Un Control PID para robots con LEGO® MINDSTORMS

Os presentamos el primer tutorial sugerido por uno de nuestros lectores, esperamos lo encontréis de interés y os animéis a mandarnos más sugerencias.

Texto y gráficos por J. Sluka

Traducción y adaptación por Jetro

Un control PID es una técnica común que se emplea para un amplio abanico de máquinas, incluyendo vehículos, robots e incluso cohetes. La descripción matemática completa de un control PID es bastante compleja, pero para poder usar un PID de manera efectiva solamente hace falta una comprensión básica.

Este documento describe cómo crear un control PID para usar con un robot LEGO® MINDSTORMS usando el lenguaje de programación NXT-G.

Será más fácil hacerlo si tenemos un objetivo claro así que describiré cómo crear un control PID para un siguelíneas. Una vez creado, el mismo PID con unas pequeñas modificaciones se puede usar para cualquier otra aplicación MINDSTORMS como por ejemplo hacer que un robot avance perfectamente en línea recta o incluso un robot que guarda equilibrio sobre dos ruedas como un Segway®.

Un PID es en realidad bastante sencillo y la descripción típica del mismo es fácil de entender para cualquiera que ha hecho cálculo. Este documento está enfocado a alumnos de hasta 14 años que participan en la FLL. *Ya que no hay muchos alumnos de esta edad que han tenido clases de cálculo intentaré explicar el concepto desde un punto de partida sencillo evitando el uso de cálculo.*

Empecemos por un diseño de robot sencillo que sirve para un ejercicio de sigue líneas. La figura 1 muestra un esquema simplificado de vista superior del robot con todos los detalles que necesitamos. Se trata de un robot con dirección diferencial - un motor conectado a cada una de las dos ruedas A y C. El robot dispone de un sensor de luz colocado en la parte delantera y enfocado hacia abajo de modo que solamente ve el suelo. El círculo rojo representa el punto relativamente pequeño del trazado que el sensor de luz realmente puede “ver”, El resto del robot es representado por el rectángulo grande con la flecha que indica la dirección en la que se desplaza.

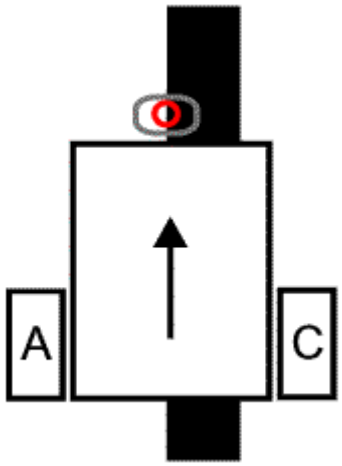


Figura 1

Nuestro objetivo es conseguir que el robot siga la gruesa línea negra. El seguimiento de una línea es un comportamiento básico para robots y a menudo es lo primero que se aprende. Un dispositivo móvil capaz de seguir una línea muestra todas las características de un auténtico robot. Emplea un sensor para recoger información acerca de su entorno y modifica su comportamiento en función de esa información.

Un sigue líneas se puede construir con uno, dos o una docena de sensores de luz. Generalmente, cuantos más sensores de luz tengas, mejor podrás seguir una línea. Aquí nos limitaremos a un solo sensor de luz MINDSTORMS. Incluso con un solo sensor

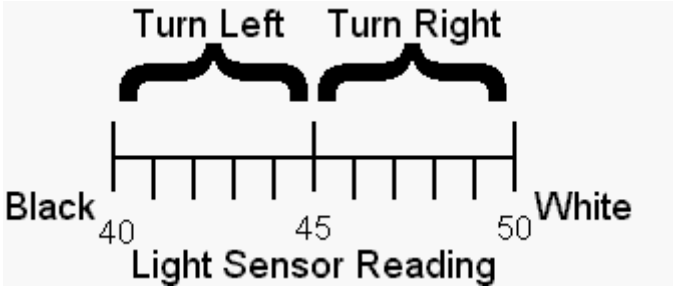
deberíamos poder hacer un robot capaz de seguir una línea con gran precisión incluso si tiene curvas.

Lo que generalmente se pierde al emplear un solo sensor es la habilidad de seguir una línea a gran velocidad. A menudo, cuantos más sensores tienes, más rápido puede seguir la línea el robot.

