Universidad Politécnica de Madrid

Robótica

MODELADO Y LOCALIZACIÓN

Potential Field Histogram y *C-Space*

Autores:

Javier Alonso Silva - javier.asilva@alumnos.upm.es

Roberto Álvarez Garrido - roberto.alvarezg@alumnos.upm.es

José Alejandro Moya Blanco - alejandro.moya.blanco@alumnos.upm.es

Última modificación: 23 de diciembre de 2019

Resumen

Hoy en día, la detección del entorno en el que se mueve un robot móvil es fundamental para poder generar trayectorias sobre el mapa modelado y así llevar a cabo tareas específicas.

Para este proyecto se ha generado un modelo del entorno en un mapa en tiempo real usando el entorno de simulación V-Rep, con el robot *Pioneer–P3DX*. Usando los diferentes sensores de los que dispone dicho robot, se han calculado unos valores de certeza ("*certainity values*", CV) los cuales muestran, bajo una probabilidad, los obstáculos que pudieran existir en dichas coordenadas. Además, dichos datos se han obtenido usando una función de probabilidad variante que se adapta a las condiciones del entorno, teniendo en cuenta aquellos datos que ya se han registrado.

Esto permitiría después generar trayectorias utilizando el concepto de "campo de potencial", donde se aplican conceptos de fuerzas virtuales que permiten al robot desplazarse por el mapa. De las diversas propuestas que se consideraron, a saber: *Quadtrees*, *Voronoy* y *C–Space*, se escogió la última por la posible aproximación que se podía hacer sobre el mundo generado en una matriz.

El uso de esta propuesta permite definir trayectorias seguras por las cuales el robot puede circular sin problemas. Para ello, se hace uso de las dimensiones del robot y se traducen en valores en celdas de la matriz representativa del mapa.

Finalmente, se obtiene así un modelado en tiempo real del entorno así como un algoritmo de descripción de trayectorias basado en el entorno *C–Space*.

Índice

Índice de figuras

1. Introducción

1.1. El problema de la evitación de obstáculos

A lo largo del tiempo, la evitación de obstáculos en robots móviles ha sido una tarea que ha sido ampliamente investigada y estudiada, buscando así que un vehículo robotizado sea capaz de moverse con la máxima velocidad posible, dentro de las diferentes situaciones de estudio, evitando los obstáculos y yendo al objetivo propuesto.

La cuestión principal es que el robot, si no es por medio de los sensores, no tiene capacidad de percepción del entorno, a diferencia de las personas. Diferentes estudios se han realizado comprobando cuáles son los sensores que mejor pueden capturar la información del entorno; en el caso particular de este proyecto, los ultrasonidos presentan los siguientes inconvenientes [1], [2]:

- Direccionalidad limitada en la detección de la posición parcial de la esquina de un obstáculo, dependiendo de la distancia del mismo y del ángulo entre el sensor y la superficie del obstáculo.
- Falsas lecturas producidas por otros sensores, cuyos ultrasonidos rebotan y son leídos por el sensor incorrecto ("*crosstalk*").
- Reflexiones especulares, las cuales ocurren cuando el ángulo formado por el frente de onda ("*wavefront*") y la superficie es demasiado grande, provocando que las ondas no se reciban y provoquen que directamente el obstáculo no sea detectado.

Pese a los problemas que presentan los ultrasonidos, se han desarrollado diversos métodos de aproximación que intentan realizar mapas del entorno para, posteriormente, poder describir trayectorias, los cuales vamos a introducir brevemente antes de comentar la aproximación realizada para el *Pioneer–P3DX*.

1.2. Distintas aproximaciones realizadas

1.2.1. Detección de bordes de los obstáculos

Una primera aproximación fue la detección de las esquinas de los obstáculos, haciendo que el robot girase entorno a ellas. Además, las líneas que conectan dos esquinas visibles se consideran que representan el contorno del objeto. Un problema que presentaba era que el robot permanecía delante de los obstáculos, tomando datos de los sensores, pero el mayor contratiempo fue que, debido a los problemas presentados en el punto anterior, muchas veces se detectaban esquinas en lugares completamente erróneos, lo cual provocaba que se creasen trayectorias muy ineficientes.

1.2.2. Matriz de probabilidades

Así pues, se pasó a utilizar una matriz que representa un modelo del mundo, este modelo fue especialmente diseñado para aprovechar sensores cuyos datos no son precisos. Dicha matriz es bidimensional donde cada celda contiene un valor de confidencialidad ("*Certainty Value*" – CV) que indica que existe un obstáculo en esa celda. Cada CV se actualiza mediante una función probabilística adaptada al tipo de sensor que se va a utilizar.

En el caso de los sensores de ultrasonidos, el campo de visión corresponde a una figura cónica, la cual obtiene la distancia con el objeto más cercano pero no la localización angular del mismo. Por esto mismo, la función probabilística C_x da mayor peso a las celdas que están en el centro respecto a las de la periferia.

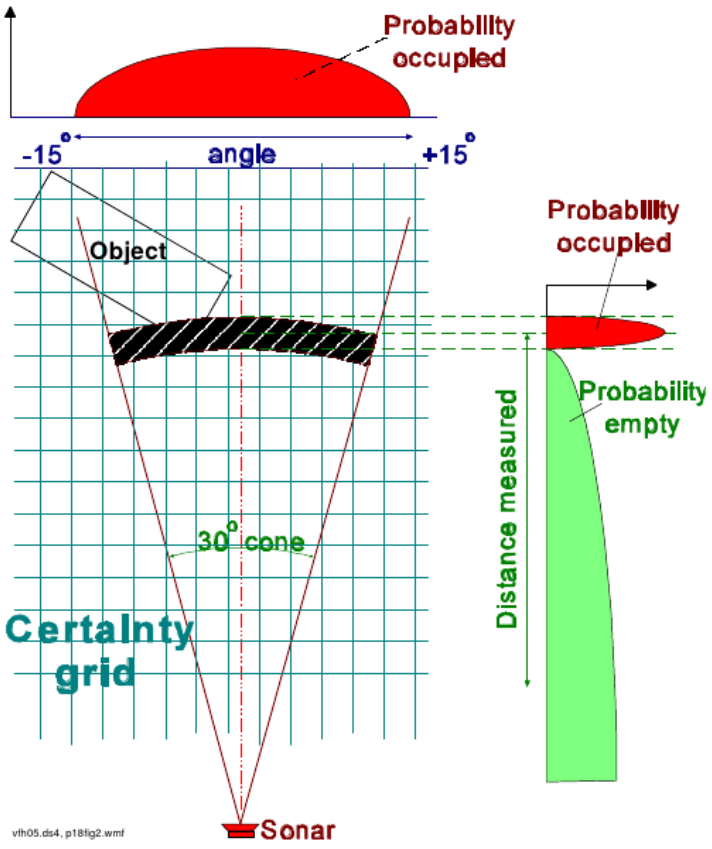

Figura 1: Visión cónica de un sensor de ultrasonidos y probabilidades [3]

De esta forma, el robot permanece quieto tomando datos con los sensores de los que dispone (en el caso del estudio, 24 [3]) y aplica la función probabilística C_x a cada uno de ellos, actualizando los valores de confidencialidad de cada celda.

1.2.3. Campo de potenciales

Este método propone el uso de fuerzas imaginarias [4]. Así, los obstáculos representan fuerzas repulsivas sobre el móvil y el objetivo una fuerza de atracción. De esta manera, se genera un vector resultante $\vec{R}$ el cual es relativo a la posición del robot, actuando como fuerza de aceleración sobre el móvil. Cuando cambia su posición, se vuelve a calcular el vector $\vec{R}$ y se repiten los pasos hasta que el robot llega a la posición deseada.

Algunas mejoras, como las propuestas por Krogh [5], tienen en cuenta la velocidad del robot a la hora de modificar las posibles fuerzas que actúan sobre el robot; de esta manera, la actuación era acorde a la distancia con los diferentes obstáculos y la velocidad del móvil.

El mayor problema de este método radica en que la magnitud de fuerzas que definen $\vec{R}$ pueden resultar negativas, provocando que el robot se detenga y se oriente según el sentido del vector, continuando su marcha por esa dirección cuando podría haber seguido con la trayectoria original.

1.2.4. *Virtual Force Field* – VFF

El método basado en campos de fuerza virtuales permite evitar obstáculos en tiempo real para robots móviles [6]. Este método define un histograma cartesiano bidimensional C el cual representa obstáculos. Al igual que en el método 1.2.2, cada celda $(i,\ j)$ del histograma contiene un valor $c_{i,j}$ que representa el nivel de confidencialidad para que haya un obstáculo en dicha celda.

La diferencia primordial del histograma cartesiano frente a la matriz de confidencialidad se encuentra en el modo de actualización. Mientras que la matriz de confidencialidad actualiza sus valores usando una función probabilística, una tarea intensiva en cómputo, el VFF usa una función de probabilidad propia la cual hace un uso mínimo de los recursos.

En el caso particular de los sensores de ultrasonidos, las celdas corresponden a la distancia medida d. Aunque pueda parecer una gran simplificación, el uso de la distribución de probabilidad aplicada de forma continua y rápida por cada sensor permite obtener un histograma de distribuciones de probabilidad bastante preciso.

Además, se aplica la idea de campos de potenciales, la explicada en el punto 1.2.3, permitiendo así que la información de los sensores pueda ser empleada para controlar eficientemente el vehículo.

Por cada instante en el que el robot se está moviendo, se observa una región de $w_s \times w_s$ celdas, correspondiendo a un espacio cuadrado del histograma C, llamada "región activa" (C^*), donde las celdas observadas son las "celdas activas" $(c^*_{i,j})$.

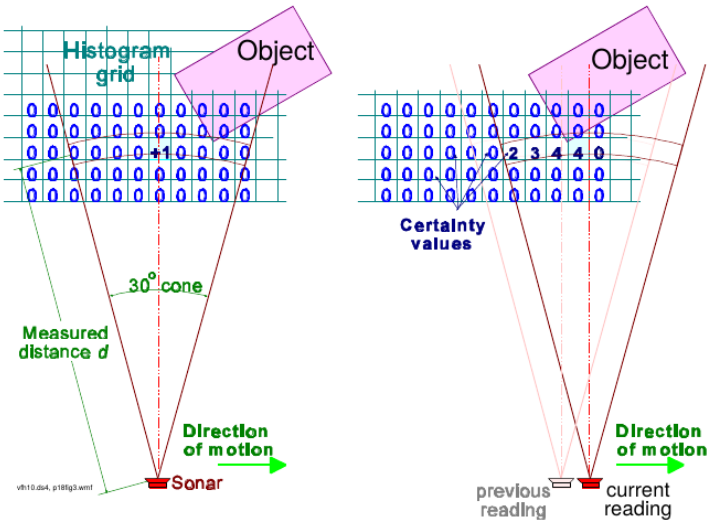

Figura 2: Representación de la región activa leída por el robot $w_s \times w_s$ [7]

Teniendo en cuenta esta perspectiva, cada celda $c^*_{i,j}$ genera una fuerza repulsiva $\vec{F}_{i,j}$ respecto al móvil, la cual es proporcional a la distancia d^x, donde $x \in \mathbb{R}_0^+$.

De esta manera, por cada iteración se genera una fuerza repulsiva $\vec{F}_r$ y, de forma simultánea, existe una fuerza constante $\vec{F}_t$ de atracción hacia el objetivo. Al igual que en el campo de potenciales (ver punto 1.2.3), se genera una fuerza resultante $\vec{R}$ que determinará el comportamiento del robot para esa iteración.

Mediante este método, es posible actualizar el histograma con los nuevos valores tan pronto como se han obtenido, permitiendo de esta forma reaccionar rápidamente a obstáculos que aparecen repentinamente.

