

Tutorial de Funciones Trigonometricas en Robocode

Para ser capaz de escribir un robot inteligente en Robocode, debes usar frecuentemente algunas funciones matemáticas:

Por Ejemplo:

- ¿En qué ángulo tengo que disparar para alcanzar una posición específica (x,y)?
- Si me muevo una distancia x en una dirección, ¿a dónde puedo llegar?
- ¿En qué dirección debo moverme para alcanzar una posición (x,y)?

Para calcular las respuestas a estas preguntas necesitamos de la trigonometría. En este documento trataremos de explicar como utilizar funciones trigonometricas para programar tu robot.

1. Preguntas Frecuentes:

- ¿Cómo puedo convertir “bearing” (dirección relativa) a “heading” (dirección absoluta)?
- ¿Cómo puedo convertir “heading” (dirección absoluta) a “bearing” (dirección relativa)?
- ¿Cómo puedo convertir radianes a grados?
- ¿Cómo puedo convertir grados a radianes?
- Tengo la dirección (en grados) y la distancia de otro robot. ¿Cómo puedo determinar las coordenadas de ese robot?
- Tengo las coordenadas cartesianas de otro robot. ¿Cómo puedo determinar la dirección hacia ese robot?

2. El Campo de Batalla

El campo de batalla en Robocode es rectangular. Cada punto tiene una coordenada x (horizontal) y una coordenada y (vertical). Las coordenadas reflejan un par (x,y), del cual el primero es x y la segunda coordenada es y. Se debe remarcar que el cuadrante se inicia en el punto de origen (0,0) en el extremo izquierdo inferior (ver imagen) y se extiende hacia la derecha y hacia arriba con los valores de `battlefield_width` (ancho) y `battlefield_height` (alto).

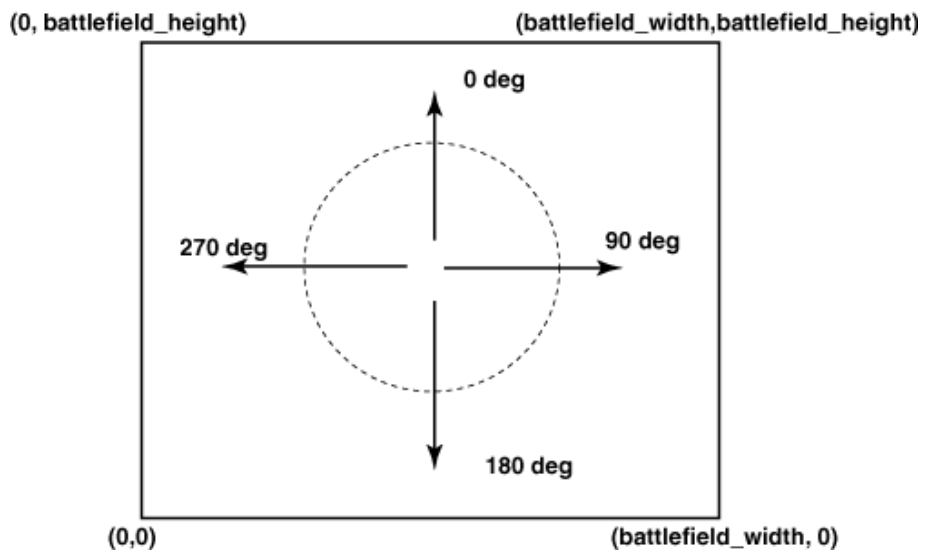
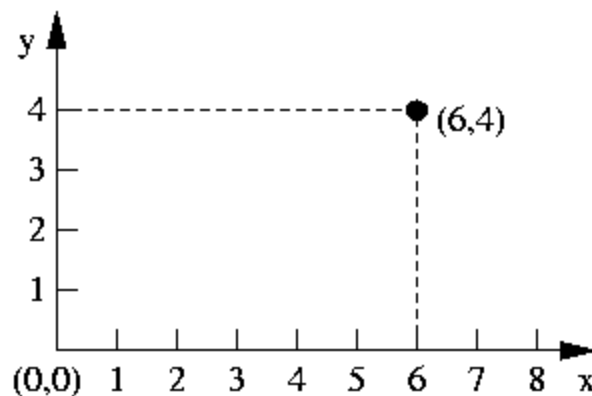


Imagen tomada de [Rock 'em, sock 'em Robocode: Round 2](#)



Un ejemplo podría ser el punto $(6,4)$ que está marcado en esta imagen.

3. Ángulos

En Robocode encontraremos 2 tipos de ángulos que usaremos frecuentemente:

Ángulos Absolutos (Heading)

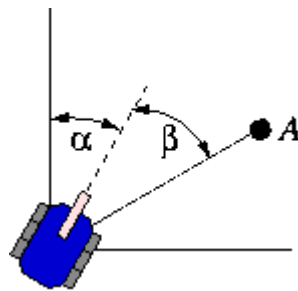
Son los ángulos con respecto a un par de coordenadas fijas. Puedes compararlas con los ejes cartesianos: Norte, Este, Sur y Oeste. Un ángulo de 0 grados tiene una dirección hacia arriba, un ángulo de 90 grados tiene una orientación hacia la derecha, etc. Los ángulos absolutos en Robocode pueden ser utilizados para determinar por ejemplo, con el método `Robot.getHeading()`, hacia donde se dirige un robot.

