PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Departamento de Ingeniería Mecatrónica

1MTR53 Robótica e Inteligencia Artificial
2024-1

Control de la Plataforma Omnidireccional

1 Introducción

La plataforma omnidireccional es uno de los módulos que conforman el kit educativo para robótica móvil. En este caso, este cuenta coma una distribución de cuatro ruedas tipo mecanum de 80 mm de diámetro con un ángulo de inclinación de 45° de los rodillos respecto al eje axial de la rueda, lo cual le brinda la capacidad de poder desplazarse en múltiples direcciones. Como se muestra en la Figura 1, la plataforma puede moverse de ocho formas distintas dependiendo de la dirección de giro de sus ruedas. La plataforma tiene 21 cm de ancho y 19.5 cm de largo, y puede cargar hasta un máximo teórico de 20 kg.

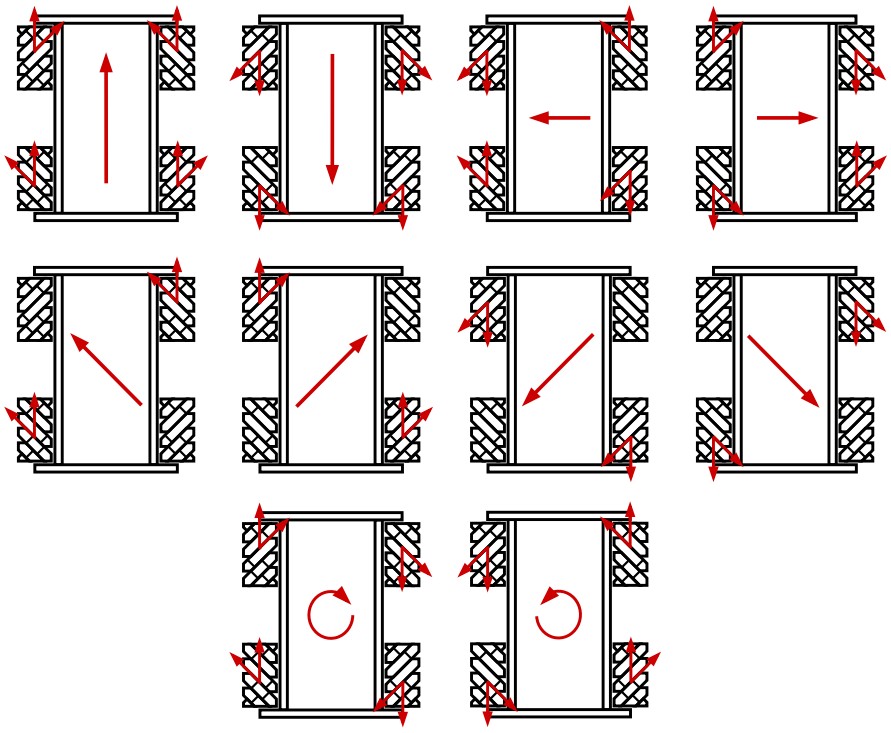

Figura 1: Movimientos del módulo omnidireccional.

1.1 Motor JGY-370

Para accionar el movimiento de cada una de las ruedas mecanum se utilizó un motor DC en cada una de ellas de tal forma que la plataforma cuente con cuatro motores que funcionen de forma independiente. El motor seleccionado es un motor DC **JGY-370**, el tiene un total de seis conexiones: dos concernientes a la alimentación de motor y cuatro que corresponden a las conexiones de un encoder incremental de sensor tipo Hall instalado (ver Figura 2).