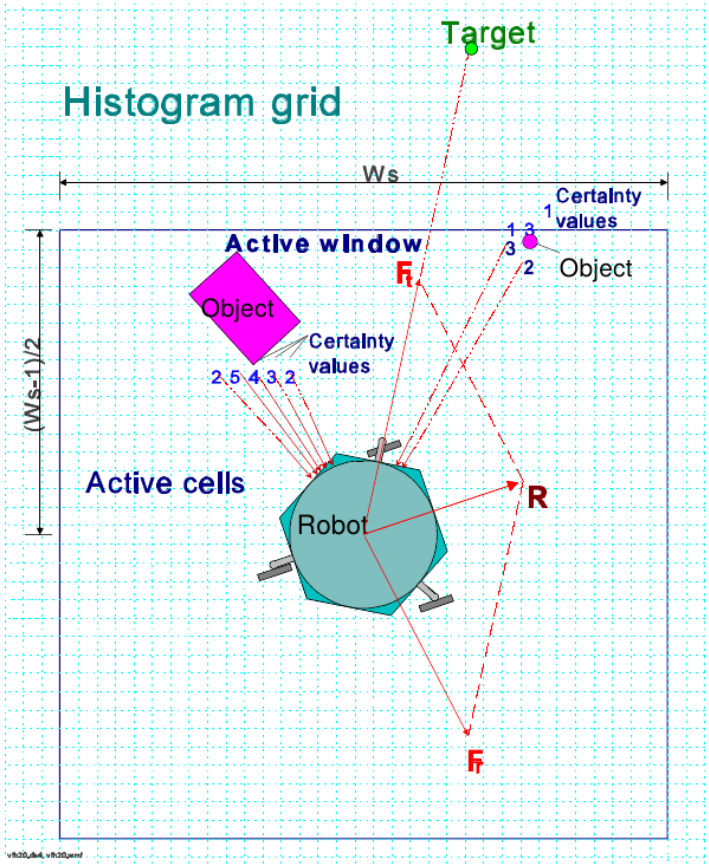

Figura 3: El método de VFF aplicado en una simulación [7]

Limitaciones del método VFF

Si bien el algoritmo trabaja especialmente bien en gran parte de condiciones, con una velocidad media de 0,5 $^{m}/_{s}$ y en situaciones donde los obstáculos estaban a una distancia de, al menos, un metro más respecto al tamaño del robot, ante otras como por ejemplo atravesar una puerta, debido a las fuerza final $\vec{R}$, donde las fuerzas repulsivas fueron mayores que la de atracción ($|\vec{F}_r| > |\vec{F}_t|$), el robot no pasaba pese a haber podido.

Por otra parte, la naturaleza discreta del histograma provoca cambios drásticos en la fuerza $\vec{R}$. Por ejemplo, se pudieron comprobar cambios drásticos en el módulo de la fuerza provocada por una celda en torno a un 42 % [7]. Una solución a este problema es el uso de filtros paso bajo, introduciendo así un mayor tiempo de reacción frente a obstáculos inesperados.

2. Representación del mapa

En esta sección se describen los métodos que se han utilizado para la representación del espacio cartesiano mediante una matriz, así como la creación del mapa de obstáculos.

El objetivo principal de este apartado, es la generación de un mapa de obstáculos que se pueda utilizar en la generación de trayectorias. Para conseguir este objetivo, se deben realizar varias acciones:

- Representar el espacio real en el que se encuentra el robot mediante coordenadas cartesianas.

- Representar dicho espacio cartesiano mediante una matriz cuadrada, es decir, asignar a cada punto cartesiano una celda de la matriz.
- Asignar a cada celda de la matriz un "*certainty value*" (CV), el cual proporciona información sobre si existe un obstáculo en dicho punto.

2.1. Representación de la escena mediante el sistema cartesiano

El espacio en el que se desplaza el robot se puede representar utilizando un sistema de coordenadas cartesianas de dos dimensiones, ya que dicho robot únicamente se desplaza por el suelo. Es por ello que la posición del robot se puede definir en todo momento utilizando un punto cartesiano de dos dimensiones: $(a,\ b) \in \mathbb{R}^2$.

En este caso, la escena por la que se desplaza el robot tiene una dimensión de 5×5 metros. Sin embargo, la zona útil de desplazamiento del robot es de 4×4 metros, dado que existe un muro delimitando la escena. Como origen de coordenadas del espacio cartesiano se toma el punto central de la escena.

Representando las distancias reales en el plano cartesiano se asigna a cada unidad del mismo una longitud de un metro exactamente. De esta forma, el punto $(1, 0)$ representa una distancia de un metro en el eje X, al igual que el punto $(0, 1)$ representa una distancia de un metro en el eje Y. De esta forma, se define la zona útil del movimiento del robot tomando como referencia el punto central de la escena $(0, 0)$ y las cuatro esquinas que delimita el muro, ubicadas en los puntos: $(-2, 2)$ $(2, 2)$ $(2, -2)$ y $(-2, -2)$ (esquinas superior izquierda, superior derecha, inferior derecha e inferior izquierda, respectivamente. Ver la figura 4):

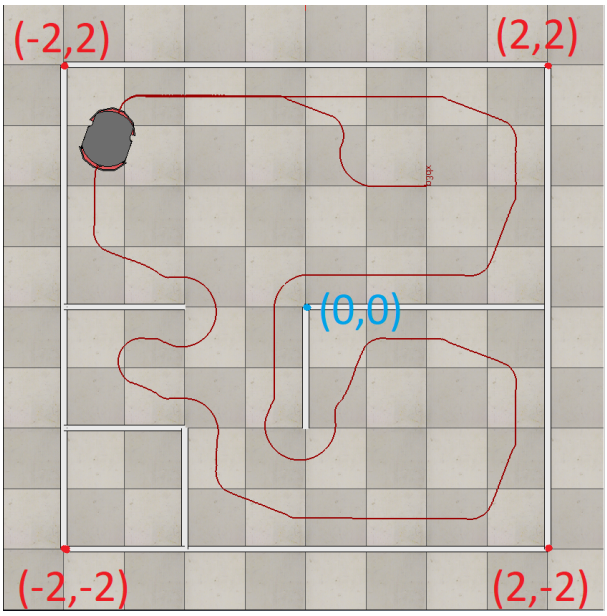

Figura 4: Escena proporcionada y recorrido

2.2. Representación del espacio cartesiano mediante una matriz

Una vez se ha definido la representación cartesiana de la escena, se debe transformar dicho espacio cartesiano en una matriz, siendo en particular para este modelo una matriz cuadrada. El objetivo de dicha transformación es generar una estructura de datos que

permita relacionar cada punto del espacio con una celda de la misma, la cual se utilizará para almacenar el "*certainty value*" asignado a dicho punto de la escena.

La dimensión de la matriz puede ser escogida libremente, dependiendo de la resolución que se quiera obtener al transformar un punto cartesiano con una celda de la matriz. De esta forma, cuanto más grande sea la matriz, mayor precisión se obtendrá al relacionar un punto cartesiano con su correspondiente celda de la matriz. Por ejemplo, para una matriz de 50×50 (2500 celdas), a cada celda de la matriz se le asignarán los puntos cartesianos situados en una superficie de 0,0064 m^2, mientras que utilizando una matriz de 200×200 (40000 celdas), a cada celda de la matriz se le asignarán los puntos cartesianos situados en una superficie de 0,0004 m^2. A pesar de que la precisión es notablemente mayor, el aumento del tamaño de la matriz se traduce en un coste computacional más alto, por lo que es necesario elegir de forma óptima esta dimensión para evitar un mal rendimiento en el mapeo. Además, tener mayor precisión no implica que el mundo vaya a ser representado más fielmente, ya que, como se explicó en el punto 1.1, los distintos problemas que presentan los sensores de ultrasonidos pueden provocar la aparición de puntos en el mapa que no son representativos, y que pueden dar lugar a confusión creando obstáculos no existentes.

La matriz que modela el espacio cartesiano de la escena no debe ser necesariamente cuadrada. Sin embargo, cuando lo es, se facilitan las operaciones matemáticas sobre la matriz y, además, la distribución de los puntos cartesianos presenta mayor uniformidad en ambos ejes.

Para la representación de los puntos cartesianos de la escena mediante una matriz se ha ideado una función matemática que toma como parámetro un punto cartesiano y devuelve su celda correspondiente en la matriz. En la siguiente ecuación se encuentran las variables y constantes siguientes:

- Punto cartesiano (x, y) que se quiere transformar.
- Celda de la matriz (i, j) resultado de la transformación.
- El punto (x_0, y_0) representa el punto cartesiano que se asocia a la celda $(i = 0, j = 0)$ de la matriz. En el caso de la escena que se utiliza en este proyecto, la esquina superior izquierda se asocia con la celda $(0, 0)$ de la matriz, por lo tanto (x_0, y_0) es el punto $(-2, 2)$.
- H_r y W_r son las constantes que definen las dimensiones en metros de la escena, largo y ancho respectivamente.
- H_v y W_v son las constantes que definen las dimensiones de la matriz que modela la el espacio cartesiano, longitud de filas y columnas respectivamente.

La función que transforma un punto cartesiano en su celda matricial equivalente es:

$$(i, j) = (x - x_0, -y + y_0) \cdot \left(\frac{W_v}{W_r}, \frac{H_v}{H_r} \right) \tag{1}$$

Un aspecto importante de esta función es que, al ser invertible, se la función correspondiente que calcula el punto cartesiano equivalente a partir de una celda de la matriz. Este proceso inverso es muy útil para generar puntos de una trayectoria a partir del mapa de obstáculos, es decir, a partir de la matriz que almacena los "*certainty values*".

La función que transforma una celda de la matriz a su punto cartesiano equivalente es:

$$(x, y) = (x_0, y_0) + \left(i \cdot \frac{W_r}{W_v}, -j \cdot \frac{H_r}{H_v} \right) \tag{2}$$

2.3. Cálculo del *certainty value* para cada punto del espacio

El motivo de relacionar cada punto cartesiano con su celda equivalente en la matriz es poder almacenar un valor que representa la certeza de que en dicho punto exista un obstáculo. Este valor se denomina "*certainty value*" (CV) y se calcula a partir de las lecturas de los sensores que realiza el robot al desplazarse por el mapa.

El objetivo de asignar un CV a cada punto del espacio es el de generar una matriz numérica que represente los obstáculos que se han detectado mediante los sensores durante la exploración del mapa.

En nuestro caso, el robot *Pioneer* realiza la exploración del mapa mediante el control reactivo creado en la practica anterior. Dado que dicho modelo funcionaba correctamente pero, sin embargo, no realizaba una exploración completa del mapa y quedaba atrapado en trayectorias repetitivas, se han incluido algunas modificaciones para que el robot se mantenga paralelo a los muros y siempre gire a la izquierda hasta encontrar otro muro. En caso detectar ninguno, el robot se desplaza por el contorno del mapa y realiza una vuelta completa a la escena.

Durante el reconocimiento, los sensores de ultrasonidos perciben obstáculos y suministran la distancia en linea recta a la que se encuentran. Empleando estas lecturas de los sensores y mediante cálculos trigonométricos, se puede obtener la posición espacial de los obstáculos detectados de forma aproximada. Para el cálculo de la posición cartesiana del obstáculo, se utiliza la distancia detectada por el ultrasonido d_{sensor_i}, el ángulo que forma dicho ultrasonido con el eje simétrico vertical del robot α_i y el punto central del mismo (x_r, y_r), que sirve como sistema de referencia local y cuya posición es proporcionada por la API:

$$(x, y)_{obst} = (x_r, y_r) + (d_{sensor_i} \cdot \cos \alpha_i \ , \ d_{sensor_i} \cdot \sin \alpha_i) \tag{3}$$

Tras esta primera aproximación, se observó que una mejor detección de los obstáculos podía ser realizada teniendo en cuenta además la orientación del robot respecto al eje de coordenadas del mapa (ver la figura 5).

