Robótica Basada en Comportamientos 2023

Informe Laboratorio 0

Grupo 01

Mauricio Berois, Gabriel Rodriguez

Índice

Parte A - Interacción con motores

Programa 1: Controles de movimiento básicos (P1.java)

Se realiza un primer acercamiento al manejo de los motores a través de la clase que permite comunicarse con los actuadores (Motor), además de la interacción con los botones (Button) y la pantalla del brick (LCD).

El programa consiste en hacer que el motor se mueva hacia adelante, hacia atrás y pare, transicionando entre cada acción al presionar el botón del brick, y desplegando en pantalla información básica como el nombre del programa.

Programa 2: Usando el tacómetro (P2.java)

Se introduce el funcionamiento del tacómetro del servomotor y se profundiza en el uso del display del brick. El programa consiste en hacer girar al motor y mostrar en pantalla las lecturas del tacómetro.

Para esto se setea la velocidad del motor a 2 rev/Seg y se lo hace girar durante 2 segundos.

Se observa la demora entre el envío de instrucciones y el efecto en los actuadores, al ser las dos lecturas de los tacómetros distintas en la secuencia:

```
LCD.drawInt(Motor.A.getTachoCount(), 0, 1);
Motor.A.stop();
LCD.drawInt(Motor.A.getTachoCount(), 0, 2);
```

Donde se imprime en pantalla la medida del tacómetro, se frena el motor, y se vuelve a imprimir en pantalla la medidas del tacómetro.

Además, luego de los dos segundos la medida del tacómetro no es exactamente 720º (4π). Esto se debe a la inercia (o histéresis) que presentan los sistemas físicos causados por la fricción estática y dinámica.

Programa 3: Control de rotación precisa (P3.java)

Se experimenta con los métodos que permiten hacer girar los motores un cierto ángulo. Se cuenta con dos métodos:

- Rotate: Hace girar el motor un ángulo especificado.
- RotateTo: Posiciona el motor en el ángulo especificado

Se observa que el motor frena con un grado de diferencia, normalmente en las pruebas siendo un grado antes del objetivo.

Este método es mucho más preciso que el anterior (programa 2) a la hora de posicionar el motor en un ángulo deseado. Esto se logra con el sistema de control de lazo cerrado del motor, que lo comienza a frenar antes de llegar a la posición deseada.

Programa 4: Interrumpiendo la rotación (P4.java)

Consiste en un programa similar al anterior, pero además de rotar se agrega la funcionalidad de detener la rotación al presionar el botón del brick.

Se nota que, de no agregarse un tiempo de delay, el programa puede funcionar de forma no deseada terminando la rotación con la primera presión del botón que la inicia. Esto se debe a que el programa es procesado más rápido que el tiempo en que una persona suelta el botón luego de presionarlo. Esto causa que la función que detecta si hay un botón presionado devuelva verdadero, haciendo que el loop principal finalice..

Se decide utilizar un delay de 200 ms por sugerencia de las soluciones del tutorial y debido a que creemos le da mayor robustez al programa, pero se experimenta con 20ms y se logran resultados satisfactorios (es posible iniciar y frenar con el botón de la forma esperada).

Programa 5: Regulando la velocidad del motor (P5.java)

El objetivo del programa es mostrar la sincronización de los motores con el reloj del sistema, para que cuando más de un motor está en movimiento a la misma velocidad, lo hagan de forma sincronizada. Por ejemplo si dos motores controlan dos ruedas, se quiere que ambas se muevan de forma sincronizada, o de lo contrario el robot giraría hacia uno de los lados cuando en realidad debería desplazarse en línea recta.

En el programa se utilizan tres motores y se toman las siguientes medidas de sus tacómetros cada 200ms

Motor A	Motor B	Motor C
87	84	87
248	247	248
391	392	391
535	353	535
679	679	679
717	721	717
721	720	721
721	720	721

Se puede observar que las diferencias no son mayores a 3 grados, y que en la mayoría de los casos coinciden exactamente o con una diferencia de 1 grado.

Actuador Lineal (actuador_lineal.java)

Aquí experimentamos con el actuador lineal. Utilizamos la clase *LnrActrFirgelliNXT* la cual instanciamos en el puerto donde se conecta el actuador.

De acuerdo a la documentación, el actuador aplica una fuerza de 25 N y se mueve a una velocidad máxima (sin carga) de 12 mm/seg.

La clase *LnrActrFirgelliNXT* provee un método de lectura del tacómetro. Este nos da una lectura de 200 cuando el actuador está extendido a su máxima longitud y 0 cuando no está desplegado.

Para probar este dispositivo, el programa *actuador_lineal.java* ejecuta un bucle que se repite hasta presionar el botón de escape del brick, y en cada ciclo se mueve el actuador 10 posiciones hacia adelante.

Notamos que la lectura del tacómetro se inicializa a 0 cada vez que se vuelve a ejecutar el programa, independientemente de la posición del actuador en ese momento.