El truco que emplearemos, y que nada tiene que ver con PID, es que no intentaremos seguir la línea. En vez de eso, intentaremos seguir el borde de la línea. ¿Porqué? Porque si seguimos la línea misma (negra), cuando nos salimos de esta y el sensor “ve blanco” no sabemos en qué lado de la línea nos encontramos. ¿Estamos a la izquierda o a la derecha? Si seguimos el borde de la línea podemos determinar hacia qué lado nos estamos desviando. Si el sensor “ve blanco” sabemos que está a la izquierda del borde de la línea (y de la línea). Si “ve negro” sabemos que está a la derecha del borde (encima de la línea). Esto se llama un “siguielíneas por la izquierda” ya que sigue el borde izquierdo de la línea.

Necesitamos saber los valores que el sensor detecta cuando “ve blanco” y cuando “ve negro”. Un sensor típico sin calibrar puede dar una lectura de 50 para blanco y de 40 para negro (sin calibrar, en una escala de 0 a 100). Es conveniente dibujar una línea con esos valores para ayudar a visualizar cómo convertimos los valores del sensor de luz en movimientos del robot. En la figura 2 se pueden ver los valores para blanco y negro.

Figura 2

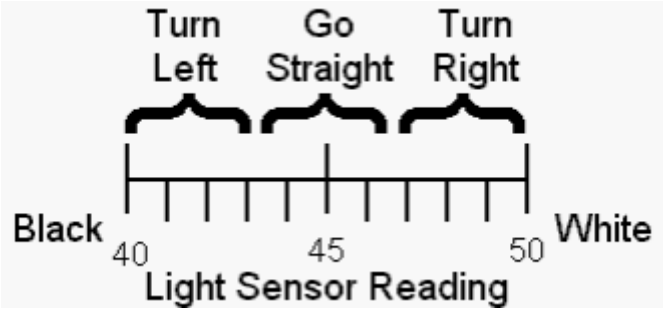


Simplemente dividiremos el rango en dos tramos iguales y diremos que si el valor de luz está por debajo de los 45 queremos que robot gire a la derecha. Si el valor es superior a 45 queremos que gire a la izquierda. No explicaré exactamente cómo hacer los giros. Solo diré que para una línea relativamente recta los giros suaves funcionan bien. Una línea con muchas curvas requerirá giros más cerrados Para giros suaves puedes usar un nivel de potencia de 50% ara la rueda rápida y de 20% para la lenta. Para giros más cerrados en una línea muy ondulada tal vez tengas que usar 30% para la rápida y 0% con o sin freno para la lenta. Sean cuales sean los niveles de potencia, serán iguales para ambos giros, solamente invirtiendo qué motor recibe la potencia mayor o menor.

Este tipo de siguelíneas cumplirá su cometido, pero no es muy elegante. Puede parecer perfecto en una línea recta y programado para giros suaves. Pero si la línea tiene alguna curva tendrás que programar el robot para giros más cerrados. Esto hace que el robot oscile continuamente de un lado a otro. El robot solo “sabe” hacer dos cosas: girar hacia la derecha y hacia la izquierda. Este enfoque puede funcionar, pero ni es muy rápido ni preciso y no es bonito de ver.

En este enfoque el robot nunca va en línea recta, incluso si está perfectamente alineado con el borde de la línea y la línea es recta. No parece muy eficiente ¿verdad?

Lo vamos a arreglar. En vez de dividir la línea de valores de luz en dos secciones, la dividiremos en tres.



Así que ahora, si el valor es inferior a 43 queremos que el robot gire a la izquierda. Si el valor está entre 44 y 47 queremos que vaya en línea recta. Si el valor es superior a 47 queremos que gire a la derecha. Esto se puede hacer de forma sencilla en NXT-G usando un conmutador (si/no) dentro de otro. De hecho solo necesitas dos, no tres.

Este enfoque funciona mejor que el primero. Al menos el robot mueve en línea recta a veces. Al igual que con el primer enfoque todavía tienes que decidir qué tipo de giros usar y eso dependerá de la línea que quieras seguir. El robot aún zigzagueará bastante.

El lector astuto ya habrá pensado “*bueno, si tres secciones funcionan mejor que dos, ¿qué tal si lo dividimos más?*” Eso es el comienzo del control PID.

La “P” de “PID”: La clave está en la proporción

¿Qué sucede si añadimos más divisiones en la escala? Bueno, lo primero que deberemos definir es el significado de ‘girar’ cuando hay más de tres secciones. En el primer enfoque el robot solo sabía hacer dos cosas: girar hacia la izquierda y hacia la derecha. Los giros siempre tenían la misma amplitud. En el segundo enfoque añadimos “ir en línea recta” a estos dos giros. Si tenemos más de tres secciones necesitamos más “tipos” de giros.