En nuestros estudios aprendimos que el ángulo de 0 grados indica hacia la derecha en vez de arriba, pero en Robocode es diferente, por lo tanto las funciones seno y coseno debes ser usadas de forma diferentes de lo que estamos acostumbrados.

Ángulos Relativos (Bearing)

Son los ángulos relativos a la dirección a la cual esta orientada tu robot. Estos pueden ser comparados con los términos como izquierda y derecha. Si un evento ocurre, con el método `getBearing()` puedes determinar desde que ángulo esta ocurre, por ejemplo: escanear un robot, golpear un muro, desde que dirección nos impacto un proyectil, etc. “Bearing” de 0 grados sería a la derecha de tu robot, “bearing” de -90 sería la izquierda, “bearing” de 90 es derecha y “bearing” de 180 o -180 grados sería atrás.

La diferencia entre heading y bearing es más transparente en el siguiente ejemplo:



Imaginemos que el objeto **A** es un tanque enemigo y nos queremos dirigir hacia el, entonces por un lado podemos determinar el ángulo α que es la dirección de nuestro robot, entonces utilizaremos el método `getHeading()=α`, de esta forma obtenemos el ángulo absoluto de dirección de nuestro tanque, por otro lado si queremos dirigir nuestro robot al objeto **A**, necesitaremos el ángulo β por lo cual deberemos ejecutar `getBearing()=β` en el evento `onScannedRobot` y luego utilizamos el método: `turnRight(β)`; o de forma completa `turnRight(event.getBearing())` y nos orientaremos hacia el objeto **A** y si ponemos `ahead(int x)`; nos acercaremos a la posición de **A**.

¿Cómo podemos convertir “bearing” (dirección relativa) a heading (dirección absoluta)?

Una dirección relativa es un ángulo relativo al ángulo absoluto en el cual esta dirigido tu robot. En el método siguiente se especifica como puedes realizar esta conversión, asegurándote que el resultado se encuentre en un rango de 0 a 360 grados.

```
public double heading(double bearing) {  
    double a = (this.getHeading() + bearing) % 360;  
    if (a < 0) a += 360;  
    return a;  
}
```

El método anterior puedes utilizarlo por ejemplo en el evento onScannedRobot para determinar la dirección o ángulo absoluto en el cual se encuentra el robot que estas escaneando.

```
double absAngle = heading(e.getBearing());
```

¿Cómo puedo convertir “heading” (dirección absoluta) en “bearing” (dirección relativa)?

Este es la inversa, puedes deducirla de la dirección absoluta de tu robot encontrando el ángulo relativo, y asegurándote que el rango se encuentre entre 180 y -180. Esto es posible con el siguiente método.

```
public double bearing(double heading) {  
    double angulo = heading - this.getHeading(); //  
    Ángulo = ángulo % 360; // aseguramos que el ángulo este entre -180 y 180  
    if (angulo > 180) angulo -= 360;  
    else if (angulo < -180) angulo += 360;  
    return angulo;  
}
```

Luego con la siguiente instrucción podemos determinar el “heading”:

```
double absAngle = bearing (e.getHeading());
```

4. Grados y Radianes

Existen muchos métodos que trabajar con ángulos. En Robocode muchas funciones utilizan grados pero en matemáticas es más usual trabajar con radianes. La mayoría de los métodos del API de Java (incluidos seno y coseno) utilizan radianes, por esa razón deberías conocer como pasar de grados a radianes y viceversa. Las formulas son:

- $\text{Radianes} = \text{grados} * (2\pi / 360)$
- $\text{Grados} = \text{radianes} * (360 / 2\pi)$

Utilizaremos los siguientes métodos en Java:

- `Math.toDegrees(double)`
- `Math.toRadians (double).`

¿Como puedo convertir radianes a grados?

Se utiliza el método `Math.toDegrees (double)`, donde el parámetro de entrada es en radianes.

```
double grados = Math.toDegrees(radianes);
```

¿Como puedo convertir de grados a radianes?

Se utiliza el método `Math.toRadians (double)`, donde el parámetro de entrada es en grados.

