# Robótica 2021-2022

1st Andrés Ventura Valiente
*aventurav@alumnos.unex.es*

2nd Carlos Sánchez García
*csancheznf@alumnos.unex.es*

3rd Yeray Bravo Díaz
*ybravodi@alumnos.unex.es*

## I. HERRAMIENTAS

En primer lugar, antes de comenzar a realizar los cambios necesarios para que la tercera práctica fuese correctamente tuvimos que añadir un componente nuevo, llamado gotoxy. Y con el cual tuvimos que crear la definición del componente: robocompdsl gotoxy.cdsl. Este componente nos permite poder hacer click en una parte del mapa, y que nuestro robot se desplace hasta ese punto.

## II. Objetivos de la entrega

Para esta entrega se precisa de que nuestro robot de limpieza sea capaz de que si clicamos en una parte del mapa vaya hacía ese punto automaticamente. Y también se precisa de que en nuestro componente recién creado, también se visualice el láser del propio robot. Para la obtención de esta práctica se han seguido unos pasos:

### A. Dibujar los datos del láser

Hemos tenido que crear el componente ”gotoxy” con el editor robocompdsl. Para que el componente pueda dibujar el láser hemos creado el método drawlaser, que tiene como parámetro la información del láser. Está representado en un póligono, por lo que creamos un póligono: QPolygonF poly, y dentro del póligono le vamos metiendo todos los datos que va procesando: poly ¡¡ QPointF(l.dist * sin(l.angle), l.dist * cos(l.angle)). Pero para poder desarrollar correctamente el polígono, debemos añadir la coordenada 0,0, para que pueda empezar y terminar el polígono desde la ubicación del láser: poly ¡¡ QPointF(0,0). Y poder visualizar el láser en todo momento, lo declaramos en nuestro método compute, que está en todo momento en funcionamiento después de obtener los datos del láser: 1. RoboCompLaser::TLaserData ldata = laserproxy-¿getLaserData() 2. drawlaser(ldata)

### B. Capturar el click

Para poder conseguir que nuestro robot aspirador pueda ir a la ubicación que queramos, primero tenemos que capturar las coordenadas de la ubicación deseada. En primer lugar, creamos una estructura donde vamos a tener tres atributos: un T, que lo inicializamos como un Vector2f, para almacenar las coordenadas, un QPointF que según clicamos las guarda y un booleano que si clicamos se pone a true y cuando llega al punto se pone a false. Primero conseguimos las coordenadas a través del método click: void SpecificWorker::click(QPointF punto), que lo que hace coge el click y lo guarda en el primer atributo de nuestro struct: t1.content=Eigen::Vector2f (punto.x(),punto.y()) y pone el booleano a true ya que ha detectado un click.

### C. Calcular la rotación y distancia

Después de obtener las coordenadas, calculamos la rotación y la distancia que el robot deberá realizar para poder llegar a la ubicación deseada. Para ello, como el robot puede estar en cualquier sitio del mapa, primero guardamos el estado del robot: Eigen::Vector2f rw(bState.x,bState.z), después debemos calcular la rotación del robot que para ello debemos calcular la diferencia vectorial geométrica, que obtemos el punto opuesto: rot¡¡std::cos(bState.alpha),(std::sin(bState.alpha)),-std::sin(bState.alpha),std::cos(bState.alpha), y con el que calculamos la orientación: auto tr=rot*(t1.content-rw) y finalmete la rotación: float beta=std::atan2(tr.x(),tr.y()). Una vez calculada la rotación debemos calcular la distancia: float dist=tr.norm().

### D. Reducir velocidad en el aproximamiento al punto

El último punto a calcular, es la velocidad de rotación y de avance. Para ello, la velocidad de rotación va a ser la beta que tenemos calculada del paso anterior. Y para calcular la velocidad de avance, tenemos que controlar que según esté llegando al punto vaya reduciendo la velocidad, y para ello aplicamos la siguiente fórmula: advspeed = MAXADVSPEED * reducespeedifturning * reducespeedif closetotarget. Dónde MAXADVSPEED está limitada a 1000: float MAXADVANCE=1000. Los otros dos factores están comprendidos entre 0 y 1, que son los factores que van a hacer que reduzca la velocidad de avance. Si esta acercándose al punto, compara con la velocidad máxima de avance, y si es mayor devuelve uno para que no influya en la velocidad, pero si es menor, devuelve el valor de la distancia entre la velocidad máxima de avance, que ahi ya implica que empiece a disminuir la velocidad. El otro factor implica a la rotación, si está rotando a 1 rad/s hace que el robot no avance, ya que devuelve un 0, y si la velocidad de rotación es de 0 rad/s hace que el factor sea 1, y no bloquea el avance. Para este último factor lo calculamos así: return exp((-beta*beta)/s) y dónde s es: static float s = pow(0.5,2)/log(0.1).

### E. Funcionamiento

Para el correcto funcionamiento, primero cargamos los datos del láser y con el que dibujamos el póligono y obtenemos el estado del robot. Después comprobamos si nuestro booleano está activo o no, si esta activo es que ha detectado un click en el mapa y entonces es cuando calculamos el punto. Después comprobamos a que distancia esta del punto, si ya está en él, hacemos que se pare: differentialrobotproxy-¿setSpeedBase(0, 0) y ponga a false la detección del click:

t1.activo = false. Y si no está en el punto, lo que hacemos es que proceda a moverse, y primero hacemos que rote con la beta calculada: differentialrobotproxy-¿setSpeedBase(0, beta) si no está ya en la orientación que tiene que desplazarse. Y en el caso de que ya esté en la orientación, hacemos que avance: differentialrobotproxy-¿setSpeedBase(adv, 0) pero teniendo en cuenta que no puede avanzar si esta rotando y que vaya reduciendo si se va acercando, mediante los cálculo que hemos hecho en el último paso: float adv = MAXADVANCE * stopifturning(beta) * stopifAttarget(dist);