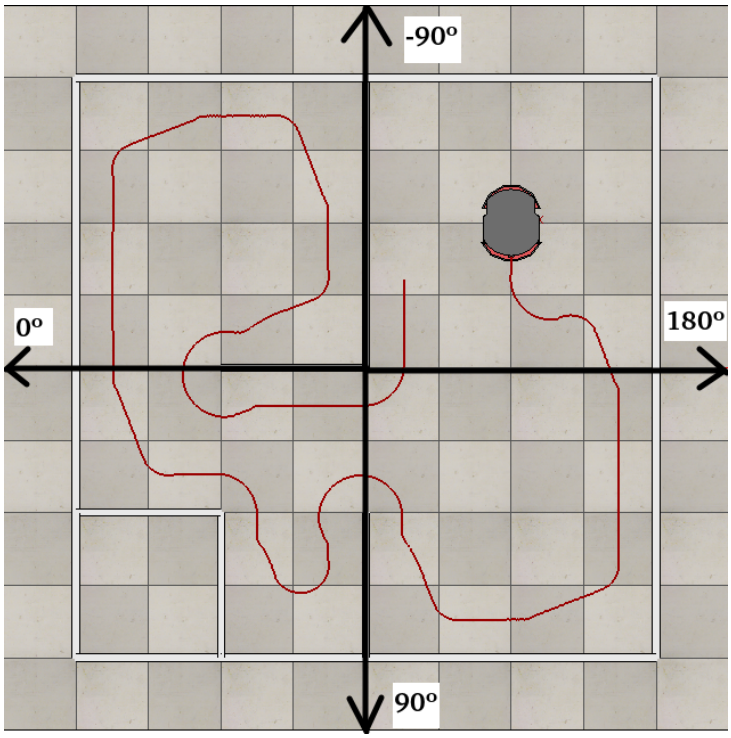

Figura 5: Orientación del mapa según el eje de coordenadas

Para ello, se tiene en cuenta además la orientación del robot θ a la hora de calcular las coordenadas (x, y) del obstáculo:

$$(x, y)_{obst} = (x_r, y_r) + (d_{sensor_i} \cdot \cos(\alpha_i + \theta), d_{sensor_i} \cdot \sin(\alpha_i + \theta)) \tag{4}$$

Una vez se ha calculado la posición espacial aproximada del obstáculo, se debe mapear dicha posición en la matriz que representa el mapa de obstáculos y se debe asignar un CV a la misma. En las sucesivas detecciones de cierto obstáculo, el CV de esa misma posición espacial no se sobrescribe por un nuevo valor sino que, por cada nueva detección del obstáculo, se suma más valor al CV de dicha posición. En consecuencia, las posiciones espaciales con un valor de CV más alto serán aquellas en las cuales se ha detectado un obstáculo en sucesivas ocasiones. Por el contrario, donde el CV sea más bajo o nulo, serán aquellas en las cuales no exista realmente un obstáculo o bien en las que haya existido un fallo de lectura por parte de los sensores.

Para el cálculo del CV por cada punto del espacio, se busca atenuar lo más posible las lecturas erróneas y anómalas. De este modo, el CV es inversamente proporcional a la distancia, dado que de esta forma se reduce el peso de las lecturas detectadas a distancias grandes, las cuales tienen mayor probabilidad de ser valores anómalos en donde el error producido es mayor. En la expresión del cálculo del CV se utilizan las variables d_{obst}, que representa la distancia a la que se encuentra el obstáculo y K, parámetro que se utiliza para regular la intensidad del CV, además de la constante d_{max}, que representa la distancia máxima de medición del ultrasonido:

$$CV_{i,j} = \frac{d_{max} - d_{obst}}{d_{max}} \cdot K \tag{5}$$

Tal y como se describe en la expresión anterior, el valor de CV para cada celda de la matriz puede tomar valor mínimo de 0, cuando la medición de obstáculo se realiza a distancia máxima, y valor máximo K, cuando la medición del obstáculo es muy cercana.

Es importante recalcar que, para mejorar al máximo la representación de los obstáculos a través del calculo del CV, se utiliza un valor umbral *threshold* que limita la asignación de CV en ciertas celdas de la matriz, para de esta forma reducir los valores anómalos:

- El valor umbral que se utiliza como *threshold* es variable y cambia, dependiendo de un ratio parametrizado a medida que existe un mayor número de celdas mapeadas.
- A medida que las celdas de la matriz se mapean con un CV, el *threshold* impide que las nuevas celdas con un valor CV muy bajo se escriban en la matriz, de esta forma, se evita el mapeado de los valores anómalos.
- El *threshold* aumenta cada vez que se detecta un obstáculo y disiminuye cuando se detectan numerosos valores anómalos, con esto se consigue que el *threshold* se auto - ajuste y finalmente se estabilice.

De forma particular, el valor de *threshold* se ha escogido de la siguiente manera:

$$th = th \cdot (1 \pm 0{,}1 \cdot r) \tag{6}$$

En el caso particular del proyecto, el valor inicial del *threshold* se ha establecido en $th = 0{,}4$, y el factor de crecimiento en $r = 1 \cdot 10^{-6}$. Pese a que el *ratio* (r) pueda parecer muy bajo, tras el período de generación del mapa el valor del *threshold* alcanzó el valor de $th = 0{,}6634$. Esto ha permitido que los datos que se han representado en el mapa han sido "seleccionados" en base a los anteriores, intentando así obtener una mejor precisión.

2.4. Elección del tamaño del mapa

A lo largo del desarrollo de este proyecto, se ha ido trabajando con diferentes tamaños de mapa, intentando conseguir una mejor aproximación y mejores resultados. Pero finalmente, debido a la configuración del espacio (*C–Space*), pensando en la planificación de trayectorias, se ha decidido usar un tamaño de *grid* de 26×26 (esta decisión se explica con detalle en el punto 3).

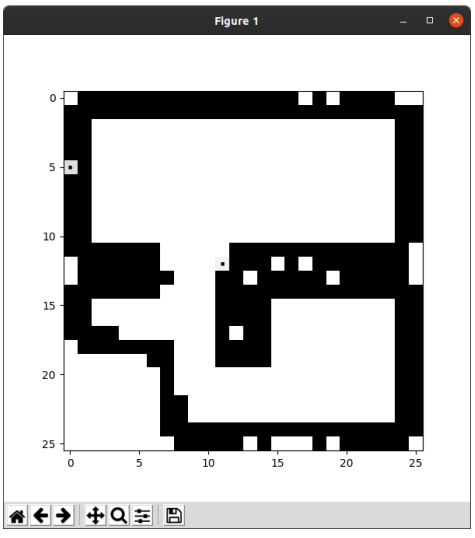

Figura 6: Mapa de tamaño 26×26

Previamente, se hizo uso de mapas de diferentes tamaños, a tener en cuenta: 50×50, 200×200, 500×500 y 1000×1000. El problema de estos tamaños de mapa alternativos se describe a continuación:

1. La falta de precisión de los sensores de ultrasonidos (explicado en el apartado 1.1) provoca lecturas erróneas, ya que mucha información producida por dichos sensores se pierde o es captada por otro sensor de la proximidad. Si bien es cierto que esto no es tan apreciable en los histogramas de 50×50 o mismamente el de 26×26, es muy notorio en los de 500×500 y 1000×1000 (ver la figura 7).

2. La carga computacional se incrementa sobremanera según crece el tamaño del mapa, obligando a dedicar mayor tiempo de cómputo en el proceso de los sensores y de los valores de la matriz. Si bien es cierto que se ha trabajado para intentar que el código se ejecute de manera óptima (esto se comenta con mayor detalle en el punto 4), la matriz de 500×500 elementos ya presentaba 250 000 elementos, y el histograma de 1000×1000 contenía un millón de valores. Pese a que los equipos de los que disponemos admiten dicha carga computacional, no nos pareció suficientemente beneficioso como para tenerlo en cuenta.

3. Finalmente, el incremento del tamaño de la matriz se traduce directamente en un incremento del tamaño de los cálculos y posibilidades que hay que hacer para el cálculo de trayectorias. Si bien es cierto que en los primeros tamaños de *grid* la trayectoria se calculaba sin demasiados problemas, al trabajar con matrices de más de 100×100 elementos el cómputo de estas tiende a infinito, debido a las diferentes opciones que puede tomar un robot para ir a un camino son muy amplias (en particular, el cálculo de una trayectoria simple en un mapa de 500×500 tomó más de una hora).

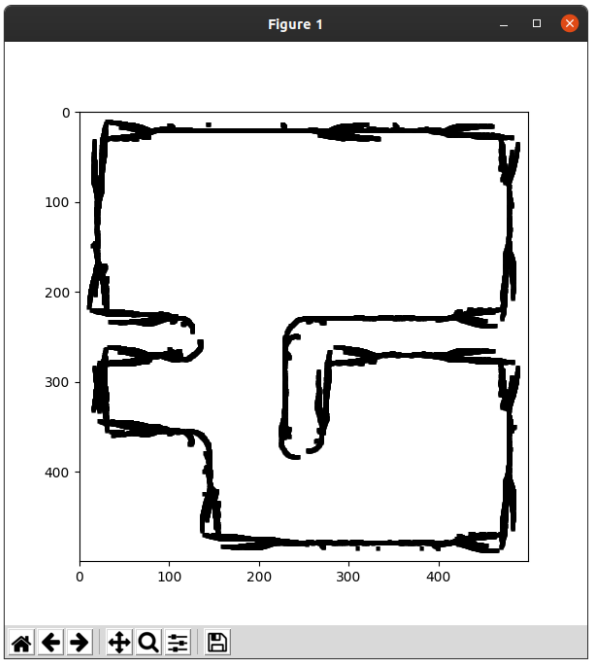

Figura 7: Mapa de tamaño 500×500. Se puede apreciar cómo hay una gran cantidad de valores anómalos producidos por las limitaciones de los sensores de ultrasonidos

Por los motivos descritos anteriormente junto con los explicados en el punto 3, se tomó la decisión de que el mapa fuera de tamaño 26×26, ya que muestra fielmente los obstáculos presentes en la escena actual así como los huecos por los que puede pasar el robot.

3. Aplicación de *C-Space* en el mapa

El método *C-Space* se utiliza en la planificación de trayectorias tanto de manipuladores como robots móviles.

El término *C-Space* significa *Configuration Space* y hace referencia al espacio en el cual el robot realiza sus movimientos. En este espacio, se suelen representar tanto los obstáculos que dificultan el movimiento del robot, así como los espacios libres por los que el mismo puede desplazarse sin riesgo. En este espacio de configuración, además de representarse los obstáculos y espacios libres, se pueden representar otros parámetros de interés que son de utilidad en la generación de trayectorias.