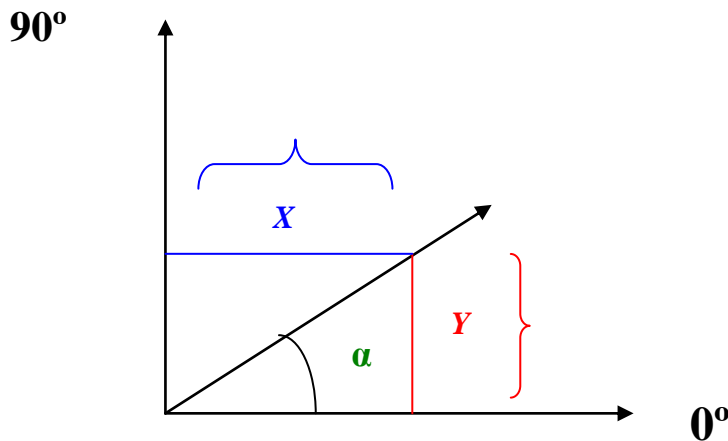
```
double radianes = Math.toRadians(grados);
```

Si te das cuenta que estas conversiones te complican aún mas y crees que es más fácil trabajar en radianes o en grados, entonces no te preocupes ya que puedes utilizar cualquiera de los dos sistemas, porque para cada método existe un equivalente. Por ejemplo `turnRight(90)` gira tu robot 90 grados a la derecha, mientras que `turnRightRadians (Math.PI/2)` gira tu robot $1/2\pi$ radianes, (es decir ambos métodos realizan la misma función).

5. Seno en vez de Coseno

Con el seno y coseno podemos calcular un punto específico (x,y) en el eje de coordenadas, si poseemos el ángulo y la distancia a la cual se encuentra ese punto. Si tenemos un punto cuya distancia es 1 y su ángulo absoluto es α entonces la coordenada en el (x) del punto será: el $\text{sen}(\alpha)$ * la distancia=1 ;y la coordenada (y) del punto será $\text{cos}(\alpha)$ * la distancia=1; Si la distancia es diferente que 1 entonces simplemente deberemos multiplicar el nuevo valor de la distancia por el seno o el coseno.

Método Original



$$\cos(\alpha) = \frac{\text{Cateto Adyacente}}{\text{Hipotenusa}} = \frac{X}{\text{Distancia}}$$

$$X = \cos(\alpha) * \text{Distancia}$$

$$\text{sen}(\alpha) = \frac{\text{Cateto Opuesto}}{\text{Hipotenusa}} = \frac{Y}{\text{Distancia}}$$

$$Y = \text{sen}(\alpha) * \text{Distancia}$$

Método en Robocode

En la secundaria aprendimos que con la función seno, obteníamos la coordenada (Y) y con la operación coseno obteníamos la coordenada (X). Pero en Robocode es diferente porque los grados comienzan desde arriba hacia la derecha en contra del sentido de las manecillas del reloj y no como estamos acostumbrados. Por lo tanto se debe utilizar la función coseno en vez de la de seno, y la de seno en vez de coseno.

$$\cos(\alpha) = \frac{\text{Cateto Opuesto}}{\text{Hipotenusa}} = \frac{Y}{\text{Distancia}}$$

$$Y = \cos(\alpha) * \text{Distancia}$$

$$\text{sen}(\alpha) = \frac{\text{Cateto Adyacente}}{\text{Hipotenusa}} = \frac{X}{\text{Distancia}}$$

$$X = \text{sen}(\alpha) * \text{Distancia}$$

Tengo la dirección absoluta “heading” y la distancia de otro robot. ¿Cómo puedo determinar las coordenadas del robot?

Puedes utilizar los siguientes métodos `Math.sin (double)` y `Math.cos (double)`. Ambos métodos solicitan el ángulo en radianes.

```
//determinamos las coordenadas del robot
double dx = Math.sin(toRadians(heading)) * distancia;
double dy = Math.cos(toRadians(heading)) * distancia;
// Aplicamos las funciones trigonometricas
double xCoordinaat = this.getX() + dx;
double yCoordinaat = this.getY() + dy;
```

Fíjense en la parte de arriba de este código que en la mayoría de los métodos o eventos solo se puede preguntar por “bearing” (que es un ángulo relativo), antes de ser utilizado este debe ser [convertido primero de un ángulo relativo a un ángulo absoluto](#).

Si tengo las coordenadas cartesianas de otro robot. ¿Como puedo determinar la dirección hacia el otro robot?

Puedes utilizar el método `Math.atan2 (double, double)` con la coordenada (x), y una coordenada (y), devolviendo el ángulo en radianes, el problema de este método es que devuelve el ángulo del punto al origen porque la función trabaja desde el origen por esa razón primero debes adaptar las coordenadas a tu posición

```
// Determinar primero las coordenadas relativa con relación a tu robot
double dx = target.getX() - this.getX();
double dy = target.getY() - this.getX();
// utiliza el método Math.atan2(dx, dy)para determinar el ángulo
double heading = Math.toDegrees(Math.atan2(dx, dy));
```

Fíjense que este código produce una dirección absoluta. Si quieres usar estos datos para controlar tu robot, [generalmente deberás convertir el ángulo absoluto en un ángulo relativo](#).

Tengo las coordenadas cartesianas de otro robot. ¿Cómo puedo determinar la distancia entre mi robot y el otro?

Con el Teorema de Pitágoras: $a^2 + b^2 = c^2$. En Java podemos calcular la distancia de la siguiente manera:

```
// Calculamos primero la posición con respecto a tu posición (calculamos a y b)
double dx = target.getX() - this.getX();
double dy = target.getY() - this.getY();
// Utilizamos Pitágoras para calcular la distancia
double distancia = Math.sqrt(dx * dx + dy * dy);
```