Para comprender mejor la idea de “más tipos de giro” convertiremos la línea en un gráfico. El eje X (horizontal) contiene los valores de luz al igual que en la línea de números. El eje Y (vertical) será el eje de “giro”.

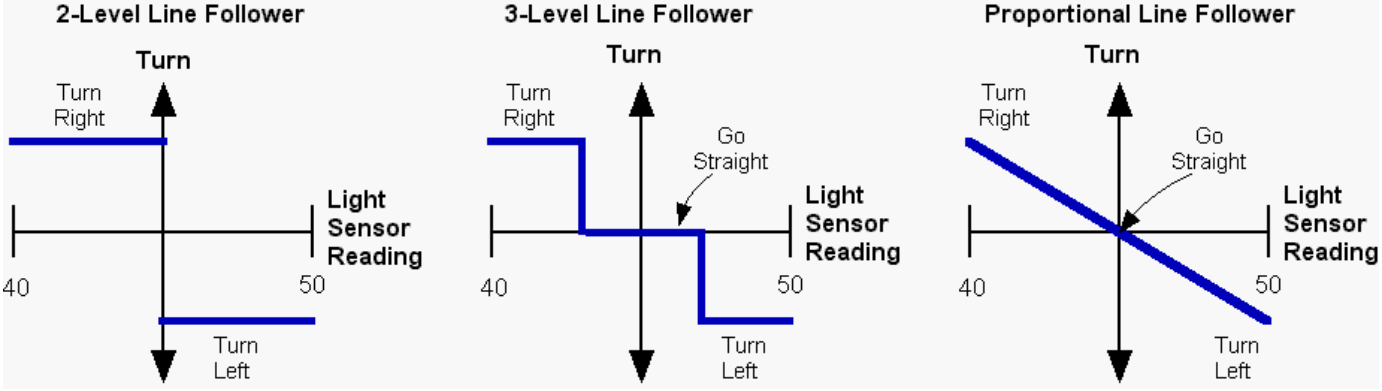


Figura 3

En la figura 3 se puede ver el primer enfoque convertido en gráfico. El robot solo puede hacer dos cosas (según muestran las líneas azules), girar hacia la derecha o hacia la izquierda, y el ángulo de giro siempre es el mismo. En el medio se ve el segundo enfoque con tres secciones. La sección adicional del centro es donde el robot va en línea recta (giro=0). Los giros son los mismos de antes. A la derecha hay un siguelíneas **Proporcional**. En un siguelíneas proporcional el ángulo de giro varía suavemente entre los dos extremos. Sin la lectura del sensor de luz dice que estamos cerca de la línea se ejecuta un giro pequeño. Si estamos lejos se ejecuta un giro grande. Esta proporcionalidad es un concepto importante.

Proporcional significa que hay una relación lineal entre dos variables. Para decirlo de manera aún más sencilla, proporcional significa que un gráfico de los variables muestra una línea recta (como en el gráfico de la derecha).

Como tal vez recuerdes, la ecuación de una línea recta es:

y = mx + b

y es la distancia hacia arriba (o abajo) en el eje y, x la distancia en el eje x, m es la inclinación de la línea y b es el punto de intersección entre la línea y el eje y. La inclinación de la línea se define como el cambio en el valor de y dividido entre el cambio en el valor x entre dos puntos de la línea.

Si no sabes mucho de líneas (o has olvidado lo que sabías) ampliaré un poco el concepto y haré algunas simplificaciones en el gráfico y la ecuación. Primero, cambiaremos el centro de nuestra línea de valores de luz (x) a cero). Eso es sencillo.

Para un rango de lectura de 40 a 50 simplemente restamos 45 (la media de 40 y 50, (40+50)/2 de todos los valores de luz obtenidos. El resultado lo llamaremos **error**. Así que, si del valor 47 restamos 45 el resultado es un **error** =2. El **error** nos dice cuanto nos hemos desviado del borde de la línea. Si el sensor está exactamente encima del borde de la línea nuestro **error** es cero ya que el valor de luz es 45 y restamos 45 a todos los valores. Si el sensor está completamente en la zona blanca, el **error** es de +5. En la zona negra el **error** es de -5.