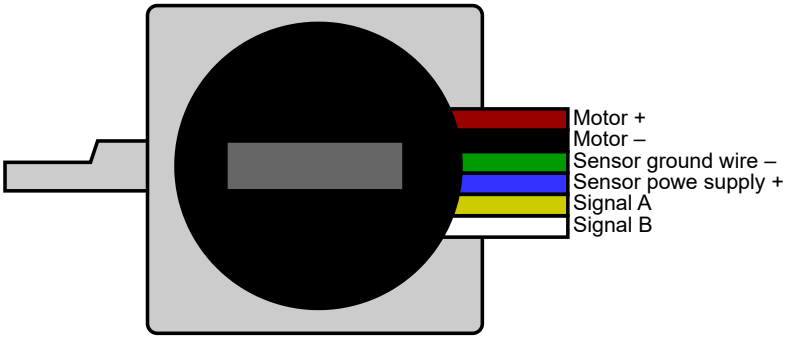

Figura 2: Salidas del motor.

En un encoder tipo Hall, se colocan sensores Hall alrededor de un imán permanente, y cuando el imán se mueve, los sensores Hall detectan los cambios en el campo magnético y generan señales eléctricas proporcionales al movimiento. Es decir, cada vez que pase un pulso, se genera un flanco de subida en su salida. En la Figura 3, se puede ver el funcionamiento del encoder incremental, donde este cuenta con dos fases (Canal A y B). En este sentido, cada una de estas salidas genera pulsos cuando el motor se empieza a mover y se utilizan dos de ellas para determinar el sentido de giro del mismo. En caso la fase A esté por delante de la fase B, el motor se encuentra girando en un sentido; en caso contrario, el motor está girando en el sentido opuesto.

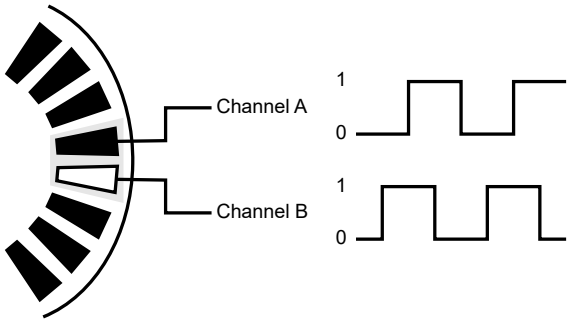

Figura 3: Funcionamiento de un encoder incremental tipo Hall.

1.2 Driver TB6612FNG

Para el control de los motores del módulo, se necesitan drivers que puedan soportar cuatro motores y manejar la cantidad de corriente que requieren los motores. En este caso, se tienen dos drivers de dos canales, es decir, cuenta con capacidad para controlar hasta dos motores cada uno. Además puede entregar hasta 3 A por canal, más que suficiente para el consumo que tendrá cada uno de los motores del módulo.
Para el control de estos drivers se requieren tres señales por cada canal. Como se aprecia en la Figura 4, cada uno de los canales se encuentran al lado derecho del driver y están denotados de la forma xINy y PWMx, donde x denota el canal (A o B) e y denota el pin de control de dirección (1 o 2). Cada uno de estos canales cuenta con dos pines de dirección y un pin de PWM. El pin PWM es una entrada para variar la velocidad del motor y los pines de dirección son entradas digitales que permiten que el motor vaya en sentido horario o antihorario. No obstante, para el control de dirección únicamente se utilizará un pin del microcontrolador, de esta manera, se logra controla cada uno de los distintos canales de forma independiente con menor número de pines y utilizando un único microcontrolador.

