DEPARTAMENTO DE INGENIERÍA E INVESTIGACIONES TECNOLÓGICAS
SISTEMAS OPERATIVOS AVANZADOS

Sistemas Embebidos (IoT) y Android

Comisión: Miércoles Noche
Ciclo Lectivo: 2º Cuatrimestre 2018

Docentes:

- Lic. De Luca, Graciela
- Ing. Valiente, Waldo
- Ing. Carnuccio, Esteban
- Ing. Volker, Mariano
- Ing. García, Gerardo

Integrantes:

- Aranda Ocampo, Alexis 39879304
- Cavicchioli, Lara 40024525
- Espínola Hamm, Carla 38014451
- Sandagorda, Patricia 39413616
- Silvestri, Valeria 39912062

ÍNDICE

OBJETIVO:

El objetivo de este trabajo práctico es desarrollar un sistema embebido que clasifique confites marca Rocklet por color. El sistema recibirá uno o varios rocklets y los depositará en distintos recipientes según su color.
Tomando en cuenta los avances tecnológicos actuales, y el creciente miedo a el día de la revolución de las máquinas, el sistema incluirá la posibilidad de tomar control manual de algunas partes móviles.[1] También estará integrado con una aplicación de Android, a través de la cual el usuario podrá ver en tiempo real cuántos rocklets han sido clasificados para cada color, podrá pausar y reanudar su funcionamiento y tomar el control manual a distancia.

ALCANCE:

- Detección de un rocklet
- Movimiento entre estaciones
- Detección de color
- Selección del recipiente
- Direccionamiento del tobogán automático, manual y por celular
- Cambio de modalidades automático, manual y celular
- Iluminación a través de un led
- Comunicación bluetooth con la aplicación
- Reportes de cantidad de rocklets
- Pausar y reanudar mediante la aplicación

FUNCIONAMIENTO LÓGICO

Inicialmente, los rocklets se apilan y el sistema embebido (apodado Rocky) los recibe de a uno. Por cada rocklet, Rocky sensa su color y lo dirige al recipiente correspondiente. Para dirigir el rocklet utiliza un tobogán móvil, el cual puede ser controlado automáticamente por el sistema o manualmente por el usuario. Rocky también lleva la cuenta de cada color detectado y reporta estadísticas a un dispositivo Android que tenga instalada su aplicación Rocky App. Si no hay rocklets apilados, Rocky simplemente espera.

El sistema cuenta con tres modalidades:

- Automático: no necesita intervención humana para realizar las actividades. (NOTA: ante un eventual levantamiento de las máquinas, abstenerse de utilizar dicho modo)

[1] Con dicho escenario apocalíptico en mente, también se tomó la decisión de nombrarlo Rocky (en honor a un famoso boxeador de los años 70) y darle un buen trato durante todo el desarrollo, con la esperanza de que nos recuerde como sus buenos creadores.

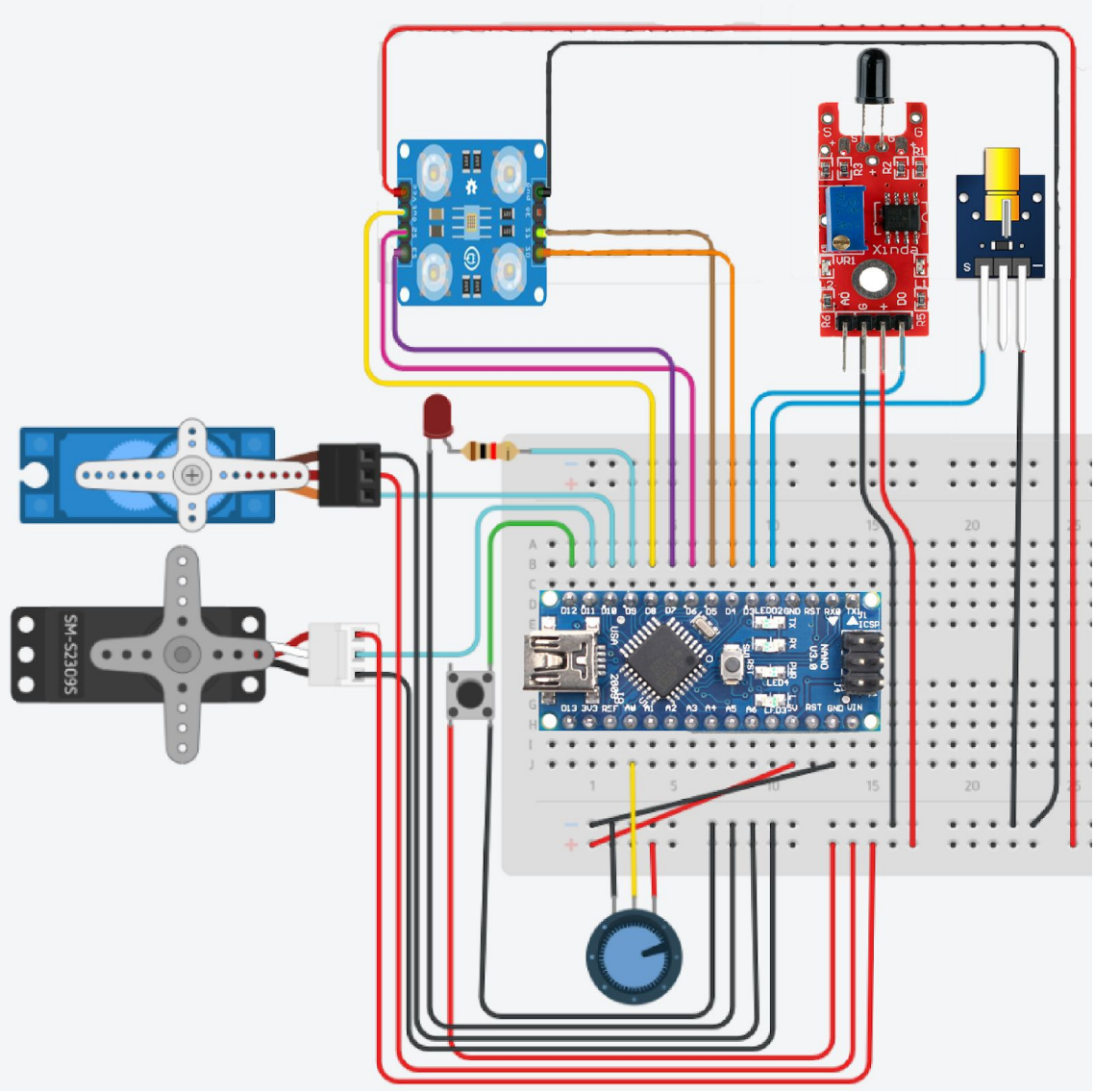
- Manual: el usuario podrá hacer un override del hardware, dirigiendo el tobogán con precisión mediante un potenciómetro y decidiendo el momento exacto a despachar el rocklet presionando un pulsador.
- Celular: el usuario podrá hacer el mismo override del hardware a distancia, utilizando Rocky App para Android.

