# Control del robot *Omnidireccional* como parte de la implementación realizada

Martin carballo,[1*]

[1]Maestría en ingeniería Electrónica y computación, Universidad veracruzana,

[*]Who correspondence should be addressed; E-mail: martincarballo93@gmail.com.

**En éste documento le describo la labor que realicé con su robot omnidireccional, partiendo con la descripcción del funcionamiento de los elementos del prototipo, para después hablar sobre cómo se relaciona la ejecución de los movimientos con los resultados de la programación hecha por Angel con `RGPPMcore.py` y finalizo describiendo el funcionamiento de la programación que hice para el arduino mega en `omniMegaBlue.ino`, que adjunto en el correo que le envío. También incluyo pensamientos personales sobre las problemáticas que encontré y así usted pueda ver el porqué de las decisiones que tomé.**

## mechatronic system elements

El prototipo consta de una tarjeta Raspberry pi 3 B+ con la cual se realiza el calculo de la ruta y asu vez obtiene la orden de movimientos que se envía al Arduino Mega. El Arduino Mega tiene la función de controlar los motores me diante una señal $PWM$ de 490 Hz, con base a la instrucción recibida de la Raspberry.

Se usan puentes H $L298N$ para poder controlar los motores con la señal $PWM$ de salida del Arduino Mega. Los motores son de DC con motorreductor y encoders Polu de 18 Kg-cm.

Y por ultimo se tienen las rueda Mecanum de 100mm, gracias a las cuales se tiene la capacidad omnidireccional.

# Path following

La salida de la programación en Python 3 es una serie de direcciones en forma de caracteres, cada uno de los cuales representan una de las direcciones mencionadas, las cuales son: $Adelante$, $Atras$, $Derecha$, $Izquierda$. Dichas direcciones se deben seguir en orden por cierta distancia, la cual es igual a la distancia entre cada nodo en el ambiente real. Creo que de ésta manera se esta manejando a la salida de la programación de RGPP como un comando de tipo $Accion$, el cual es interpretado directamente por el Arduino Mega y con ello se lleva al Arduino Mega a trabajar a un nivel más alto de planificación que el mero control, lo cual no ocurriría si hubiese programado que desde la Raspberry se enviara una dirección a la vez y el Arduino se encargara de controlar al robot durante el tiempo necesario para la dirección requerida. Lo cual no hice para evitar problemas de sincronización que no eran necesarios ya que no estaba revisando obstáculos en tiempo real.

Para que el robot se pueda mover hacia adelante simplemente se mueven las cuatro ruedas hacia adelante y en sentido contrario para cuando se necesite mover hacia atrás. En la tabla 1 le muestro las combinaciones del sentido de las ruedas para lograr el resto de direcciones. Abrevie las posiciones de las llantas, ejemplo: left front wheel (LFW).

El tiempo que se mantiene al robot en una dirección depende de la distancia que se haya establecido entre cada nodo y, por lo tanto, también de la velocidad que se haya establecido. La velocidad a la que se probó fue 10 cm/seg para las direcciones de modelado cuadrado y 7 cm/seg en diagonal, aunque ésta última no se grabó. La problemática que tuve para aumentar la velocidad fue que no use interruciones para revisar los encoders, si no que reviso los encoders cáda vez que se ejecuta el programa; había decidido no trabajar con interrupciones ya que me

| Direction | LFW | RFW | LRW | RRW |
|---|---|---|---|---|
| forward left diagonal | off | forward | forward | off |
| forward right diagonal | forward | off | off | forward |
| Backward left diagonal | backward | off | off | backward |
| Backward right diagonal | off | backward | backward | off |
| Right | forward | backward | backward | forward |
| Left | backward | forward | forward | backward |

Table 1: No estaba seguro como llamar las diagonales.

estába dando lecturas incluso menores que las que obtenia de la forma en que lo hice al final y aun desconozco el porqué, sabiendo que con interrupciones debería ser más rápido que con la frecuencia de ejecución, ya que cuando una interrupción se activa se ejecuta la función de interrupcuón independientemente de qué parte del programa se esté ejecutando; entonces, ya que dependo de la frecuencia de ejecución, tengo un efecto de aliasing al aumentar la velocidad deseada.