Existen numerosos enfoques del método *C-Space*,sin embargo, sus principales ámbitos de aplicación son los siguientes:

- El método *C-Space* suele aplicarse para analizar el espacio de trayectorias que se pueden realizar con un manipulador, en este caso, el espacio de configuración esta formado por una representación N-dimensional en la cual se representan las variables asignadas al control de los grados de libertad del mismo (ángulos de giro, movimientos prismáticos,etc)

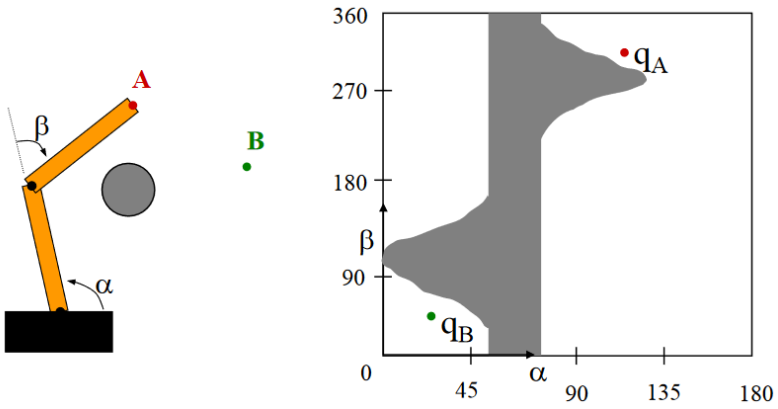

Figura 8: Manipulador usando C-Space para describir trayectorias

- El método *C-Space* también puede aplicarse en robótica móvil, en este caso, el espacio de configuración representa el mapa de obstáculos por el que se desplaza el robot. En esta aplicación de *C-Space*, el espacio de configuración suele ser un mapa de obstáculos, en el cual, dichos obstáculos están ampliados de tal manera que el robot móvil pueda ser considerado un punto unidimensional.

En este caso, el método *C-Space* se aplica sobre una situación del robótica móvil y en consecuencia, el objetivo de su uso será el de crear un espacio de configuración en el cual se amplíen los obstáculos a partir de las medidas del robots y con lo cual, se simplificará el proceso de planificación de trayectorias.

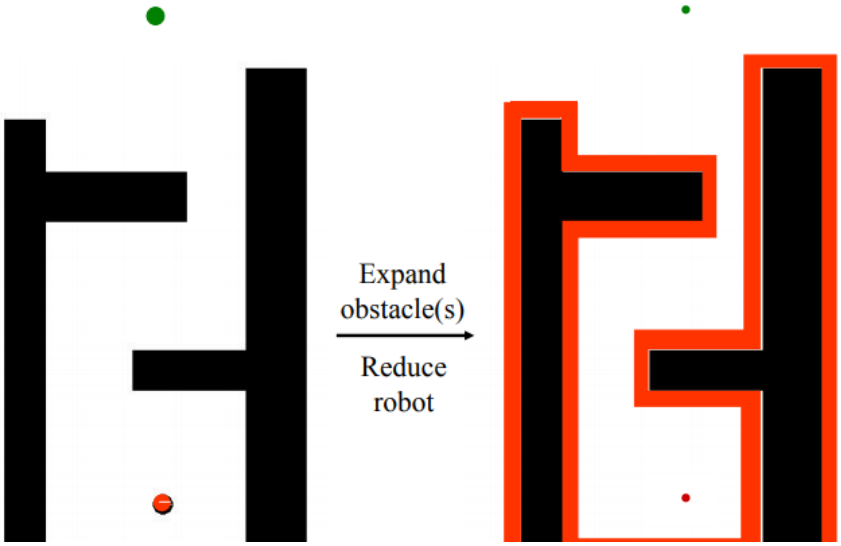

Figura 9: Planificación usando C-Space para describir trayectorias en robótica móvil

Cabe destacar que, dado que el diseño de trayectorias se realiza en este caso utilizando el concepto de 1.2.3, es decir el campo de potenciales, los valores asignados a este método también se incluyen dentro del espacio de configuración.

Para el caso particular que se presenta en este proyecto, se ha conseguido mapear un mapa con obstáculos de forma exitosa a partir de una exploración del robot. En un primer momento, este mapa de obstáculos se mapeaba con una matriz de gran tamaño y eso, acarreaba gran precisión y resolución, sin embargo, su uso a la hora de generar trayectorias no era exitosa, dado que, era difícil definir los lugares por los cuales el robot podía circular sin riesgos.

Mediante el uso de *C-Space*, se busca facilitar el diseño de trayectorias, dado que, al ampliar el tamaño real de los obstáculos en función del tamaño del robot, se genera una zona segura de desplazamiento alrededor de los mismos. El concepto esencial es que, al amplificar el tamaño real de los obstáculos, y dado que este margen añadido es de un tamaño que depende del las dimensiones del robot, los mismos pueden ser usados como referencia a la hora de generar trayectorias, debido a que, en el espacio de configuración creado, los bordes de los obstáculos realmente no coinciden con un obstáculo en la realidad, sino con, trayectorias seguras entorno a los mismos.

Al aplicar *C-Space*, se genera un espacio de configuración basado en el mapa de obstáculos reales, pero adaptado para así facilitar la creación de trayectorias.

En particular y entrando en detalles técnicos, dado que el robot *Piooner* siempre se desplaza de forma paralela a las paredes y su ancho tiene una longitud de $38cm$, se desea agregar un margen de $19cm$ alrededor de los obstáculos para que su desplazamiento entorno a los obstáculos sea completamente seguro. Este margen añadido es de exactamente la mitad de su ancho, dado que se busca que el robot se ajuste lo máximo posible a las paredes.´

Para aplicar *C-Space* en este caso, se necesita agregar un margen de $19cm$ alrededor de los obstáculos, en consecuencia, se debe tratar la matriz que representa el mapa de obstáculos para conseguir este espacio de configuración.

Durante el desarrollo del proyecto, se han pensado en varias opciones para realizar este tratamiento:

- Como primera opción, generar una matriz mapa de obstáculos y tratarla de tal forma que se amplíen los obstáculos detectados mediante el análisis y modificación de las celdas colindantes a ellos. Este método aporta gran resolución en el mapa, pero su complejidad es alta.

- Como segunda opción, se ha ideado generar un mapa de obstáculos en el cual estos se representen directamente con la dimensión que busca, es decir con una anchura de 19 cm. Esto se puede conseguir ajustando exactamente la dimensión de la matriz del mapa de obstáculos en función de la dimensión real de la escena que se modela. La dimensión N de la matriz cuadrada que verifica que el tamaño de obstáculo S_{obst} requerido por *C-Space* en función de la dimensión de la escena S_r se calcula mediante la expresión:

$$N = \frac{S_{obst}}{S_r} \tag{7}$$

Finalmente se ha decidido optar por la segunda opción, ya que es más sencilla y útil a la hora de crear trayectorias, dado que la información sobre obstáculos se concentra en menos celdas. Aplicando la expresión anterior, se obtiene una dimensión de la matriz cuadrada de 26x26, la cual garantiza que cada obstáculo se representa con un grosor de 19cm (ancho y largo de cada celda de la matriz).

En conclusión, los obstáculos que se representan en el mapa utilizando una dimensión 26x26 realmente ocupan un grosor de 19cm, y esto garantiza que el robot se pueda mover por el borde de los mismos de forma segura, dado que existe un espacio de seguridad hasta colisionar, se consigue así una referencia segura en cada obstáculo que será muy útil a la hora de generar trayectorias.

3.1. Generación de trayectorias en una configuración de *C–Space*

Una vez se ha generado completamente el mapa con el tamaño buscado, se puede entonces planificar trayectorias para que el móvil se mueva con total seguridad por el mapa. Usando el *grid* de 26×26, se puede definir un punto inicial P_0 y el objetivo P_t y utilizando un algoritmo iterativo se computan las celdas adyacentes comprobando el potencial de las mismas para llegar al punto deseado.

Previamente, por cada celda se ha computado, usando la distancia al punto P_t, un campo de potencial el cual pretende orientar al robot en el desarrollo de una trayectoria hasta el objetivo. De esta manera, el campo de potencial $V_{i,j}$ se calcula como:

$$V_{i,j} = \frac{1}{\sqrt{(P_{i_t} - i)^2 + (P_{j_t} - j)^2}} \tag{8}$$

Este campo de potencial representa una relación ascendente en donde en el punto más lejano su potencial tiende a cero y el objetivo tiene un potencial infinito ($V_{P_t} \to \infty$). De esta manera, se puede así llegar al punto objetivo yendo de menor a mayor potencial (cuando sea posible) hasta llegar al punto P_t objetivo y llevando la cuenta de en qué celdas el robot ya ha circulado, evitando así posibles ciclos en la trayectoria.

La planificación de las mismas se aborda directamente buscando el camino hasta el objetivo y generando una lista con los puntos que el robot tendrá que recorrer (ver la figura 10).

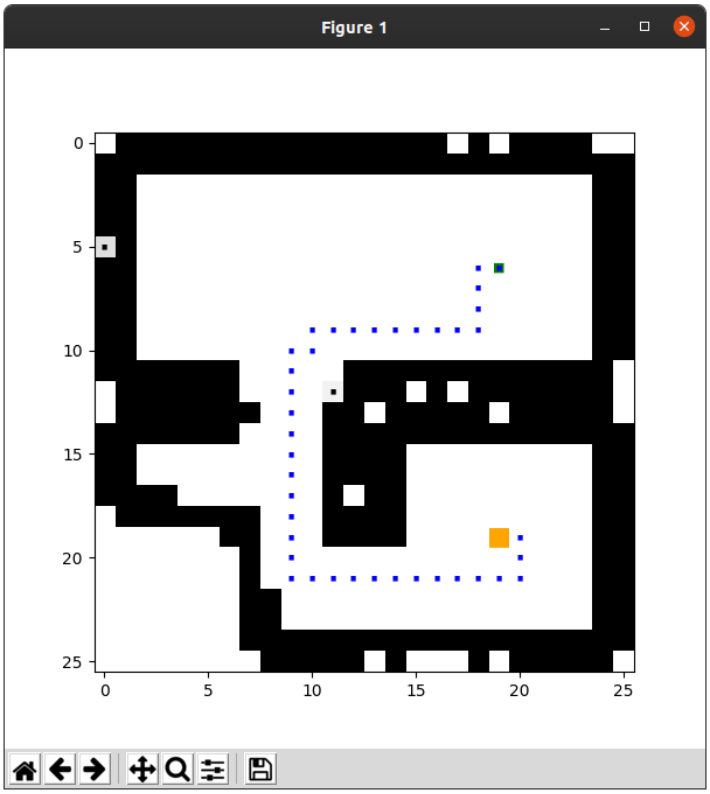

Figura 10: Trayectoria desde el punto $(-1, -1)$ hasta el punto $(1, -1)$. El punto verde representa el inicio y el azul el objetivo.

Como se explicó anteriormente en el punto 2.4, hay otros tamaños de mapa que permiten igualmente definir trayectorias (ver la figura 11), pero el tiempo de cálculo de las mismas era excesivamente grande para el resultado obtenido finalmente.

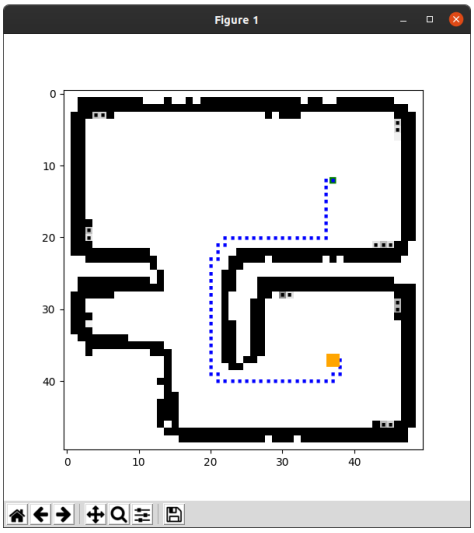

Figura 11: Trayectoria generada en un mapa de tamaño 50×50

Por otro lado, el tamaño usado en el mapa no afecta en demasía a la trayectoria descrita, añadiendo mayor redundancia e incluso desinformación, frente a lo ya obtenido en una matriz de tamaño 26×26.

4. Resultados, conclusiones y futuras mejoras

En esta sección, se muestran los resultados generales del proyecto y se realiza una revisión del mismo con vistas a plantear posibles futuras mejoras y conclusiones.