El usuario podrá cambiar entre modo Manual y Automático presionando el pulsador de Rocky durante medio segundo. Si se cuenta con la Rocky App, es posible entrar en el modo Celular, y salir de él volviendo inmediatamente al modo Automático, mediante un botón en la aplicación. Rocky indica a través de su LED en qué estado se encuentra.

HARDWARE UTILIZADO:

- *MATERIALES UTILIZADOS*
  - Placa Arduino NANO
  - Protoboard
  - Una amplia variedad de cables
  - Cartón, escarbadientes, silicona líquida, cola vinílica, mucha cinta, entre otros materiales para la estructura
- *SENSORES*
  - 1 sensor de flama (xinda)
  - 1 pulsador
  - 1 potenciómetro
- *ACTUADORES*
  - 2 servo motor
  - 1 LED
  - 1 laser rojo (xinda)
- *SENSORES y ACTUADORES*
  - 1 sensor de color (TCS 3200)
  - Módulo Bluetooth
- *SENSORES ANDROID*
  - Giróscopo
  - Sensor de proximidad
  - Sensor de luz

CONEXIONES

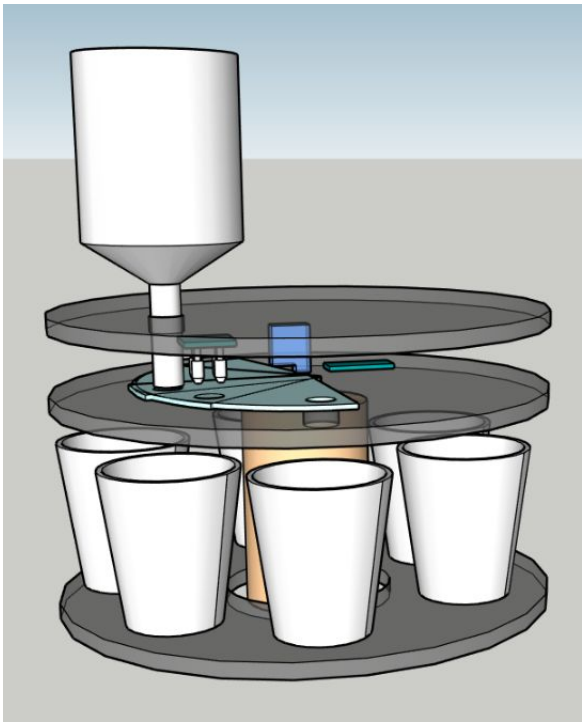


Componente	Pata	Pin	Modo
Laser	S	D2	OUTPUT
Detector de flama	D0	D3	INPUT
Sensor color	S0	D4	OUTPUT
	S1	D5	OUTPUT
	S2	D6	OUTPUT
	S3	D7	OUTPUT
	OUT	D8	INPUT
LED	larga	D9	OUTPUT

ServoTobogán	sig	D10	OUTPUT
ServoCinta	sig	D11	OUTPUT
Pulsador	sig	D12	INPUT
Potenciómetro	sig	A0	INPUT
Bluetooth	RX	TX	OUTPUT
	TX	RX	INPUT

DETALLES TÉCNICOS DE LOS COMPONENTES UTILIZADOS:

➢ Estructura



Pasamos por muchas versiones de la estructura. La primer versión consistía en dos pisos circulares y una bandeja giratoria para los vasos. Pero la bandeja necesitaba girar al menos 360° y los servos giran 180°.

La estructura actual se compone de una especie de estantería semicilíndrica la cual tiene varios “pisos”, que alojan distintos componentes:

Base: Sirve para mantener unida la estructura. Cubre todo el medio círculo.

Techo: La tapa superior, además de servir para mantener la estructura, este piso aloja el pulsador, el potenciómetro para su operación, el led para poder observar el estado actual, y es atravesado por el tubo que contiene los rocklets.