# Control

**Descripción del programa.** Como mencioné antes, se tiene una cadena de caracteres para llevar acabo la *Acción* de moverse hacia el destino. Entonces durante el trayecto de un nodo a otro se ejecuta el control PID para cada una de las senãles PWM de cada uno de los motores, es decir que el control de un motor es independiente del resto de motores. Por lo tanto se cuenta con arreglos de cuatro elementos para las señales PWM, los valores de error de cada motor, las derivadas de cada error, las integrales de cada error y arreglos para las constantes del control PID.

El algoritmo 1 es el usado para hacer el seguimiento de la ruta, las letras `F, B, R y L` hacen referencia a la dirección. Para construir el algoritmo quise usar $case$ pero no pude encontrar una manera de ponerlo en LaTeX . `encoders` es el arreglo para los conteos de

los contadores, `ErrorIntegral` es el arreglo para almacenar las integrales de cada error con respecto al tiempo y `lastError` es el arreglo del ultimo valor de error registrado, éste se usa para comparar un error más reciente y usar esa diferencia para obtener `rateError`, que és la derivada del error con respecto al tiempo de muestreo, el cual se manejó de 25 milisegundos. Al final del algoritmo se observa que dichos registros se ponen en cero, para evitar errores al ejecutar el control PID la siguiente ocasión que se requiera, esto se hace siempre y se vera en otras partes del código.

**Algorithm 1** Algoritmo para el seguimiento de la ruta.

**Require:** $Directions \leftarrow [char_0, char_1, ..., char_{n-1}]$
  $i \leftarrow 0$
  **for** $i < Directions.length$ **do**
    **if** $Directions[i] == F$ **then**
      $control(byteHigh, byteLow, distance, Directions[i], i)$
    **else if** $Directions[i] == B$ **then**
      $control(byteHigh, byteLow, distance, Directions[i], i)$
    **else if** $Directions[i] == R$ **then**
      $control(byteHigh, byteLow, distance, Directions[i], i)$
    **else if** $Directions[i] == L$ **then**
      $control(byteHigh, byteLow, distance, Directions[i], i)$
    **end if**
    $i \leftarrow i + 1$
  **end for**
  **for** $i < 4$ **do**
    $encoders[i] = 0$
    $ErrorIntegral[i] = 0$
    $lastError[i] = 0$
    $i \leftarrow i + 1$
  **end for**
  $turnOffmotors()$

En la función `control()` se recibe `byteHigh, byteLow`, que son bytes usados para configurar los registros de pines y con ello configurar las direcciones de los motores que llevan a la dirección deseada del robot. `distance` es la distancia entre los nodos, enviada en forma de entero que mantendrá al control PID ejecutandose hasta que un contador `R` sea igual a

`distance`. Lo hice así en vez que poner un retardo, por que como estoy revisando encoders sin interrupción quise evitar perder cambios de estado que ocurrieran mientras el microcontrolador estubiera en medio de un retardo.

En el algoritmo 2 se muestra la forma en que se lleva a cabo el control, primero se toma el tiempo actual y se guarda como `t(0)` para posterior calculo de la derivada del tiempo en la funcion `pid()` y lo mismo se hace con los contadres de los encoders, `encoders[]`. A continuación se lleva a cabo el control en el `while` que se mensionó anteriormente, dentro del cual se configuran los puertos con `byteHigh` y `byteLow`: `byteHigh` contiene los bits que deben ponerse en alto y `byteLow` los que deben ponerse en bajo o apagado y así manipular el sentido de la corriente en los puentes $H$; se procede a enviar las señales $PWM$ a los puertos de salida, despues de ésto se revisa si hubo algún cámbio de estado de los encoders; y se ejecuta la función `pid()`, cuyo argumento depende de qué dirección se este trabajando, los `True` le indican qué motores estan habilitados o deshabilitados para el caso de los `False`. El primer `if` es el caso que usamos cuando hicimos las pruebas en Puebla, los otros `else if` son para cuando se necesita mover en diagonal. Al final se tiene un `if` que comprueba si la siguiente dirección que se trabajará sera diferente a la actual y de ser así, entonces se reinician los registros que se usan en el cálculo de la función `pid()`, se detienen lo smotores y se espera detiene el al robot por aproximadamente 500 msec para evitar derrapes que dificultarían el control y generan corrientes por inducción que podrían dañar los circuitos.