En primer lugar, los resultados generales de la práctica son los siguientes:

- Comprender extensivamente como modelar un mapa de obstáculos usando matrices.
- Obtención de las funciones que relacionan el plano cartesiano y el matricial.
- Aplicar el concepto de *certainty value* en la representación de obstáculos mediante matrices.
- Construcción de un algoritmo de exploración del mapa basado en control reactivo y conseguir mapear correctamente el mapa de obstáculos.
- Representar gráficamente el mapa de obstáculos de forma fiable.
- Construcción de un generador de trayectorias utilizando un mapa matricial de obstáculos y el concepto de campo de potencial.
- Desarrollo de un código Python estructurado que implementa las funcionalidades anteriormente descritas.

Dentro de los objetivos de la práctica, se pretendía implementar un controlador del robot orientado a que el mismo llevase a cabo trayectorias de forma física. La lógica de este controlador se ha implementado, pero debido a la complejidad del proyecto y a la falta de tiempo, no se ha conseguido un funcionamiento completamente correcto y funcional, por lo cual esta funcionalidad se plantea como futura mejora.

Con respecto a otras mejoras futuras, se plantea el refinamiento de la aplicación de *C-Space*, así como la mejora y optimización del planificador de trayectorias.

En general el desempeño del equipo durante el proyecto ha sido bueno y consideramos que los resultados han sido satisfactorios.

4.1. Optimización del código Python

Como se ha descrito anteriormente en los distintos apartados, el tiempo de cómputo de los distintos apartados es bastante elevado dada la magnitud de las operaciones. Para abordar esta problemática, se ha decidido hacer el código de la aplicación en conjunción de Python y Cython, un módulo del lenguaje de programación que permite definir funciones del lenguaje C desde la interfaz Python y permitir la comunicación entre ellos.

Esto ha permitido una gran mejora en los tiempos de cómputo así como en el consumo de memoria. Estamos hablando de una reducción de más de un 20 % en uso de memoria RAM y en un uso masivo del paralelismo para realizar operaciones sobre *arrays* y permitir así una gran mejora en los cálculos de los diferentes valores que se han ido necesitando. Esto se ha conseguido, por ejemplo, usando estructuras de datos colindantes entre sí, permitiendo un acceso más eficaz a los datos contenidos, así como los tipos de datos nativos de la máquina en uso. Se detalla en el apéndice cómo poder compilar el código escrito en Cython y usarlo de forma nativa en la máquina de cada persona.

Como una futura mejora, se propone intentar escribir en Cython las partes del código restantes intentando reducir la carga de trabajo destinada a Python lo máximo posible, para obtener mayores ganancias en rendimiento y memoria.

Se puede ver en el siguiente vídeo cómo funciona el algoritmo de detección en tiempo real del mapa: https://youtu.be/Iit4AHccAAc.

Referencias

[1] A. Elfes, «Sonar-based real-world mapping and navigation», *IEEE Journal on Robotics and Automation*, vol. 3, n.º 3, págs. 249-265, jun. de 1987, ISSN: 2374-8710. DOI: 10.1109/JRA.1987.1087096.

[2] J. Borenstein e Y. Koren, «Obstacle avoidance with ultrasonic sensors», *IEEE Journal on Robotics and Automation*, vol. 4, n.º 2, págs. 213-218, abr. de 1988, ISSN: 2374-8710. DOI: 10.1109/56.2085.

[3] H. Moravec y A. Elfes, «High resolution maps from wide angle sonar», en *1985 IEEE International Conference on Robotics and Automation Proceedings*, ISSN: null, vol. 2, mar. de 1985, págs. 116-121. DOI: 10.1109/ROBOT.1985.1087316.

[4] O. Khatib, «Real-time obstacle avoidance for manipulators and mobile robots», en *1985 IEEE International Conference on Robotics and Automation Proceedings*, ISSN: null, vol. 2, mar. de 1985, págs. 500-505. DOI: 10.1109/ROBOT.1985.1087247.

[5] B. Krogh y C. Thorpe, «Integrated path planning and dynamic steering control for autonomous vehicles», en *1986 IEEE International Conference on Robotics and Automation Proceedings*, ISSN: null, vol. 3, abr. de 1986, págs. 1664-1669. DOI: 10.1109/ROBOT.1986.1087444.

[6] J. Borenstein e Y. Koren, «Real-time obstacle avoidance for fast mobile robots», *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, n.º 5, págs. 1179-1187, sep. de 1989, ISSN: 2168-2909. DOI: 10.1109/21.44033.

[7] ——, «The vector field histogram-fast obstacle avoidance for mobile robots», *IEEE Transactions on Robotics and Automation*, vol. 7, n.º 3, págs. 278-288, jun. de 1991, ISSN: 2374-958X. DOI: 10.1109/70.88137.

A. Compilación del código

A.1. Instalación de Python 3.8.0

Debido a los mecanismos de sincronización, la versión que se ha decidido usar para este proyecto es la de *Python 3.8.0*, ya que permite intercambiar fácilmente los datos entre procesos.

Para ello, se puede usar *pyenv*, un *software* que permite tener instaladas varias versiones de Python a la vez sin que interfieran unas con otras. Se detallan las instrucciones de instalación en su repositorio: https://github.com/pyenv/pyenv

A.2. Instalación de las librerías necesitadas

Una vez se haya instalado *pyenv* y se compruebe que en efecto es la versión de Python que se está usando, se podrán instalar las librerías necesarias:

- Numpy
- Matplotlib
- Cython

Estas librerías pueden ser instaladas usando el comando `pip`. Además, es necesario instalar las librerías de compilación de Numpy. En sistemas DEB, se puede hacer con `sudo apt install python3-numpy`, pudiendo así compilar el código fuente.

A.3. Compilación

Para compilar el código, basta con clonar el fuente del repositorio y ejecutar los siguientes comandos:

- `git clone https://github.com/UPM-Robotics/Pioneer-OA.git`
- `cd Pioneer-OA/src/PioneerOA`
- `python setup.py build_ext -inplace`

Estos comandos deberían permitir la compilación del código para la máquina local. Igualmente, por comodidad, en el repositorio se proveen los archivos `.so` para usarlo directamente.

Es importante además modificar el archivo de cliente para, en las cabeceras, incluir la localización de la versión de Python 3.8 instalado con *pyenv*. Esto se puede hacer con el comando `which python`, teniendo la versión de *pyenv* activada.

El repositorio está disponible aquí: https://github.com/UPM-Robotics/Pioneer-OA

B. Código Python

```python
#!/home/javinator9889/.pyenv/shims/python
# /usr/bin/python3

print('### Script:', __file__)

import os
import sys
import math

import vrep

import pickle

import numpy as np
from time import sleep
import matplotlib.pyplot as plt

from mapper import Mapper
from motion import Motion
from planner import Planner
from map_wrapper import Wrapper
from map_printer import start_printing


# from .robot_control import Sensor

# ------------------------------------------------------------------------

def getRobotHandles(clientID):
    # Robot handle
    _, rbh = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx',
                                      vrep.simx_opmode_blocking)

    # Motor handles
    _, lmh = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx_leftMotor',
                                      vrep.simx_opmode_blocking)
    _, rmh = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx_rightMotor',
                                      vrep.simx_opmode_blocking)

    # Sonar handles
    str = 'Pioneer_p3dx_ultrasonicSensor%d'
    sonar = [0] * 16
    for i in range(16):
        _, h = vrep.simxGetObjectHandle(clientID, str % (i + 1),
                                        vrep.simx_opmode_blocking)
        sonar[i] = h
        vrep.simxReadProximitySensor(clientID, h, vrep.simx_opmode_streaming)

    return [lmh, rmh], sonar, rbh


# ------------------------------------------------------------------------

def setSpeed(clientID, hRobot, lspeed, rspeed):
    vrep.simxSetJointTargetVelocity(clientID, hRobot[0][0], lspeed,
```

```
                                    vrep.simx_opmode_oneshot)
    vrep.simxSetJointTargetVelocity(clientID, hRobot[0][1], rspeed,
                                    vrep.simx_opmode_oneshot)


# ------------------------------------------------------------------------

def getSonar(clientID, hRobot):
    r = [1.0] * 16
    for i in range(16):
        handle = hRobot[1][i]
        e, s, p, _, _ = vrep.simxReadProximitySensor(clientID, handle,
                                                     vrep.simx_opmode_buffer)
        if e == vrep.simx_return_ok and s:
            r[i] = math.sqrt(p[0] * p[0] + p[1] * p[1] + p[2] * p[2])

    return r


# ------------------------------------------------------------------------

def getRobotPosition(clientID, hRobot):
    _, rpos = vrep.simxGetObjectPosition(clientID, hRobot[2], -1,
                                         vrep.simx_opmode_streaming)
    return rpos[0:2]


# ------------------------------------------------------------------------

def getRobotHeading(clientID, hRobot):
    _, reul = vrep.simxGetObjectOrientation(clientID, hRobot[2], -1,
                                            vrep.simx_opmode_streaming)
    return reul[2]


# ------------------------------------------------------------------------
# ------------------------------------------------------------------------
clr = "
     "


def avoid(robot):
    if not robot.is_any_obstacle_front():
        # print(clr, end="\r")
        # print(" >>> Looking for wall", end="\r")
        if any(x < 10 for x in robot.sensors.parallel_left):
            # print(clr, end="\r")
            # print(" ! Obstacle parallel to the left side", end="\r")
            if robot.sensors.parallel_left[0] < robot.sensors.parallel_left[1]:
                lspeed, rspeed = 0.25, 0
            elif robot.sensors.parallel_left[0] == robot.sensors.parallel_left[
                1]:
                lspeed, rspeed = 1, 1
            else:
                lspeed, rspeed = 0.25, 1

        elif any(x < 10 for x in robot.sensors.parallel_right):
            # print(clr, end="\r")
```

```
113             # print(" ! Obstacle parallel to the right side", end="\r")
114             if robot.sensors.parallel_right[0] < \
115                     robot.sensors.parallel_right[1]:
116                 lspeed, rspeed = 0, 0.5
117 
118             else:
119                 lspeed, rspeed = 1, 1
120         else:
121             lspeed, rspeed = 0.25, 1
122 
123     else:
124         if robot.is_any_obstacle_right():
125             # print(clr, end="\r")
126             # print("! Obstacle right", end="\r")
127             if robot.is_any_obstacle_left():
128                 # print(clr, end="\r")
129                 # print("! Obstacle left", end="\r")
130                 lspeed, rspeed = 10, -10
131             else:
132                 lspeed, rspeed = 0.1, 1
133         elif robot.is_any_obstacle_left():
134             # print(clr, end="\r")
135             # print("! Obstacle left", end="\r")
136             if robot.is_any_obstacle_right():
137                 lspeed, rspeed = -10, 10
138             else:
139                 lspeed, rspeed = 1, 0.1
140         else:
141             # print(clr, end="\r")
142             # print(" >>> Looking for wall", end="\r")
143             lspeed, rspeed = 2, 2
144 
145     return lspeed, rspeed
146 
147 
148 # --------------------------------------------------------------------------
149 
150 def main():
151     print('### Program started')
152 
153     print('### Number of arguments:', len(sys.argv), 'arguments.')
154     print('### Argument List:', str(sys.argv))
155 
156     vrep.simxFinish(-1)  # just in case, close all opened connections
157 
158     port = int(sys.argv[1])
159     clientID = vrep.simxStart('127.0.0.1', port, True, True, 2000, 5)
160 
161     if clientID == -1:
162         print('### Failed connecting to remote API server')
163 
164     else:
165         print('### Connected to remote API server')
166         hRobot = getRobotHandles(clientID)
167         # Recover later generated data
168         if os.path.exists("map.cls"):
169             try:
170                 with open("map.cls", "rb") as created_map:
```

```
                map_wrapper = pickle.load(created_map)
        except Exception as _:
            robot = Mapper(X0=-2,
                           Y0=2,
                           map_width=4,
                           map_height=4,
                           grid_size=(26, 26),
                           k=1.0,
                           initial_threshold=0.4,
                           max_read_distance=0.5,
                           ratio=1E-6)
        else:
            robot = map_wrapper.restore()
    else:
        robot = Mapper(X0=-2,
                       Y0=2,
                       map_width=4,
                       map_height=4,
                       grid_size=(26, 26),
                       k=1.0,
                       initial_threshold=0.4,
                       max_read_distance=0.5,
                       ratio=1E-6)
    sensors = robot.sensors
    printer = start_printing(robot.lock)

    while vrep.simxGetConnectionId(clientID) != -1:
        # Perception
        sonar = getSonar(clientID, hRobot)
        x, y = getRobotPosition(clientID, hRobot)

        heading = getRobotHeading(clientID, hRobot)
        sensors.set_sonar(sonar)

        # Planning
        lspeed, rspeed = avoid(robot)

        # Action
        setSpeed(clientID, hRobot, lspeed, rspeed)
        # Update robot position and map location
        robot.update_robot_position(x, y,
                                    np.asarray(sonar, dtype=np.float_),
                                    heading)
        print(clr, end="\r")
        print(f"Threshold: {robot.threshold}", end="\r")
        # time.sleep(0.1)

    print('### Finishing...')
    vrep.simxFinish(clientID)

    # As we are using Cython, use a wrapper for saving it into a file
    wrapper = Wrapper(robot)
    with open("map.cls", "wb") as file_map:
        pickle.dump(wrapper, file_map, protocol=pickle.HIGHEST_PROTOCOL)

    # Close the process we have just created for printing in real-time
    printer.terminate()
    printer.join()
```

```
        printer.close()

        figure = plt.figure(figsize=(6, 6))
        axis = figure.add_subplot(111)
        image = axis.imshow(np.random.randint(0, 10, size=robot.grid.shape),
                            cmap="gray_r")

        annotations_grid = np.zeros(robot.grid.shape)

        plt.show(block=False)
        image.set_data(robot.grid)
        for index in np.ndindex(robot.grid.shape):
            if robot.threshold < robot.grid[index] != annotations_grid[index]:
                axis.annotate('',
                              xy=(index[1], index[0]),
                              horizontalalignment="center",
                              verticalalignment="center",
                              color="black",
                              size=4)
            annotations_grid[index] = robot.grid[index]
        figure.canvas.draw_idle()
        plan = Planner(robot)
        path = plan.calculate_path((-1, -1), (1, -1))
        axis.annotate(' ',
                      xy=(path[0][1], path[0][0]),
                      horizontalalignment="center",
                      verticalalignment="center",
                      color="green",
                      size=8)
        for index in plan.calculate_path((-1, -1), (1, -1)):
            axis.annotate('',
                          xy=(index[1], index[0]),
                          horizontalalignment="center",
                          verticalalignment="center",
                          color="blue",
                          size=4)
            figure.canvas.draw_idle()
            plt.pause(0.3)
        axis.annotate(' ',
                      xy=(path[len(path) - 1][1], path[len(path) - 1][0]),
                      horizontalalignment="center",
                      verticalalignment="center",
                      color="orange",
                      size=16)
        plt.pause(0.01)
        plt.show()

        del robot

    print('### Program ended')


# ------------------------------------------------------------------------

if __name__ == '__main__':
    main()
```

