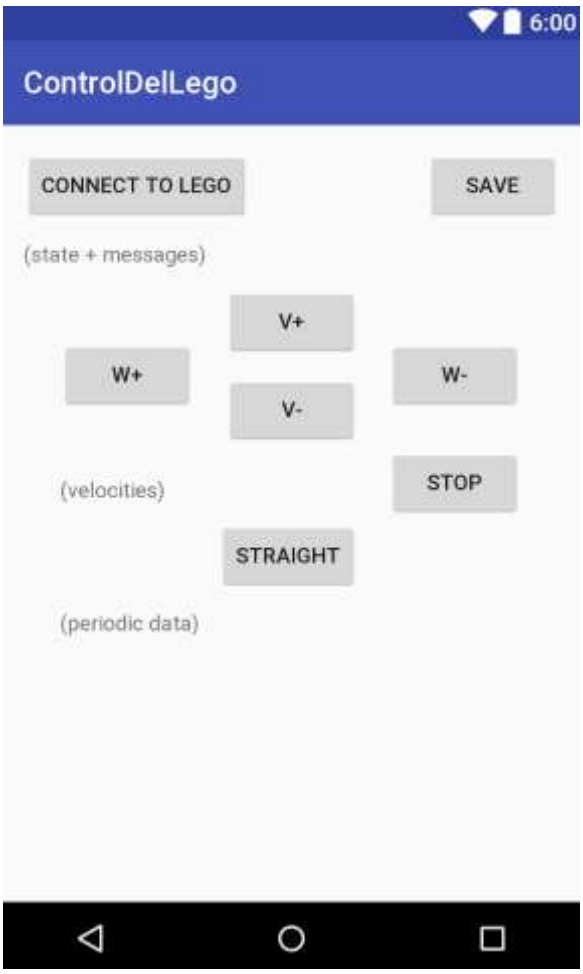
Práctica 7: Diseño de aplicaciones de tiempo real para dispositivos móviles

[deadline: 30 de enero]

Apartado 1. Diseño de la máquina de estados de Mealy y requisitos de tiempo.

Diseña una *app* en base a una máquina de estados de Mealy para el *thread* principal y el número de *threads* secundarios que estimes oportunos, que, con unos *deadlines* de tiempo real suave razonables elegidos por ti, permita al usuario enviar comandos simples de movimiento a un robot de Lego de forma remota a través de Bluetooth. La aplicación debe tener los siguientes elementos en su actividad:



El comportamiento de la *app* debe ser el siguiente:

- Todos los botones deben responder a las pulsaciones del usuario, aunque no hagan nada porque la *app* no esté en el estado adecuado para ello.
- El usuario debe comenzar pulsando el botón "CONNECT TO LEGO" (el resto no deben hacer nada en esta fase inicial). Una vez pulsado, la *app* debe abrir una conexión Bluetooth directa con un robot concreto (sin hacer descubrimiento de dispositivos) del que se conozca su MAC.
- Cuando la conexión se haya establecido exitosamente, el botón "CONNECT TO LEGO" debe cambiar su leyenda a "DISCONNECT FROM LEGO" (usa el método `setText()` para ello). A partir de ese momento, si el usuario pulsa el botón, se debe cerrar la conexión con el robot y volver a poner "CONNECT TO LEGO" por si se quiere reconectar en el futuro.
- Si la conexión no se establece correctamente, debe escribirse en la caja de texto "(state + messages)" un error explicativo de las causas. El usuario podrá volver a intentar conectar en el futuro.
- Si la conexión está establecida correctamente, cada vez que el usuario pulse uno de los botones de cambio de velocidad del robot, tanto angular (W) como lineal (V), se deben enviar comandos al robot para cambiar las dos potencias de las ruedas. La rutina que convierte una pareja (W,V) de velocidades angular y lineal en la potencia necesaria para una rueda es la siguiente:

```
byte powerForWheel(boolean leftorright, double v, double w)
// return the power to give to the wheel motor in order to get those velocities
// only works well if the robot has its battery full!
{
   final double d = 0.115; // distance between wheels
   final double r = 0.03; // wheel radius

   double vwheel; // m/s  -> linear speed of the center of the wheel on the plane
   if (leftorright) // left wheel
      vwheel = v - d * w;
   else // right wheel
      vwheel = v + d * w;

   double alpha = vwheel / r; // -> rad/s -> angular speed of rotation for the wheel

   double power;
   if (Math.abs(alpha)<1e-6) power=0.0;
   else power = (alpha - 0.031571)/0.14149;

   if (power<-100.0) return(-100);
   else if (power>100.0) return(100);

   return((byte)Math.round(power));
}
```