Por último describo el algoritmo de la función `pid()`, el control PID que programé utiliza un arreglo `desired[]` que contiene los valores de contadores deseados por un tiempo `dt`. Al inicio `ErrorIntegral` es igual a `0`, así como el arreglo `pwms[]` y para el caso del error y su derivada se recalculan cada ciclo del `while` de la función `control`. El algoritmo 3 recibe la indicación de a qué motores se les aplicará el control, se toma el tiempo actual como `t(f)` y se calcula `dt`; entonces, si la derivada del tiempo es 25 msec, se calcula el PID para los motores

**Algorithm 2** Control.

**Require:** $byteHigh = Byte, byteLow = Byte, distance, direction, intex = int$

```
t(0) ← time()
R ← 0
i ← 0
for i < 4 do
    encoder0[i] ← encoder[i]
    i ← i + 1
end for
while R < distance do
    portConfig(byteHigh, byteLow)
    sendPWM(pwms[pw[0], pw[1]..., pw[n − 1]])
    checkEncoders()
    if (direction == F)OR(direction == B)OR(direction == R)OR(direction == L)
    then
        pid(True, True, True, True)
    else if (direction == FLD)OR(direction == BRD) then
        pid(True, False, False, True)
    else if (direction == FRD)OR(direction == BLD) then
        pid(False, True, True, False)
    end if
    R ← R + 1
end while
if directions[indx + 1] ≠ direction then
    i ← 0
    R ← 0
    turnOffmotors()
    for i < 4 do
        encoders[i] ← 0
        ErrorIntegral[i] ← 0
        lastError[i] ← 0
        i ← i + 1
    end for
    while R < 500 do
        delay(1msec)
        R ← R + 1
    end while
end if
```

que estén habilitados. Si el control para el motor $i$ está habilitado se toma la cantidad de conteos hechos desde la ultima vez que se calculo el PID, se compara con `desired[i]` y la diferencia será `error[i]`; se calculan `rateError[i]`, `ErrorIntegral[i]` y la senñal PWM `i` en base a la expresión del PID de la ecuación 1.

**Algorithm 3** PID.

**Require:** $enabPID \leftarrow [enable[0], enable[1], enable[2], enable[3]]$
  $t(f) \leftarrow time()$
  $dt \leftarrow (tf - t(0))/1000$
  **if** $dt == 0.025$ **then**
    $i \leftarrow 0$
    **for** $i < 4$ **do**
      **if** $enabPID[i] == True$ **then**
        $encoderf[i] \leftarrow encoder[i]$
        $counters[i] \leftarrow encoderf[i] - encoder0[i]$
        $error = desired[i] - counters[i]$
        $ErrorIntegral[i] \leftarrow ErrorIntegral[i] + ErrorIntegral[i] * dt$
        $rateError \leftarrow (error[i] - lastError[i])/dt$
        $pwms[i] \leftarrow error[i] * Kp[i] + ErrorIntegral[i] * ki[i] + kd[i] * rateError[i]$
        $lastError[i] \leftarrow error[i]$
        $encoder0[i] \leftarrow encoterf[i]$
        $i \leftarrow i + 1$
      **end if**
    **end for**
    $t(0) \leftarrow tf$
  **end if**

$$PID = k_p * e(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{1}{dt} de(t) \tag{1}$$

# References and Notes


1. RONALD SIEGWART and ILLAH R. NOURBAKSHSH, *Introducion to Autonomous Mobile Robots*, The MIT Press, 2004.

2. La única referencia que usé fue la que uso como guia básica para ordenar mis ideas sobre

los enfoques en robótica, además de usar la página de productos Polu y la hoja de datos del microcontrolador ATmega2560.