Parte B - Interacción con el sensor de distancia ultrasónico

Se experimentó con el sensor de distancia ultrasónico de LEGO, el cual es controlado por la clase *lejos.nxt.UltrasonicSensor*. Al igual que antes, esta clase se instancia con el puerto del brick al que está conectado el sensor. El método utilizado fue *getDistance()* que nos devuelve la distancia, en centímetros, al objeto que se encuentre bloqueando la señal ultrasónica al frente del sensor.

Se trata de un sensor activo, porque emite una onda de sonido que rebota sobre el objeto y luego se calcula la distancia al mismo midiendo el tiempo que tarda dicha onda en volver al sensor.

Según la especificación, el sensor tiene una precisión de +/- 3 cm, y un máximo de distancia de 255 cm. También se indica que es menos preciso en objetos curvos, lo cual pudimos comprobar al comparar las distancias medidas tanto a cubos como esferas..

Experimentalmente pudimos corroborar esta información midiendo la distancia a dos bloques de 5x5 cm, uno sobre el otro, y a una pelota de ping pong de aproximadamente 3.5 cm de diámetro, de donde se obtienen los siguientes datos:

Al medir contra los bloques:

Distancia real	Distancia leída del sensor
20 cm	22 cm
5 cm	5 cm
3 cm	7 cm

Donde se observa que consistentemente el error es de 2 cm, y el sensor brinda una medición aceptable a distancias cortas.

Al medir contra la pelota de ping pong:

Distancia real	Distancia leída del sensor
10 cm	14 cm

20 cm	24 cm

Podemos ver que la diferencia en este caso es de 4 cm, que efectivamente es mayor que el error obtenido contra una superficie plana como la de los bloques.

Adicionalmente se evaluó la incidencia de objetos de una altura menor al centro del sensor. Esto se llevó a cabo colocando el sensor a una cierta altura, la torre de bloques del experimento anterior a 30 cm y la pelota de ping pong entre ellos a 15 cm.

De este experimento se obtienen los siguientes datos:

Altura del sensor	Distancia real	Distancia leída del sensor
6 cm	22 cm	30 cm
4 cm	15 cm	15 cm

De este experimento concluimos que el sensor toma las medidas a la altura del centro de sí mismo, y aunque en los datos se muestre un mayor nivel de error en la medida por encima de la pelota, donde el error en la distancia a los bloques es de 8 cm en comparación a los 2 cm de los experimentos anteriores, consideramos que esta diferencia puede deberse a una interferencia por la pelota y/o por errores en la medición, como una leve inclinación del sensor con respecto a la mesa donde se encontraban los demás elementos.

La distancia máxima que pudimos detectar fue de 1.90 metros. Más allá de este valor, el sensor devuelve el valor por defecto de 255.

Según la especificación, este sensor tiene la capacidad de detectar hasta 8 objetos simultáneamente (*getDistances*), pero desafortunadamente no pudimos testearlo por falta de tiempo.

Parte C - Interacción con el sensor de color

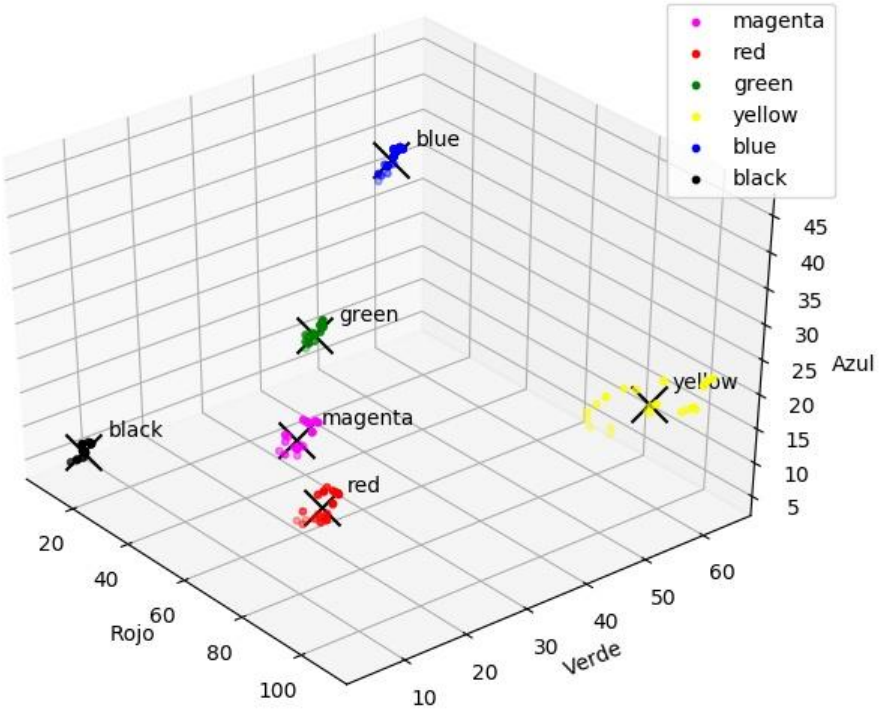
El sensor de color permite la deteccion de colores en formato RGB, a una distancia de 1 cm. Posee también 3 LEDs que emiten luz roja, azul y verde.

