

ROBOT: Daneel

Carlos Tardón Rubio
Marcos Herrero Agustín
Martín Fernández de Diego

Resumen

Este documento expone los pasos seguidos para la construcción y programación del robot (bautizado como Daneel) que constituye el proyecto final para la asignatura de Robótica de sus tres autores. El robot ha de ser capaz de desplazarse y, utilizando los estímulos de su entorno, superar los retos propuestos por el profesor.

1. Objetivos

- Construir un robot capaz de desplazarse sobre sus ruedas y de detectar estímulos externos por medio de los sensores CNY70 y GP2D12.
- Dotar al robot anterior del código necesario que le permita superar los retos de *Seguir la Línea* y del *Laberinto*.

2. Construcción del robot

En esta sección, se explicarán los pasos seguidos para acoplar los componentes del robot (Raspberry, placa de conexiones, baterías, motores y sensores) sobre la plataforma elegida, así como las causas y consecuencias de las distintas modificaciones sufridas.

Durante todo el proceso de diseño y construcción, se han ido proponiendo y efectuando modificaciones en el robot en base, principalmente, a las siguientes finalidades generales:

- Conseguir una arquitectura estable, modular y asequible
- Conseguir un movimiento fluido del robot
- Conseguir una detección fiable y suficientemente precisa de los estímulos

Se hablará, pues, de los cambios sufridos por el robot a lo largo del proceso, por qué se decidió aplicarlos y por qué, algunos de ellos, fueron finalmente descartados.

2.1. Planteamiento inicial

Desde un principio, nuestro objetivo fue abastecernos del material necesario para el robot de lo que tuviéramos por casa para, de esta forma, con materiales reciclados, abaratrar al máximo el coste del robot.

Tras una investigación inicial en la que se trajeron varios ejemplos de Internet, barajamos dos modelos de robot: uno que usaba un táper como chasis y otro que sostenía los componentes sobre una chapa. Por supuesto, en ambos casos el robot tendría una planta rectangular, con el fin de evitar artificios poco beneficiosos en plantas con otras geometrías.

Usar un táper como chasis del robot tendría la ventaja de, por ejemplo, poder atornillar los motores en las paredes laterales y de poder cubrirse para proteger u ocultar los componentes internos del robot. Pero temíamos que el grosor de sus paredes nos diera problemas de estabilidad y que los bordes redondeados que tienen la gran mayoría nos supusiera complicaciones en el futuro.

Sin embargo, finalmente nos decantamos por utilizar una placa de contrachapado que teníamos sobrante de un proyecto anterior, porque pese a perder la superficie de los laterales que nos brindaba un táper, solucionábamos los problemas de estabilidad y obteníamos una base sin complicaciones en su forma. En aquel momento, también pensamos que su poco peso jugaría en nuestro favor. Hubo que cortarla un poco para que tuviera las dimensiones adecuadas, de 20cm × 15cm.



Figura 1: Grosor del contrachapado

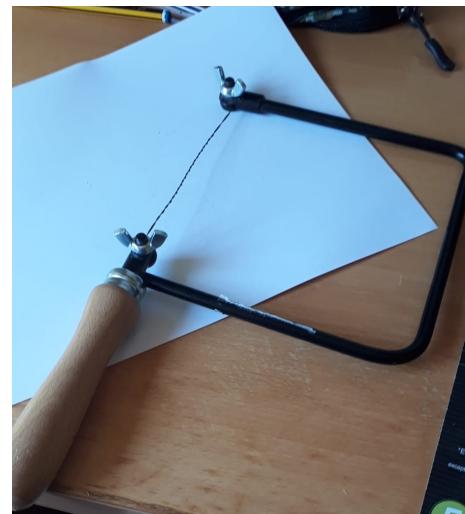


Figura 2: Segueta para cortar contrachapado

Otros elementos de los que ya disponíamos (aparte de los proporcionados por los laboratorios) eran algunas PowerBank, aunque no estábamos seguros de que fueran adecuadas, y cables de conexión USB.



Figura 4: PowerBank, con botón para cortar el suministro e impedir que se malgaste la energía.

Figura 3: Salida 5v

Partiendo de esta decisión, comenzó un proceso de debate de cómo anclar los elementos a la tabla.

2.2. Daneel 1: Robot de bridas

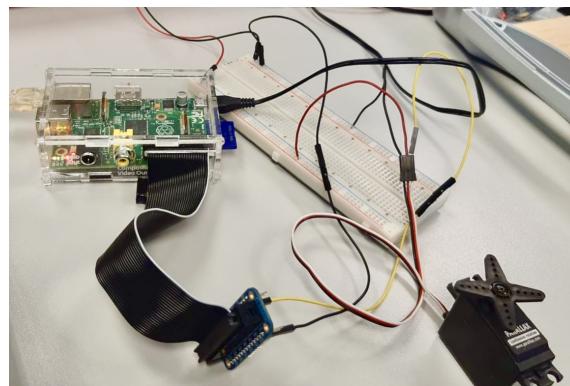


Figura 5: Probando el circuito de los servomotores, antes de instalarlos en el robot

Los motores parecían pensados para acoplarse al chasis mediante tornillos pero, dada la naturaleza de nuestra plataforma, esto no parecía una opción (material demasiado fino para atornillar y ausencia

de paredes verticales). Por tanto, decidimos utilizar briduras para colocar los motores y mantenerlos alineados. Con este tipo de sujeción conseguíamos acoplar todos los elementos al soporte sin realizar modificaciones permanentes sobre el mismo. De esta idea, surgió la primera versión de nuestro robot: **Daneel 1, el robot de briduras**.