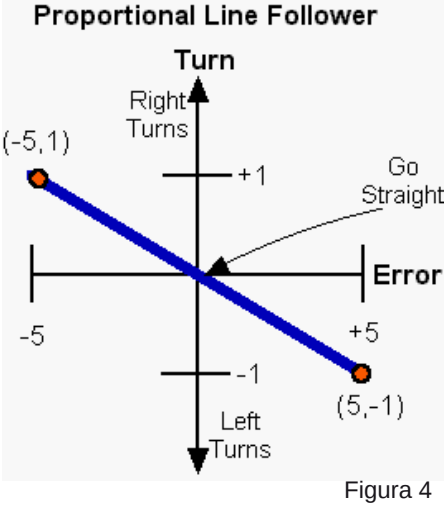


Figura 4

En la figura 4 he movido el eje, convirtiéndolo en una escala de error. Como ahora la línea cruza el eje y en cero, la ecuación de la línea es algo más sencilla;

y = mx

o utilizando nuestra terminología

Giro =m* **error**

Aún no hemos definido lo que significa el eje de giro así que de momento diremos que los giros varían de -1 (giro cerrado hacia la izquierda) a +1 (giro cerrado hacia la derecha) y un giro de 0 significa que vamos en línea recta. La inclinación de la línea del gráfico se puede calcular usando los dos puntos marcados en rojo (funcionará para cualquier combinación de puntos que se encuentran en la línea);

inclinación = m = (cambio en y)(cambio en x) = (1-(-1) / (-5-5) = 2/10 = 0,2

La inclinación es una **proporcionalidad constante** y es el factor por el cual hay que multiplicar el **error** (valor x) para convertirlo en un **Giro** (valor y). Es importante recordar eso.

La "inclinación" tiene varios nombres pero todos significan lo mismo, al menos en este contexto. En la documentación sobre PID a la inclinación se le llama "**K**" (¿ viene de escribir mal "constante"? Se encuentran varios **K**s en la documentación sobre PIDs y son todos muy importantes. A **K** (o m, o la inclinación, o la constante de proporcionalidad) se le puede considerar un *factor de conversión*. Se usa **K** para convertir un número que significa una cosa (valores de luz o error en nuestro caso) en otra cosa como un giro. Eso es todo lo que hace una **K**. Muy sencillo y muy potente.

Así que usando estos nuevos nombres para nuestras variables la ecuación de la línea es;

Giro = K*(error)

En palabras significa "toma el **error** y multiplícalo por la constante de proporcionalidad **K** para conseguir el giro que necesitas". El valor **Giro** es la salida de nuestro controlador P y se le llama el "**término P**" ya que es solo un controlador proporcional.

Tal vez hayas reparado en que en el último gráfico la línea no sale fuera del rango de **error** de entre -5 y +5. Fuera del rango de -5 a +5 no sabemos cuanto se ha desviado el sensor de la línea. Todo lo "blanco" tiene el mismo aspecto una vez que el sensor ya no ve nada de negro. Recuerda que este rango es arbitrario. Tu rango se determinará por cómo colocas el sensor de luz, el color del suelo etc. Una vez que el sensor de luz se aleje demasiado del borde de la línea empezará a dar una lectura constante. Eso significa que la lectura del sensor de luz ya no es proporcional al **error**. Solo sabemos cuanto se ha desviado el sensor de la línea cuando está relativamente cerca. Dentro de ese pequeño margen, la lectura del sensor de luz es proporcional a la distancia. Así que nuestro sensor tiene un rango limitado dentro del cual da información proporcional. Fuera de ese rango nos informa de la dirección correcta pero con una magnitud (distancia) equivocada. La lectura del sensor de luz, o el **error**, es mas pequeña de lo que debería ser y no nos da una buena idea de qué giro hace falta para corregirlo.

En la documentación sobre PID el rango sobre el cual el sensor da una respuesta proporcional se llama el "**rango proporcional**" (hablando de evidencias :D). El rango proporcional es otro concepto importante en PID. En nuestro sigue líneas el rango proporcional para el sensor de luz es de 40 a 50, para el **error** es de -5 a +5. Nuestros motores también tienen un rango proporcional, de -100 (plena potencia hacia atrás) a +100 (plena potencia hacia delante). Comentaré unas pocas cosas sobre la importancia del rango proporcional:

(1) Quieres tener un rango proporcional todo lo amplio posible. El rango proporcional de nuestro sensor de luz es bastante pequeño, es decir, el sensor tiene que estar bastante cerca de la línea para conseguir información proporcional. Exactamente cómo de amplio es el rango depende principalmente de la altura que el sensor tiene respecto del suelo. Si el sensor está muy cerca del suelo, digamos 2mm, entonces solo ve un pequeño círculo del suelo. Un pequeño movimiento de lado a lado hará pasar el valor del **error** de -5 a +5, atravesando todo el rango proporcional. Se podría decir que el sensor tiene visión de túnel y que solo puede ver una pequeña parte del suelo. El sensor tiene que estar muy cerca del borde de la línea para conseguir una lectura que no sea "blanco" o "negro". Si el sensor se eleva más, puede ver un círculo más grande. A una altura de aprox. 1,5 cm el sensor parece observar un círculo con un diámetro de aprox 1,5 cm. Con el sensor tan arriba el rango proporcional es mucho más amplio, ya que el sensor solo necesita quedarse a +/- 1,5 cm del borde de la línea para seguir dando información proporcional Por desgracia hay dos inconvenientes con un sensor tan alto. Primero, un sensor alto "ve", y responde a, la luz de la habitación mucho más que un sensor bajo. Un sensor alto también tiene menos diferencia entre negro y blanco que uno bajo. A suficiente distancia la lectura será la misma con blanco y con negro.

(2) Fuera del rango proporcional el controlador moverá las cosas en la dirección correcta, pero con una corrección inferior a la necesaria. La respuesta proporcional del controlador se ve limitada por el rango proporcional.

De P a valores de potencia para los motores

¿Cómo haremos los giros? ¿Cuales serían los niveles de potencia de los motores? Una manera de hacer los giros es definir un "objetivo (**T**arget) de nivel de **p**otencia que llamaremos **Tp**". **Tp** es el nivel de potencia de ambos motores cuando el robot va en línea recta, a saber, cuando el **error**=0. Cuando el **error** no es cero usamos la ecuación **Giro = K*(error)** para calcular

cómo cambiar los niveles de potencia de ambos motores. A un motor recibe un nivel potencia de **Tp+Giro**, y el otro recibe un nivel de potencia **Tp-Giro**. Ya que el **error** puede tener un valor de entre -5 y +5, el **Giro** puede ser positivo o negativo lo que corresponde con giros en direcciones contrarias. Resulta que eso es exactamente lo que necesitamos ya que designará automáticamente qué motor es el rápido o el lento. Un motor (supondremos que es el motor de la izquierda conectado al puerto A) siempre recibirá **Tp+Giro** como nivel de potencia. El otro motor (derecha del robot, puerto C) siempre recibirá **Tp-Giro** como nivel de potencia. Si **error** es positivo entonces **Giro** es positivo y **Tp+Giro** es mayor que **Tp** de modo que el motor izquierdo acelera mientras que el derecho se ralentiza. Si el **error** cambia de signo y es negativo (lo que significa que hemos cruzado el borde de la línea y "vemos negro") entonces **Tp+Giro** será menor que **Tp** y el motor izquierdo se ralentizar mientras que el derecho acelera ya que **Tp-Giro** es mayor que **Tp**. (Recuerda que negativo más negativo es igual a positivo). Sencillo ¿verdad? Espero que se vaya aclarando a medida que avanzamos.

Pseudo código para el Controlador P

Primero tenemos que medir los valores que el sensor de luz da para blanco y negro. Partiendo de estos dos números podemos calcular el **offset**, es decir, cuanto debemos restar del valor para convertirlo en un valor de **error**. El **offset** es simplemente la media de los valores para blanco y negro. Para simplificar las cosas supondremos que el **offset** ha sido medido y guardado en una variable llamada **offset**. (Una interesante mejora sería hacer que el robot midiese los valores para blanco y negro y luego calculara el **offset**.)

También necesitamos un sitio donde guardar el valor de la constante **K**, que llamaremos **Kp** (la **K**onstante del controlador proporcional). Además necesitamos una aproximación inicial de ese valor **Kp**. Hay muchas maneras de conseguir ese primer valor para **Kp**. Puedes estimarlo y luego refinarlo mediante prueba y error. O puedes intentar estimar un valor, basado en las características del sensor y el robot. Haremos lo segundo. Usaremos un **Tp** (objetivo de potencia) de 50: cuando el error es cero, ambos motores tendrán una potencia de 50. El rango de error es de -5 a 5+. Haremos que la potencia vaya de 50 a 0 cuando el error va de 0 a -5. Eso significa que **Kp** (recuerda: la inclinación, el cambio en y dividido entre el cambio en x) es;

Kp = (0 - 50)/(-5 - 0) = 10.

Usaremos un valor de **Kp**=10 para convertir un **error** en un valor de **giro**. En palabras esa conversión es "para cada unidad de cambio en el **error** la potencia de motor se incrementa con 10". La potencia del otro motor se reduce en 10.

