Laboratorio RPM

Laura Camila Blanco Gómez
Escuela de Ingeniería
Universidad Sergio Arboleda
Bogotá, Colombia
laura.blanco01@usa.edu.co

Santiago Cáceres Linares
Escuela de Ingeniería
Universidad Sergio Arboleda
Bogotá, Colombia
santiago.caceres01@usa.edu.co

I. Introducción

El laboratorio se enfoca en el desarrollo de un sistema de control de RPM mediante la modulación por ancho de pulso (PWM). La implementación de esta técnica permitirá ajustar la velocidad de un motor, lo cual es esencial para el posicionamiento preciso de un dispositivo mecánico, en este caso, una rueda.

Se busca realizar movimientos precisos y controlados en función de la información proporcionada por sensores. Dichos sensores están diseñados para detectar cambios en la posición de la rueda, generando así señales que son procesadas por un controlador PID (Proporcional, Integral y Derivativo). Este controlador desempeña un papel crucial al calcular la corrección necesaria para mantener la posición deseada de la rueda, logrando así un control preciso y eficiente del sistema.

El movimiento de la rueda puede ser en dirección izquierda o derecha, y la cantidad de pasos a mover se determinará en función de la retroalimentación de los sensores y la lógica de control implementada. Es importante destacar que, en caso de que la posición deseada no se haya actualizado, se espera que la rueda regrese a su posición original.

II. Marco teórico

PWM: La modulación por ancho o de pulso (en inglés pulse width modulation PWM) es un tipo de señal de voltaje utilizada para enviar información o para modificar la cantidad de energía que se envía a una carga.

Esta acción tiene en cuenta la modificación del proceso de trabajo de una señal de tipo periódico.

Puede tener varios objetivos, como tener el control de la energía que se proporciona a una carga o llevar a cabo la transmisión de datos.

Si el ciclo de trabajo es del 25% se pasa el 25% de su periodo arriba y el 75% abajo. El periodo es lo que dura la onda sin repetirse. Por eso se va repitiendo con el tiempo porque el periodo se repite durante todo el tiempo. El periodo es la suma de la parte alta y baja una vez, cuando vuelve a subir ya es otro periodo y la onda vuelve a empezar otra vez.

Control PID: El controlador PID, una herramienta clave en el control de sistemas en lazo cerrado, emplea acciones Proporcional, Integral y Derivativa para lograr un comportamiento deseado. La acción Proporcional busca reducir el error actual, aumentando la velocidad de respuesta y disminuyendo el error permanente. Sin embargo, un incremento excesivo puede provocar inestabilidad en el sistema. La acción Derivativa contrarresta oscilaciones y sobrepulsos al considerar la velocidad del cambio del error. Ajustar la constante derivativa mejora la estabilidad, pero puede afectar la velocidad de respuesta.

Por otro lado, la acción Integral busca reducir el error permanente acumulando la integral del error a lo largo del tiempo. Aumentar la constante integral disminuye el error permanente, pero puede introducir inestabilidades adicionales. La sintonización manual del PID implica ajustar estas constantes incrementalmente para lograr la respuesta deseada sin comprometer la estabilidad.

En la práctica, la ecuación del controlador PID combina estas acciones para determinar la señal de control. No obstante, en sistemas reales, existen limitaciones físicas, como la saturación del actuador, que impactan la capacidad del controlador para alcanzar respuestas ideales. Considerar estos límites es esencial en el diseño y la sintonización de un controlador PID para aplicaciones prácticas, garantizando un rendimiento óptimo sin comprometer la estabilidad del sistema.

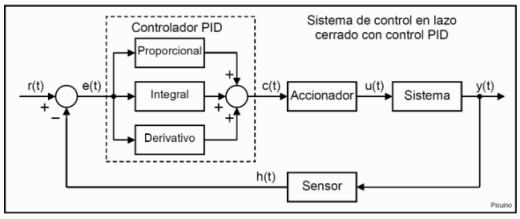

Figure 1: PID

RPM(revolución por minuto): Las revoluciones por minuto son una medida de frecuencia. Nos muestra la cantidad de vueltas completadas en un tiempo determinado en la máquina. Entre más revoluciones por minuto tenga un motor, podrá trabajar a más velocidad aprovechando al máximo la potencia del motor.