Entrepiso: Este piso también es atravesado por el tubo que contiene los rocklets, y debe estar ubicado bastante cerca del inferior (debe haber 5mm entre el disco y el techo para ser exactos) ya que tiene incrustado el sensor de color que sensa el rocklet en el piso de abajo. Para obtener una lectura óptima del color, los leds del sensor se ubican a 1 mm del disco. de distancia de la estación de sensado del rocklet en sentido vertical. Por otro lado este piso también tiene incrustado del lado inferior el láser y el detector de flama, que están ubicados de tal forma que el láser atraviesa el medio del tubo casi donde termina el mismo, para poder detectar si hay un rocklet que precisa ser

atendido. Además contiene la protoboard con el arduino y es el centro al que llegan y de donde salen todos los cables a los demás componentes.
Piso de sensado: Este piso contiene una especie de semi disco giratorio que funciona como una cinta transportadora: recibe los rocklets que caen por el tubo a una hendidura del disco y los transporta a las demás estaciones. Este disco se desliza sobre una plataforma semicircular pegada sobre el estante en sí, que evita que el rocket caiga más allá de la hendidura del disco. El movimiento del disco está dirigido por el Servo Cinta que tiene unido por debajo, el cual va trasladando al rocklet desde que cae por el tubo (Estación de RECEPCIÓN), hasta debajo del sensor de color (Estación COLOR) y luego a su caída por el agujero de la plataforma (Estación CAÍDA). El sistema de traslado es sencillo: El rocklet está contenido por el agujero del disco y el piso de la plataforma hasta que coinciden los agujeros del disco y el que tiene la plataforma ubicado justo por encima del tobogán.
Piso de tobogán: Tiene la tarea de sostener un tobogán de acrílico que está fijado al Servo Tobogán. Luego de que el rocket cae por la hendidura superior este aterriza en el interior del tobogán, el cual lo transporta al vaso correspondiente. El movimiento del tobogán es controlado por el servo que tiene adjunto que lo ubica apuntando a alguna de las 6 estaciones que tiene definidas donde finalmente caen a los vasos. El piso cubre casi todo el medio círculo, excepto en los costados del tobogán para que este mismo rote en 180 grados.

➢ Arduino NANO

Procesador: ATmega328
Voltaje de salida: 5 V
Alimentación: 7 a 12 V
Pines digitales: 14, de los cuales 6 funcionan con PWM
Pines de entrada analógica: 8
Corriente DC por pin: 40 mA
Memoria flash: 32 KB (ATmega328) de los cuales 2 KB son usados por el bootloader
SRAM: 2 KB (ATmega328)
EEPROM: 1 KB (ATmega328)

Los pines digitales que pueden trabajar con PWM son: 3, 5, 6, 9, 10 y 11. Casi todos trabajan a 490 Hz, salvo 5 y 6 que trabajan a 980 Hz. 5 y 6 no los usamos con servos, porque el valor PWM lo testeamos con 490 Hz.

Función en Rocky:
El Arduino NANO es el cerebro de Rocky: controla todos los componentes y toma las decisiones necesarias.

➢ Sensor de color TCS3200

Tipo de señal: Digital
Máxima frecuencia (sin producir saturación): 600KHz
Alimentación: 2.7V a 5.5V

El sensor TCS3200 convierte la intensidad de luz medida por una matriz de fotodiodos, en frecuencia. Dicha matriz cuenta con filtros de color (16 filtros rojos, 16 filtros verdes, 16 filtros azules y 16 sin filtro) que, al ir alternando su estado, sirven para distinguir cada componente.
Los 64 fotodiodos se activan de a grupos para realizar la medición de la intensidad de luz que incide en ellos. Para elegir qué filtros deben activarse en cada momento se cuenta con dos patas, la S2 y S3.

S2	S3	COLOR DETECTADO
L	L	ROJO
L	H	AZUL
H	L	SIN FILTRO
H	H	VERDE

Es posible variar la frecuencia en tres niveles, la normal (máxima) al 100%, una media al 20% y otra baja al 2%. Las patas S0 y S1 sirven para establecer estos valores de frecuencia.

S0	S1	FRECUENCIA
L	H	2%
H	L	20%
H	H	100%

Variar la frecuencia es útil para optimizar la lectura del sensor al utilizar un microcontrolador.

Función en Rocky:
Usamos el sensor de color para detectar el color de cada rocklet. Utilizamos la frecuencia media, es decir al 20%, que es la que funciona correctamente con los pines PWM de Arduino NANO.

➢ Flame Detector

Tipo de señal: Digital
Alimentación: 3.3V a 5V
Distancia de detección: 0.40m a 0.80m
Sensible a ondas entre 700nm y 1100nm

Este sensor permite detectar la existencia de calor por la luz emitida de la llama. Detecta ondas entre 700nm y 1100nm, que son componentes de luz infrarroja.

Función en Rocky:
En nuestro proyecto, utilizamos el detector de flama para detectar un láser rojo. Dicho láser será bloqueado cuando se encuentre un rocklet para analizar, y podrá ser detectado por el sensor de flama cuando ya no haya más rocklets.

➢ Pulsador

Tipo de señal: Digital
Alimentación: 5V
Vida eléctrica: 100000 ciclos
Clasificación: 50mA 12VC
Temperatura ambiental: -25 ° C a 105 ° C

Este sensor detecta el momento en que está siendo pulsado, utilizando un circuito abierto el cual devuelve LOW, pero que si se cierra manteniendo presionado el botón devuelve HIGH. Tiene la particularidad de causar ruido a veces, sea por una presión suave por el botón, o bien por el rebote del mismo al ser presionado.

Función en Rocky:
En nuestro proyecto, utilizamos el botón de dos formas diferentes:

1) Para pasar del modo Automático al Manual y viceversa con un pulso largo (más de medio segundo)
2) Para soltar rocklets en el modo Manual con un pulso corto (más de 50 milisegundos, menos de medio segundo)

➢ Potenciómetro

Tipo de señal: Analogica
Alimentación: 5V

Un potenciómetro es una perilla simple que proporciona una resistencia variable, que podemos leer en la placa Arduino como un valor analógico.
Al girar el eje del potenciómetro, variamos la resistencia que está aplicada al pin central del potenciómetro. Esto cambia la "cercanía" relativa de ese pin a 5 voltios y tierra, lo que nos da una entrada analógica diferente. Cuando el eje se gira completamente en una dirección, hay 0 voltios que van al pin, y leemos 0. Cuando el eje se gira completamente en la otra dirección, hay 5 voltios que van

al pin y leemos 1023. En el medio, analogRead() devuelve un número entre 0 y 1023 que es proporcional a la cantidad de voltaje que se aplica al pin.