Listing 1: client-pr2.py

```
#                            src
#                  Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#               GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.
import os

import numpy as np

import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

from multiprocessing import Lock
from multiprocessing import Process
from multiprocessing import dummy as mp
from multiprocessing import shared_memory


def start_printing(robot_lock: Lock) -> Process:
    """
    Process launcher that executes a new process which is reading data
    from a shared memory location, so it is not necessary to share it
    directly from the owner process.
    :param robot_lock: the lock mechanism for avoiding race conditions.
    :return: the created process.
    """

    def _print_map(lock: Lock):
        """
        Internal function that prints the map into a matplotlib window
        until it is closed.
        It is specially designed to update all matrix elements as fast
        as possible by using multithreading mechanisms.
        :param lock: the synchronization mechanism.
        """
        sh_args = shared_memory.ShareableList(name="shared_args")
        shape = (sh_args[0], sh_args[1])
        figure = plt.figure(figsize=(6, 6))
        axis = figure.add_subplot(111)
        image = axis.imshow(np.random.randint(0, 10, size=shape),
                            cmap="gray_r")

        sh_memory = shared_memory.SharedMemory(name="pioneer-oa")
        annotations_grid = np.zeros(shape)

        def annotate(index):
```

```
58            if threshold < grid[index] != annotations_grid[index]:
59                axis.annotate('',
60                              xy=(index[1], index[0]),
61                              horizontalalignment="center",
62                              verticalalignment="center",
63                              color="black",
64                              size=4)
65                annotations_grid[index] = grid[index]
66
67        nCores = os.cpu_count()
68        if nCores is None:
69            nCores = 1
70        grid_pool = mp.Pool(nCores)
71
72        plt.show(block=False)
73
74        while True:
75            with lock:
76                grid = np.ndarray(shape,
77                                  dtype=np.float_,
78                                  buffer=sh_memory.buf)
79                threshold = sh_args[2]
80            image.set_data(grid)
81            grid_pool.map(annotate, np.ndindex(grid.shape))
82            figure.canvas.draw_idle()
83            plt.pause(0.01)
84
85    _print_task = Process(target=_print_map, args=(robot_lock,))
86    _print_task.start()
87    return _print_task
```

Listing 2: map$_p$*rinter.py*

```
1  #                           src
2  #                 Copyright (C) 2019 - Javinator9889
3  #
4  #    This program is free software: you can redistribute it and/or modify
5  #    it under the terms of the GNU General Public License as published by
6  #      the Free Software Foundation, either version 3 of the License, or
7  #                    (at your option) any later version.
8  #
9  #       This program is distributed in the hope that it will be useful,
10 #       but WITHOUT ANY WARRANTY; without even the implied warranty of
11 #        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 #                GNU General Public License for more details.
13 #
14 #     You should have received a copy of the GNU General Public License
15 #    along with this program. If not, see <http://www.gnu.org/licenses/>.
16 import numpy as np
17
18 from mapper import Mapper
19
20
21 class Wrapper:
22     """
23     Wrapper class that encapsulates a given Mapper class - it is useful for
24     saving and restoring data from a created mapping.pyx object, as Cython is
25     not providing object serialization for custom data types.
```

```
    """

    def __init__(self, mapper: Mapper):
        self.X0 = mapper.X0
        self.Y0 = mapper.Y0
        self.map_grid = (mapper.w, mapper.h)
        self.grid_size = (mapper.mw, mapper.mh)
        self.grid = np.asarray(mapper.grid, dtype=np.float_)
        self.k = mapper.k
        self.min = mapper.min
        self.max = mapper.max
        self.heading = mapper.heading
        self.max_read_distance = mapper.max_read_distance
        self.ratio = mapper.ratio
        self.threshold = mapper.threshold

    def restore(self) -> Mapper:
        """
        Restores the class into a Mapper class, setting the old params into
        the new class instance.
        :return: the recovered mapper class.
        """
        mapper = Mapper(X0=self.X0,
                        Y0=self.Y0,
                        map_width=self.map_grid[0],
                        map_height=self.map_grid[1],
                        grid_size=self.grid_size,
                        k=self.k,
                        max_read_distance=self.max_read_distance,
                        ratio=self.ratio)
        mapper.grid = self.grid
        mapper.min = self.min
        mapper.max = self.max
        mapper.heading = self.heading
        mapper.threshold = self.threshold

        return mapper
```

Listing 3: map$_w$*rapper.py*

```
#                           PioneerOA
#                 Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#                GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.
import numpy as np

from mapping import PioneerMap
```

```
19
20 from multiprocessing import Lock
21 from multiprocessing import shared_memory
22
23
24 class Mapper(PioneerMap):
25     """
26     Python-wide class for accessing the C interface and offering the
27     necessary synchronization mechanisms in order to interact with other
28     processes.
29     This class creates a C instance of the PioneerMap class and a shared
30     memory location for communicating with the map printer class.
31     """
32
33     def __init__(self,
34                  X0: float,
35                  Y0: float,
36                  map_width: float,
37                  map_height: float,
38                  grid_size: tuple,
39                  sonar: list = None,
40                  initial_threshold: float = 1.0,
41                  k: float = 1.0,
42                  min_cv: float = 0.5,
43                  max_cv: float = 30,
44                  max_read_distance: float = 1.0,
45                  ratio: float = 1.5):
46         print(initial_threshold)
47         super().__init__(X0, Y0, map_width, map_height, grid_size, sonar,
48                          initial_threshold, k, min_cv, max_cv,
49                          max_read_distance, ratio)
50         self.lock = Lock()
51         self.sh_memory = shared_memory.SharedMemory(name="pioneer-oa",
52                                                     create=True,
53                                                     size=self.grid.nbytes)
54         self.shared_np = np.ndarray(self.grid.shape,
55                                     dtype=np.float_,
56                                     buffer=self.sh_memory.buf)
57         self.shared_args = shared_memory.ShareableList([self.mw, self.mh,
58                                                         self.threshold],
59                                                        name="shared_args")
60         self.shared_np[:] = self.grid[:]
61
62     def update_robot_position(self,
63                               robotX: float,
64                               robotY: float,
65                               sonar: list,
66                               heading: float):
67         """
68         Updates the robot position and sets the shared data required
69         information.
70         :param robotX: double robot X position.
71         :param robotY: double robot Y position.
72         :param sonar: the list of the sensor data.
73         :param heading: the robot heading (orientation).
74         :return:
75         """
76         super().update_robot_position(robotX, robotY, sonar, heading)
```

```
77          with self.lock:
78              self.shared_np[:] = self.grid[:]
79              self.shared_args[2] = self.threshold
80
81      def __del__(self):
82          """
83          Safely finishes the class instance, removing the unnecessary and
84          opened shared memory locations.
85          :return:
86          """
87          try:
88              self.sh_memory.close()
89              self.shared_args.shm.close()
90          finally:
91              self.sh_memory.unlink()
92              self.shared_args.shm.unlink()
93              del self.shared_args
94
95      def __reduce__(self):
96          _, values = super().__reduce__()
97          return Mapper, values
```