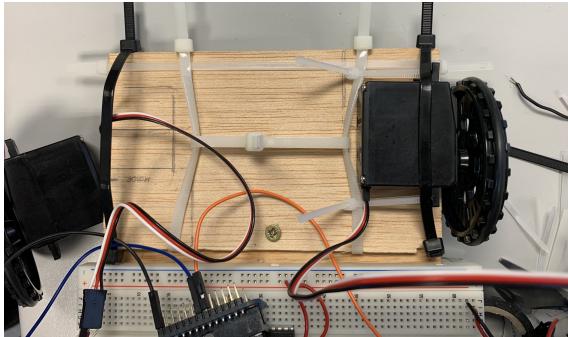


Figura 6: Planta del modelo de briduras con un motor anclado

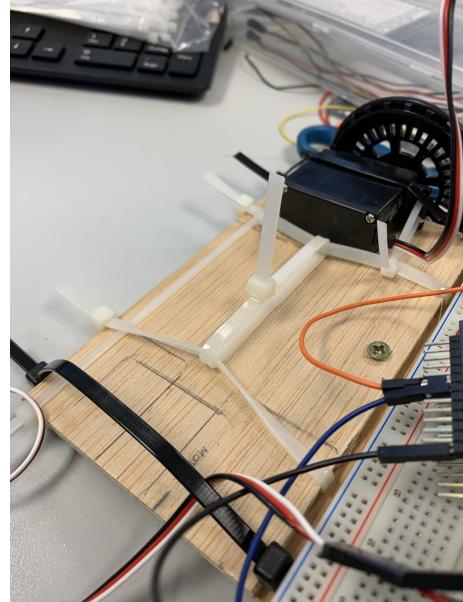


Figura 7: Perspectiva del modelo de briduras con un motor anclado

El acoplamiento de los motores al soporte se realizó con una tabla auxiliar que, sujetada a la principal, los amarraba a la estructura sin necesidad de realizar agujeros que supusieran una modificación irreversible en la chapa o daños en los componentes. De esta forma, conseguíamos anclar los motores con suficiente estabilidad.

La posición de la rueda libre o rueda loca fue analizada detenidamente considerando las dos principales opciones: colocarla en la parte trasera y que las ruedas tractoras fueran en la delantera o viceversa. Finalmente, la rueda loca se colocó en la parte inferior trasera del robot porque así podríamos minimizar una salida errática del robot si la rueda loca no estuviera alineada con la trayectoria planificada. Se sujetó al chasis con una potente cinta adhesiva de doble cara porque los tornillos habrían atravesado la placa.

La Raspberry Pi, la placa protoboard de conexiones y las baterías se sujetaron sin complicaciones extra a la base.

Este primer modelo no llegó a implementar ningún sensor en su arquitectura dado que nos encontramos con fallos previos que solucionamos, antes de continuar el montaje, en una segunda versión.

2.2.1. Problemas encontrados

El robot de briduras pretendía principalmente evitar, de la forma más sencilla posible, daños en los componentes prestados y asegurar que se pudieran deshacer construcciones no acertadas. Pero esto nos abrió otros problemas.

- Elevaba considerablemente la dificultad de realizar leves cambios en las posiciones de los motores y con ello disminuía su modularidad, uno de los objetivos iniciales de nuestra implementación.

- Aumentaba el uso de artificios como planchas auxiliares o bridas de refuerzo.

2.2.2. Material y componentes del modelo

- Prestado por los laboratorios
 - Raspberry Pi 3B+
 - 2 × Servomotor Parallax
 - 2 × Sensor CNY70
 - 2 × Sensor GP2D12
 - Placa de conexiones protoboard de 165mm × 55mm × 10mm
 - Cables MM / Cables MH / Cables HH
- Reutilizado
 - Baterías portátiles con salida de 5V y 1A
 - Plancha de contrachapado
 - Bridas pequeñas de 10cm
 - Tornillo de 10mm
 - Cinta adhesiva de doble cara
 - Cinta aislante negra
 - Destornillador
 - Alicates
 - Tijeras
- Comprado
 - Rueda loca de 30mm: 1,59€
 - Paquete de bridas de 20cm: 2,60€
 - Paquete de bridas de 30cm: 2,90€

En total, el precio del robot de bridas ascendió a 7,09€.

2.3. Daneel 2: Robot de velcro

La Raspberry y la placa de conexiones proporcionadas por el laboratorio ya tienen pegada una tira de cinta de fijación¹, así que decidimos adquirir más y utilizarla para pegar estos elementos a la plataforma. Fue un poco problemático encontrarla, ya que no sabíamos dónde se podía comprar algo así (ni siquiera sabíamos con qué nombre pedírsela a los dependientes). Finalmente, lo obtuvimos en Leroy Merlin, donde pudimos encontrarlo gracias a enseñarle la muestra que teníamos. El precio fue mucho más caro de lo esperado, con lo que se acabaron nuestras esperanzas de hacer un robot barato. Pero al final hemos rentabilizado la compra, ya que este velcro ha sido el elemento de sujeción insignia del robot.

En este modelo ya pudimos incorporar los sensores. Antes de integrarlos en el diseño del robot, los montamos por separado, para asegurarnos del correcto funcionamiento del circuito, como se observa en las siguientes imágenes:

¹De aquí en adelante nos tomaremos la licencia de llamarla velcro. Aunque este no es el nombre correcto, es conciso y es el que hemos usado entre nosotros para referirnos a este elemento durante el desarrollo del proyecto

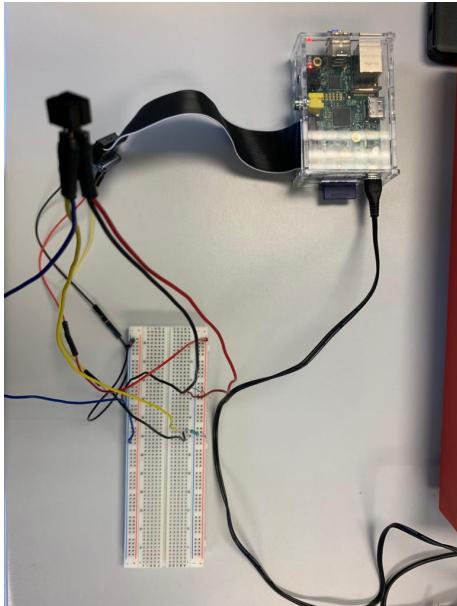


Figura 8: Sensor cny70, antes de incorporarlo al robot Daneel

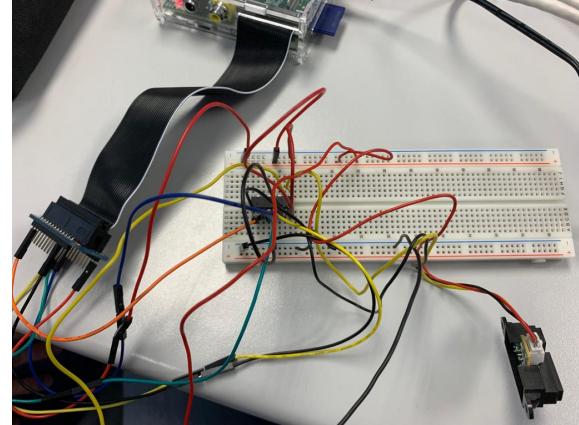


Figura 9: Sensor GP2D12, antes de incorporarlo al robot Daneel

En cuanto al GP2D12, en sensor de medición de distancia, tuvimos que realizar unas mediciones iniciales, anotando los voltajes para cada una de las distancias posibles. Una vez hecho esto, graficamos los puntos obtenidos, y probamos distintas formas de interpolación, como se ve en las siguientes figuras. Al final, decidimos quedarnos con la interpolación por segmentos lineales, ya que es muy sencilla y da buenos resultados.