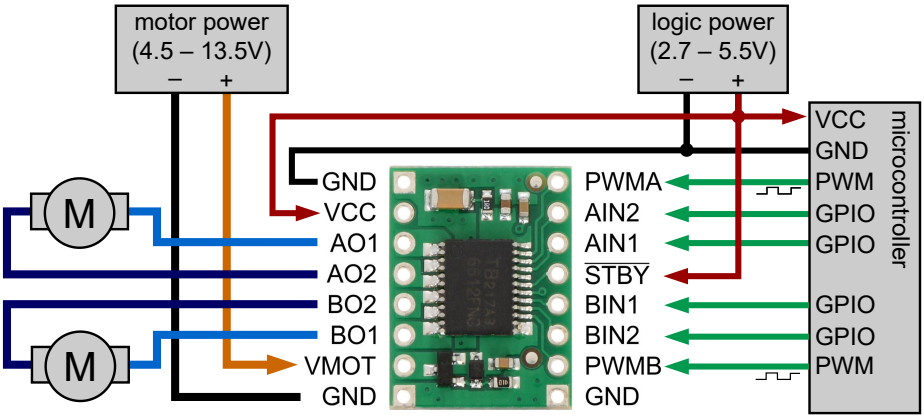

Figura 4: Driver de los motores DC.

1.3 Arduino Uno

El microcontrolador empleado para este módulo es el Arduino UNO, el cual es uno de los microcontroladores más utilizados para prototipado en el mundo. Su principal fortaleza radica en la cantidad de recursos disponibles en internet, lo que facilita y motiva su uso en el desarrollo de proyectos. En la Figura 5, se muestran las conexiones que se realizan al Arduino UNO para el control de los drivers de motores.

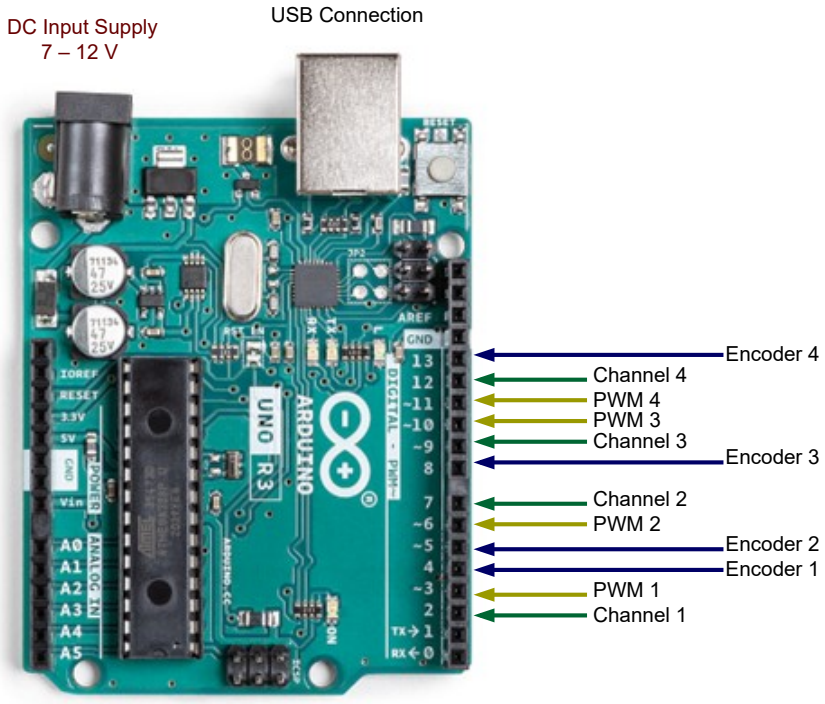

Figura 5: Conexiones en el Arduino UNO.

2 Controles Básicos de la Plataforma

A continuación, se muestran los códigos del control de la plataforma omnidireccional, ejemplos de códigos de la lectura de los encoders, control de los motores y control PID.