Sensores de herradura: Es un dispositivo que responde al cambio en la intensidad de la luz. Se componen de un emisor que genera la luz y un receptor que percibe la luz generada. Se suelen usar para detectar, clasificar y posicionar objetos en condiciones ambientales extremas.

III. Procedimiento

Para este apartado se divide en cuatro partes, las cuales son.

- **Montaje** Para este apartado del procedimiento se hizo la conexión entre la STM, los sensores de herradura y el programa en QT como se ve en la figura 2, adicionalmente se conecta un cable USB - C para la trasmisión de datos por el puerto serial.

- **Protocolo** para este apartado se definió un protocolo el cual es enviado por medio del puerto serial del computador en donde se ejecuta el código generado en el QT, el protocolo fue formado de la siguiente manera.
  - head
  - tamaño
  - data
  - runStop
  - byte de validación
  - end

  Algunos de estos bytes de este protocolo fueron quemados debido a que estos nunca cambian, estos bytes fueron head y end porque cada uno indica el inicio y el fin de la transmisión del protocolo respectivamente, otro dato que no tiende a cambiar su valor a la hora hacer el envío de datos es el tamaño debido a que este se compone de la suma del tamaño de la data y el **overhead**, este último tiene un valor de 5 debido a que quitando los bytes de la data los cuales en este caso fueron 4. A pesar de que la el tamaño de la data para el protocolo de QT y la STM son lo mismo, se cambia el valor a enviar como se ve a continuación.
  - 4 bytes para la posición deseada enviada desde QT.
  - 4 bytes para la posición actual enviada desde la STM.

  Los otros componentes del protocolo ocupan 1 byte y al sumar estos nos dan 5 el cual es el valor del overhead finalmente se suman el tamaño de la data y el overhead y así nos da un total de 13 por lo cual es valor para este caso siempre será 13. Por otro lado, los bytes runStop y byte de validación tiene el siguiente propósito en el protocolo, el byte runStop sirve para saber si se desea encender el motor o apagar desde el QT esto lo logra por medio de los comandos (0x11 y 0x12) los cuales son para encender y apagar el motor respectivamente. Finalmente, el byte de verificación nos sirve para saber si el paquete llego de manera correcta, esto lo logra por medio de una operación xor desde la el byte head hasta el byte runStop, en caso de que el protocolo llegue bien se retorna un ACK en caso contrario se retorna un NACK.

- **QT** Este apartado se divide en dos partes, las cuales son:
  - Interfaz gráfica: Este apartado consta de unos labels los cuales nos sirven para mostrar los siguientes datos.
    - Posición actual.
    - Posición deseada.

    Además de estos labels se tiene un botón el cual se encarga de realizar él envidó de la posición deseada. Finalmente, la interfaz final se ve de la siguiente manera

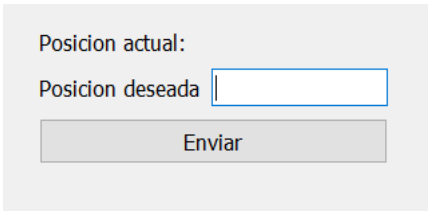

Figure 2: Interfaz QT

  - Código: El código consta de 3 funciones principales les cuáles son
    - readSerial()

      Esta función se encarga de recibir los datos que llegan por medio del puerto serial, esto lo hace por medio de una variable global de tipo **qByteArray** la cual funciona como un buffer de datos que llegan a QT, la función principal de esta función es validar que el protocolo haya sido enviado de manera correcta esto lo hace verificando el valor de los bytes de inicio y fin, si estos valores son correctos entonces se realiza el cálculo para el valor del byte de verificación si el resultado de esta operación es igual al valor que se encuentra en la penúltima posición del protocolo se procede a la toma de datos que viene en el protocolo.

```
1  if(serialData.at(0) == 0x02 && ...
       serialData.at(serialData.at(1)-1)==0x3A){
2          qDebug()<<"Paquete con ...
               cabeza y cola iguales";
3          // calculate(serialData);
4          if(verify_xor(serialData,
5          serialData.length(),
6          serialData.at(serialData.length()-2)) ...
               == ACK) {
7              qDebug()<<"xor igual";
8              calculate(buffer);
9              serialData.clear();
10         }
11     }
12 }
```

    - create_protocol(uint8_t operation)