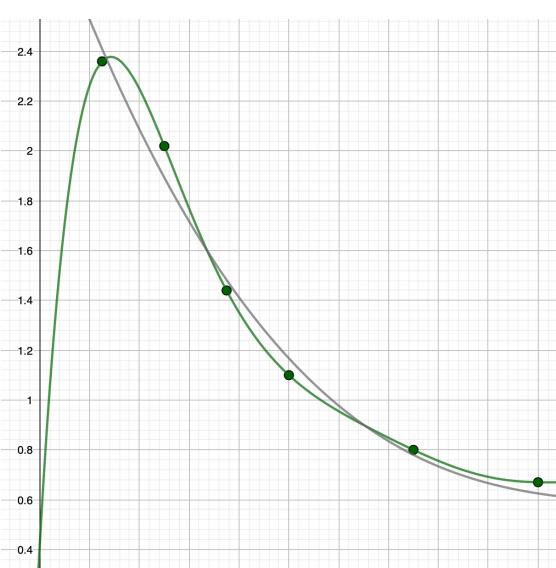


Figura 10: Interpolación polinomial

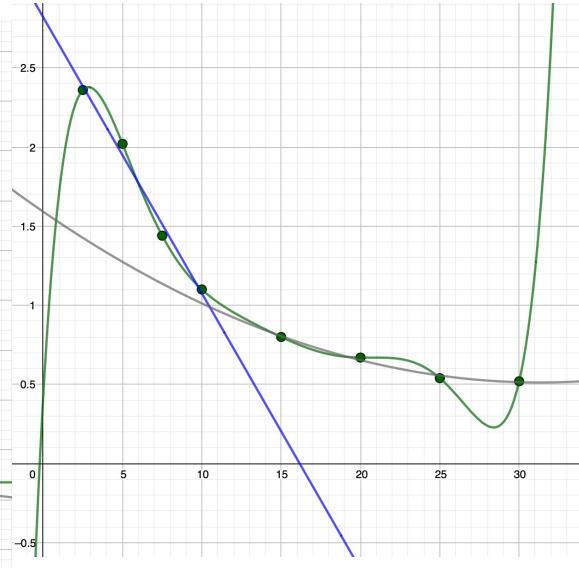


Figura 11: Distintas interpolaciones: polinomial de distintos grados, y lineal

Tras terminar todo el montaje, el robot había conseguido ser totalmente modular sin llegar a realizar modificaciones permanentes en ningún elemento. Así, el robot de velcro se puede desmontar y montar completamente sin necesidad de ninguna herramienta.

Este nuevo modelo sigue la línea perfectamente. Aún así, tenía un ligero problema, y es que la pista, al ser tan lisa, hacía que las ruedas derraparan. El problema era el balance de pesos, pues como se ve en la figura, la Raspberry y la PowerBank (debajo de esta) hacían que el peso apoyara sobre la rueda loca, y no sobre las ruedas motrices. La flecha verde indica el rediseño que planteábamos hacerle:

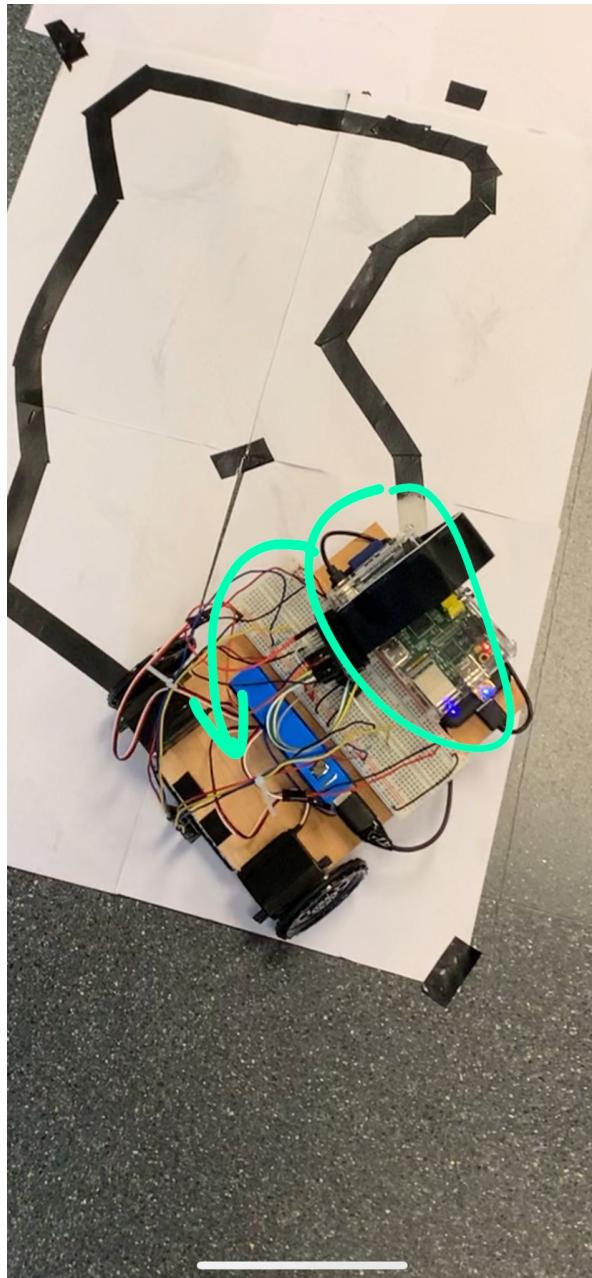


Figura 12: Robot

Tras el rediseño, el robot quedó así:

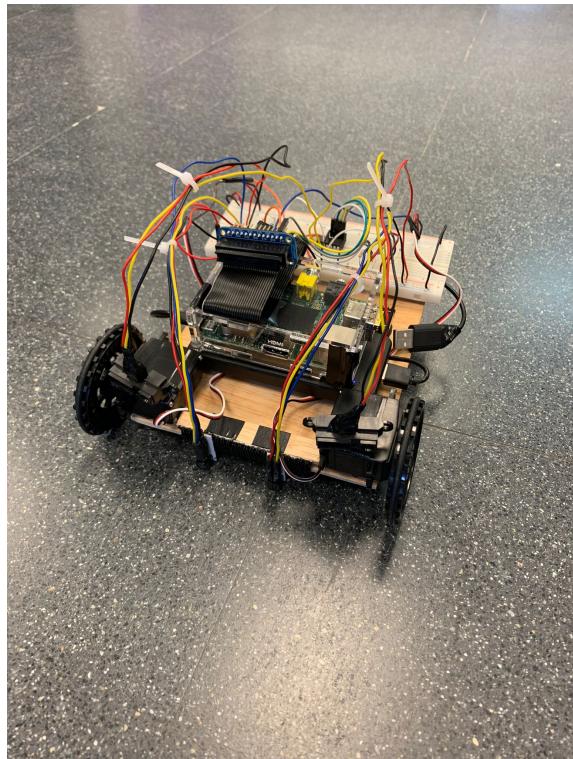


Figura 13: Resultado final del robot.

2.3.1. Material y componentes del modelo

- Prestado por los laboratorios
 - Raspberry Pi 3B+
 - 2 × Servomotor Parallax
 - 2 × Sensor CNY70
 - 2 × Sensor GP2D12
 - Placa de conexiones protoboard de 165mm × 55mm × 10mm
 - Cables MM / Cables MH / Cables HH
- Reutilizado
 - Baterías portátiles con salida de 5V y 1A
 - Plancha de contrachapado
 - Bridas pequeñas de 10cm
 - Tornillo de 10mm
 - Cinta adhesiva de doble cara
 - Cinta aislante negra

- Destornillador
 - Alicates
 - Tijeras
- Comprado
- Rueda loca de 30mm: 1,59€
 - Rollo de velcro de 1m: 16,79€

En total, el precio del robot de velcro ascendió a 18,38€.

Pese a que el coste del robot de velcro casi triplica al coste del robot de bridás, la cuantía sigue siendo correcta dado el ámbito y la magnitud del proyecto y las mejoras de este nuevo diseño han supuesto un claro avance en modularidad y fluidez de trabajo.

3. Programación del robot

La programación del robot se realizó en paralelo con su construcción. Dado que estamos mucho más acostumbrados a trabajar con software que con hardware, nos resultó mucho más sencillo que esta. No obstante, sí que hemos tenido problemas de que un código que en abstracto no tiene ninguna pega da problemas al llevarlo al mundo real.

Si bien contábamos con un programa inicial para mover al robot en línea recta, los programas del robot que conservamos son los que le permiten realizar los retos de Seguir la Línea y del Laberinto.

3.1. Programa de Seguir la Línea

Como sabemos, el reto del siguelíneas consiste en que el robot capte una línea negra sobre fondo blanco por medio de los sensores CNY70 y la siga sin perderse lo más rápido posible. A la hora de programar esto existen dos posibilidades: buscar que los CNY70 lean constantemente la línea negra (versión negro-negro) o buscar que lean constantemente el blanco lateral (versión blanco-blanco). Por supuesto, para usar seguir el paradigma negro-negro los sensores CNY70 deben situarse mucho más juntos que para seguir el blanco-blanco, pero esto no resulta problemático en nuestro caso porque nuestros CNY70 se pueden recolocar muy fácilmente por medio de velcro gracias a su modularidad. Nosotros hemos probado ambos enfoques y valorado sus ventajas e inconvenientes. Aquí presentamos el código para los dos, pero para la realización del examen hemos decidido usar blanco-blanco, que, en general, permite mayor velocidad.

En ambas versiones, la lectura de los valores de los sensores se realiza en *Threads* separados. Esto permite que el hilo principal lleve a cabo las operaciones necesarias para el control del robot.

Versión negro-negro

Esta fue la primera forma en que intentamos resolver el Sigue la Línea, inspirados por la idea inicial que teníamos del robot siguelíneas. Es decir, esto consiste en colocar los CNY70 muy juntos entre sí, menos de 8 mm. En cuanto a código, lo que hay que hacer es ir rectos cuando leamos negro-negro (de ahí el nombre de esta versión del robot), girar a la derecha si leemos blanco en el izquierdo (pues nos hemos salido), y viceversa para la izquierda. El primer problema que surge en este aspecto es qué hacer cuando el robot se sale completamente en una curva y lee dos blancos simultáneamente. El enfoque que nosotros dimos fue dar marcha atrás hasta volver a leer la cinta, pues esto garantizaría volver al camino que estábamos siguiendo.

Con este enfoque, a nuestro robot le costaba mucho hacer los giros, pues la cinta negra suele tener una anchura pequeña, así que tenía que rectificar muchas veces (al salirse de la cinta), y detectamos que eso era un problema. Por ello, hicimos un algoritmo para **predecir el giro** que tendría que realizar el robot, inspirados por los predictores de salto que tienen todos los microprocesadores. De esta forma, se guarda el último giro que se ha realizado en una variable *last*, donde 0 indica que no hay predicción disponible, 1 indica que el último giro es hacia la izquierda y 2 derecha. Entonces, cuando nos salimos de la cinta, es decir, cuando leemos dos blancos, en vez de dar marcha atrás (lo que ralentiza el giro); se pone en marcha el predictor de giro. Si el último giro fue hacia la derecha, giramos hacia la derecha, lo cual tiene sentido pues significa que estamos en una curva hacia la derecha. Ídem hacia la izquierda. Si no tenemos predicción (*last*=0) no nos queda más remedio que dar marcha atrás, pues nos habíamos salido. El código del predictor es el siguiente:

```

int last = 0
if(der == 1 && izq == 1){ //ambos negros , seguimos rectos
    softPwmWrite(MotRight,13);
    softPwmWrite(MotLeft,16);
    last = 0;
} else if(der == 1 && izq == 0){ // derecho se ha salido , giramos a la izquierda
    softPwmWrite(MotRight,13);
    softPwmWrite(MotLeft,13);
    last = 1;
} else if(der == 0 && izq == 1){ // izquierdo se ha salido , giramos a la derecha
    softPwmWrite(MotRight,16);
    softPwmWrite(MotLeft,16);
    last = 2;
} else{ //Ambos blancos , nos hemos salido . Prediccion de giro
    if(last == 1){
        softPwmWrite(MotRight,13);
        softPwmWrite(MotLeft,13);
        last = 0;
    } else if(last == 2){
        softPwmWrite(MotRight,16);
        softPwmWrite(MotLeft,16);
        last = 0;
    } else{
        softPwmWrite(MotRight,16);
        softPwmWrite(MotLeft,13);
    }
}