- Los dos comandos de potencia que se envían al robot al pulsar los botones mencionados en el punto anterior deben enviarse justo uno a continuación del otro, sin esperar las respuestas hasta que no hayan sido enviados ambos. Una vez enviados ambos, deben esperarse las dos respuestas.
- Para manejar los mensajes de comandos y respuestas con el robot, así como la conexión con el mismo, tienes disponible dos ficheros que puedes descargar de la web de la asignatura (`Command.java` y `Response.java`); además, se te proporciona allí el código de una clase auxiliar `SharedSocket` que hace el manejo más cómodo de estas comunicaciones (es la clase que deberías usar).
- Las velocidades V,W deben estar entre ciertos rangos para que el robot pueda ejecutarlas, y los incrementos/decrementos de las mismas producidos al pulsar los botones deben ser los siguientes (puedes cambiar estos incrementos más adelante si crees que así funcionará mejor, pero las velocidades máximas no las cambies):

```
private final static double MAXABSV = 0.4; // max linear speed of Lego
private final static double MAXABSW = 7; // max angular speed of Lego
private final static double ABSINCV = MAXABSV / 5.0;
private final static double ABSINCW = MAXABSW / 20.0;
```

- Si hay errores en el envío de comandos o en la recepción de respuestas, debes escribirlos en "(state + messages)" y dar el envío de los comandos por terminado.
- El botón "Stop" funciona análogamente a los botones de velocidad, sólo que envía potencias 0 a ambas ruedas.
- El botón "Straight" funciona análogamente a los botones de velocidad, sólo que deja V como está y pone W a 0 antes de calcular las potencias a enviar a las ruedas.
- Todos los botones que cambien las velocidades V,W del robot deben actualizar el valor de las mismas en la caja de texto "(velocities)".
- Mientras haya conexión correcta con el robot, la *app* debe enviar periódicamente, cada medio segundo, peticiones de lectura del estado de los dos motores de las ruedas (en este caso, envía un comando, espera su respuesta, envía otro comando y espera su respuesta), mostrando en el texto "(periodic data)" los valores de los *encoders* de ambas ruedas. Mientras la *app* esté realizando estas peticiones periódicas, los botones de control de movimiento no deben tener ningún efecto.
- El botón "SAVE" se podrá utilizar en cualquier momento para grabar en un fichero de texto legible por Matlab datos de tiempos tardados en ejecutar diversos trozos de código. Por tanto, los trozos de interés deberán medir sus tiempos de ejecución y guardarlos en memoria para que el botón pueda recuperarlos y grabarlos en fichero, de forma análoga a como lo hiciste en la práctica 5. **NOTA**: También puedes usar el botón "SAVE" para grabar un *log* del programa, lo que puede ser muy útil en depuración; ese *log* debes generarlo tú, sin usar las rutinas de *logging* de Java, ya que no podremos ejecutar la *app* en simulación debido al Bluetooth.
- La *app* debe crear todas las variables de interés en el evento `onRestore()` (no en el `onCreate()`) y cerrarlas en el evento `onStop()` . Esto último incluye también la conexión Bluetooth si está activa.
- Los métodos de actividad que sobreescribas (`onCreate()` , `onStop()` , ...) deben llamar, en primer lugar, a los de la clase padre: `super.onCreate()` , `super.onStop()` , ...

Apartado 2. Implementación y verificación del software.

Escribe el código de tu *app* a partir del diseño del apartado 1, y comprueba con el botón "SAVE" qué tiempos de cómputo (distribuciones de probabilidad) tienen al menos las siguientes partes de la misma:

- Envío de comandos al robot y recepción de respuestas (tanto de movimiento como de consulta de estado de las ruedas).
- Establecimiento de conexión con el robot.

Justifica en base a las distribuciones de probabilidad obtenidas si tu *app* cumple los requisitos de tiempo real que especificaste en el diseño. Si tienes que hacer algún cambio para que lo hagan, explícalo igualmente.

¿Hasta qué punto puede cambiarse la periodicidad de recogida de datos de los motores? ¿Cómo habría que cambiar los *deadlines* en el diseño?

Ilustra este apartado con capturas/fotos de la aplicación y un vídeo del robot siendo manejado con la misma por el usuario. Incluye asimismo el código completo de la actividad en la *wiki*.

Última modificación: lunes, 19 de diciembre de 2016, 13:06