Este sensor se controla mediante la clase *lejos.nxt.ColorSensor*, la cual debe instanciarse con el puerto al que está conectado el sensor.

Se calibró el dispositivo para poder detectar 6 colores diferentes: negro, rojo, azul, verde, amarillo y magenta. Para esto se presentó al sensor un cubo de cada color y se tomaron 30 lecturas consecutivas del mismo. Esto se realizó con la luz ambiente del salón de clase procurando que la misma no variara entre una lectura y la siguiente.

Para poder observar los resultados en forma más clara, se hizo un ploteo de las muestras dentro del cubo RGB y se calculó el promedio para cada color. El resultado se muestra en el siguiente gráfico, en donde cada cruz representa el color promedio de las 30 muestras, esto es, el promedio en cada una de las 3 dimensiones.

Luego, para reconocer un color, se toma una nueva muestra RGB y se calcula la distancia euclidiana (Norma L2) a cada uno de los 6 colores promedio. El color reconocido es aquel que resulte más cercano a la muestra dada.



Para la etapa de calibración se implementó la clase *color_calibration* que imprime en la consola las muestras obtenidas junto con el promedio calculado.

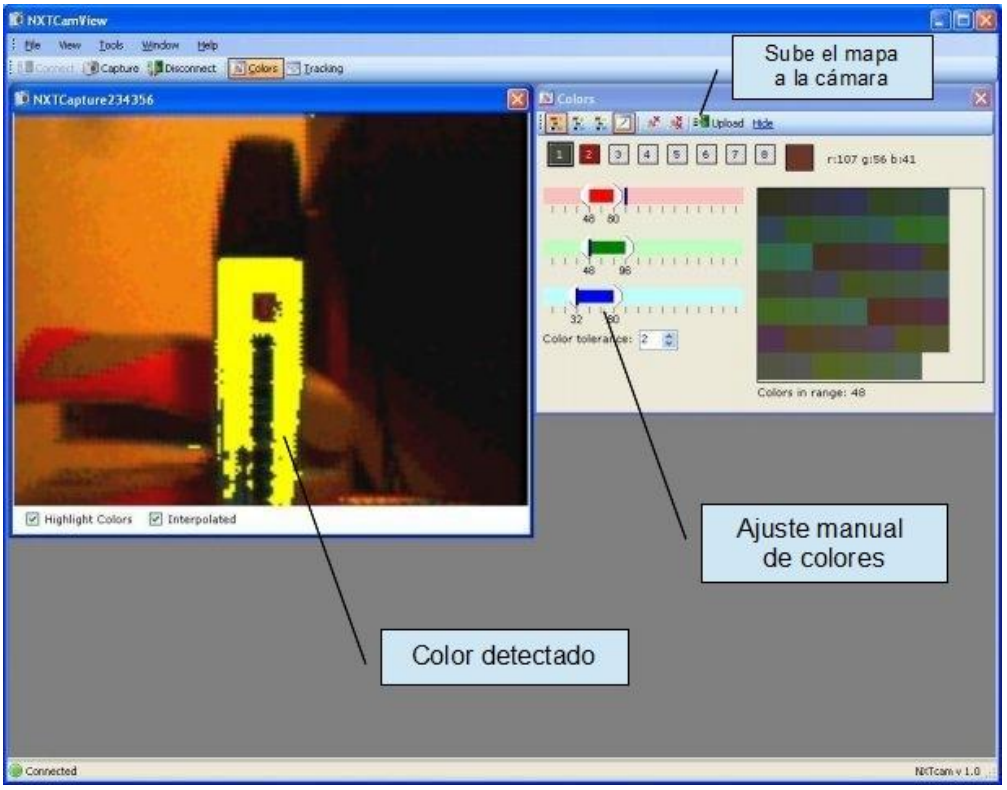
En la etapa de reconocimiento (o clasificación), se implementó la clase *color_sensor* que básicamente calcula la distancia a cada promedio y selecciona el color más cercano.