      Esta función se encarga de crear el protocolo que enviara la posición deseada para el encoder, a diferencia de otros laboratorios donde la data que poseía el protocolo siempre era un byte teniendo este byte el valor de 1, para este caso porque no se necesita enviar un comando la data enviada como se dijo anteriormente será la posición deseada teniendo un tamaño de 4 bytes como se ve en el siguiente código.

```
        uint8_t total_len = ...
            overhead + 1;
    protocol.append(0x02);
    protocol.append(total_len);
    protocol.append(0x01);
    protocol.append(operation); // ...
        OPERATION
    protocol.append(
    xor_operation(protocol,total_len));
    protocol.append(0x3A);
```

- **Código STM** Para el apartado de la tarjeta, la cual va a servir como el dispositivo, el cual se va a encargar de recibir la posición deseada y enviar la posición actual del encoder, para que esto se pueda realizar, se debe configurar el **Milddleware** de la tarjeta como **USB_DEVICE**, adicionalmente en el apartado de conectividad de la tarjeta se le debe escoger la opción **device_only** una vez se hacen esta configuración se debe configurar el reloj el cual tendrá como frecuencia máxima 96MHz esto debido a la conexión USB.

  Una vez finalizadas estas configuraciones previas se debe realizar la configuración de los tims que para este laboratorio se usaron el **TIM2**, **TIM4** y **TIM5**. todos estos tims servirán como interrupciones y se usarán para las siguientes funciones.

  – TIM2: Se encarga de enviar el protocolo.
  – TIM4: Se encargará de dar la señal PWM para el motor reductor.
  – TIM5: Se encargará de llamar a la función que realiza el control PID.

  las configuraciones realizadas para los tims fueron las siguientes.

  – **TIM2**
    * Preescaler: 960 - 1
    * CP: 2000 - 1
  – **TIM4**
    * Preescaler: 960 - 1
    * CP: 200 - 1
  – **TIM5**
    * Preescaler: 100 - 1
    * CP: 960 - 1

  Después de realizar estas configuraciones se procede a generar las funciones para el código, las cuales son:

  – readSensor(): Esta función se encarga de actualizar la posición a partir de los estados que se pueden tener con el encoder y los dos sensores de herradura, estos estados son

  $$\begin{matrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{matrix}$$

  con estos estados se puede saber si el motor está girando en sentido del reloj o en contra reloj, las reglas que sigue este esta función es que solo se pueda pasar del estado actual a un estado anterior o al estado siguiente, esto quiere decir que si se está en el estado 1 solo se puede pasar al estado 2 o al estado 4, no se podría pasar directamente al estado 3, esta regla se aplica para cualquier estado. esto se puede ver en el siguiente código.

```
    if (HAL_GPIO_ReadPin(GPIOB, ...
        GPIO_PIN_1) == ...
        GPIO_PIN_RESET
        && ...
            HAL_GPIO_ReadPin(GPIOB, ...
            GPIO_PIN_2) == ...
            GPIO_PIN_RESET) {

        if (estadoAntiguo == 4) {
            posicion++;
        }
        if (estadoAntiguo == 2) {
            posicion--;
        }
        estadoAntiguo = 1;
    } else if ...
        (HAL_GPIO_ReadPin(GPIOB, ...
        GPIO_PIN_1) == ...
        GPIO_PIN_RESET
        && ...
            HAL_GPIO_ReadPin(GPIOB, ...
            GPIO_PIN_2) == ...
            GPIO_PIN_SET) {
        if (estadoAntiguo == 1) {
            posicion++;
        }
        if (estadoAntiguo == 3) {
            posicion--;
        }
        estadoAntiguo = 2;
    } else if ...
        (HAL_GPIO_ReadPin(GPIOB, ...
        GPIO_PIN_1) == GPIO_PIN_SET
        && ...
            HAL_GPIO_ReadPin(GPIOB, ...
            GPIO_PIN_2) == ...
            GPIO_PIN_SET) {
        if (estadoAntiguo == 2) {
            posicion++;
        }
        if (estadoAntiguo == 4) {
            posicion--;
        }
        estadoAntiguo = 3;
    } else if ...
        (HAL_GPIO_ReadPin(GPIOB, ...
        GPIO_PIN_1) == GPIO_PIN_SET
        && ...
            HAL_GPIO_ReadPin(GPIOB, ...
            GPIO_PIN_2) == ...
            GPIO_PIN_RESET) {
        if (estadoAntiguo == 3) {
            posicion++;
        }
        if (estadoAntiguo == 1) {
            posicion--;
        }
        estadoAntiguo = 4;
    }
```