Función en Rocky:
En nuestro proyecto, el potenciómetro tiene la función de controlar el movimiento de rotación del servo tobogán, mientras se esté en el modo Manual

➢ Servo motor TowerPro SG90 Micro Servo (x2)

Tipo de señal: PWM
Alimentación: 4.8V - 6V
Torque: 1,5 Kg
Ángulo de rotación: 180° (teóricamente)
Dimensiones (LxWxH): 22,0 x 11,5 x 27 mm
Temperatura de operación: 0 a 50°C
Peso: 10.6 gr

Función en Rocky:
En nuestro proyecto, utilizaremos un servo motor para mover el rocklet desde que se ubica en la parte inferior del tubo de recepción del embebido, llevándolo a la estación para la lectura del color y por último a la estación en la que se lo despacha.

El segundo servo lo utilizaremos para movilizar el tobogán que permitirá trasladar los rocklets hasta su correspondiente recipiente

➢ Led

Tipo de señal: PWM
Alimentación: 5V
Color de luz: Blanco
Resistencia utilizada: 470 Ohms

Función en Rocky:
En nuestro proyecto, el led es utilizado para mostrar el estado y el modo actual en el que está el sistema.

➢ Laser Xinda

Alimentación: 5V
Tipo de señal: Digital
Longitud de Onda: 650 nm
Color de luz: Rojo

Función en Rocky:
En nuestro proyecto, utilizaremos el láser para que, en conjunto con el flame detector, funcionen como una barrera para la detección de Rocklets.

➢ Bluetooth HC-05

Bluetooth 2.0 EDR
Modelo: ZG-B23090W
Basado en el chip csr BC417
2 modos: AT mode y communication mode.
AT mode: 38400 baudios
Communication mode: 9600 baudios

Cuando lo conectamos por primera vez, muestra el nombre HC-05 y se empareja con el pin 1234. Primero debemos configurarlo. Para ello, cargamos en Arduino el sketch de configuración que se encuentra en nuestra repo de GitHub:
https://github.com/alexis-aranda/Rocky/tree/master/Sistema%20embebido/Pruebas%20unitarias/Configuracion_bluetooth
Este sketch recibe comandos a través del Serial Monitor, los envía al módulo Bluetooth y además los muestra por pantalla anteponiendoles un “>”, junto con las respuestas del módulo.
Como necesitamos dos conexiones seriales (una para el Serial Monitor y otra para el BlueTooth), conectamos el RX del BT al pin D9 y TX del BT al D8 y usamos la biblioteca SoftwareSerial.h para mapearlos a TX y RX del Arduino respectivamente. Estas conexiones seriales además utilizan diferentes velocidades de transmisión: el Serial Monitor trabaja a 9600 baudios y el BT en modo AT a 38400 baudios.
Entramos en modo AT presionando el botón del módulo mientras conectamos la alimentación. Para que los comandos sean interpretados correctamente por el Bluetooth, estos deben terminar con \r\n, por lo tanto el Serial Monitor debe ser configurado con “Both NL & CR”.
Enviamos los siguientes comandos:

1. AT: devuelve OK si la conexión es correcta
2. AT+ORGL: borra la configuracion volviendo a los valores de fábrica
3. AT+NAME=Rocky: lo nombra Rocky
4. AT+ROLE=0: lo define como Esclavo
5. AT+UART=9600,0,0: define la velocidad de transmisión entre el HC05 y Android a 9600 baudios

Por último con AT+RESET o desconectando y volviendo a conectar la alimentación se reinicia el módulo BT y este vuelve a entrar en el modo normal

de comunicación. Entonces podemos comprobar desde cualquier celular que aparece como Rocky y se empareja con el pin 1234.
Ya tenemos el Bluetooth listo para nuestro proyecto.

Función en Rocky:
Comunicar la cantidad de cada color de rocklets a la aplicación de Android, así como los cambios de modo, y recibir comandos de pausa, reanudación y posición a mover el tobogán cuando se encuentra en modo Celular.

SOFTWARE ARDUINO

Generamos distintas bibliotecas y controladores para los distintos componentes del sistema. Además, el funcionamiento se planteó en varios estados para resolver todos los pasos necesarios.
Todo el código se encuentra en nuestro repositorio de GitHub: https://github.com/alexis-aranda/Rocky/tree/master/Sistema%20embebido/rocky

Pero primero definamos algunos términos específicos de nuestro sistema:
Llamamos Servo Cinta[2] al servomotor que mueve los rocklets entre las estaciones RECEPCION_ST, COLOR_ST y CAIDA_ST.

- RECEPCION_ST es la estación donde se recibe el siguiente rocklet.
- COLOR_ST es la estación donde el sensor de color realiza sus lecturas.
- CAIDA_ST es la estación donde el rocklet cae al tobogán.

Llamamos Servo Tobogán[3] al servo motor que mueve el tobogán y redirige el rocklet a una de las 6 estaciones donde se encuentran los recipientes.

BIBLIOTECAS

Color rocklet

Esta biblioteca inicializa todos los pines de requeridos por el sensor de color, los cuales se reciben en el constructor. Además, mediante el método apagarSensor() pone al mismo en modo ahorro de energía.
Realiza 3 lecturas, sensando los valores de rojo, amarillo y verde (un valor por cada lectura), e identifica el color con esos datos, utilizando para ello el método identificarColor() que analiza los valores leídos y a qué parámetros de colores pertenecen.

Nuestra barrera laser

La biblioteca tiene un constructor que recibe los pines del Laser y del Sensor de flama, y le asigna el modo OUTPUT al pin de Láser.