Así que en el pseudo código (pseudo código quiere decir que no es NXT-G ni ningún otro tipo de código de programa sino una lista detallada de lo que queremos que el programa haga):

```
Kp = 10                          ! Inicializar nuestras tres variables
offset = 45
Tp = 50
Bucle infinito
  ValorLuz = leer sensor de luz    ! ¿cual es el valor de luz actual?
  error = ValorLuz - offset        ! calcular el error restando el offset
  Giro = Kp * error                ! el "término P", o cuanto queremos que varíe la potencia del motor
  potenciaA = Tp + Giro              ! el nivel de potencia para el motor A
  potenciaC = Tp - Grio              ! el nivel de potencia para el motor C
  MOTOR A dirección=hacia delante potencia=potenciaA ! dar la orden con el nuevo nivel de potencia en el bloque MOTOR
  MOTOR C dirección=hacia delante power=powerC ! lo mismo para el otro motor, pero usando el otro nivel de potencia
fin de bucle infinito                ! bucle terminado: vuelve al inicio del bucle y ejecútalo otra vez
```

Eso es todo, bueno, casi. Hay un pequeño problema que hay que corregir. Pero inténtalo de todos modos. Si parece que tu robot evita la línea en vez de intentar encontrarla, lo más probable es que haya intercambiado las direcciones de giro. Cambia **Kp** a -10 y observa lo que pasa. Si eso corrige el problema vuelve a cambiar **Kp** a +10 y cambia el signo en las dos líneas de potencia;

```
potenciaA = Tp - Giro
potenciaC = Tp + Giro
```

Hay dos "parámetros ajustables" y una constante en este controlador **P**. La constante es el **offset** (el promedio entre la lectura de blanco y negro). Tendrás que escribir un pequeño programa para medir esos valores con tu robot. Necesitas un valor "blanco" y otro "negro". Calcula el promedio y coloca el resultado en la variable **offset** del programa del controlador P. Casi todos los programas siguelíneas necesitan que tu o el programa que escribas para tu robot haga este paso.

Los parámetros ajustables son el valor **Kp** y el objetivo potencia **Tp**. Los parámetros ajustables se averiguan mediante prueba y error. **Kp** controla la velocidad con la que el controlador intenta volver al borde de la línea cuando se aleja. **Tp** controla la velocidad del robot a lo largo de la línea.

Si la línea es bastante recta puedes usar una **Tp** grande para que el robot se mueva a gran velocidad y una **Kp** para que los giros (correcciones) sean suaves.

Si la línea tiene curvas cerradas habrá un valor máximo de **Tp** que funcione. Si **Tp** es mayor que ese máximo dará igual el valor de **Kp**, el robot perderá la línea cuando encuentre una curva porque se mueve demasiado deprisa. Si **Tp** es muy pequeño entonces casi cualquier valor **Kp** funcionará ya que el robot se mueve muy despacio. El objetivo es conseguir que el robot se

mueva todo lo rápido posible sin perder la línea.

Estimamos un valor inicial para **Kp** de 10. Para **Tp** podrías empezar con un valor incluso inferior que lo sugerido anteriormente, tal vez 15 (el robot se moverá muy despacio). Prueba a ver qué tal funciona. Si pierdes la línea porque el robot parece moverse de manera muy lenta aumenta el valor **Kp** un poco y prueba otra vez. Si pierdes la línea porque el robot parece hiperactivo al buscar la línea entonces reduce **Kp**. Si parece que el robot hace un buen trabajo, incrementa **Tp** y mira a ver si puedes seguir la línea a mayor velocidad. Para cada nuevo **Tp** tendrás que averiguar un **Kp** nuevo, aunque **Kp** no suele variar mucho.

Seguir una línea recta normalmente es bastante sencillo. Seguir una línea con curvas suaves es algo más difícil. Seguir una línea con curvas cerradas es lo más difícil. Si el robot es lo suficientemente lento se puede seguir casi cualquier línea, incluso con un controlador muy básico. Queremos conseguir un buen siguelíneas, una buena velocidad y la capacidad de seguir curvas suaves. (Líneas con curvas cerradas suelen requerir siguelíneas más especializados).

Es probable que el mejor controlador P sea diferente para cada tipo de línea (ancho de línea, ángulo de las curvas) y para diferentes robots. En otras palabras, un controlador P (o incluso un controlador PID) se ajusta para un tipo de línea específico y no necesariamente funcionará bien par otras líneas o robots El código será válido para muchos robots (y muchas tareas), pero los parámetros, **Kp**, **Tp** y el **offset** tendrán que ser ajustados para cada robot y para cada tipo de uso.

