DESARROLLO DE SOFTWARE PARA SISTEMAS EMPOTRADOS

Práctica 3: Uso básico y configuración de motores

Héctor Pérez
Michael González

Informe

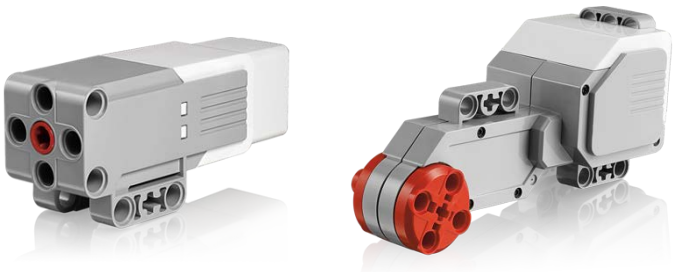
- Sección 1: Configuración del entorno de desarrollo
- Sección 2: Uso de dispositivos de entrada/salida
- Sección 3: Uso y configuración de motores
  - Objetivos
  - Gestión de motores en el kernel
  - Uso y configuración de motores
- Sección 4: Caracterización de la plataforma
- Sección 5: Desarrollo de un sistema empotrado
- Sección 6: Conclusiones

Objetivos

- Uso básico de motores
  - desarrollo de aplicaciones sencillas en C que nos permitan manejar motores
- Configuración de motores
  - obtención y ajuste de parámetros característicos en una aplicación particular
- Threads planificados con prioridades
  - configuración de atributos de los threads
- Anexo: Ejecución de threads periódicos planificados por prioridades

Introducción

- Motores EV3



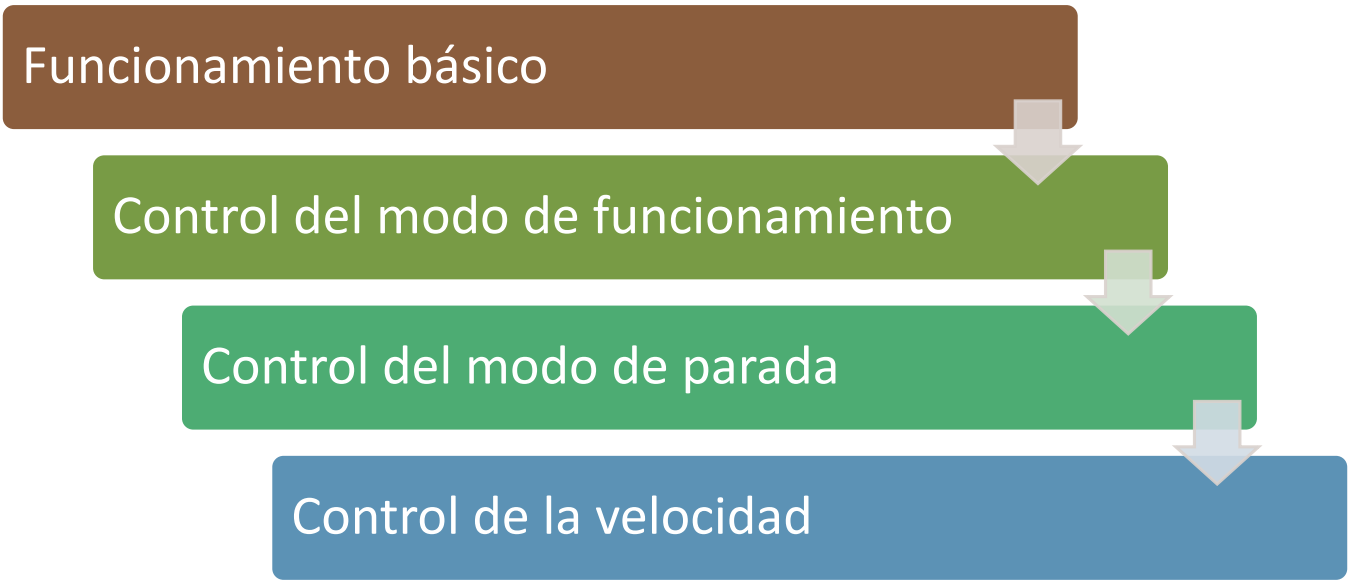
  - dispositivos que transforma energía eléctrica en energía mecánica
  - proporciona retroalimentación para un control preciso de posición y velocidad
    - sensor de rotación integrado
  - Posibles usos:
    - coches, cintas transportadoras, brazos robóticos, etc.