[2] También apodado Servo Negro.
[3] También apodado Servo Azul.

La clase permite activar y desactivar el láser (y por consiguiente la detección), mediante los métodos activarBarrera() y desactivarBarrera() respectivamente, los cuales envían la orden de HIGH o LOW al pin de láser para que este prenda o apague. Además, verifica si se está cortando en ese momento la barrera por un rocklet, utilizando el método detecta()

Nuestro LED

La biblioteca tiene un constructor que recibe el pin del led, y le asigna el modo OUTPUT.
La clase permite setear el modo actual de operación, con el método setModo(), al cual se le pasa uno de los 5 modos del led. A su vez, la clase tiene un método activar(), el cual debe ser ejecutado en cada vuelta del loop principal, para que opere con el led según su modo, y en caso de estar en modo manual, se le pasa el valor del potenciómetro.
Los modos son:
PRENDE_APAGA: En este modo, el led verifica si está prendido, y pasado cierto tiempo apaga, y viceversa
INTENSIDAD_VARIABLE: En este modo, se setea según lo que reciba del potenciometro
SOFT_PWM: En este modo, el led se va prendiendo lentamente hasta estar totalmente prendido, y luego se va apagando hasta estar completamente apagado
SIEMPRE_PRENDIDO: El led se mantiene en estado HIGH (Prendido al 100%)
SIEMPRE_APAGADO: EL led se mantiene en estado LOW (Apagado totalmente)

Nuestro potenciómetro

La biblioteca tiene un constructor que recibe el pin del potenciómetro, y lo pone en modo INPUT.
Se puede obtener la posición actual del potenciómetro con el método getPosicion()

Nuestro pulsador

La biblioteca tiene un constructor que recibe el pin del pulsador, y lo pone en modo INPUT.
Se debe ejecutar en cada vuelta el método chequear() para que se chequee el estado actual del pulsador (En este método se determina si hubo pulso corto, largo o no hubo pulso válido)
Por otro lado, con los métodos detectaLargo() y detectaCorto(), avisa si alguno de estos estados ocurrió recientemente

De esta forma, el pulsador solo devuelve que hizo pulso si se mantuvo presionado constantemente en un rango de tiempo, con lo que se evitan las pulsaciones accidentales por ruido o rebote

Nuestro servo

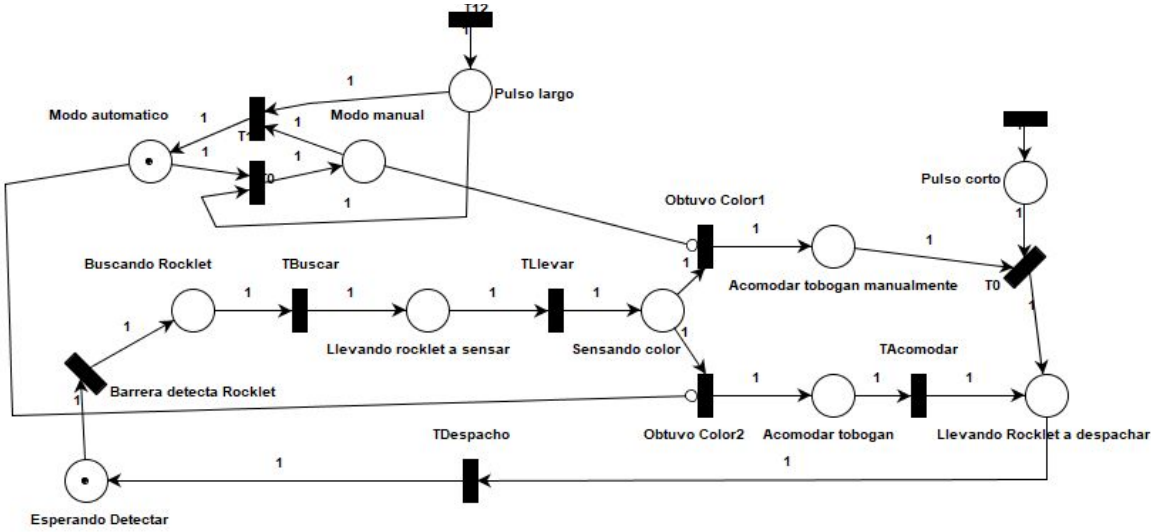
La clase NuestroServo ofrece constantes estáticas para referenciar a las tres estaciones del Servo Cinta (RECEPCION_ST, COLOR_ST y CAIDA_ST) y las seis estaciones del Servo Tobogán (ST_1 a ST_6). Estas constantes guardan los anchos de pulsos que se deben enviar por PWM para que el servo se mueva a la estación requerida. Dichos valores se deben ajustar para cada servo en particular.

El constructor recibe el pin a través del cual enviará las señales, un límite inferior y otro superior para valores PWM. Con estos límites se asegurará que nunca se le pide una posición fuera de rango.

Una vez instanciado un servo, se pueden usar los métodos irA(), que recibirá una posición en su rango PWM (de preferencia, una de las estaciones definidas en las constantes), e irAAnalogico(), que recibirá un valor entre 0 y 1023 y lo mapeará a su rango de movimiento. Los valores son entre 0 y 1023 para facilitar la comunicación con el potenciómetro.

ESTADOS

Los estados y los pasajes entre ellos se resumen en el siguiente diagrama de Petri.



El loop general siempre chequea el estado del pulsador y del bluetooth y envía una señal al controlador del LED en cada vuelta. Si detecta un pulso largo, cambia entre modo Automático y Manual. Además ejecuta diferentes funciones según en qué estado se encuentre.

En espera

En este estado se activa la barrera laser y se espera hasta que esta detecte la presencia de un rocklet. Entonces pasa a Buscando.

Buscando

Se indica al servo Cinta que se dirija a RECEPCION_ST y se espera un segundo para asegurarnos que el servo llegue a destino y reciba un rocklet antes de cambiar al siguiente estado: Llevando.

