
              OSError("Init Error!\n", 0 , FALSE);
              return -1;
       }

       SERVOSet (servo_camara,50);
       OSWait(200);
       SERVOSet (servo_camara,200);
       OSWait(200);

       SERVORelease (servo_camara);
      return 0;
}
```

4 CONTROL DE LOS MOTORES DC DEL ROBOT

Para el movimiento del robot se dispone de dos motores de corriente continua equipados con una reductora para aumentar el par sobre las ruedas. Para medir la posición del eje y su velocidad se dispone de dos encoders diferenciales en cuadratura acoplados directamente al eje del motor (no a la rueda). La velocidad de los motores se controla mediante dos drivers de potencia ubicados en la tarjeta Eyebot que nos permiten regular la corriente (velocidad) en un rango entero entre ±100.

Destacar que, a pesar de la reductora, el par no es muy elevado para el peso del robot por lo que el comportamiento de la velocidad respecto a la entrada de control (corriente suministrada) no es lineal.

Para el manejo de los motores que mueven el Robot tenemos dos opciones:

1) Trabajar con las funciones de bajo nivel que nos permiten ajustar la velocidad de ambos motores de forma independiente en bucle abierto. Realmente se fija la corriente transmitida a los motores y no su velocidad, que depende de la carga dinámica.

2) Trabajar con las funciones del control V-Omega que utilizan los encoders y un regulador de velocidad PI conjuntamente con los dos motores, teniendo en cuenta la cinemática del robot. Permite ajustar de forma precisa trayectorias y velocidades para el robot.

Las primera opción tienen la ventaja de la simplicidad de uso pero obliga a tener en cuenta la no linealidad del comportamiento del robot. Es útil cuando la trayectoria seguida por el robot no es crítica (control reactivo). El manejo directo mediante estas funciones no está disponible en el simulador EyeSim.

La segunda opción nos permite fijar trayectorias más precisas pero exige un costo computacional al precisar cerrar un bucle de control de forma periódica. También nos da una estimación de la posición y orientación del robot en cada momento basada en la cinemática del sistema diferencial y en las lecturas de los encoders. Estas funciones si que están disponibles en el simulador EyeSim

4.1 Funciones de bajo nivel

En la tabla 2 se resumen las funciones para el manejo de los motores. La tabla 3 muestra la funciones para la lectura del desplazamiento a partir de los de saltos registrados en los encoders.

Motors

```
MotorHandle MOTORInit(DeviceSemantics semantics);
        Input:          (semantics) semantic (see hdt.h)
        Output:         (returncode) MotorHandle
        Semantics:      Initialize given motor

int MOTORRelease (MotorHandle handle)
        Input:          (handle) sum of all MotorHandles which should be released
        Output:         (returncode)
                     0 = ok
                     errors (nothing is released):
                     0x11110000 = totally wrong handle
                     0x0000xxxx = the handle parameter in which only those bits remained
                                  set that are connected to a releasable TPU-channel
        Semantics:      Release given motor

int MOTORDrive (MotorHandle handle,int speed);
        Input:  (handle) sum of all MotorHandles which should be driven
                   (speed) motor speed in percent
                     Valid values: -100 - 100 (full backward to full forward)
                        0 for full stop
        Output:         (returncode)
                     0 = ok
                    -1 = error wrong handle
        Semantics:      Set the given motors to the same given speed
```

Tabla 2. Librería ROBIOS para manejo de los motores

Encoders

```
QuadHandle QuadInit(DeviceSemantics semantics);
        Input:          (semantics) semantic
        Output:         (returncode) QuadHandle or 0 for error
        Semantics:      Initialize given Quadrature-Decoder (up to 8 decoder are possible)

int QuadRelease(QuadHandle handle);
        Input:          (handle) sum of decoder-handles to be released
        Output:         0 = ok
                        -1 = error wrong handle
        Semantics:      Release one or more Quadrature-Decoder

int QuadReset(QuadHandle handle);
        Input:          (handle) sum of decoder-handles to be reseted
        Output:         0 = ok
                        -1 = error wrong handle
        Semantics:      Reset one or more Quadrature-Decoder

int QuadRead(QuadHandle handle);
        Input:          (handle) ONE decoder-handle
        Output:         32bit counter-value (0 to 2^32-1)
                       a wrong handle will ALSO result in an 0 counter-value!!
        Semantics:      Read actual Quadratue-Decoder counter


float QUADODORead(QuadHandle handle);
        Input:     (handle) ONE decoder-handle
        Output:     meters since last odometer-reset
        Semantics:  Get the distance from the last resetpoint of a single motor!
                    It is not the overall meters driven since the last reset!
                     It is just the nr of meters left to go back to the startpoint.
                    Usefull to implement a PID-control

int QUADODOReset(QuadHandle handle);
        Input:          (handle) sum of decoder-handles to be reseted
        Output:         0 = ok
                        -1 = error wrong handle
        Semantics:      Resets the simple odometer(s) to define the startpoint
```

Tabla 3. Librería ROBIOS para manejo de los encoders

Ejemplo: compilar y transferir este programa de ejemplo (*motor1.c*) al robot**.**