  – PID(): Esta función se encarga de calcular el valor de PWM a partir de la **ecuación diferencial de un PID discreto**, este control que da el valor a

PWM se logra por medido de sus tres constantes **p, d, i** y los errores. como se observa en el código para calcular la fórmula, esta se divide en 3 para finalmente sumar estos 3 cálculos por separada y así obtener el valor de PWM el cual si es mayor a 200 el PWM irá al máximo de su capacidad porque gracias a su configuración este es su valor máximo, cuando se obtiene el valor de la fórmula de ser este negativo se pasara a positivo, ya que el registro **CCR** no admite valores negativos. Finalmente, para obtener los valores de las constantes estas se obtiene por medio de prueba y error

```
error = posicion - ...
    posicionDeseada;

PWMvalue = PWMvalueAnterior ...
    + ((p + (d / t)) * error);
PWMvalue2 = ((-p + (i * t) - ...
    (2 * (d / t))) * errorP);
PWMvalue3 = ((d / t) * errorPP);
PWMvalueTotal = PWMvalue + ...
    PWMvalue2 + PWMvalue3;

if (PWMvalueTotal > 0) {
    HAL_GPIO_WritePin(GPIOA, ...
        GPIO_PIN_4, ...
        GPIO_PIN_SET);
    HAL_GPIO_WritePin(GPIOA, ...
        GPIO_PIN_5, ...
        GPIO_PIN_RESET);
    if (PWMvalueTotal > 200) {
        PWMmotor = 200;
    }else{
        PWMmotor = ...
            PWMvalueTotal;
    }
} else if (PWMvalueTotal < ...
    0) {
    HAL_GPIO_WritePin(GPIOA, ...
        GPIO_PIN_4, ...
        GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, ...
        GPIO_PIN_5, ...
        GPIO_PIN_SET);

    if ((PWMvalueTotal * -1) ...
        > 200) {
        PWMmotor = 200;
    }else{
        PWMmotor = ( ...
            PWMvalueTotal * -1);
    }
} else {
    HAL_GPIO_WritePin(GPIOA, ...
        GPIO_PIN_4, ...
        GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOA, ...
        GPIO_PIN_5, ...
        GPIO_PIN_RESET);
    PWMmotor = 0;
}

PWMvalueAnterior = ...
    PWMvalueTotal;
errorPP = errorP;
errorP = error;
TIM4->CCR1 = PWMmotor;
```

– createBuffer(): Esta función se encarga de tomar la posición actual para luego enviar este dato en un paquete de 4 bytes, posterior a esto se hace el llamado de la función **createPakage()**, lo anteriormente mencionado se ve en el siguiente código.

```
uint8_t packageData[size];
uint32_t data = ...
    *(uint32_t*) &posicion;

packageData[0] = data & ...
    0x000000FF;
packageData[1] = (data & ...
    0x0000FF00) >> 8;
packageData[2] = (data & ...
    0x00FF0000) >> 16;
packageData[3] = (data & ...
    0xFF000000) >> 24;

createPakage(packageData, ...
    size);
```

– createPakage(): Esta función se encarga de la construcción del protocolo de comunicación y el envío de este último por medio del puerto serial hacia QT, esto lo logra tomando los datos de la función **createBuffer()** y anexando estos datos desde la tercera posición del protocolo porque los dos primeros bytes están ocupados por el inicio y el tamaño del protocolo, una vez es agreda la información del protocolo se agregan los bytes de runStop, Byte de verificación y final del protocolo, para dar valor al byte de verificación se hace por medio de la función **valueXorValidation()** el cual realizara un cálculo de xor desde el inicio hasta el byte runStop, una vez el protocolo es creado se llama la función **CDC_Transmit_FS()** la cual es la función que envía los datos a QT, esto se pueden ver en el siguiente código.

```
uint32_t totaLen = len + ...
    overHead;
uint8_t *dataEnvida =
malloc(sizeof(uint8_t)*totaLen);
dataEnvida[0] = 0x02;
dataEnvida[1] = totaLen;
for (int i = 0; i < len; ...
    i++) {
    dataEnvida[i + 2] = ...
        buffer[i];
}
dataEnvida[totaLen - 3] ...
    = RunStop;
dataEnvida[totaLen - 2] =
valueXorValidation(dataEnvida, ...
    totaLen - 2);
dataEnvida[totaLen - 1] ...
    = 0x3A;
CDC_Transmit_FS(dataEnvida, ...
    totaLen);
```