Hacer cálculos en un ordenador que no sabe lo que son decimales causa algunos problemas

NOTA: Este trabajo se hizo con NXT-G versión 1,1 que solo tiene soporte para enteros. NXT-G 2 tiene soporte para coma flotante así que esto puede que esto no sea necesario si tienes la versión 2 o superior.

Durante el ajuste del controlador P habrá que subir y bajar el valor **Kp**. El rango de valores **Kp** esperado puede depender de exactamente qué está haciendo el controlador P. ¿Cual es la amplitud del rango de entrada y del de salida? Para nuestro controlador P siguelíneas, el rango de entrada es de aprox. 5 unidades de luz y el de salida de 100 unidades de potencia, así que parece razonable que **Kp** está cerca de 100/5=20. En algunos casos el valor **Kp** esperado no seá tan grande. ¿Qué sucede si el **Kp** esperado es uno? Como las variables en NXT-G solo admiten números enteros, cuando intentas ajustar el valor **Kp** solo puedes probar con ... -2, -1, 0, 1, 2, 3, ... No puedes meter 1,3, así que no puedes probar **Kp**=1,3. No puedes usar ningún número con decimales. Pero probablemente haya una gran diferencia en el comportamiento del robot si cambias el valor de **Kp** con el menor cambio posible de 1 a 2. Con **Kp**=2 el robot intenta corregir el error el doble comparado con **Kp**=1. El nivel de potencia de motor cambia el doble para el mismo cambio en el valor de la luz. Nos gustaría tener un control más ajustado de **Kp**.

Es bastante sencillo corregir esto. Todo lo que haremos es multiplicar **Kp** por una potencia de diez para incrementar el rango disponible dentro de las restricciones de la variable. Si sospechamos que **Kp** estará cerca de 1, entonces un valor de 100 como multiplicador será una buena opción. De hecho, probablemente sea mejor usar siempre 100***Kp** como número que usamos en el programa. Una vez hayamos multiplicado **Kp** por 100 podemos escribir el valor 1,3 como 130. 130 no tiene parte decimal así que ese número le gusta a NXT-G.
¿Pero eso no echa al traste el cálculo? Si, pero se puede corregir fácilmente. Una vez tengamos el valor P, lo dividiremos entre 100 para eliminar la multiplicación. Recuerda la ecuación con la que definimos el controlador P antes;

Giro = K*(error)

Multiplicaremos **Kp** por 100, lo que significa que el **Giro** que calculamos es 100 veces mayor de lo que debe ser. Antes de usar el **Giro** lo dividiremos entre 100 para corregir eso.
Así que nuestro nuevo y mejorado pseudo código para un controlador P siguelíneas es:

```
Kp = 1000                          ! RECUERDA que estamos usando Kp*100 así en realidad esto es 10 !
offset = 45                        ! Inicializar las otras dos variables
Tp = 50
Bucle infinito
  ValorLuz = leer sensor de luz    ! ¿cual es el valor de luz actual?
  error = ValorLuz - offset        ! calcular el error restando el offset
  Giro = Kp * error                ! el "término P", cuanto queremos que cambie la potencia del motor
 Giro = Giro/100                   ! RECUERDA deshacer el efecto de la ultiplicación por 100 en Kp !
  potenciaA = Tp + Giro                ! el nivel de potencia para el motor A
  potenciaC = Tp - Giro                ! el nivel de potencia para el motor C
  MOTOR A dirección=hacia delante potencia=potenciaA ! ejecutar el comando en el bloque MOTOR
  MOTOR C dirección=hacia delante potencia=potenciaC ! lo mismo para el otro motor pero usando el otro nivel de potencia
fin de bucle infinito                ! bucle terminado: vuelve al inicio del bucle y ejecútalo otra vez
```

¡Espera! ¿Cual era es "pequeño problema" que se mencionó con la primera versión del controlador P?

Hay algunos pequeños problemas. A veces son importante y otras no. ;)
En este caso, un problema consiste en que cuando calculamos el nivel de potencia del motor (ej. potenciaC=Tp-Giro) es posible obtener un número negativo. Queremos que un número negativo signifique que el motor cambio de sentido. Pero la entrada de datos en el bloque MOTOR de NXT-G no entiende eso. El nivel de potencia siempre es un número entre 0 y +100. La dirección del motor se controla con otro conector de entrada. Para conseguir que el motor reaccione de la manera correcta cuando el valor es negativo habrá que ajustar el programa. Aquí hay una manera de hacer eso;

```
Si potenciaA > 0 entonces                    ! un valor de potencia de motor positivo no es un problema
  MOTOR A dirección=hacia delante potencia=potenciaA
de lo contrario
potenciaA = potenciaA * (-1)                 ! un valor negativo de potencia de motor debe convertirse en
  MOTOR A dirección=inversa potencia=potenciaA  ! un número positivo y la dirección del motor
finde la condición                           ! hay que invertirlo en el panel de control
```

El bloque MOTOR recibe la potencia (potenciaA para el motor A) mediante un cable de datos. La dirección se determina con las casillas en el panel de configuración del bloque.