```

El principal problema de este método de predicción es el mismo que presentan los predictores de saltos en los que nos basamos para idearlo: los saltos mal predichos (*misprediction*). Esto en la práctica supuso que el robot en algunas (muy pocas) ocasiones iba hacia el lado contrario del que debía ir, lo que suponía perder la pista negra. Esto se puede solucionar, y de hecho lo que hacen los microprocesadores modernos es tener sistemas muy eficientes de predicción, con una rápida detección de los *misprediction*, a veces usando varios tipos de predictores al mismo tiempo. Valoramos esta posibilidad, y teniendo en cuenta el limitado tiempo del que disponíamos, decidimos desecharla e ir por otras vías de investigación distintas a esta: leer blanco-blanco.

El código completo de esta fase puede verse en [5](#).

Versión blanco-blanco

Esta segunda versión consiste en leer el blanco que rodea a la cinta negra. Nos aseguramos de separar más los sensores CNY70 (como hemos dicho, esto no nos supone ningún problema, pues nuestro diseño es totalmente modular), unos 18 mm entre sí. Esto permite que la cinta negra quede entre medias de los dos sensores. De esta forma, si leemos dos blancos, significa que debemos avanzar. Si leemos blanco-negro, o negro-blanco, deberemos girar de tal forma que corrijamos el error. De este modo, el número de veces que tenemos que corregir es mucho menor que en la versión negro-negro, pues ya no damos marcha atrás, y la separación entre los sensores permite suficiente espacio como para no leer negro constantemente (lo que produciría giros que lastrarían el avance).

El código final puede verse en [4](#)

3.1.1. Programa del Laberinto

Este programa es fundamentalmente distinto a los anteriores, ya que la prueba que tiene que resolver es otra muy distinta. En este caso, el robot se sumerge en un laberinto formado por paredes, y tiene que escapar del mismo utilizando únicamente los sensores GP2D12 que tiene a ambos lados. En nuestro caso, los sensores no miran al frente, sino que cada sensor está levemente girado hacia fuera, lo que permite detectar las paredes del laberinto de forma más precisa.

El funcionamiento de nuestro programa es muy simple: cuando se encuentra una pared por la derecha, gira a la izquierda (para dejar un cierto margen entre la pared y el robot) y luego sigue avanzando. Cuando se encuentra una pared por la izquierda, gira a la derecha (para dejar un cierto margen entre la pared y el robot) y luego sigue avanzando. En caso de encontrarse una pared de frente, hace un giro a ciegas: prueba con la derecha, y si hay espacio sigue avanzando. Por último, si se encuentra un objeto muy cerca (menos de 10 cm), el robot se para, pues en ese caso no sería seguro girar para intentar esquivarlo.

Con este planteamiento, al robot le costaba girar, pues debido a la posición de los ejes de las ruedas (muy adelantados), al girar desplazaba mucho su parte trasera, y se golpeaba con los objetos. Así, decidimos hacer un cambio de última hora: girar los motores para que su eje estuviera un poco más cerca del centro del robot. El resultado se puede ver en las dos figuras siguientes:

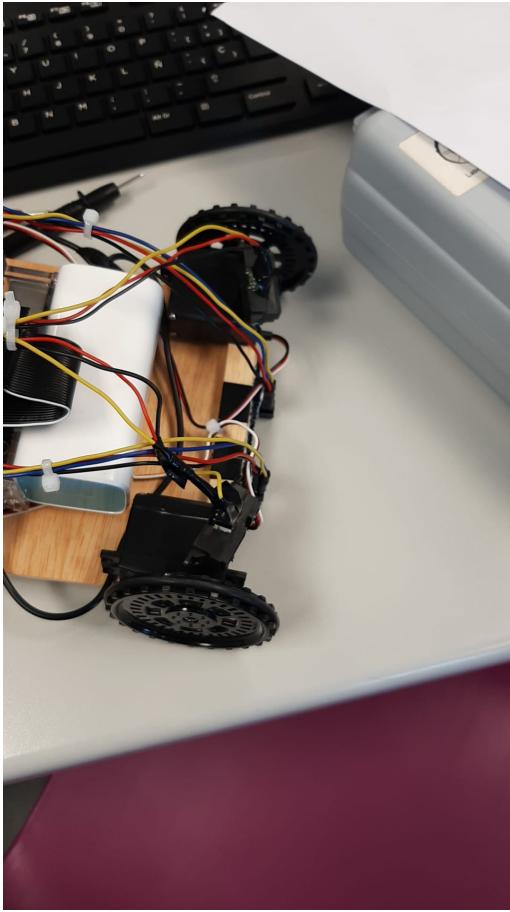


Figura 14: Motores antes del cambio

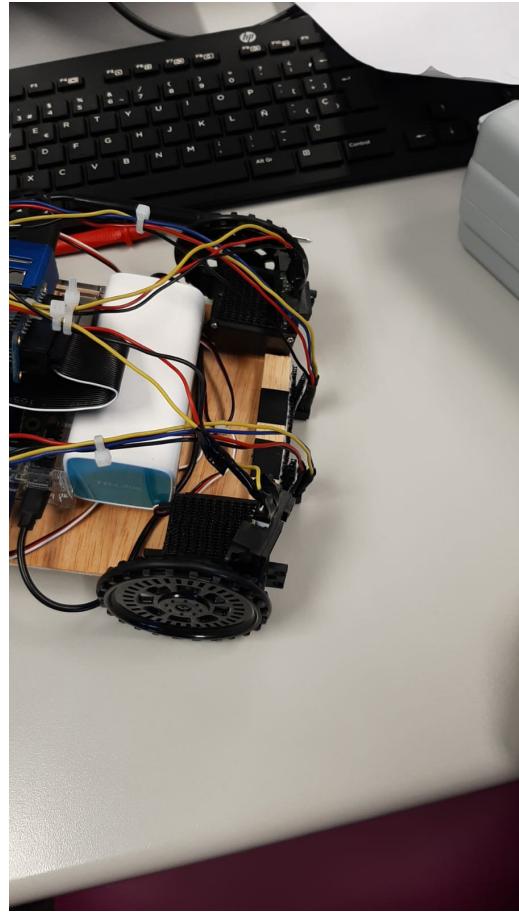


Figura 15: Motores después del cambio

El código final del robot para el laberinto puede verse en [6](#)