Introducción

- Gestión de dispositivos en el kernel *ev3dev*

/sys/class/tacho-motor/motor<N>	
Fichero	***Descripción***
commands	Listado de los comandos aceptados
stop_commands	Listado de los comandos de parada aceptados
count_per_rot	Número de unidades en una rotación completa del motor
position	Posición actual del motor en unidades del sensor de rotación
speed	Velocidad del motor en unidades del sensor de rotación por segundo
duty_cycle	Porcentaje de tiempo en un intervalo en el que se transmite energía al motor
state	Estado actual del motor (running, ramping, holding o stalling)
polarity	Sentido de giro
*_sp	Valores finales de las variables asociadas al terminar de ejecutar el comando adecuado

Uso básico y configuración de motores



Motores: funcionamiento básico

- Ejecutar la aplicación *practica3.c* proporcionada
  - indicar brevemente qué realiza el código propuesto
    - ¿qué función tiene el *usleep* utilizado en la aplicación?
    - ¿qué comprueba la siguiente operación en el código?

      ev3_motor_state (belt_motor) & MOTOR_RUNNING
- Se proporciona el script *reset_motors.sh* por si se pierde el control de los motores
  - ejecutarlo desde un terminal remoto para realizar una parada de emergencia

Motores: modos de funcionamiento (1/2)

- Los motores tienen diferentes modos de funcionamiento
  - **run-to-abs-pos**: activa el motor hasta la posición determinada por la variable *position_sp*
  - **run-to-rel-pos**: activa el motor hasta alcanzar una posición igual a la posición actual + *position_sp*
  - **run-timed**: activa el motor durante el tiempo indicado por la variable *time_sp*
  - **run-direct**: activa el motor hasta alcanzar la potencia indicada por la variable *duty_cycle_sp*

Motores: modos de funcionamiento (2/2)

- Control del modo de funcionamiento del motor
  - a partir del código del apartado anterior, crea una aplicación *practica3b.c* para que el motor realice 4 rotaciones completas
    - consulta el fichero *count_per_rot* para saber las unidades de una rotación completa del motor
    - configura el modo de funcionamiento apropiado
      - razona la elección del comando utilizado
    - muestra por pantalla la *posición final* del motor al finalizar la aplicación y razona los resultados obtenidos

Motores: modos de parada (1/2)

- Los motores de LEGO pueden operar con distintos modos de parada:
  - *coast*
  - *brake*
  - *hold*
    - solo es compatible con los modos de funcionamiento basados en el posicionamiento: **run-to-***
- El siguiente ejercicio permite conocer las diferencias de funcionamiento que hay entre ellos

Motores: modos de parada (2/2)

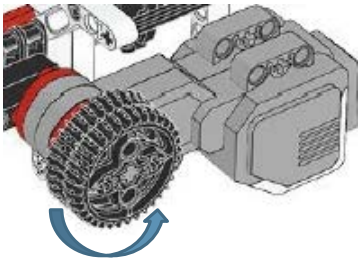
- Modifica el código de *practica3b.c* para configurar el modo de parada del motor
  - ejecuta la aplicación con los distintos modos de parada e indica la ***posición final real*** del motor en cada caso
    - razona si existen diferencias al mostrar la posición final del motor:
      - inmediatamente después de que el motor abandone el estado MOTOR_RUNNING
      - un segundo después de que el motor abandone el estado MOTOR_RUNNING
    - indica la diferencia de funcionamiento entre los modos de parada disponibles

Motores: control de velocidad (1/3)

- Los motores de LEGO tienen dos formas de controlar la velocidad del motor
  - mediante el parámetro *speed*
    - ev3_set_speed_sp (rango 0-max_speed)
      - *LARGE MOTOR*: aprox. 900 deg/sec
      - *MEDIUM MOTOR*: aprox. 1200 deg/sec
  - mediante el parámetro *duty cycle,* que representa el porcentaje de tiempo que se transmite energía al motor en un intervalo de tiempo determinado
    - ev3_set_duty_cycle_sp (rango 0-100%)
    - solo puede utilizarse con el modo de funcionamiento **run-direct**

Motores: control de velocidad (2/3)

- Crea una aplicación practica3c.c que mueva los motores durante 5 segundos y que controle la velocidad con el parámetro *duty cycle* :
  - utiliza el modo de funcionamiento **run-direct**
  - por sencillez, se puede utilizar sleep para suspender el thread y posteriormente parar el motor
  - configura el *duty_cycle* para que opere al 25% de la potencia
  - estima aproximadamente la velocidad media de giro
    - para ello, puede obtenerse el valor de la posición final y dividirlo entre el tiempo de operación del motor
- Aplica una pequeña resistencia al engranaje para simular una carga pesada
  - estima de nuevo la velocidad media de giro

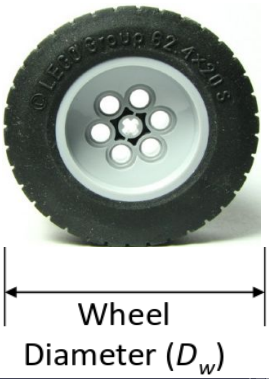


Motores: control de velocidad (3/3)