2.1 Lectura de encoders

Para la lectura de encoders se utilizará una interrupción por cada motor, por lo cual, se debe incluyir la librería `PinchangeInterrupt.h` ya que el Arduino Uno solo cuenta con dos pines de interrupción (los pines 2 y 3). En este sentido, las variables de entrada para la rutina de interrupción son: el pin de interrupción, la función que se ejecuta (ISR - Rutina de servicio de interrupción), la cual no debe tomar parámetros ni retornar valores, y el modo, el cual indica cuándo se debe activar la interrupción. Esta función se declara de la siguietne forma:

Declarar ISR

```
#include "PinChangeInterrupt.h"
attachPinChangeInterrupt ( digitalPinToPinChangeInterrupt (pin), handleFunc, FALLING);
```

La función de servicio de interrupción donde se suman la cantidad de flancos se muestra a continuación:

Actualizar Contador

```
void handleFunc (){
    encNcount++;  // Se cuenta cada vez que se detecte un pulso en el encoder
}
```

Para medir las revoluciones por minuto de cada motor se realiza la lectura de los encoders cada 100 milisegundos y se convierte a rpm. Para los motores se cuenta con aproximadamente 1390 pulsos por revolución.

Obtener Velocidad

```
unsigned long t = millis();
dt = (double) t - t_ant;

if (dt >= 100){
    t_ant = t;
    w = 10*encNcount*(60.0/1390.0);  // La variable w es la variable de salida en rpm
    encNcount = 0;  //Reset del contador, pues la muestra se toma cada 100 milisegundos
}
```

2.2 Accionamiento de Motores

Para controlar al motor se debe hacer uso de los pines de PWM, lo cual nos permite obtener una señal cuadrada con frecuencia de 490 Hz y ciclo de trabajo (Duty Cycle) variable. El ciclo de trabajo de una onda PWM es la relación que existe entre el tiempo en que la señal se encuentra en estado activo (T_{on}) y el periodo de la misma (T), ver Figura 6.

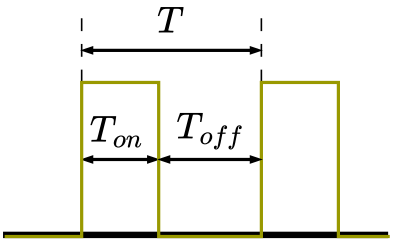

Figura 6: Onda cuadrada PWM.

Para variar la velocidad de cada motor se puede variar la tensión aplicada y utilizando la función `digitalWrite()`, la cual cuenta con una salida con resolución de 8 bits, es posible variar la señal de control hacia el driver con valores de 0 - 255 para realizar variaciones de anchura del pulso de 0 a 100%. Además, la asignación de la dirección del movimiento de las ruedas se invierte en el sentido del flujo de corriente, lo cual se puede variar a través del pin de dirección del driver colocando en `LOW` o `HIGH` como se muestra en el siguiente código:

Accionar los Motores

```
int valorPWM;
int dirchn1 = 3;
int speedch1 = 2;  // Pin PWM

void setup() {
    pinMode(dirchn1,OUTPUT);
    pinMode(speedch1,OUTPUT);
}

void loop() {
    valorPWM = 255;  // En el rango de 0 a 255
    digitalWrite(dirchn1,HIGH);  // Se asigna la   direccin
    analogWrite(speedch1,valorPWM);
}
```

3 Sintonización de PID e Implementación

En esta sección se explica la sintonización Ziegler-Nichols en lazo abierto, para obtener los valores de K_p, K_d y K_i del controlador PID de cada motor.

3.1 Sintonización Ziegler-Nichols

Para determinar los valores de las constantes proporcional,integral y derivativa (K_p, K_d y K_i) se utiliza el método de Ziegler-Nichols, pues no se cuenta con el modelo matemático de la planta. Este método se realiza con el lazo abierto y se requiere que la respuesta no tenga sobre impulsos. Para poder determinar la respuesta a la planta, se debe tener una señal escalón aplicada en la entrada y, a partir de la respuesta de salida del sistema, se obtienen los parámetros del controlador.
Este método consiste en trazar una línea tangente a la respuesta del sistema o curva de reacción en el punto de inflexión o punto de máxima pendiente. Para obtener el modelo, se debe identificar el tiempo muerto (T_u) y el tiempo de estabilización (T_g). En la Figura 7 se presenta una gráfica que muestra la sintonización en tiempo continuo; sin embargo, en este laboratorio se trabajará con tiempo discreto.