3.1.2. Problemas encontrados

Aunque la programación de Daneel dio muchos menos problemas que su construcción, sí que dio algunos. Los citamos a continuación:

- En una fase temprana del desarrollo del robot, no éramos capaces de conseguir que el robot se moviera en línea recta por mucho tiempo, siempre se desviaba. Aunque el desplazamiento en línea recta más allá de unos pocos metros no era algo que se pidiera como objetivo de la práctica, considerábamos que era importante. El problema se debía a que, como cada rueda debía girar en un sentido distinto, no había forma de conseguir que rotaran a exactamente la misma velocidad, por lo que el robot siempre se desviaba. Estuvimos muchas horas cambiando el número escrito en cada uno de los motores y haciendo pruebas por los pasillos de la facultad. Inicialmente intentábamos poner ambas ruedas a la misma velocidad y, cuando vimos que no se podía, intentamos corregir el movimiento cada cierto tiempo para compensar la desviación que el robot sufría de manera natural. Finalmente, nos dimos por vencidos sin haber logrado nuestro objetivo. Actualmente, después de haber cambiado las ruedas, la forma de sujetar los motores y la colocación

de los componentes sobre el robot (varias veces) va claramente mejor que entonces, es decir, aguanta mucho más sin desviarse. Pero sigue sin ser capaz de recorrer una distancia grande en perfecta línea recta. En cualquier caso, este episodio nos sirvió como enseñanza de que a menudo no merece la pena intentar corregir un hardware defectuoso (o que simplemente no presta la funcionalidad deseada) mediante software.

- Cuando Daneel ya era capaz de seguir la línea perfectamente y a buena velocidad, hubo un día que nos lo encontramos mucho más lento de lo normal. Durante un momento pensamos que podía deberse al hecho de tener múltiples threads, pero quitarlos no parecía afectar. Al final se debía a que uno de nosotros había puesto `printf`s en el código para informar por pantalla de los datos leídos por los sensores en cada momento, y eran estas instrucciones las que estaban lastrando el rendimiento. Por supuesto, los quitamos y volvió a funcionar perfectamente.
- Tras un uso prolongado (hora y media o más) el robot se nos desconectaba del WiFi automáticamente. Sin embargo, el aparato USB que usábamos para conectar la Raspberry al WiFi (aunque dañado) funcionaba perfectamente. Por ello, decidimos probar con una batería de mayor capacidad, por si el causante de este problema era el sobreesfuerzo al que le sometíamos, y resultó ser ese el problema.



Figura 16: Nueva PowerBank

Referencias

4. Anexo A: Código blanco

```
#include <stdio.h>
#include <wiringPi.h>
#include <time.h>
#include <sys/time.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <string.h>
#include <softPwm.h>
#include <wiringPiSPI.h>
#include <mcp3004.h>

// LED Pin - wiringPi pin 0 is BCM_GPIO 17.

#define Tldr    100
#define Tcny   30
#define MAXTIME 3000000 // 10s en microsegundos
#define LDR    0
#define CNY    1
#define MotLeft 2
#define MotRight 3
#define RANGE          100
#define NUM_LEDS        1
#define BASE 100
#define SPI_CHAN 0

int Vgp2 = 0;
int Vcny = 0;
int cnyRight; //blanco 1 negro 0
int cnyLeft;
double gp2Right;
double gp2Left;

PI_THREAD (gp2Thread){ // Usa el lock con key=0
    //pinMode (LDR, INPUT);

    int i = 0;
    //struct timeval tv;
    //int last;
    double left_gp , right_gp;
    while(1{
        //gettimeofday(&tv,NULL); //llamada al sistema para obtener el tiempo actual
        //last = tv.tv_usec/1000; // el tiempo actual
        left_gp = ((double)analogRead(BASE+0)/ (double)1024) * 3.3;
        right_gp = ((double)analogRead(BASE+1)/ (double)1024) * 3.3;

        piLock(0);
    }
}
```

```

Vgp2 = 1; // indicamos que hay lectura disponible
gp2Right = right_gp;
gp2Left = left_gp;
piUnlock(0);
i++;
delay(1);
//gettimeofday(&tv ,NULL);
//delay(Tldr - (tv.tv_usec/1000 - last)); // Queremos que cada iteracion del bucle dure Tld
}

PI_THREAD (cnyThread){ // Usa el lock con key=1
//pinMode (LDR, INPUT);

int i = 0;
//struct timeval tv;
//int last;
while(1{
    //gettimeofday(&tv ,NULL); //llamada al sistema para obtener el tiempo actual
    //last = tv.tv_usec/1000; // el tiempo actual

    //usaremos el valor intermedio 1.5 para ser capaces de contemplar distintos escenarios
    double right_cny = ((double)analogRead(BASE+2)/ (double)1024) * 3.3;
    double left_cny= ((double)analogRead(BASE+3)/ (double)1024) * 3.3;
    piLock(1);
    Vcny = 1; // indicamos que hay lectura disponible
    if(right_cny > 1.5){
        cnyRight = 1;
        //printf("cny derecho: blanco \n");
    }else{
        cnyRight = 0;
        //printf("cny derecho: negro; hay que girar a la derecha \n");
    }
    if(left_cny > 1.5){
        cnyLeft = 1;
        //printf("cny izquierdo: blanco \n");
    }else{
        cnyLeft = 0;
        //printf("cny izquierdo: negro; hay que girar a la izquierda \n");
    }
    piUnlock(1);
    i++;

    //gettimeofday(&tv ,NULL);
    //delay(Tldr - (tv.tv_usec/1000 - last)); // Queremos que cada iteracion del bucle dure Tld
}
}

int main (void){
    //char buf[80];
    wiringPiSetup();
    printf("wiringPiSPISetup RG=%d\n", wiringPiSPISetup(0,500000));

    mcp3004Setup(BASE,SPI_CHAN);
}