Parte D - Interacción con la cámara

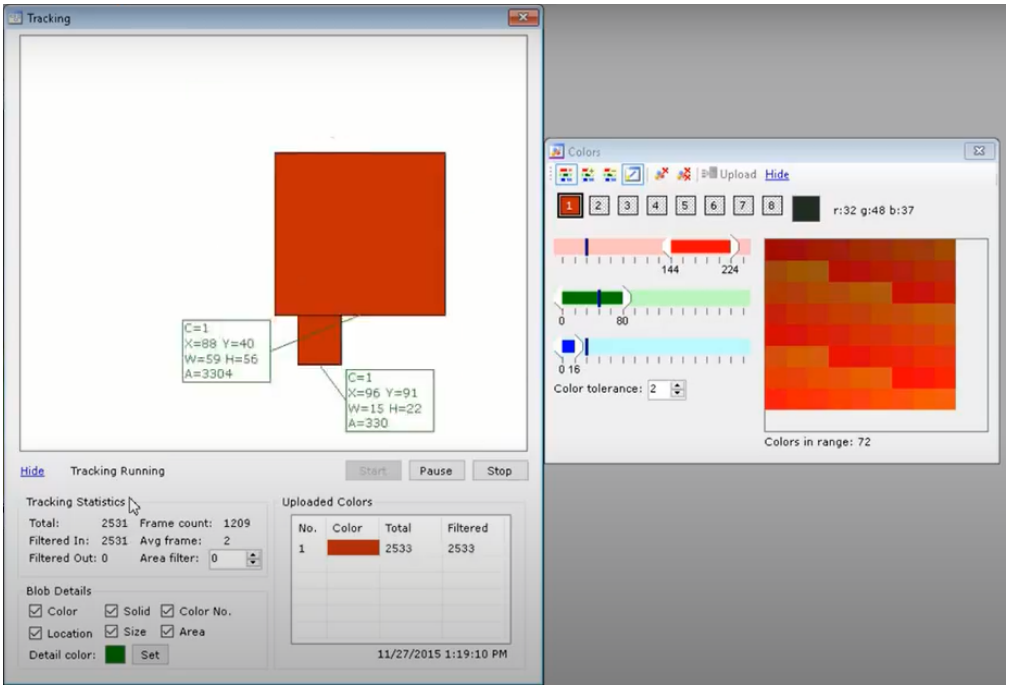
En este ejercicio utilizamos la cámara NXTCam-v3, la cual es capaz de detectar y rastrear hasta 8 objetos (o líneas) de diferentes colores. Esta funcionalidad captura 30 imágenes por segundo de 176x144 píxeles cada una.

Por cada objeto detectado, la cámara calcula un rectángulo que lo contiene y que está definido dentro de las coordenadas de la imagen. En concreto es devuelve: Largo, ancho y posición x e y de la esquina superior izquierda del rectángulo.

Para poder utilizar la cámara, primero es necesario calibrarla con el software NXTCamView. Este software se ejecuta en un PC con Windows XP, y la cámara se conecta vía USB al mismo. El proceso de calibración consiste en capturar una imagen de los objetos a detectar y seleccionar manualmente los píxeles adecuados para indicar qué color o colores queremos que se detecten. En caso de ser necesario, se pueden ajustar los rangos RGB de cada color seleccionado. Una vez definido, el mapa de colores se almacena en la memoria de la cámara. Esta funcionalidad se muestra en la siguiente imagen.



Una vez calibrado los colores deseados, estamos en condiciones de utilizar la funcionalidad de tracking, que puede testearse con el software NXTCamView, donde es posible ver los rectángulos detectados y evaluar los posibles filtros de área para los mismos:



Luego, con el mapa de colores ya almacenado en la cámara, procedemos a conectarla al brick de Lego y la controlamos con la clase Lejos *NXTCam*. Como es usual, instanciamos esta clase con el puerto de sensores al cual la hemos conectado.

Construimos el programa *camera.java* para probar este dispositivo, cuyo pseudocódigo se detalla a continuación:

```
Inicializar la camara (NXTCam cam = new NXTCam)
Habilitar funcionalidad de tracking (cam.enableTracking(true))
Mientras no se presione escape hacer
    Obtener número de objetos (cam.getNumberOfObjects())
    Para cada objeto detectado hacer
        Obtener rectángulo del objeto
        Si largo X ancho > 200px entonces
            Obtener color de objeto (cam.getObjectColor(i))
            Acumular contador del color correspondiente
            Imprimir en consola los detalles del rectángulo
        Fin Si
    Fin Para
    Imprimir resumen de colores detectados (cantidades)
    Esperar a que se presione un botón
Fin Mientras
```

El código descarta los rectángulos pequeños (menos de 200 pixeles) por considerarlos falsos positivos. Otro detalle, no reflejado en el pseudocódigo, es que la obtención del número de objetos (*getNumberOfObjects)* se realiza dentro de un bucle que se repite mientras el número de objetos sea cero. Esto se debe a que notamos que la cámara suele devolver 0 objetos detectados durante ciertos periodos.

El siguiente es un ejemplo de salida en consola:

```
Presionar boton para reconocer:
numOBjects: 3
Objeto 0: Azul, rectangulo (W, H, X, Y): (31.0, 14.0, 24.0, 96.0)
Objeto 2: Verde, rectangulo (W, H, X, Y): (19.0, 63.0, 148.0, 12.0)
Objeto 5: Rojo, rectangulo (W, H, X, Y): (83.0, 17.0, 56.0, 96.0)
#Negro: 0
#Rojo: 1
#Azul: 1
#Verde: 1
#Amarillo: 0
#Magenta: 0
#Otro: 0
```

Lo primero que comprobamos es que el resultado no es muy preciso. En el caso mostrado aquí, se presentaron los 6 cubos a la cámara.