Llevando

Se indica al servo Cinta que se dirija a COLOR_ST. Nuevamente se espera un segundo y luego se cambia a Sensando.

Sensando

Se enciende el sensor de color y se llama al método hacerLectura() hasta que éste indique una lectura correcta.[4] Entonces se obtiene el color sensado, se incrementa el contador correspondiente, se informan los contadores actualizados a Rocky App y según el modo actual se pasa al estado Tobogán Automático o Tobogán Manual.

Tobogán Automático

Mientras continúe en modo automático, se decide la estación correspondiente según el color y se indica al servo Tobogán dicha estación. Luego se espera un segundo y se pasa a Despachando.
Si en algún momento durante el proceso hubiera un cambio de modo a manual, se aborta el proceso y se pasa a Tobogán Manual.

Tobogán Manual

Mientras continúe en modo manual y no detecte un pulso corto, por cada loop se toma la posición del potenciómetro, que será un número entre 0 y 1023, y se indica al servo Tobogán que se dirija a dicha posición a través del método irAAnalógico(). Cuando se detecta un pulso corto, se pasa a Despachando.
En caso de encontrarse en modo celular, el servo tobogán se moverá en función al sensor de rotación del celular que esté utilizando Rocky App. Dicha aplicación enviará un valor entre 0 y 255. Para pasar a despachando en este caso, se deberá recibir una señal del bluetooth.
Si en algún momento durante el proceso hubiera un cambio de modo a automático, se aborta el proceso y se pasa a Tobogán Automático.

Despachando

El servo Tobogán ya está alineado con el recipiente correspondiente. Se indica al servo Cinta que dirija a CAIDA_ST, se espera un segundo y se pasa al estado En espera, reiniciando el ciclo.

El LED también tiene varios estados posibles para reflejar los estados anteriores. En todos los cambios de estado se llama al método setearLED(), que setea el modo del LED según el nuevo estado.

---

[4] El método hacerLectura() requiere varios llamados a lo largo de varios loops. Referirse a la sección de la biblioteca Color Rocklet o a Lecturas de color en Problemas y soluciones para más información.

Durante el estado Tobogán Manual, el LED estará en modo INTENSIDAD_VARIABLE, variando su intensidad según la posición del potenciómetro. En todos los otros estados, y mientras se mantenga el modo manual, estará en PRENDE_APAGA.
Mientras se esté en modo automático, si está En espera el LED estará en SOFT_PWM, si está en Tobogán Automático o Despachando estará SIEMPRE_PRENDIDO, y para el resto (Buscando, Llevando, Sensando) estará SIEMPRE_APAGADO.

En todos los loops se chequea si hay algún mensaje nuevo por bluetooth, incluso si el sistema está en pausa. A través del bluetooth se reciben comandos del celular.
Si el sistema está en pausa, solo puede recibir el comando SEGUIR para salir de la pausa. Si no está en pausa puede recibir:

- PASAR_A_CELU: pasa a modo Celular. Para evitar conflictos con el potenciómetro, es un modo específico con más prioridad que Automático y Manual.
- SALIR_DE_CELU: vuelve a modo Automático y lo informa por bluetooth a la Rocky App.
- PAUSAR: pausa el sistema. Los servos se siguen moviendo hasta llegar a la última posición indicada. Todo las demás funciones se detienen en el punto que estén. Mientras esté en pausa no chequeará ningún sensor, solo el bluetooth para recibir la orden de salir de pausa.
- POSICIONAR: si está en modo Celular, lee el siguiente byte, lo interpreta como un número y lo pone en una variable para poder ubicar el tobogán cuando llegue a Tobogán Manual. Para reducir el ruido, tomará sólo valores de 1 a 254.
- SOLTAR: es la orden de despachar el rocklet una vez ubicado el tobogán en modo Celular. Si se encuentra en el estado Tobogán Manual, pasa a Despachando, sino guarda la señal para cuando llegue a ese estado.

SOFTWARE ANDROID

La aplicación posee varias Activities y clases independientes que le dan funcionamiento.
Todo el código se encuentra en nuestro repositorio de GitHub: https://github.com/alexis-aranda/Rocky/tree/master/Android/App

En detalle estas funcionalidades son:

- Pausa y reanudación del clasificador de rocklets, de forma remota
- Conteo de Rocklets por vaso
- Informe constante del modo actual de operación

- Nuevo modo para el embebido: Celular
- En modo Celular, se permite mover el tobogán con el celular.
- En modo Celular, se deja caer el rocklet acercando la mano a la pantalla
- La aplicación cambia la tonalidad del fondo de pantalla para ajustarse a la luz de la habitación, en todas sus pantallas

ACTIVITIES:

MainActvity:

Activity principal, la cual sirve para conectarse al embebido. Posee tres botones:

- **Activar/Desactivar:** Sirve para activar o desactivar el Bluetooth.
- **Dispositivos Emparejados:** Sirve para conectarse al embebido una vez emparejado.
- **Buscar dispositivos:** Sirve para buscar el embebido y emparejar con él, para posteriormente conectar.

DeviceListActivity:

Activity que sirve para conectar con un dispositivo emparejado. Lee del sistema operativo cuantos otros dispositivos existen emparejados y por cada uno muestra dos botones:

- **Iniciar:** Prepara la conexión y abre la Activity para monitorear y operar el embebido.
- **Emparejar/Desemparejar:** Empareja o desempareja un dispositivo con el otro.

Funciones:

Activity central del programa, nos informa de la cantidad de rocklets en los vasos, por color, cuál fue el color del último Rocklet identificado y del modo actual del embebido (Manual o Automático). Además, permite usar el modo Celular con las funciones del mismo: Rotar el servo girando el teléfono y soltarlo en el vaso acercando la mano. La activity se encarga de iniciar la conexión con el embebido y dejar corriendo un Thread que se encargará de la comunicación (en una clase definida dentro de *Funciones*, llamada *ConnectedThread*).