```

```

softPwmCreate(MotRight,0,RANGE);
softPwmCreate(MotLeft,0,RANGE);
softPwmWrite(MotRight,13);
softPwmWrite(MotLeft,16);
/*while(1){
    softPwmWrite(0,13);
    softPwmWrite(2,16);
    delay(5000);
    softPwmWrite(0,0);
    softPwmWrite(2,16);
}
*/
printf("Iniciando R.Daneel Olivaw\n");
piThreadCreate(gp2Thread);
piThreadCreate(cnyThread);
struct timeval tv;
gettimeofday(&tv,NULL);
int initialTime = tv.tv_usec;
int der = 0, izq = 0;
double left_gp = 0, right_gp = 0;
int last = 0;
int parar = 0;
int i = 0;//pinMode (LDR, INPUT);
while( tv.tv_usec - initialTime < MAXTIME){

    piLock(0);
    if(Vgp2 == 1){
        //printf("Leyendo ldr en MAIN: %d\n", ldr);
        Vgp2 = 0;
        left_gp = gp2Left;
        right_gp = gp2Right;
    }
    piUnlock(0);
//printf("left_gp: %f right_gp: %f\n", left_gp , right_gp );
parar = 0;
    if(left_gp > 1.3 || right_gp > 1.3) parar = 1;
piLock(1);
    if(Vcny == 1){
        //printf("Leyendo cny en MAIN: right %d left %d\n", cnyRight , cnyLeft );
        der = cnyRight;
        izq = cnyLeft;
    Vcny = 0;
    }
    piUnlock(1);
    if(left_gp > 1.3 || right_gp > 1.3){
        //printf("PARANDO: %i \n", i);
        i +=1;
        softPwmWrite(MotRight,100);
        softPwmWrite(MotLeft,100);
    }else if(der == 1 && izq == 1){ //ambos blancos , seguimos rectos
        softPwmWrite(MotRight,13);
        softPwmWrite(MotLeft,16);
        last = 0;// printf("| \n");
    }else if(der == 1 && izq == 0){ // izquierdo lee linea , giramos a la izquierda
        softPwmWrite(MotRight,13);
        softPwmWrite(MotLeft,13);
}
}

```

```

        last = 1;
    }else if(der == 0 && izq == 1){ // izquierdo se ha salido , giramos a la derecha
        softPwmWrite(MotRight,16);
        softPwmWrite(MotLeft,16);
        last = 2;// printf(">");
    } // else: ambos negros, no mandamos orden nueva a los motores, asi que siguen funcionando,
delay(1);
gettimeofday(&tv,NULL);
//delay(Tldr - (tv.tv_usec/1000 - last));
}

return 0 ;
}

```

5. Anexo B: Código negro negro

```

#include <stdio.h>
#include <wiringPi.h>
#include <time.h>
#include <sys/time.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <string.h>
#include <softPwm.h>
#include <wiringPiSPI.h>
#include <mcp3004.h>

// LED Pin - wiringPi pin 0 is BCM_GPIO 17.

#define Tldr      100
#define Tcny     30
#define MAXTIME  3000000 // 10s en microsegundos
#define LDR      0
#define CNY      1
#define MotLeft  2
#define MotRight 3
#define RANGE      100
#define NUM_LEDS   1
#define BASE     100
#define SPI_CHAN 0

int Vgp2 = 0;
int Vcny = 0;
int cnyRight; //blanco 1    negro 0
int cnyLeft;
double gp2Right;
double gp2Left;

```

```

PI_THREAD (gp2Thread){ // Usa el lock con key=0
    //pinMode (LDR, INPUT);
}

```

```

int i = 0;
//struct timeval tv;
//int last;
double left_gp, right_gp;
while(1){
    //gettimeofday(&tv,NULL); //llamada al sistema para obtener el tiempo actual
    //last = tv.tv_usec/1000; // el tiempo actual
    left_gp = ((double)analogRead(BASE+0)/ (double)1024) * 3.3;
    right_gp = ((double)analogRead(BASE+1)/ (double)1024) * 3.3;

    piLock(0);
    Vgp2 = 1; // indicamos que hay lectura disponible
    gp2Right = right_gp;
    gp2Left = left_gp;
    piUnlock(0);
    i++;
    delay(1);
    //gettimeofday(&tv,NULL);
    //delay(Tldr - (tv.tv_usec/1000 - last)); // Queremos que cada iteracion del bucle dure Tld
}
}

PI_THREAD (cnyThread){ // Usa el lock con key=1
    //pinMode (LDR, INPUT);

    int i = 0;
    //struct timeval tv;
    //int last;
    while(1){
        //gettimeofday(&tv,NULL); //llamada al sistema para obtener el tiempo actual
        //last = tv.tv_usec/1000; // el tiempo actual

        //usaremos el valor intermedio 1.5 para ser capaces de contemplar distintos escenarios
        double right_cny = ((double)analogRead(BASE+2)/ (double)1024) * 3.3;
        double left_cny= ((double)analogRead(BASE+3)/ (double)1024) * 3.3;
        piLock(1);
        Vcny = 1; // indicamos que hay lectura disponible
        if(right_cny > 1.5){
            cnyRight = 1;
            //printf("cny derecho: blanco \n");
        }else{
            cnyRight = 0;
            //printf("cny derecho: negro; hay que girar a la derecha \n");
        }
        if(left_cny > 1.5){
            cnyLeft = 1;
            //printf("cny izquierdo: blanco \n");
        }else{
            cnyLeft = 0;
            //printf("cny izquierdo: negro; hay que girar a la izquierda \n");
        }
        piUnlock(1);
        i++;
    }
}