Esto probablemente se debe a que el proceso de calibración necesita ser refinado. También notamos que el algoritmo funcionaba relativamente bien cuando se presentaba un sólo cubo a la cámara, mientras que los resultados eran bastante malos cuando habían varios cubos en la escena.

Colores como el negro y el magenta no se lograron identificar en varios casos debido a que no toda la cara del cubo era reconocida con su color en la captura, causando que su área total fuera menor al filtro y por tanto eliminada. Esto consideramos puede ser resuelto ajustando el filtro a un valor menor para esos colores.

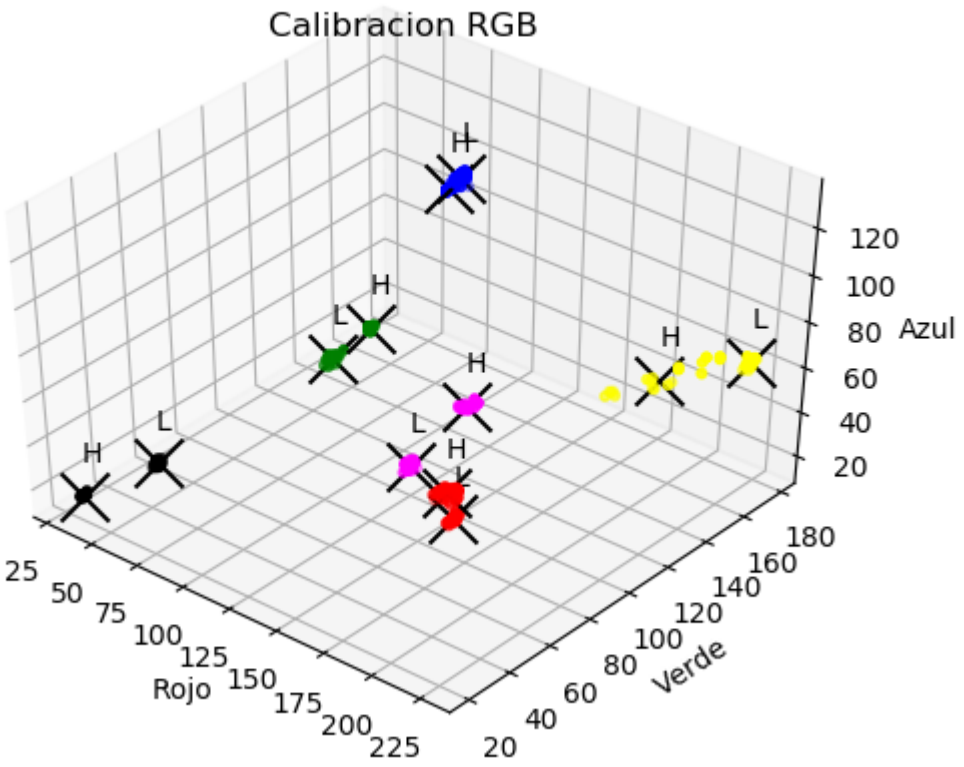
Por falta de tiempo, no pudimos evaluar otras estrategias, como por ejemplo diferentes condiciones de luz o la realización de un calibrado con un mejor grado de verificación, esto es, utilizando múltiples capturas y comprobando que los colores se reconozcan en forma consistente entre diferentes imágenes.

Lo que resultó claro es que si elegimos utilizar este dispositivo para el proyecto final, será necesario dedicar bastante más tiempo a este sensor.

