PRÁCTICA 3

ROBÓTICA

Trabajo realizado por:
Carmen Carrero Hurtado
Andrés Fernández Pérez

INTRODUCCIÓN:

En esta entrega se nos pedía modificar el componente de la práctica anterior para que cuando pulsaramos sobre cualquier parte del tablero el robot se moviera hasta allí. Si pulsamos varias veces el ratón en distintos lugares, el robot irá a la última posición pulsada sin tener en cuenta los obstáculos, pudiente en esta entrega atravesarlos.

Para llegar a esto, incluimos unas líneas en el archivo controller.cdsl para que reconociera el click del ratón y regeneramos el componente. Después editamos el archivo config y después de compilar de nuevo se creó un nuevo método setPick.

Lo siguiente que había que hacer era definir una estructura thread-safe, por lo que creamos una clase nueva llamada Target, que explicaremos más adelante. El objetivo de esta estructura es almacenar las coordenadas del mundo real a las que tiene que dirigirse el robot.

Una vez que teníamos la clase implementada, nos pusimos a desarrollar el código.

CÓDIGO DESARROLLADO:

Vamos a empezar explicando la clase implementada para almacenar las coordenadas, llamada Target.

Esta clase tendrá como atributos un vector de tamaño dos que almacenará las coordenadas, un mutex para garantizar la exclusión mutua y un boolean, inicialmente a falso, que se utilizará para comprobar si el robot tiene coordenadas pendientes o no.

Target también cuenta con cuatro métodos: insert(float x, float y), extract(), setPendiente(bool pend) y getPendiente(). El primer método inserta los valores pasados por parámetro en el vector de coordenadas y pone el boolean a true para indicar que el robot tiene coordenadas pendientes. El método extract devuelve el vector de coordenadas, así como los métodos set y getPendiente que modifican y retornan el boolean, respectivamente.

Todos los métodos garantizan una estructura segura para hilos mediante la variable mutex.

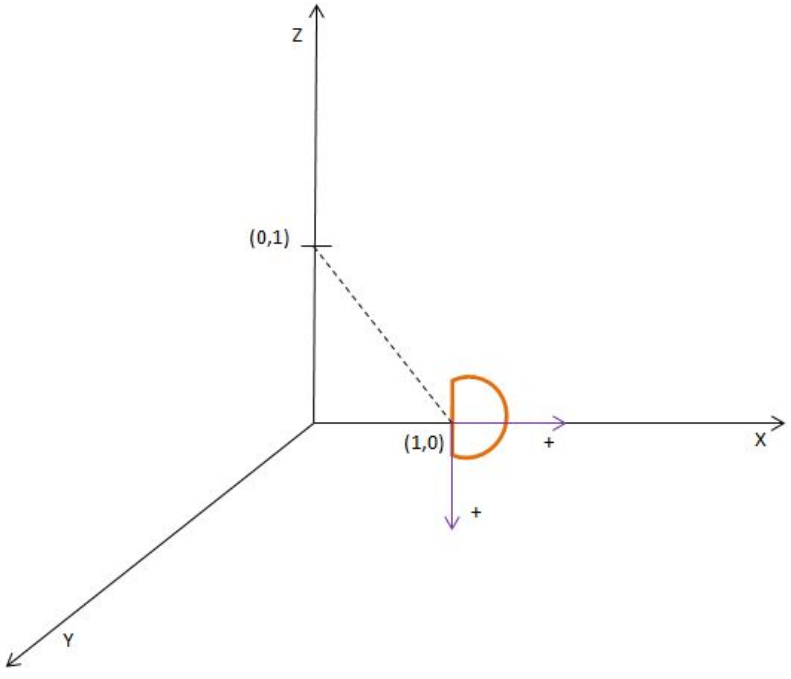
Una vez explicada la clase Target, comentamos la clase specificWorker, ya que fue la única que modificamos para la creación de este componente.

El algoritmo seguido para el desarrollo de la práctica fue el siguiente:

Teniendo el cuenta que el mundo está en 3D y el robot ve en 2D, tenemos que pasar las coordenadas del mundo real (las que recibe del ratón) a las del robot. Para tener una mejor concepción de esto se desarrolla un ejemplo ilustrativo.

Como vemos en la figura, el robot se encuentra en las coordenadas (1,0) orientado hacia la derecha, de tal manera que el eje del robot será el marcado en lila. Ponemos como coordenadas destino el punto (0,1), que si transformamos eso a cómo lo vería el robot sería el punto (-1, -1).

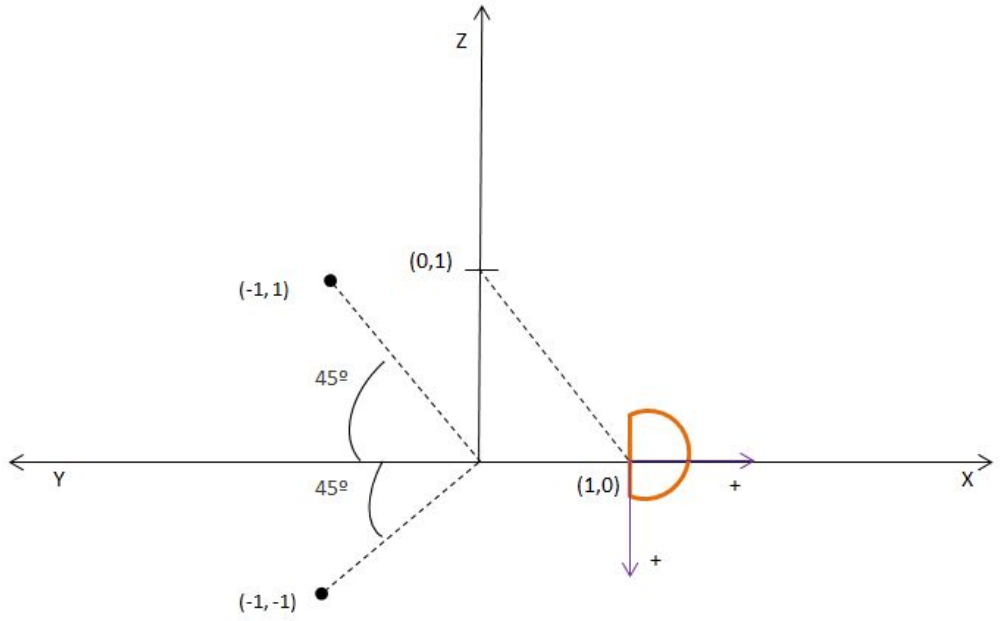
La cuestión ahora es encontrar una fórmula para realizar eso.



La fórmula escogida fue:

$$X_R = R^T (Z_W - T_W)$$

Para representar esto gráficamente, seguimos con el ejemplo anterior donde Z - T = (0,1) - (1,0) = (-1, 1) y R = 90º (ya que el robot está mirando a la derecha y solo gira una vez -90º- para "mirar recto").



Efectivamente, comprobamos visualmente que una vez que te situas en el punto (-1, 1) giras 90º y acabas en el punto (-1, -1) que serían las coordenadas (0, 1) del mundo real en la posición del robot.

Trasladamos esto al código y añadimos una fórmula más para calcular la velocidad que llevaría el robot en cada momento.
Finalmente, suponemos que si la distancia es menor que 50 se ha llegado al objetivo, por lo que paramos al robot y actualizamos el boolean de la clase Target a false, pues no hay coordenadas pendientes.