- Ejecuta la aplicación controlando la velocidad con el parámetro *speed*:
  - cambia el modo de funcionamiento a **run-forever**
  - establece una velocidad objetivo igual a la velocidad media de giro calculada al utilizar el *duty_cycle*
    - utilizar el primer valor obtenido (esto es, sin aplicar resistencia)
    - estima aproximadamente la velocidad media de giro
- Aplica una pequeña resistencia al engranaje para simular una carga pesada
  - estima de nuevo la velocidad media de giro
- Indica las posibles diferencias entre ambos métodos de control de velocidad

Motores: control preciso del movimiento

- Realiza el montaje de la **cinta transportadora** proporcionado en *Moodle*
  - se incorpora al montaje realizado en la práctica anterior
- Crea una aplicación ***cinta.c*** para que la cinta transportadora desplace un objeto parando en cada engranaje:
  - posteriormente, el objeto debe volver a su posición original
  - indica las configuraciones utilizadas para el motor
  - calcular la longitud de movimiento:
    - por cada rotación completa, la distancia recorrida viene dada por $L = \pi \cdot D_w$
    - el número π está definido en la librería matemática



Planificación por prioridades

- **Revisa** los apuntes de políticas de planificación en sistemas POSIX
  - *configuración previa* de atributos de planificación

```
int pthread_attr_setinheritsched (pthread_attr_t *attr,
                                  int inheritsched);
int pthread_attr_setschedparam (pthread_attr_t *attr,
                                const struct sched_param *param);
int pthread_attr_setschedpolicy (pthread_attr_t *attr,
                                 int policy);
```

  - *cambio dinámico* de atributos de planificación

```
int pthread_setschedparam (pthread_t thread,
                           int policy
                           const struct sched_param *param);
```

  - los apuntes utilizan la macro ***CHK (f)*** para mejorar la legibilidad del código
    - se incluye en el fichero de cabecera *misc/error_checks.h* proporcionado

Planificación por prioridades

- La asignación de prioridades en Linux **requiere** privilegios de administrador (***sudo***)
  - opciones para la ejecución remota con privilegios de administrador
    1. Ejecutar la aplicación en un terminal *ssh* externo
    2. *Ejecución en eclipse*: conectarse directamente con la cuenta de ***root***
       - cuenta de *root* desactivada por defecto en *ev3dev*
         - activación mediante el comando *sudo passwd*
       - configurar *ssh* para permitir el acceso mediante la cuenta de *root*
         - opción *PermitRootLogin* en el fichero de config. /etc/ssh/*sshd_config*
         - reiniciar el servicio *ssh* mediante *sudo systemctl restart ssh*
       - configurar la conexión con el usuario *root* en *eclipse* y reiniciar el IDE

Aplicación planificada por prioridades (1/2)

- A partir del fichero **cinta.c**, crear una aplicación **cinta_sched.c** que configure el thread que controla el motor (*thread principal*) para:
  - utilizar una política de planificación basada en prioridades fijas
    - en un sistema POSIX, la política se denomina **SCHED_FIFO**
  - ejecutar con una prioridad igual a 20
- Se proporciona un fichero zip con herramientas adicionales para simplificar el desarrollo
  - macro ***CHK (f)*** de ayuda para la gestión de errores en ***misc/error_checks.h***
  - herramienta de monitorización ***rtinfo_threads.sh***

```
include
├── misc
│   ├── error_checks.h
│   └── timespec_operations.h
├── ev3c_battery.h
├── ev3c_button.h
├── ev3c_core.h
├── ev3c.h
├── ev3c_lcd.h
├── ev3c_led.h
├── ev3c_motor.h
└── ev3c_sensor.h
```

Aplicación planificada por prioridades (2/2)

- La herramienta ***rtinfo_threads.sh*** permite obtener información sobre los threads de un proceso en ejecución

  rtinfo_threads.sh process_name

  - la herramienta monitoriza el sistema hasta que encuentra un proceso en ejecución con el nombre indicado
  - posteriormente obtiene la información de planificación POSIX y LINUX
- Ejecutar la herramienta para obtener información sobre la aplicación original ***cinta*** y la aplicación ***cinta_sched***
  - comprueba que se ha modificado la prioridad y política de planificación en cinta_sched
    - ejecuta *rtinfo_threads en un terminal del robot y la aplicación a monitorizar en eclipse u otro terminal del robot*
    - si no obtienes datos, puede ser neceario añadir una suspensión al código una vez realizada la configuración del thread

Anexo: Aplicación periódica planificada por prioridades

- **Revisa** los apuntes sobre la gestión de tiempo en sistemas POSIX
  - se proporcionan un conjunto de *macros* para facilitar las operaciones con el tipo de dato *timespec*
    - se incluye en el zip de recursos (fichero de cabecera *misc/timespec_operations.h*)