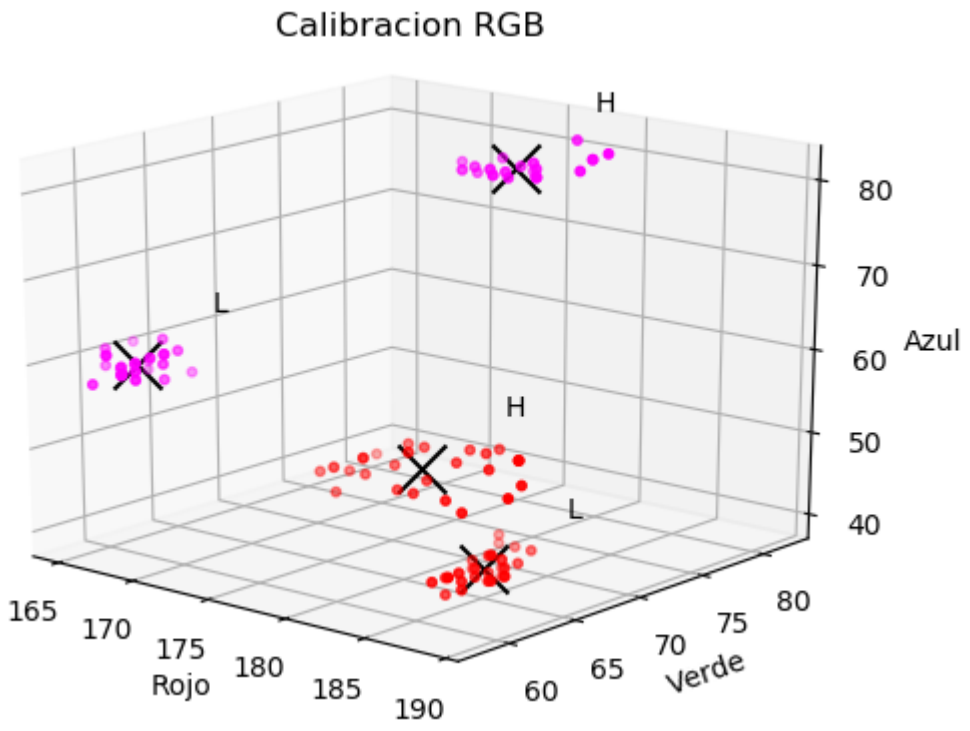
Parte E - Variando el ambiente

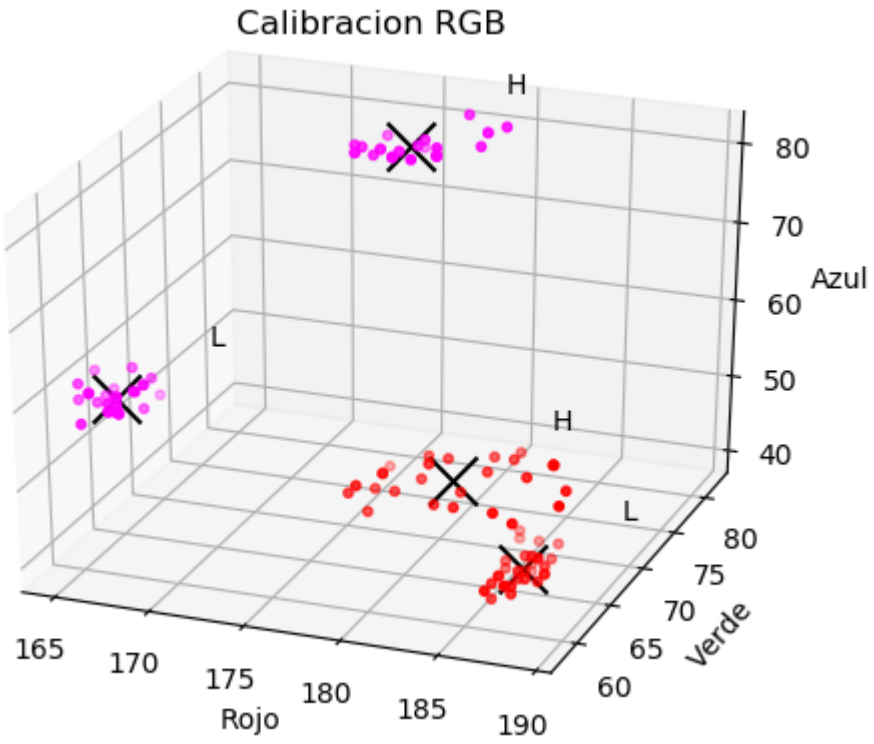
Los experimentos realizados en la parte C se repitieron pero con una luz ambiente reducida, esto es, cubriendo parcialmente el sensor.

Se obtuvieron así dos conjuntos de muestras, que denominamos H (High light) y L (Low light), y nuevamente se plotearon dentro del cubo RGB a modo de comparación, resultando en el siguiente gráfico:



Observamos que los valores H y L resultan relativamente cercanos entre sí con respecto al resto de los colores. Solo el magenta (L) parece acercarse más al rojo (H). Para verlo más en detalle, hacemos un zoom de esta área desde dos ángulos diferentes:





Los valores con mayor luz ambiente (H), parecen estar más dispersos (mayor desviación estándar) que los de menor luz ambiente (L).

A la hora de clasificar colores bajo diferentes condiciones de luz, una opción es obtener una medida de la luz ambiente y a partir de cierto umbral utilizar uno de los dos mapas de colores representados: H o L.

Esto tiene como ventaja el ser un algoritmo muy sencillo de implementar. La principal desventaja es que se necesita tener, idealmente, un gran número de mapas de colores: uno para cada tipo o intensidad de luz ambiente en la que se encuentre el sensor, lo cual no es muy práctico de implementar. De todas formas, en las condiciones del laboratorio, este algoritmo funcionó en forma aceptable.