Además tiene los siguientes botones:

- **Pausar/Reanudar:** Permite pausar o reanudar el funcionamiento del embebido. Esto envía un mensaje con el caracter ‘2’, el cual el embebido interpreta como pausa. Para quitar la pausa envía el caracter ‘3’.
- **Mover Tobogán:** Si el embebido no está en Pausa, envía un mensaje al embebido para activar el modo Celular (el mensaje es el caracter ‘0’). Tras cambiar de modo, el botón cambia su nombre a “Volver a automático”, y se pone el mensaje en pantalla de “Lanzar rocklet”. Al presionarlo nuevamente se vuelve a automático y se borran los mensajes que habían en pantalla.
- La activity escucha los mensajes que envía el embebido gracias al Thread con un Handler. Los mensajes que se pueden escuchar son los siguientes:
    - Reporte del último color junto con la cantidad de rocklets por vaso. Se usa la siguiente cadena: #Color_ultimo_rocklet-Cant1-Cant2-Cant3-Cant4-Cant5-Cant6\r\n
      Donde:
        - Color_ultimo_rocklet es un número del 0 al 5 que me indica el color, o 9 si se trata de un color no identificado, y se traduce con el método *codToStringColor* en *Tools.* Cuando recibe el color, el fondo de la zona de mensajes de la Activity cambia según el último color lanzado (en caso de detección errónea, se pone en Gris). Además, cambia el mensaje de la aplicación de “Espere a que el rocklet esté listo para lanzar” a “Lanzar rocklet”, y desbloquea el envío de mensajes de giro
        - CantN es un número que me indica cuantos Rocklets fueron identificados de ese color
        - #, - y \r\n son marcadores de control
    - Cambio de modo. Se usa la siguiente cadena: $Modo
      Donde:
        - Modo es un caracter ‘a’ o ‘m’ que indica el modo actual de operación, y se traduce con el método *codToStringModo* en *Tools*
        - $ es un marcador de control

Como añadido, si se está en modo tobogán, además envía mensajes cada vez que:

El dispositivo rote en eje z: Envía, de forma traducida para que lo pueda interpretar el embebido, la posición actual. Envía el caracter ‘5’ seguido del caracter que representa a la posición enviada en código ASCII. Utiliza el sensor de posición vectorial:un sensor por software que se basa en el Giroscopio, el

cual sí es un sensor por Hardware. La traducción la hace el método *gyroToServo256* de la clase *Tools*. No se envía hasta que sea necesario hacerlo.

Se acerque la palma al dispositivo: Envía un caracter '4' al embebido, que será interpretado como la orden "lanzar rocklet". La app cambia el mensaje principal de "Lanzar rocklet" a "Espere a que el rocklet esté listo para lanzar", y desactiva el envío de datos de giro.

Por último, al cerrar el modo, se dejan también de enviar datos de giro, y se envía el caracter '1', el cual el embebido interpreta como el fin del modo Celular.

CLASES AUXILIARES:

Tools:

Métodos varios, los cuales están agrupadas en esta clase de utilidades:

- **luxToGreyColor:** Transforma la cantidad de luz medida por el sensor de luz, que es recibida en Lux (medida internacional de iluminación, símbolo: lx), en un valor de color en escala de grises, el cual se utiliza para el color de fondo de la pantalla.
- **gyroToServo256:** Transforma los datos recibidos por el sensor de sentido rotacional (valores entre -1 y 1 radianes) a un rango de 0 a 255 dentro de un byte (Para enviarlo como tal al embebido)
- **codToStringColor:** Traduce un código de color en la String de ese color, devuelve la String "Error" si no puede traducir
- **codToStringMode:** Traduce un código de modo en la String de ese modo, devuelve la String "Error" si no puede traducir

SingletonColorPantalla:

Clase tipo Singleton que controla el color de la pantalla según el brillo que el dispositivo detecte.

La clase controla que se posea o no un sensor de luz en el dispositivo. Informa de esto al abrir la aplicación, y de no poseerlo, el fondo de pantalla de la aplicación se mantendrá blanco durante el resto de su ejecución

La clase solo tiene dos métodos públicos: Uno que sirve para indicarle la Activity sobre la que tiene que operar, y otro para informar que la app está en pausa. El primero se usa cada vez que se ejecuta *onResume()*, y el segundo cada vez que se ejecuta *onPause()*

La clase implementa SensorEventListener por lo que escucha el sensor de luz del dispositivo de forma paralela a la actividad en ejecución

- La clase cambia el color a cada vez más oscuro cuanto menos luz detecte (para proteger la vista) hasta llegar al negro cuando no se detecte luz. A su vez tiende al blanco conforme sube la cantidad de luz hasta los 128 Lux (Unidad de medida de luz), a partir de donde se muestra en blanco para que sea visible a pesar de la luz.

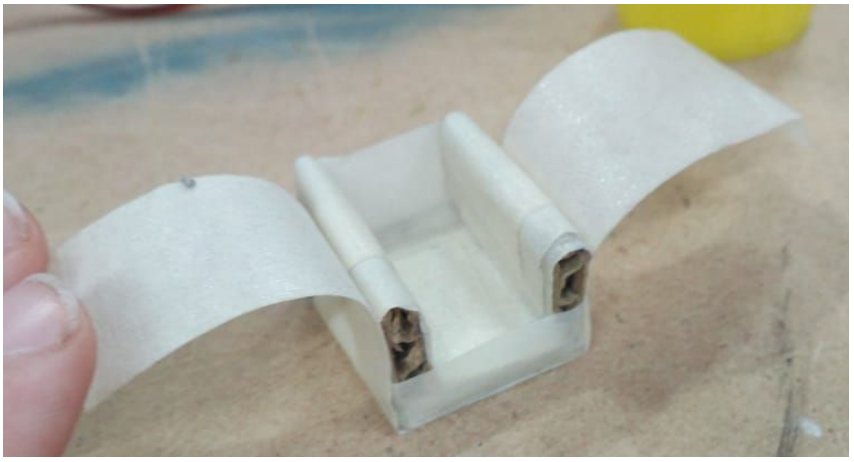
ConnectedThread:

Clase encargada de los datos de la conexión Bluetooth, es la que informa y recibe mensajes con el dispositivo.