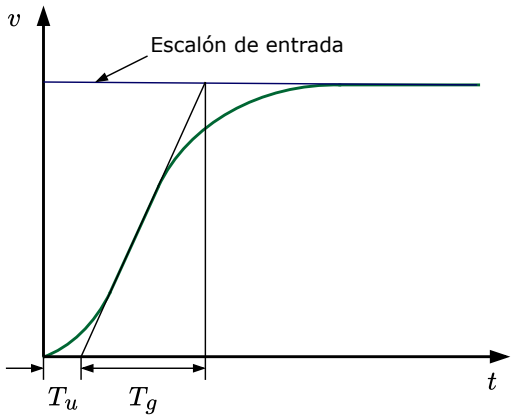

Figura 7: Método Ziegler-Nichols en tiempo continuo.

En la Figura 8, se observa una gráfica de la sintonización en tiempo discreto, la coordenada en el eje horizontal representa al tiempo en segundos y la coordenada en el eje vertical representa a la velocidad en rpm.

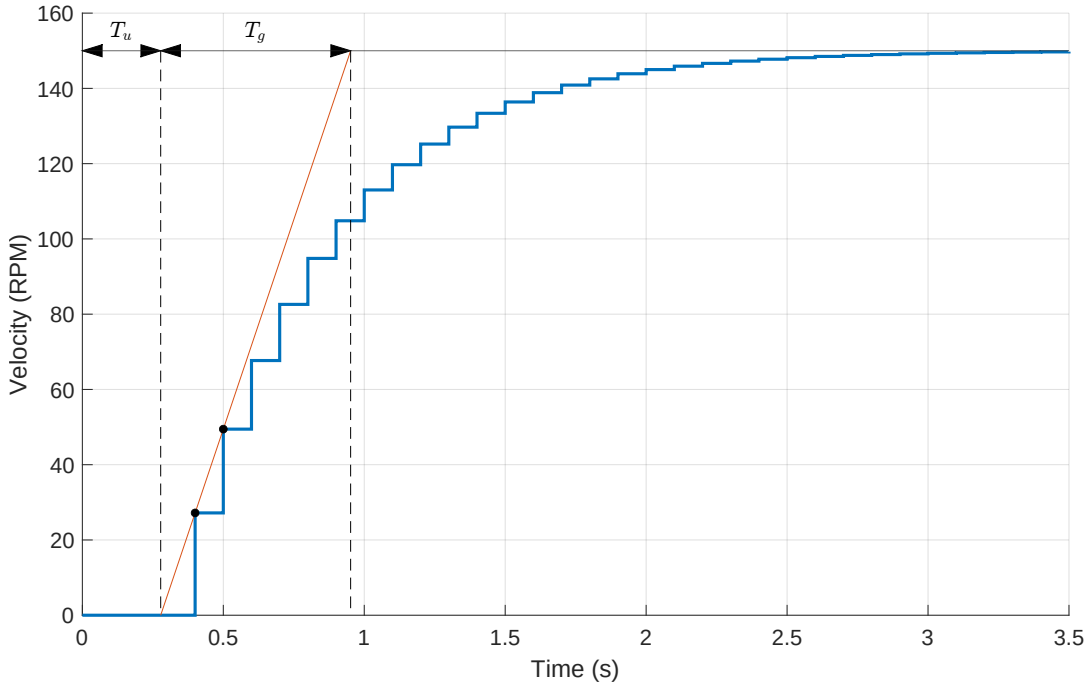

Figura 8: Método Ziegler-Nichols en tiempo discreto.

Para determinar los valores de T_u y T_g se considera que la relación (T_u/T_g) está entre 10% a 30%. A partir de ello, se determinará los valores de las ganancias de controlador, considerando $K_i = K_p/T_i$ y $K_d = K_p \cdot T_d$, los cálculos se muestran en la Tabla 1.