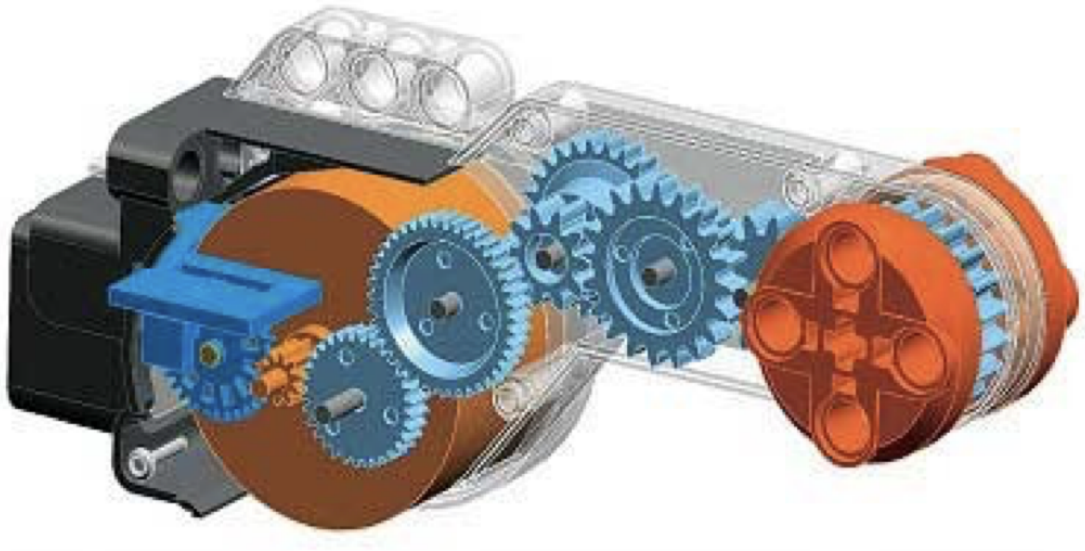
Una alternativa a este método es la de utilizar la capacidad del sensor de emitir luz de un color determinado, mediante sus tres LEDs RGB. Con esta funcionalidad se puede aprovechar el hecho de que al emitir luz de cierto color sobre una superficie de ese mismo color, la luz reflejada tendrá más intensidad que sobre una superficie de un color diferente. De este modo se podrían emitir, en secuencia, los 6 colores a reconocer y simultáneamente obtener una lectura de la luz reflejada. A continuación se determinaría qué color reflejó más luz.

Esta alternativa se intentó pero los resultados fueron muy erráticos. No estamos seguros de cuál fue el problema, pero creemos que probablemente se debió a un uso incorrecto de los métodos de la clase *ColorSensor*. Por otro lado notamos que, al tomar lecturas, el dispositivo emite haces de luz en forma intermitente y no sabemos si efectivamente estábamos tomando una lectura cuando el haz de luz estaba encendido o apagado. Por cuestiones de tiempo, no pudimos profundizar en el análisis de esta funcionalidad.

Parte F - Cuestionario

Motores

El kit robótico utilizado posee servomotores. Estos están compuestos por motores DC con escobillas, un encoder y un sistema de control, a través de pulsos de período variable (PWM), donde el motor puede posicionarse de forma precisa según una cierta posición o velocidad y posee la capacidad de devolver como feedback su posición (rotación). Además se disponen de engranajes para regular la fuerza transmitida por el motor, vistos en celeste en la siguiente imágen.



El tacómetro es un sensor que mide la velocidad, en el caso de nuestro motor se hace uso del encoder, la pieza azúl oscura que vemos en la imágen anterior, que devuelve la posición. Podemos clasificar al tacómetro en las siguientes categorías:

- Interno/Introceptivo - Mide el giro del motor, el cuál pertenece al robot.
- Pasivo - Ya que no afecta al entorno. Se utiliza el encoder, que emite un rayo de luz entre las ranuras de su engranaje y es recibido por sí mismo para medir la posición del motor, sin que ese rayo salga de la estructura del motor y afecte al ambiente.
- Local - Ya que se encuentra montado en el robot, específicamente en el motor, que como mencionamos anteriormente es parte de este.
- Digital - Como mencionamos anteriormente utiliza los mecanismos discretos del encoder para medir la posición del motor.

Sensor de color

Las características del ambiente que notamos que afectaron al sensor fueron la distancia al objeto medido y el ángulo con respecto al sensor. También la iluminación del ambiente, aunque en los experimentos realizados con la metodología y los valores finales de parametrización utilizados esta resultó ser muy poco influyente, mientras que la distancia y ángulo del objeto tuvieron un impacto notorio en el resultado, con tan sólo una pequeña variación de estos valores.

Cámara

La ventaja más notoria de la cámara sobre el sensor de luz es la amplitud y la distancia de medición mayores, lo que a su vez le permite detectar más de un objeto.

Esta ventaja también trae consigo la desventaja de tener una calibración más compleja y artesanal que la encontrada para el sensor de color, ya que se requiere ajustar los rangos de RGB medidos para cada color manualmente de forma que cubra todas las posibles condiciones del ambiente, además de un ajuste en el filtrado de áreas para la detección de objetos si se trabaja con distancias suficientemente grandes donde los objetos lejanos puedan verse muy próximos en tamaño a las medidas erroneas filtradas.