IV. Resultados y análisis de resultados

una vez realizado varias pruebas en el sistema con distintos valores de en la posición deseada enviada desde qt se llegaron a los siguientes resultados

- **QT**
  Gracias a que el envío de datos se realiza cada 20 ms desde la stm al empezar a llegar los datos, el label que muestra la posición actual no se pueden percibir con mucha claridad a valores pequeños, esto se da debida a que el valor de la posición cambia bastante rápido por lo cual en un inicio no se pueden percibir estos valores de una manera agradable.

- **Protocolo**
  Con respecto al protocolo se decidió tomar la decisión que se quemen las posiciones donde va la data debido a que si se usaba por medio de un ciclo como el for generaba que los paquetes se pagaran uno tras otro, por lo cual al momento de llegar a la stm el valor del byte de verificación nunca daba, por lo cual la stm siempre devolvía un **NACK**.

  Por otra parte, al momento de realizar las primeras pruebas con el envío de datos desde QT a la STM se presentó un error en el cual siempre hacía que el valor de llegada del byte de validación no considera con el calculado que se realizaba en la STM, esto se presentaba porque no se enviaba bien el tamaño del protocolo.

  Siguiendo con el punto anterior, otra medida que se tomó para qué la llegada de los datos no se viera afectada por el envío tan rápido que se hacía de la información fue crear un arreglo global en el cual se copiaban los datos para posteriormente hacer tomar los datos de este array.

- **Código STM**
  como se mencionó anteriormente se tuvo que separar la fórmula del cálculo de PWM, se tuvo que dividir en 3 partes, ya que cuando se generaba la fórmula en una solo variable se presentaban errores en el cálculo. Con respecto al valor que se toma desde el GPIO este es inversamente proporcional a la velocidad a la que vaya el motor, esto se da debido a que a mayor velocidad menor es el tiempo en el que se da una vuelta completa y así mismo a menos velocidad mayor es el tiempo en que se da una vuelta entera por parte del disco

- **¿Para qué se utiliza el controlador de la manera planteada y no solamente prender el motor, contar los pulsos?** se usa el control PID, ya que con este se controla de una mejor manera la velocidad que necesita el motor para llegar a la posición deseada, ya que entre más lejos se esté de la posición más rápido irá el motor y entre más cerca se esté de la posición más lenta irá el motor. esto no se podría lograr si se hace solo con el conteo de pulso porque no existiría una manera de hacer la disminución de la velocidad cuando se esté cerca de la posición deseada.

- **Explicar el procedimiento que se utilizó para determinar el óptimo tiempo de muestreo.** El procedimiento que se usó para poder determinar el tiempo óptimo de muestreo fue utilizar el tiempo mínimo que se necesita para poder realizar el cálculo del PMW.

- **Explique el procedimiento que realizó para determinar el valor de la constante P** Para poder obtener el valor de la constante P se empezó probando poniendo esta constante con un valor de 1 y así se fue aumentando hasta que se puede mover el motor, siendo este el valor mínimo de P para este caso es de 30, finalmente, debido a que el comportamiento de P multiplica el error se trató de dejar una P que no fuera muy grande para que el control fuera más preciso y el error no creciera demasiado.

- **Explique el procedimiento que realizó para determinar el cálculo de la ecuación de error** El error se puede calcular por medio de la grafica de respuesta del motor, con esta se puede comparar la respuesta brindada una vez implementados el pid y así ver los puntos en los que se desfasa la grafica.

V. Conclusiones

- Si se usan ciclos al momento de crear la data, el protocolo desde QT se unen los paquetes, lo cual genera errores a la hora de validar si el protocolo llega de manera adecuada.

- Al momento de hacer la estructura se deben tener en cuenta las medidas exactas y los materiales usados, al no quedar el motor estable empieza a chocar con el sensor lo que causa detenimiento en el proceso o una mala lectura en los datos, en el caso de los materiales se pudo ver que al crear la pieza en plástico por culpa de la fricción se derrite un poco, pero esto genera que no quede justo el disco por lo que cuando se aumenta el voltaje se sale el disco.

- Cuando se necesita que se ejecute una función cuando se ejecuta una interrupción es mejor tener una variable que habilite la llamada en el main que

llamar a la función en la interrupción porque este proceso son demorados.