Controlador	$\mathbf{K_p}$	$\mathbf{T_i}$	$\mathbf{T_d}$
PID	$1.2 \cdot T_g/T_u$	$2 \cdot T_u$	$0.5 \cdot T_u$

Tabla 1: Tabla de controladores PID.

Así, el controlador PID en su representación continua en el tiempo tiene su forma en función de las ganancias proporcional, derivativa e integral como

$$u(t) = K_p\, e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \triangleq K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(t)dt + T_d \frac{de(t)}{dt} \right)$$

donde $e(t)$ es el error entre la referencia y la señal medida (velocidad en rpm del motor), y $u(t)$ es la entrada de control (señal de control PWM). Sin embargo, en el dominio discreto con un tiempo de muestreo $T = 0.1$ s, es decir, para su implementación como algoritmo, se debe trabajar en ecuación de diferencias que utiliza valores de muestras pasadas para general la señal de control, la cual tiene la forma

$$\begin{cases} u(k) = K_p\, e(k) + K_d \left(\frac{e(k) - e(k-1)}{T} \right) + K_i\, e_i(k) \\ e_i(k) = e_i(k-1) + e(k)T, \qquad e_i(0) = 0 \end{cases}$$

A continuación, se muestra un ejemplo de código de Arduino UNO aplicando una entrada escalón a los motores. Asimismo, se muestran en el puerto serial las velocidades que alcanza la planta (en rpm) cada 0.1 s. En base a ello, se puede realizar una gráfica y determinar el tiempo muerto/retardo y el tiempo de subida.

Prueba de Zieger-Nichols

```
#include "PinChangeInterrupt.h"
#include<math.h>
// Variables de tiempo para interrupciones del calculo de los rpm de los motores
volatile unsigned muestreoAnterior = 0;
unsigned long muestreoActual = 0;
volatile double deltaMuestreo = 0;
// Variables de tiempo para mostrar
volatile unsigned muestreoAnteriorPrint = 0;
unsigned long muestreoActualPrint = 0;
volatile double deltaMuestreoPrint = 0;
volatile float pv1;
// Pines de pwm
int speedch1 = 3;
// Direction pins
int dirch1 = 2;
// Variables para la cueta de encoders
volatile long encoder1Count = 0;
// Se define el numero de pines para el encoder
#define encoder1 4
volatile float rpm1;

void setup() {
    Serial.begin(115200);
    // Setear Pines como salida
    pinMode(dirch1,OUTPUT);
    // Pines del encoder como entrada
    pinMode(encoder1,INPUT);
    // Interrupciones para el encoder, con pines que no admiten interrupcin se coloca el pintopinchange
    attachPinChangeInterrupt(digitalPinToPinChangeInterrupt(encoder1), handleEncoder1,FALLING);
    analogWrite(speedch1,0);
    delay(2000);
    Serial.println("Set up terminado");
    muestreoAnterior = 0;
}

void loop() {
    // Entrada escalon
    digitalWrite(dirch1,LOW);
    analogWrite(speedch1,255);
    muestreoActual = millis();
    deltaMuestreo = (double) muestreoActual-muestreoAnterior;
    muestreoActualPrint = millis();
    deltaMuestreoPrint = (double) muestreoActualPrint-muestreoAnteriorPrint;
    // Mayor a 100 milisegundos
    if (deltaMuestreo >= 100){
        muestreoAnterior = muestreoActual;
        pv1 = 10*encoder1Count*(60.0/1390.0); // La variable pv es la vaiable de salida en rpm
        encoder1Count = 0; // Reset del contador
    }
    // Observamos el mensaje cada 100 ms
    if (deltaMuestreoPrint >= 10){
    muestreoAnteriorPrint = muestreoActualPrint;
    Serial.print("El valor de la velocidad en rpm es : ");Serial.println(pv1);
    }
}

void handleEncoder1(){
    encoder1Count++;
}
```