Listing 4: mapper.py

```
1  #                            PioneerOA
2  #                  Copyright (C) 2019 - Javinator9889
3  #
4  #    This program is free software: you can redistribute it and/or modify
5  #    it under the terms of the GNU General Public License as published by
6  #      the Free Software Foundation, either version 3 of the License, or
7  #                    (at your option) any later version.
8  #
9  #       This program is distributed in the hope that it will be useful,
10 #       but WITHOUT ANY WARRANTY; without even the implied warranty of
11 #        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 #                GNU General Public License for more details.
13 #
14 #     You should have received a copy of the GNU General Public License
15 #    along with this program. If not, see <http://www.gnu.org/licenses/>.
16
17 # cython: infer_types=True
18 # distutils: language=c++
19 import cython
20
21 import numpy as np
22 cimport numpy as np
23
24 from libc.math cimport sin
25 from libc.math cimport cos
26
27 from sensors cimport Sensors
28 from pioneer cimport Pioneer
29 from PioneerSensor cimport PioneerSensor
30
31
32 cdef class PioneerMap(Pioneer):
33     """
34     C interface for storing, computing and evaluating the robot data
```

```
    and map location (grid). It uses the native C interface of the computer
    in order to optimize and speed-up both memory and CPU usage.
    """
    def __init__(self,
                 double X0,
                 double Y0,
                 double map_width,
                 double map_height,
                 tuple grid_size,
                 list sonar = None,
                 double initial_threshold = 1.0,
                 double k = 1,
                 double min_cv = 0.5,
                 double max_cv = 30.0,
                 double max_read_distance = 1.0,
                 double ratio = 1.5):
        super().__init__(Sensors(sonar))
        assert grid_size[0] > 0 and grid_size[1] > 0
        self.X0 = X0
        self.Y0 = Y0
        self.w = map_width
        self.h = map_height
        self.mw = grid_size[0]
        self.mh = grid_size[1]
        self.sensor = PioneerSensor(sonar)
        self.grid = np.zeros(shape=grid_size, dtype=np.float_)
        self.k = k
        self.min = min_cv
        self.max = max_cv
        self.heading = 0
        self.max_read_distance = max_read_distance
        self.ratio = ratio
        self.threshold = initial_threshold

    cpdef tuple translate_to_matrix_position(self, double x, double y):
        """
        Translates a given (x, y) position to the matrix correspondent cell.
        :param x: the X position.
        :param y: the Y position.
        :return: a tuple of integer values.
        """
        return int((x - self.X0) * (self.mw / self.w)), \
               -int((y - self.Y0) * (self.mh / self.h))

    @cython.boundscheck(False)
    @cython.wraparound(False)
    @cython.locals(x=cython.double, y=cython.double, mX=cython.int,
                   mY=cython.int, cv=cython.double)
    cpdef update_robot_position(self,
                                double robotX,
                                double robotY,
                                double[:] sonar,
                                double heading):
        """
        Updates the robot position and the CV values by using the required data
        in order to update the threshold.
        :param robotX: the robot X position.
        :param robotY: the robot Y position.
```

```
        :param sonar: the sonar list of data.
        :param heading: the heading (orientation) of the robot.
        :return:
        """
        assert len(sonar) == 16
        for i in range(16):
            self.sensor[i].value = sonar[i]
            x = self.sensor[i].value * \
                cos(self.sensor[i].angle + heading) + robotX
            y = self.sensor[i].value * \
                sin(self.sensor[i].angle + heading) + robotY
            mX, mY = self.translate_to_matrix_position(x, y)
            if mX < self.grid.shape[0] and mY < self.grid.shape[1]:
                if self.sensor[i].value > self.max_read_distance:
                    continue
                cv = self.k * (self.max_read_distance -
                               self.sensor[i].value) / \
                     self.max_read_distance
                if cv >= self.threshold:
                    self.grid[mX, mY] += cv
                    self.threshold *= (1 + 0.1 * self.ratio)
                elif cv != 0:
                    self.threshold *= (1 - 0.1 * self.ratio)

    def __reduce__(self):
        return PioneerMap, (self.X0, self.Y0, self.w, self.h, self.mw, self.mh,
                            self.sensor, np.asarray(self.grid, dtype=np.float_),
                            self.k, self.min, self.max, self.heading,
                            self.max_read_distance, self.ratio,
                            self.threshold)
```

Listing 5: mapping.pyx

```
#                           PioneerOA
#                 Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#               GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.
# distutils: language=c++
import cython

from pioneer cimport Pioneer
from PioneerSensor cimport PioneerSensor


cdef class PioneerMap(Pioneer):
    cdef public int mw, mh
    cdef public double[:, :] grid
```

```
    cdef public double k, heading, max_read_distance, min, max, ratio, \
        grid_max, threshold, X0, Y0, w, h
    cdef public PioneerSensor sensor

    cpdef tuple translate_to_matrix_position(self, double x, double y)

    @cython.locals(x=cython.double, y=cython.double, mX=cython.int,
                   mY=cython.int,
                   cv=cython.double)
    cpdef update_robot_position(self, double robotX, double robotY,
                                double[:] sonar, double heading)
```

Listing 6: mapping.pxd

```
#                           src
#                Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#               GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.
from math import sin
from math import cos

from mapping import PioneerMap


class Motion:
    def __init__(self, robot: PioneerMap, movements: list):
        self.robot = robot
        self.movements = movements

    def move(self, get_position, get_heading, set_speed, **kwargs):
        for movement in self.movements:
            x, y = get_position(kwargs["client"], kwargs["robot"])
            heading = get_heading(kwargs["client"], kwargs["robot"])
            i, j = self.robot.translate_to_matrix_position(x, y)
            diffi, diffj = movement[0] - i, movement[1] - j
            if diffi != 0:
                if abs(round(cos(heading), 2)) != 0:
                    if cos(heading) < 0:
                        if diffi > 0:
                            lspeed, rspeed = 0.5, 0
                            fval = 1
                        else:
                            lspeed, rspeed = 0, 0.5
                            fval = -1
                    else:
                        if diffi < 0:
                            lspeed, rspeed = 0, 0.5
```

```
                            fval = 1
                        else:
                            lspeed, rspeed = 0.5, 0
                            fval = -1
                    while round(sin(get_heading(kwargs["client"],
                                                kwargs["robot"])), 2) != fval:
                        set_speed(kwargs["client"], kwargs["robot"],
                                  lspeed, rspeed)
            elif diffj != 0:
                if abs(round(sin(heading), 2)) != 0:
                    if sin(heading) < 0:
                        if diffj > 0:
                            lspeed, rspeed = 0.5, 0
                            fval = 1
                        else:
                            lspeed, rspeed = 0, 0.5
                            fval = -1
                    else:
                        if diffj < 0:
                            lspeed, rspeed = 0, 0.5
                            fval = 1
                        else:
                            lspeed, rspeed = 0.5, 0
                            fval = -1
                    while round(cos(get_heading(kwargs["client"],
                                                kwargs["robot"])), 2) != fval:
                        set_speed(kwargs["client"], kwargs["robot"],
                                  lspeed, rspeed)
            while get_position(kwargs["client"], kwargs["robot"]) != \
                    movement:
                set_speed(kwargs["client"], kwargs["robot"], 1, 1)
```

Listing 7: motion.py

```
#                            robot_control
#                  Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#                GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.

# cython: infer_types=True
# distutils: language=c++
import cython

from libc.math cimport cos
from libc.math cimport M_PI

from typing import Tuple
```

```
25
26 from sensors cimport Sensors
27
28 cpdef inline float radians(float degrees):
29     return (degrees * M_PI) / 180.0
30
31 cdef class Pioneer:
32     """
33     Direct access the robot information wrapping the different values
34     contained in sensors.
35
36     With this class, it is possible to know the exact situation of the
37     robot and actuate in base on it.
38
39     The accessible parameter is:
40      - sensors: an instance of Sensors class wrapping the different sonar values.
41     """
42
43     def __init__(self, Sensors sensors):
44         self.sensors = sensors
45
46     cpdef double nearest_obstacle_at(self, str orientation):
47         """
48         With the given orientation, finds the nearest obstacle to it.
49
50         Parameters
51         ----------
52         orientation : str
53             The orientation in which the robot must look for
54             obstacles - possible values are: `{right, left, front}`.
55         Returns
56         -------
57         out : float
58             The nearest obstacle to the given orientation. If the orientation
59             is not valid, then the result is -1.
60         """
61         if orientation == "left":
62             return self.sensors.left_sensors.min()
63         elif orientation == "right":
64             return self.sensors.right_sensors.min()
65         elif orientation == "front":
66             return self.sensors.front_sensors.min()
67         else:
68             return -1
69
70     @cython.locals(i=cython.int, min_dist=cython.float, dist=cython.float)
71     cpdef int is_any_obstacle_front(self):
72         """
73         Checks if there is any obstacle in front of the robot (sensors 3 to 5).
74
75         Returns
76         -------
77         out : bool
78             True if some of the sensors is less or equal to the minimum distance.
79             If all of the sensors do not detect any obstacle, then returns False.
80         """
81         for i in range(2, 6):
82             min_dist, dist = self.distance_in_xaxis(i)
```

```
83              if dist <= min_dist:
84                  return 1
85          return 0
86
87      @cython.locals(i=cython.int, min_dist=cython.float, dist=cython.float)
88      cpdef int is_any_obstacle_left(self):
89          """
90          Checks if there is any obstacle next to the robot on the left (sensors 2 to 4)
        .
91
92          Returns
93          -------
94          out : bool
95              True if some of the sensors is less or equal to the minimum distance.
96              If all of the sensors do not detect any obstacle, then returns False.
97          """
98          for i in range(1, 5):
99              min_dist, dist = self.distance_in_xaxis(i)
100             if dist <= min_dist:
101                 return 1
102         return 0
103
104     @cython.locals(i=cython.int, min_dist=cython.float, dist=cython.float)
105     cpdef int is_any_obstacle_right(self):
106         """
107         Checks if there is any obstacle next to the robot on the right (sensors 6 to
        8).
108
109         Returns
110         -------
111         out : bool
112             True if some of the sensors is less or equal to the minimum distance.
113             If all of the sensors do not detect any obstacle, then returns False.
114         """
115         for i in range(5, 8):
116             min_dist, dist = self.distance_in_xaxis(i)
117             if dist <= min_dist:
118                 return 1
119         return 0
120
121     @cython.locals(min_distance=cython.float,
122                    angle=cython.float,
123                    dist=cython.float)
124     cpdef tuple distance_in_xaxis(self, int sensor):
125         """
126         Calculates the distance in the x-axis by using the cosine of the angle
        multiplied
127         by the measured distance.
128
129         Parameters
130         ----------
131         sensor : int
132             the position of the sensor (from 0 to 7) from which the distance must be
133             measured.
134
135         Returns
136         -------
137         out : Tuple[float, float]
```

```
138             the minimum distance, for comparing, and the projection of the distance in
139             the x-axis.
140         """
141         min_distance, angle, dist = self[sensor]
142         return min_distance, dist * cos(angle)
143 
144     def __getitem__(self, key) -> tuple:
145         assert isinstance(key, int)
146         dist = {
147             7: 0.5,
148             6: 2,
149             5: 3,
150             4: 7,
151             3: 7,
152             2: 3,
153             1: 2,
154             0: 0.5
155         }.get(key)
156         angle = {
157             7: radians(0),  # 90 - 90
158             6: radians(40),  # 90 - 50
159             5: radians(60),  # 90 - 30
160             4: radians(80),  # 90 - 10
161             3: radians(80),  # 90 - 10
162             2: radians(60),  # 90 - 30
163             1: radians(40),  # 90 - 50
164             0: radians(0)  # 90 - 90
165         }.get(key)
166         return dist, angle, self.sensors[key]
167 
168     def __reduce__(self):
169         return Pioneer, (self.sensors,)
```

Listing 8: pioneer.pyx

```
1  #                            robot_control
2  #                  Copyright (C) 2019 - Javinator9889
3  #
4  #    This program is free software: you can redistribute it and/or modify
5  #    it under the terms of the GNU General Public License as published by
6  #      the Free Software Foundation, either version 3 of the License, or
7  #                    (at your option) any later version.
8  #
9  #       This program is distributed in the hope that it will be useful,
10 #       but WITHOUT ANY WARRANTY; without even the implied warranty of
11 #        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 #                GNU General Public License for more details.
13 #
14 #     You should have received a copy of the GNU General Public License
15 #    along with this program. If not, see <http://www.gnu.org/licenses/>.
16 
17 # distutils: language=c++
18 cimport cython
19 
20 from sensors cimport Sensors
21 
22 cpdef inline float radians(float degrees)
23 
```

```
cdef class Pioneer:
    cdef public Sensors sensors

    cpdef double nearest_obstacle_at(self, str orientation)

    cpdef int is_any_obstacle_front(self)

    cpdef int is_any_obstacle_left(self)

    cpdef int is_any_obstacle_right(self)

    cpdef tuple distance_in_xaxis(self, int sensor)
```