- El PID a pocos cambios en su valor generan grandes cambios en el movimiento, calcularlo a punta de prueba y error es complicado por lo que se puede calcular a partir de la gráfica de tiempo de respuesta del motor.

- Se debe tener en cuenta los valores del PWM, estos deberán ser generados a partir de los resultados de la formula de PID para llevar el control correcto del funcionamiento del motor.

REFERENCES

[1] Motorysa. Qué son las revoluciones por minuto en un motor. [Online]. Available: https://mitsubishi-motors.com.co/blog/revoluciones-por-minuto-que-son/

[2] MovilTronics. Sensor de herradura klh512. [Online]. Available: https://moviltronics.com/tienda/sensor-klh512/

[3] shoptronica. Que es pwm y como funciona. [Online]. Available: https://www.shoptronica.com/curiosidades-tutoriales-y-gadgets/4517-que-es-pwm-y-como-funciona-0689593953254.html#:~:text=La%20modulaci%C3%B3n%20por%20ancho%20o,una%20se%C3%B1al%20de%20tipo%20peri%C3%B3dico.

[4] Picuino. Controlador pid. [Online]. Available: https://www.picuino.com/es/control-pid.html

VI. ANEXOS

1

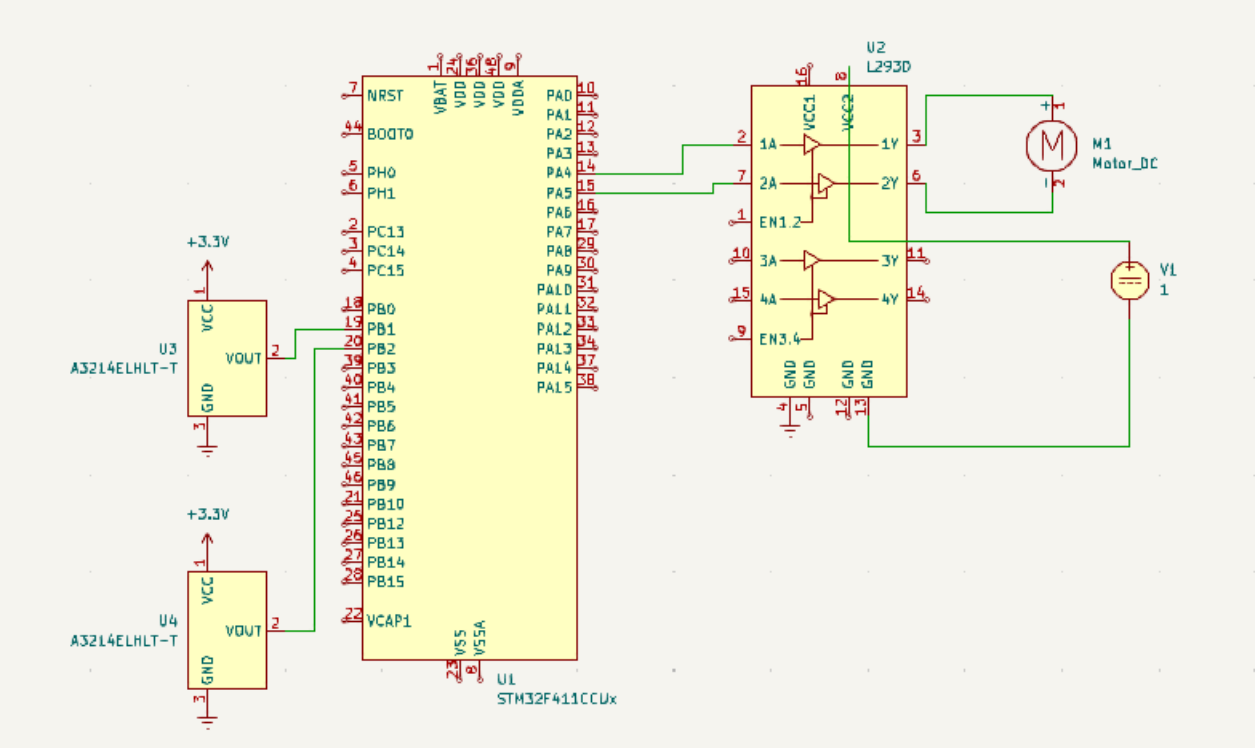

Figure 3: Esquematico