Necesitamos un trozo de código similar para el motor C. Ahora cuando la potencia calculada es negativa los motores responderán de la manera apropiada. Una cosa que conseguimos de esta manera es permitir al controlador P usar un “giro de radio cero” y el robot puede girar sobre su propio eje si es necesario. Por supuesto, puede que eso no ayude.

Hay algunas otras cosas que pueden causar problemas. ¿Qué sucede cuando envías un nivel de potencia mayor que 100? Resulta que el motor simplemente trata el número como si fuera 100. Eso es bueno para el programa pero no es la mejor solución par un controlador P (o PID). Realmente sería preferible que el controlador nunca pidiese que el motor hiciera algo imposible. Si la potencia requerida no supera en mucho los 100 (o -100) probablemente no pasa nada. Si la potencia requerida supera en mucho los 100 (o es muy inferior a -100) eso a menudo significa que el controlador está perdiendo el control. ¡Así que asegúrate de tener el extintor a mano!

Resumen del controlador P

Espero que haya entendido o suficiente para comprender cómo funciona un controlador P (proporcional). Es bastante sencillo. Utiliza un sensor para medir algo que intentas controlar. Convierte esa medición en un error. Para el siguelíneas hicimos esto retando el promedio de los valores para banco y negro. Multiplica por el **error** con un factor de escala llamado **Kp**. El resultado es la corrección para el sistema. En nuestro siguelíneas la corrección se aplica como un incremento/decremento del nivel de potencia de los motores. El factor de escala **Kp** se determina por estimación seguido de ajuste por prueba y error.
Un controlador P puede resolver una muy amplia gama de problemas de control, no solo seguir una línea con un robot de LEGO. En general, los controladores P funcionan muy bien cuando se dan algunas condiciones.

1. El sensor tiene que tener un amplio rango dinámico (lo que desgraciadamente no aplica al siguelíneas)

2. Lo que se controla (en este caso los motores) también tiene que tener un amplio rango dinámico, lo que significa que debe haber un amplio rango de niveles de potencia y que estos niveles estén próximos entre si.(los motores del NXT son bastante buenos en este aspecto).

3. Tanto el sensor como lo que se controla debe responder de manera rápida. “Rápido” en este contexto es “mucho más rápido que cualquier otra cosa que esté pasando en el sistema”. A menudo cuando controlas motores no es posible conseguir una respuesta “rápida” ya que los motores requieren tiempo para reaccionar a un cambio en la potencia. Puede tomar varias decimas de segundo para que un motor de LEGO® reaccione a un cambio en el nivel de potencia. Eso significa que las acciones del robot van por detrás de las órdenes emitidas por el controlador P. Eso hace que sea difícil conseguir un control preciso con un controlador P.

¿Donde está el código?

Podría dártelo pero tendría que matarte.

Ya que este documento está enfocado a participantes mayores de la FLL realmente no quiero dar el código. Deberían poder escribirlo ellos mismos.

El pseudo código contiene todo lo que necesitas para el PID en sí. Puede que tengas que añadir algo de inicialización y tal vez un método para parar el bucle del siguelíneas.

Como ayuda he hecho un MiBloque que tiene dos entradas - la potencia objetivo **Tp** y el **Giro**, y que controla dos motores. Este bloque además maneja niveles de potencia negativos correctamente. Incluso emite un pitido cada vez que un motor cambia de dirección, lo cual ayuda en el ajuste. Un PID para seguir una línea correctamente ajustado rara vez tendrá que cambiar el sentido de los motores.

PID_LF_MotorControl.rbt es el archivo RBT para NXT-G v1.1
http://www.inpharmix.com/jps/_images/PID_LF_MotorControl.rbt

Y esto es un pantallazo del programaPID_LF_MotorControld.png
http://www.inpharmix.com/jps/_images/PID_LF_MotorControld.png

Si **realmente** quieres mi código PID para NXT-G, mándame un email a Lego at InPharmix dot com
#