Listing 9: pioneer.pxd

```
#                           robot_control
#                Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#                GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.

# distutils: language=c++
from sensor cimport Sensor
from pioneer cimport radians

cdef class PioneerSensor:
    """
    Wrapper class that contains the sensor data of the Pioneer robot.
    """
    def __init__(self, list sonar = None):
        angles = {
            0: radians(90),
            1: radians(50),
            2: radians(30),
            3: radians(10),
            4: radians(-10),
            5: radians(-30),
            6: radians(-50),
            7: radians(-90),
            8: radians(-90),
            9: radians(-130),
            10: radians(-150),
            11: radians(-170),
            12: radians(170),
            13: radians(150),
            14: radians(130),
            15: radians(90)
        }
```

```
        if sonar is None:
            sonar = [1] * 16
        assert len(sonar) == 16
        self.sonar = [Sensor(angle, value) for angle, value in zip(
            angles.values(), sonar)]
        self.parallel_left = [self.sonar[15], self.sonar[0]]
        self.parallel_right = [self.sonar[7], self.sonar[8]]

    def __getitem__(self, item) -> Sensor:
        assert isinstance(item, int)
        return self.sonar[item]

    def __reduce__(self):
        return PioneerSensor, (
        self.sonar, self.parallel_right, self.parallel_left)
```

Listing 10: PioneerSensor.pyx

```
#                            robot_control
#                 Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#                GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.

# distutils: language=c++

cdef class PioneerSensor:
    cdef public list sonar, parallel_left, parallel_right
```

Listing 11: PioneerSensor.pxd

```
#                            src
#                 Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#                GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.
import numpy as np

```

```
18 from math import sqrt
19
20 from mapping import PioneerMap
21
22
23 class Planner:
24     """
25     Class that uses a given grid for planning a path to the robot.
26     """
27
28     def __init__(self, grid_map: PioneerMap):
29         self.map = grid_map
30         self.potential_map = grid_map.grid.copy()
31         self.visited_map = np.zeros(grid_map.grid.shape, dtype=bool)
32
33     def calculate_path(self, origin: tuple, target: tuple) -> list:
34         """
35         Calculates a path for the robot by using the origin and target points.
36         :param origin: the origin point.
37         :param target: the target point.
38         :return: a list with the movements.
39         """
40
41         def is_valid_cell(i, j):
42             return 0 <= i < self.potential_map.shape[0] and 0 <= j < \
43                    self.potential_map.shape[1] and not self.visited_map[i, j]
44
45         i0, j0 = self.map.translate_to_matrix_position(x=origin[0],
46                                                        y=origin[1])
47         it, jt = self.map.translate_to_matrix_position(x=target[0],
48                                                        y=target[1])
49
50         if self.map.grid[i0, j0] != 0 or self.map.grid[it, jt] != 0:
51             return []
52
53         for i, j in np.ndindex(self.map.grid.shape):
54             if self.map.grid[i, j] != 0:
55                 self.potential_map[i, j] = -1
56             else:
57                 if not ((i, j) == (it, jt)):
58                     self.potential_map[i, j] = 1 / sqrt((it - i) ** 2
59                                                         + (jt - j) ** 2)
60                 else:
61                     self.potential_map[i, j] = float("inf")
62
63         path = [(i0, j0)]
64         self.visited_map[i0, j0] = True
65         i, j = i0, j0
66         while not ((i, j) == (it, jt)):
67             self.visited_map[i, j] = True
68             if (i, j) not in path:
69                 path.append((i, j))
70             potentials = {
71                 "i + 1": 0,
72                 "i - 1": 0,
73                 "j + 1": 0,
74                 "j - 1": 0
75             }
```

```
            if is_valid_cell(i + 1, j):
                potentials["i + 1"] = \
                    self.potential_map[i + 1, j]
            if is_valid_cell(i - 1, j):
                potentials["i - 1"] = \
                    self.potential_map[i - 1, j]
            if is_valid_cell(i, j + 1):
                potentials["j + 1"] = \
                    self.potential_map[i, j + 1]
            if is_valid_cell(i, j - 1):
                potentials["j - 1"] = \
                    self.potential_map[i, j - 1]

            movement = max(zip(potentials.values(), potentials.keys()))[1]
            if movement == "i + 1" and potentials[movement] >= 0:
                i += 1
            elif movement == "i - 1" and potentials[movement] >= 0:
                i -= 1
            elif movement == "j + 1" and potentials[movement] >= 0:
                j += 1
            elif movement == "j - 1" and potentials[movement] >= 0:
                j -= 1
            else:
                break

        path.append((it, jt))

        return path
```

Listing 12: planner.py

```
#                           src
#                 Copyright (C) 2019 - Javinator9889
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#      the Free Software Foundation, either version 3 of the License, or
#                    (at your option) any later version.
#
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#               GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.

# distutils: language=c++


cdef class Sensor:
    """
    Sensor wrapper class for fast access the ultrasonic sensor information.
    """
    def __init__(self, double angle, double value = 0.0):
        self.angle = angle
        self.value = value

```

```
28      def __reduce__(self):
29          return Sensor, (self.angle, self.value)
```

Listing 13: sensor.pyx

```
1  #                            src
2  #                  Copyright (C) 2019 - Javinator9889
3  #
4  #    This program is free software: you can redistribute it and/or modify
5  #    it under the terms of the GNU General Public License as published by
6  #      the Free Software Foundation, either version 3 of the License, or
7  #                    (at your option) any later version.
8  #
9  #       This program is distributed in the hope that it will be useful,
10 #       but WITHOUT ANY WARRANTY; without even the implied warranty of
11 #        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 #                GNU General Public License for more details.
13 #
14 #     You should have received a copy of the GNU General Public License
15 #    along with this program. If not, see <http://www.gnu.org/licenses/>.
16 # distutils: language=c++
17 cdef class Sensor:
18     cdef public double value
19     cdef readonly double angle
```

Listing 14: sensor.pxd

```
1  #                            sensors
2  #                  Copyright (C) 2019 - Javinator9889
3  #
4  #    This program is free software: you can redistribute it and/or modify
5  #    it under the terms of the GNU General Public License as published by
6  #      the Free Software Foundation, either version 3 of the License, or
7  #                    (at your option) any later version.
8  #
9  #       This program is distributed in the hope that it will be useful,
10 #       but WITHOUT ANY WARRANTY; without even the implied warranty of
11 #        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 #                GNU General Public License for more details.
13 #
14 #     You should have received a copy of the GNU General Public License
15 #    along with this program. If not, see <http://www.gnu.org/licenses/>.
16
17 # distutils: language=c++
18
19 import cython
20
21 import numpy as np
22 cimport numpy as np
23
24 cdef class Sensors:
25     """
26     Wrapper for managing the data input received by the robot sensors.
27     As the range of each sonar is, at most, one meter, this wrapper converts each
28     value to centimeters by multiplying by 100.
29
30     In addition, this model is using only the first eight sensors as the back ones
31     are not being used.
```

```
32
33      The accessible params are:
34       - sonar: a numpy array containing the sonar data (len == 8).
35       - parallel_left: the sensors 1 and 16 for controlling the distance to the wall.
36       - parallel_right: the sensors 8 and 9 for controlling the distance to the wall.
37      """
38      def __init__(self, list sonar = None):
39          self.sonar = np.ones(8) * 100 if sonar is None else \
40              np.asarray(sonar[:8]) * 100
41          """
42          numpy array containing the values for each sensor, in centimeters, starting
    from zero.
43          """
44          self.parallel_left = int(sonar[0] * 100) if sonar is not None else 1, \
45                               int(sonar[15] * 100) if sonar is not None else 1
46          """
47          Tuple[int, int] containing the distance to the wall on the left, in
    centimeters.
48          """
49          self.parallel_right = int(sonar[7] * 100) if sonar is not None else 1, \
50                                int(sonar[8] * 100) if sonar is not None else 1
51          """
52          Tuple[int, int] containing the distance to the wall on the right, in
    centimeters.
53          """
54
55      @cython.boundscheck(False)
56      @cython.wraparound(False)
57      cpdef set_sonar(self, list sonar):
58          """
59          Updates the sonar data. Input list contains the read data from sonar in meters
    .
60
61          Parameters
62          ----------
63          sonar: list of float data
64              measured in meters, whose length is 16.
65
66          Raises
67          ------
68          AssertionError
69              when the length of the list is distinct 16.
70          """
71          assert len(sonar) == 16
72          self.sonar = np.asarray(sonar[:8]) * 100
73          self.parallel_left = int(sonar[0] * 100), int(sonar[15] * 100)
74          self.parallel_right = int(sonar[7] * 100), int(sonar[8] * 100)
75
76      @property
77      def front_sensors(self) -> np.float_:
78          """
79          Fast access for the front sensors of the robot.
80
81          :return a numpy array from sensor number 3 to sensor number 5.
82          """
83          return self.sonar[2:6]
84
85      @property
```

```
86      def left_sensors(self) -> np.float_:
87          """
88          Fast access for the left sensors of the robot.
89
90          :return a numpy array from sensor number 1 to sensor number 3.
91          """
92          return self.sonar[0:4]
93
94      @property
95      def right_sensors(self) -> np.float_:
96          """
97          Fast access for the front sensors of the robot.
98
99          :return a numpy array from sensor number 5 to sensor number 7.
100         """
101         return self.sonar[5:8]
102
103     def __getitem__(self, int key):
104         assert isinstance(key, int)
105         return self.sonar[key]
106
107     def __reduce__(self):
108         return Sensors, (self.sonar, self.parallel_left, self.parallel_right)
```

Listing 15: sensors.pyx

```
1  #                            src
2  #                 Copyright (C) 2019 - Javinator9889
3  #
4  #    This program is free software: you can redistribute it and/or modify
5  #    it under the terms of the GNU General Public License as published by
6  #      the Free Software Foundation, either version 3 of the License, or
7  #                    (at your option) any later version.
8  #
9  #       This program is distributed in the hope that it will be useful,
10 #       but WITHOUT ANY WARRANTY; without even the implied warranty of
11 #        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12 #               GNU General Public License for more details.
13 #
14 #     You should have received a copy of the GNU General Public License
15 #    along with this program. If not, see <http://www.gnu.org/licenses/>.
16
17 # distutils: language=c++
18 cdef class Sensors:
19     cdef public double[:] sonar
20     cdef public tuple parallel_left, parallel_right
21     cpdef set_sonar(self, list sonar)
```

Listing 16: sensors.pxd

```
1  #                            src
2  #                 Copyright (C) 2019 - Javinator9889
3  #
4  #    This program is free software: you can redistribute it and/or modify
5  #    it under the terms of the GNU General Public License as published by
6  #      the Free Software Foundation, either version 3 of the License, or
7  #                    (at your option) any later version.
8  #
```

```
#       This program is distributed in the hope that it will be useful,
#       but WITHOUT ANY WARRANTY; without even the implied warranty of
#        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
#               GNU General Public License for more details.
#
#     You should have received a copy of the GNU General Public License
#    along with this program. If not, see <http://www.gnu.org/licenses/>.
import os
import numpy as np

from distutils.core import setup
from Cython.Build import cythonize

sourcefiles = ["mapping.pyx", "sensors.pyx", "pioneer.pyx", "sensor.pyx",
               "PioneerSensor.pyx"]
threads = os.cpu_count()
if threads is None:
    threads = 1

print("----------------------------------")
print(f"Compiling with {threads} threads")
print("----------------------------------")

setup(
    ext_modules=cythonize(sourcefiles,
                          nthreads=threads,
                          compiler_directives={"language_level": "3",
                                               "infer_types": True,
                                               "optimize.use_switch": True},
                          include_path=[np.get_include()],
                          annotate=True)
)
```

Listing 17: setup.py