```
/* motor1.c */
#include "eyebot.h"

MotorHandle     leftmotor;
MotorHandle     rightmotor;

int main ()
{
       LCDClear();
       LCDMode(SCROLLING|NOCURSOR);
       LCDMenu("","","","END");

       leftmotor = MOTORInit(LEFTMOTOR);
       rightmotor = MOTORInit(RIGHTMOTOR);

       if(!leftmotor || !rightmotor)
       {
              OSError("Init Error!\n", 0 , FALSE);
              return -1;
       }

       LCDPutString("Girando ...\n");
       MOTORDrive (rightmotor,50);
       MOTORDrive (leftmotor,-50);
        OSWait(200);

       /* para el motor */
       MOTORDrive (rightmotor | leftmotor,0);
       LCDPutString("\nParado ...\n");

       KEYWait(KEY4);   /* espera que se pulsa la tecla 4 */

       MOTORRelease (rightmotor|leftmotor);
      return 0;
}
```

4.2 Control V-Omega

Se trata de un conjunto de funciones que inician y configuran un proceso (thread) de control de los dos motores para realizar trayectorias precisas con el robot. Tienen en cuenta la cinemática del robot para controlar las velocidades de giro. Estas funciones permiten:

1) Mantener velocidades traslacionales y rotacionales del robot especificadas
2) Conocer la posición actual del robot en coordenadas cartesianas
3) Conocer la orientación del robot

Las funciones de la librería V-Omega son bastante extensas por lo que no se reproducen en este documento. Se encuentran detallas en el fichero ***library.pdf***, por lo que se recomienda su estudio en profundidad dada la extensa lista de parámetros. Los tipos de datos específicos de esta libraría son:

```
/* Drive (VW) types */
typedef float meterPerSec;
typedef float radPerSec;
typedef float meter;
typedef float radians;
```

```
typedef struct
{
  meter x;
  meter y;
  radians phi;
} PositionType;

typedef struct
{
  meterPerSec v;
  radPerSec w;
} SpeedType;
```

Tabla 4. Tipos de datos para la API V-Omega

Ejemplo: compilar y transferir este programa de ejemplo (*motor2.c*) al robot**.**

Estudiar el uso de las diferentes opciones del controlador:

1) velocidad constante
2) conocer la posición y orientación actual (respecto a la posición inicial)
3) giros puros a una velocidad dada
4) Traslación pura
5) Trayectorias en arco

Como ejercicio se propone ajustar los parámetros de regulador de velocidad PI

```
/*
   motor2.c ejemplo de utilización del controlador V-Omega para los motores
*/

#include "eyebot.h"

meter distancia=0.4;
radPerSec vel_rot = 5.0;
meterPerSec vel_tras=10.0;
radians angulo=1.5;
PositionType pos;

VWHandle vw;

int main ()
{
       LCDClear();
       LCDMode(SCROLLING|NOCURSOR);
       LCDMenu("","","","END");

       vw = VWInit(VW_DRIVE,1);
       if(!vw)
       {
            OSError("Init Error!\n", vw , FALSE);
            return -1;
       }
       VWSetSpeed(vw, 0.0, 0.0);
       VWStartControl(vw,7,0.3,7,0.1);

       LCDPutString("Me muevo ...\n");
       VWSetSpeed(vw, vel_tras, vel_rot);
       OSWait(50);
       VWSetSpeed(vw, 0.0, 0.0);
       OSWait(100);

       VWGetPosition(vw, &pos);
       LCDPrintf("\nx: %f\ny: %f\nphi: %f\n", pos.x, pos.y, pos.phi);

       LCDPutString("Estoy girando ...\n");
       VWDriveTurn(vw, angulo, vel_rot);
       VWDriveWait(vw);

       LCDPutString("Me traslado ...\n");
       VWDriveStraight(vw, distancia, vel_tras);
       VWDriveWait(vw);

       LCDPutString("Hago una curva ...\n");
       VWDriveCurve(vw, distancia, angulo, vel_tras);
       VWDriveWait(vw);
       LCDPutString("Me he cansado ...\n");

       VWStopControl(vw);
       VWRelease(vw);
        return 0;
}
```

En el fichero ***ejmplos.zip*** se dispone de una aplicación de ejemplo: (drive.c, drive.sim).