```

```

    //gettimeofday(&tv ,NULL);
    //delay(Tldr - (tv.tv_usec/1000 - last)); // Queremos que cada iteracion del bucle dure Tld
}

int main (void){
    //char buf[80];
    wiringPiSetup();
    printf("wiringPiSPISetup RG=%d\n", wiringPiSPISetup(0,500000));

    mcp3004Setup(BASE,SPI_CHAN);

    softPwmCreate(MotRight,0,RANGE);
    softPwmCreate(MotLeft,0,RANGE);
    softPwmWrite(MotRight,13);
    softPwmWrite(MotLeft,16);
    printf("Iniciando R.Daneel Olivaw\n");
    piThreadCreate(gp2Thread);
    piThreadCreate(cnyThread);
    struct timeval tv;
    gettimeofday(&tv ,NULL);
    int initialTime = tv.tv_usec;
    int der = 0, izq = 0;
    double left_gp = 0, right_gp = 0;
    int last = 0;
    int parar = 0;
    int i = 0;//pinMode (LDR, INPUT);
    while( tv.tv_usec - initialTime < MAXTIME){

        piLock(0);
        if(Vgp2 == 1){
            //printf("Leyendo ldr en MAIN: %d\n", ldr);
            Vgp2 = 0;
        left_gp = gp2Left;
        right_gp = gp2Right;
        }
        piUnlock(0);
        //printf("left_gp: %f right_gp: %f\n", left_gp , right_gp );
        parar = 0;
        if(left_gp > 1.3 || right_gp > 1.3) parar = 1;
        piLock(1);
        if(Vcny == 1){
            //printf("Leyendo cny en MAIN: right %d left %d\n", cnyRight , cnyLeft );
            der = cnyRight;
            izq = cnyLeft;
        Vcny = 0;
        }
        piUnlock(1);
        if(left_gp > 1.3 || right_gp > 1.3){ // hay un obstaculo. Paramos
            //printf("PARANDO: %i \n", i);
            i +=1;
            softPwmWrite(MotRight,100);
            softPwmWrite(MotLeft,100);
        }else if(der == 0 && izq == 0){ //ambos negros, seguimos rectos
            //printf("Palante bro \n");
        }
    }
}

```

```

        softPwmWrite(MotRight,13);
        softPwmWrite(MotLeft,16);
        last = 0; // printf("|\n");
    }else if(der == 0 && izq == 1){ // derecho se ha salido , giramos a la izquierda
        softPwmWrite(MotRight,16);
        softPwmWrite(MotLeft,16);
        last = 1; // printf("<-");
    }else if(der == 1 && izq == 0){ // izquierdo se ha salido , giramos a la derecha
        softPwmWrite(MotRight,13);
        softPwmWrite(MotLeft,13);
        last = 2; // printf("->");
    }else{
        if(last == 1){
            softPwmWrite(MotRight,13);
            softPwmWrite(MotLeft,13);
            last = 0;
        }else if(last == 2){
            softPwmWrite(MotRight,16);
            softPwmWrite(MotLeft,16);
            last = 0;
        }else{
            softPwmWrite(MotRight,16);
            softPwmWrite(MotLeft,13);
        }
    }

    delay(1);
    gettimeofday(&tv,NULL);
//delay(Tldr - (tv.tv_usec/1000 - last));
}

return 0 ;
}

```

6. Anexo C: Código laberinto

```

#include <stdio.h>
#include <wiringPi.h>
#include <time.h>
#include <sys/time.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <string.h>
#include <softPwm.h>
#include <wiringPiSPI.h>
#include <mcp3004.h>

// LED Pin - wiringPi pin 0 is BCM_GPIO 17.

#define Tldr    100
#define Tcny   30

```

```

#define MAXTIME 3000000 // 10s en microsegundos
#define LDR 0
#define CNY 1
#define MotLeft 2
#define MotRight 3
#define RANGE 100
#define NUM_LEDS 1
#define BASE 100
#define SPI_CHAN 0
#define THRESHOLD_OBJECT 1
#define T_STOP 1.2

int Vgp2 = 0;
double gp2Right;
double gp2Left;

PI_THREAD (gp2Thread){ // Usa el lock con key=0
    //pinMode (LDR, INPUT);

    int i = 0;
    //struct timeval tv;
    //int last;
    double left_gp , right_gp;
    while(1){
        //gettimeofday(&tv ,NULL); //llamada al sistema para obtener el tiempo actual
        //last = tv.tv_usec/1000; // el tiempo actual
        left_gp = ((double)analogRead(BASE+0)/ (double)1024) * 3.3;
        right_gp = ((double)analogRead(BASE+1)/ (double)1024) * 3.3;

        piLock(0);
        Vgp2 = 1; // indicamos que hay lectura disponible
        gp2Right = right_gp;
        gp2Left = left_gp;
        piUnlock(0);
        i++;

        //gettimeofday(&tv ,NULL);
        //delay(Tldr - (tv.tv_usec/1000 - last)); // Queremos que cada iteracion del bucle dure Tld
    }
}

int main (void){
    //char buf[80];
    wiringPiSetup();
    printf("wiringPiSPISetup_RG=%d\n", wiringPiSPISetup(0,500000));

    mcp3004Setup(BASE,SPI_CHAN);

    softPwmCreate(MotRight,0,RANGE);
    softPwmCreate(MotLeft,0,RANGE);
    softPwmWrite(MotRight,13);
    softPwmWrite(MotLeft,16);
    printf("Iniciando_R.Daneel_Olivaw\n");
}

```

```

piThreadCreate(gp2Thread);
struct timeval tv;
gettimeofday(&tv, NULL);
int initialTime = tv.tv_usec;
double left_gp = 0, right_gp = 0;
int parar = 0;
int i = 0;//pinMode (LDR, INPUT);
while( tv.tv_usec - initialTime < MAXTIME){

    piLock(0);
    if(Vgp2 == 1){
        //printf("Leyendo ldr en MAIN: %d\n", ldr);
        Vgp2 = 0;
        left_gp = gp2Left;
        right_gp = gp2Right;
    }
    piUnlock(0);
//printf("%f %f", left_gp, right_gp);
if(left_gp > T_STOP && right_gp > T_STOP){
    softPwmWrite(MotRight,0);
    softPwmWrite(MotLeft,0);
} else if(left_gp > THRESHOLD_OBJECT && right_gp > THRESHOLD_OBJECT){
// printf("PARED DELANTE: %i \n", i);
    i +=1;
    softPwmWrite(MotRight,16);
    softPwmWrite(MotLeft,16);
} else if(left_gp > THRESHOLD_OBJECT){
    softPwmWrite(MotRight,16);
    softPwmWrite(MotLeft,16);
// printf("GIRANDO DERECHA\n");
} else if(right_gp > THRESHOLD_OBJECT){
// printf("GIRANDO IZQUIERDA\n");
    softPwmWrite(MotRight,13);
    softPwmWrite(MotLeft,13);
} else{
// printf("RECTO\n");
    softPwmWrite(MotRight,13);
    softPwmWrite(MotLeft,16);
}

    gettimeofday(&tv, NULL);
//delay(Tldr - (tv.tv_usec/1000 - last));
}

return 0 ;
}

```