3.2 Control PID en Arduino

Para realizar el control del PID en Arduino UNO, se pueden usar librerías, como por ejemplo `PID.h`, la cual facilita la programación. Sin embargo, en este ejemplo se realiza la programación del control discreto de PID desde cero, para lo cual se necesitan los valores de K_p, K_d y K_i anteriormente hallados. Así, para limitar los picos por la lectura de los encoders, se limita a la cuenta de rpm a 300 como máximo y en lugar de ello se coloca al valor de la referencia (setpoint), el lazo de control se muestra en la Figura 9.

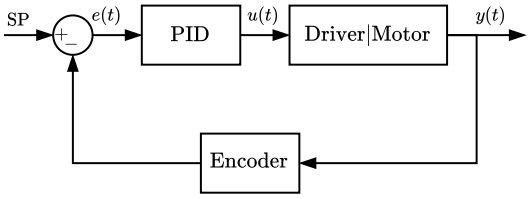

Figura 9: Lazo de control de motores.

El algoritmo de control implementado como clase y con la ecuación de diferencias del controlador PID se muestra a continuación:

Control PID

```
class SimplePID{
    private:
        float kp, kd, ki, umax; // Parameters
        float eprev, eintegral; // Storage

    public:
    // Constructor
    SimplePID() : kp(1), kd(0), ki(0), umax(255), eprev(0.0), eintegral(0.0){}

    // A function to set the parameters
    void setParams(float kpIn, float kdIn, float kiIn, float umaxIn){
        kp = kpIn; kd = kdIn; ki = kiIn; umax = umaxIn;
    }

    // A function to compute the control signal
    void evalu(int value, int target, float deltaT, int &pwr){
        // error
        int e = target - value;
        // derivative
        float dedt = (e-eprev)/(deltaT);
        // integral
        eintegral = eintegral + e*deltaT;
        // control signal
        float u = kp*e + kd*dedt + ki*eintegral;
        pwr = (int) fabs(u);
        if (pwr > umax) {
            pwr = umax;
        }
        if (pwr < 0) {
            pwr = 0;
        }
        eprev = e;
    }
};
```

Como se puede apreciar, uno de los componentes que mayor influencia tiene en la respuesta de control de motor es el sensor, si este componente no está correctamente calibrado o no es lo suficientemente preciso para los requerimientos operacionales de uso, no se llegará al valor de deseado real. En este sentido, se puede modelar al sensor con un componente que es afectado por el ruido y, a través de filtros y procesamiento en conjunto con otros sensores, es posible mejorar y reducir el error en estos dispositivos.

4 Ejercicio

Se les pide realizar la cinemática directa e inversa de la configuración omnidireccional. La configuración es con ruedas mecanum y considerarla igual a la mostrada en la Figura 10.
Datos: Ancho = $2b$, Largo = $2d$, Radio de ruedas = R y considerar los ejes tal como se muestran en la gráfica.

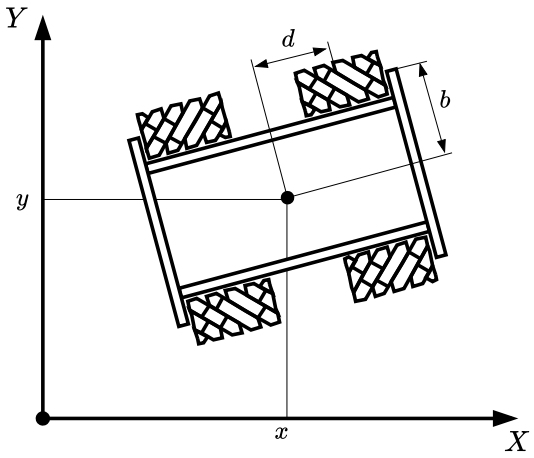

Figura 10: Configuración omnidireccional con ruedas mecanum.