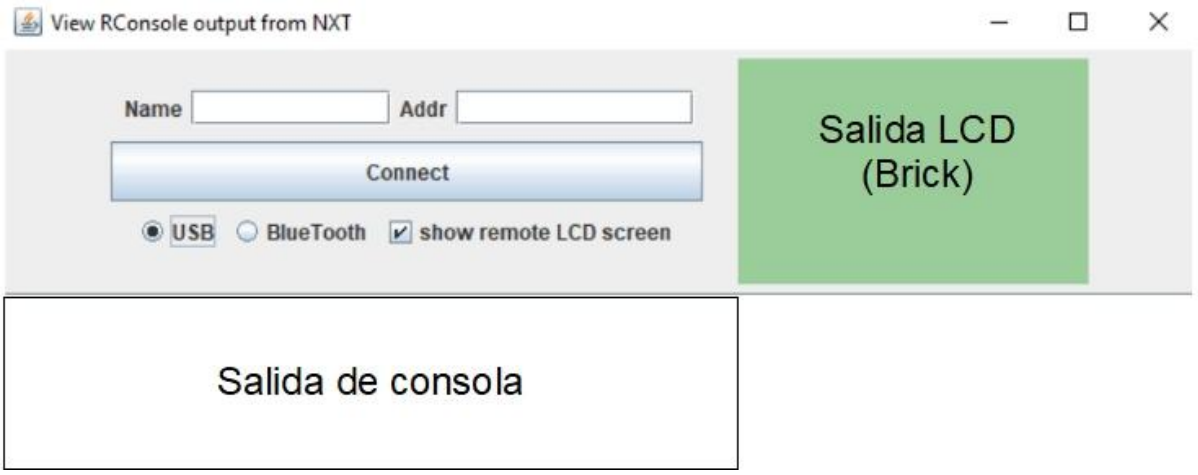
Por otro lado el sensor de color requiere, para su correcto funcionamiento y de acuerdo a la especificación del dispositivo, que la lectura se haga a 1cm de distancia aproximadamente, lo cual fue comprobado durante los experimentos. Bajo estas condiciones, la metodología utilizada para clasificar colores descrita previamente alcanza un buen desempeño en una variedad de condiciones de luz, al menos dentro del laboratorio.

Anexo: Compilación y consola

Los programas java detallados previamente se han compilado, linkeado y cargado al brick LEGO utilizando los siguientes pasos:

c:> nxjc <*archivo*>.java
c:> nxjlink -o <*archivo*>.nxj <*archivo*>
c:> nxjupload <*archivo*>

Una vez iniciado el programa en el brick, éste espera una conexión con el programa de consola *nxjconsoleviewer* que corre en el PC:



Referencias

“Clase 02: Robótica móvil, de RBC.” n.d. Accessed March 25, 2023. https://eva.fing.edu.uy/pluginfile.php/25433/mod_resource/content/6/teorico/RBC23_Clase02_Robotica%20Movil.pdf.

“LEGO Color Sensor (9694) : Toys & Games.” n.d. Amazon.com. Accessed March 25, 2023. https://www.amazon.com/LEGO-9694-Color-Sensor/dp/B005GXTD2M.

“LEGO Linear actuator.” n.d. Accessed March 25, 2023. https://www.actuonix.com/l12-ev3-100.

“LEGO Mindstorm User Guide.” n.d. Lego. Accessed March 25, 2023. https://www.lego.com/cdn/product-assets/product.bi.core.pdf/4589647.pdf.

“LEGO NXT Ultrasonic sensor : Amazon.co.uk: Toys & Games.” n.d. Amazon UK. Accessed March 25, 2023. https://www.amazon.co.uk/LEGO-4297102-NXT-Ultrasonic-sensor/dp/B000PM8I8O/ref=sr_1_7?keywords=Lego+MINDSTORMS+Ultrasonic+Sensor&qid=1679782087&sr=8-7.

“The leJOS NXJ Tutorial.” n.d. leJOS. Accessed March 25, 2023. https://lejos.sourceforge.io/nxt/nxj/tutorial/.

“NXTCamView.” n.d. Accessed March 25, 2023. https://nxtcamview.sourceforge.net/.

“NXT® motor internals.” n.d. Philo's Home Page. Accessed March 25, 2023. https://www.philohome.com/nxtmotor/nxtmotor.htm.

“Vision Subsystem - Camera for NXT or EV3 (NXTCam-v4).” n.d. mindsensors.com. Accessed March 25, 2023. http://www.mindsensors.com/ev3-and-nxt/14-vision-subsystem-camera-for-nxt-or-ev3-nxtcam-v4.

“Clase Elementos Estructurales - Parte 2, de FRA.” https://eva.fing.edu.uy/pluginfile.php/113707/mod_resource/content/4/2.2%20-%20Elementos%20estructurales.pdf