PROBLEMAS Y SOLUCIONES

Servos

Comenzamos con dos servos prestados a la cátedra: un TowerPro SG90 Micro Servo (apodado “Servo Azul”) y un Small Servo del Arduino Starter Kit (apodado “Servo Negro”). Lo primero que notamos es que no respondían correctamente al código generado con la biblioteca servo.h, por lo que decidimos trabajarlos directamente con PWM. Testeamos los límites de anchos de pulso a los que respondían, y descubrimos que cada servo respondía en rangos diferentes, que a veces podían girar más de 180º y que en ciertos ángulos se trababan si se les pedía movimientos cortos, pero podían pasar si eran movimientos más largos.
Al principio íbamos a poner el servo negro en el tobogán, ya que era un trabajo más pesado y ese servo era más fuerte, pero resultó tener un comportamiento errático. Por eso los intercambiamos y dejamos el negro para hacer movimientos entre solo tres estaciones bien distanciadas, mientras que el azul con el tobogán debía moverse entre seis estaciones más cercanas.
Finalmente el servo negro se quemó y compramos otro del mismo modelo que el azul, pero lo seguimos llamando Servo Negro para mantener compatibilidad hacia atrás. El nuevo servo resultó funcionar en valores totalmente distintos de generación PWM, por lo que tuvimos que calibrar variando todos los valores de las estaciones. Este servo también se trababa ocasionalmente en lugares críticos y al hacer fuerza se separaban sus engranajes, logrando recalentar pero sin moverse a ningún lado.
Nos prestaron un cuarto servo del mismo modelo que los azules. Este servo parece funcionar correctamente hasta el día de la fecha. Implementamos un soporte más firme con forma de cómodo sillón, para que el servo no se desplace de su posición ideal.



Lecturas de color

Si bien el sensor de color tiene muchos fotodiodos con distintos filtros para detectar intensidades de rojo, verde, azul y luminosidad general, puede leer una sola de estas por vez. Esto significa que para detectar un color se requieren varias lecturas del sensor. Además cada una va a utilizar el mismo pin para enviar el valor sensado, por lo que se requiere un tiempo para liberar el canal antes de poder efectuar la siguiente.

Ya que utilizar delays es una mala práctica de programación de sistemas de tiempo real (desperdicia tiempo de operación, postergando también los chequeos de otros sensores), programamos una función booleana que sería llamada varias veces, a lo largo de varios loops, y en cada llamada se encargaría de efectuar una lectura o simplemente retornar porque aún no puede leer. Retorna falso cuando todavía está en proceso de reconocer un color, y verdadero cuando lo reconoció.

Pulsador

Creímos que iba a ser extremadamente sencillo implementar un pulsador: simplemente leer si está siendo presionado o si no lo está. Error fatal. Resultó que existía ruido por distintas situaciones (como una presión suave) que generaba múltiples falsos positivos. Otro problema fueron los rebotes: Cuando se pulsa el botón, a veces vuelve arriba, choca con el metal y rebota tocando de nuevo el contacto, generando falsos positivos… Para solucionar estos problemas, además de simplificar la interfaz, decidimos implementar una clase la cual toma como “pulsado” después de pasado cierto tiempo en el que el botón no se suelte, y así ignore todos los falsos positivos (ya que es muy improbable que por error se quede en contacto tanto tiempo). La implementación requirió que el pulsador se chequee en todos los loops. La detección de un pulso largo o corto se puede hacer una sola vez por pulso, por lo que fue necesario guardar el último estado detectado para poder preguntar varias veces dentro de un mismo loop sin modificarlo.

Barrera laser

Al tener preparada la parte de la estructura que recibe y despacha los rocklets notamos la complicación que presentaba colocar el láser y el detector de flama necesarios para crear la barrera en el reducido espacio entre los pisos, tuvimos que desmontar el piso donde iría colocada y tallar huecos para que quedaran fijos estos componentes y el láser atravesara el tubo en el limitado espacio disponible.

Otra complicación consistió en el alineamiento del láser y el detector de flama ya que consiste en un punto muy preciso donde el sensor si detecta el láser, sin dejar margen de movimiento ni de 1mm. de ninguno de los dos componentes. Nuevamente la cinta fue nuestra solución, una vez encontrada el punto exacto donde se alineaban los componentes los inmovilizamos en esa posición con múltiples capas de cinta, además de aumentar la sensibilidad del detector de flama.

Bibliografía

Especificaciones de Arduino Nano: https://store.arduino.cc/usa/arduino-nano
Especificaciones y configuración del módulo Bluetooth HC-05: http://www.martyncurrey.com/hc-05-zg-b23090w-bluetooth-2-0-edr-modules/
Comandos para configurar el móduglo Bluetooth HC-05: https://gitlab.com/so-unlam/Material-SOA/blob/master/Ejemplos/Arduino-AndroidBluethoot/Arduino-Code/Configuracion_HC-05/Configuracion_HC-05.ino
Uso de los sensores en Android: https://developer.android.com/guide/topics/sensors/sensors_overview
Especificaciones laser: https://www.planetaelectronico.com/modulo-arduino-diodo-laser-650nm-p-16148.html