Taller 3 - IRB2001

Profesor: Jorge Díaz
jjdiaz6@uc.cl

Ayudante: Alonso Rivera
adrivera1@uc.cl

Publicación: Viernes 20 de Octubre.
Entrega: Viernes 24 de Noviembre.

1. Indicaciones

- El puntaje asociado a cada pregunta está incluído al lado de ella, la nota final consiste en la suma delpuntaje de todas las preguntas (+1 base).
- La entrega es en canvas y consiste de un archivo .zip que contenga un informe y la estructura de la tarea (puedes descargarla clonando **este repositorio**)
- La tarea es individual, pudiendo discutirla con sus pares. Toda referencia externa debe citarse.

2. Parte práctica

Para la realización de esta tarea es necesaria la utilización del programa **Webots**, compatible con Windows, Linux y macOS (**Guía de instalación**), también es necesario contar con Python 3 (preferentemente **Python3.11**) y se recomienda fuertemente el uso de la librería **Numpy** para agilizar los cálculos.

2.1. Simulación (Webots)

En este caso utilizaremos un robot modelo "Turtlebot Burger" y un laberinto de 10m x 10m.

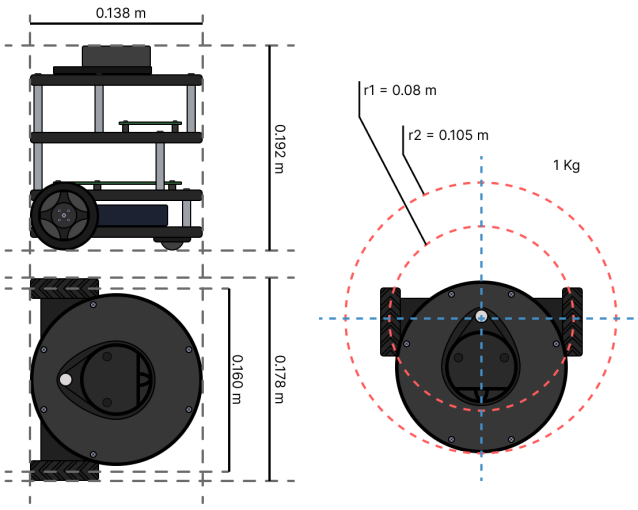

Figure 1: Turtlebot burger

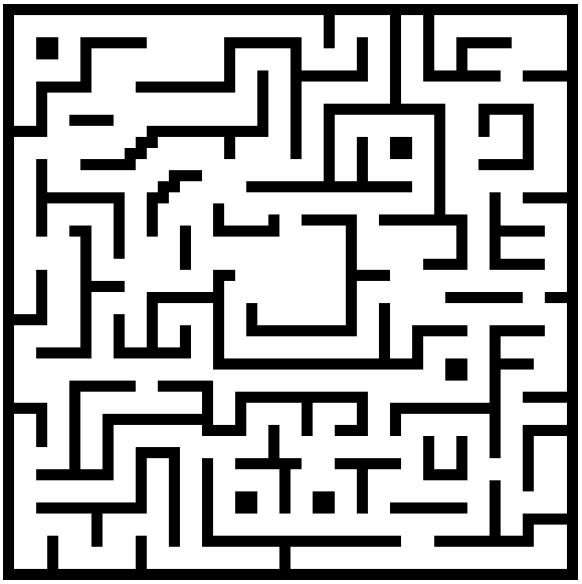

Figure 2: Laberinto

HINT: el robot cuenta con un módulo **"gps"** y un módulo **"gyro"** que junto con la función integrada en Python **"dir()"** te serán útiles.

2.1.1. Mapeo (SLAM) (0.25 ptos.)

En la carpeta "simulador/controllers/01_mapping" debes implementar un controlador que de manera autónoma se ubique y registre **todo** el mapa del *World file* "maze.wbt" a partir de los sensores integrados en el robot.

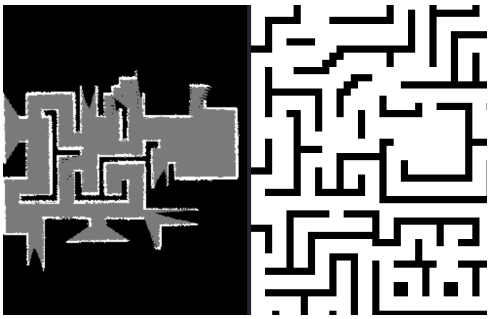

Figure 3: Ejemplo mapa odométrico

Deberás hacer uso del módulo **LiDAR LDS-01** integrado en tu Turtlebot, este toma N = 240 muestras, una cada θ = 360/N grados, con una distancia radial maxima de L = 1.8 m, ruido artificial en la medición con un coeficiente de 0.0086 y un periodo de muestreo *timeStep*.

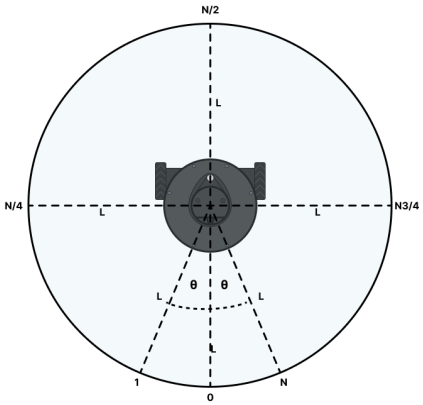

Figure 4: LDS-01

2.1.2. Localización (0.5 ptos.)

En la carpeta "simulador/controllers/02_location" debes implementar un controlador que sea capaz de encontrar la ubicación del robot en cualquier parte del *World file* "maze.wbt" en función de los datos del sensor LiDAR, en este caso se te entrega un código donde la función gps y gyro estan modificados para que solo puedas utilizarlo para detectar el movimiento del robot y no obtener la posicion exacta, también se te entrega un mapa en formato .png (10m:1px) por lo que puedes decidir usar este o el que generaste en la pregunta anterior.

2.1.3. Localizacion con obstaculos (0.75 ptos.)

En la carpeta "simulador/controllers/03_location_obstacle" debes implementar un controlador que sea capaz de encontrar la ubicación del robot en cualquier parte del *World file* "maze-obstacle.wbt" en función de los datos del sensor LiDAR. En este caso, se te entrega un código donde las funciones gps y gyro están modificadas, y el mapa cuenta con obstáculos que harán que tus mediciones tengan errores mucho más grandes. Se te entrega el mismo mapa que en la pregunta anterior que no cuenta con información de estos obstáculos. Puedes asumir que estos últimos son estáticos.

*No es válido el uso de la cámara integrada.
*Se espera que se pruebe moviendo manualmente el robot a posiciones que tengan obstáculos.

2.1.4. Planificación de Trayectorias (1.25 ptos.)

En la carpeta REDACTED debes implementar un controlador capaz de moverse desde un punto A a un punto B en el *World file* "maze.wbt", contaras con una variable "MODO", si su valor es igual a 1 deberas encontrar el camino mas rapido (considerando pesos para cada movimiento teniendo en cuenta el costo

de los giros y la aceleracion en rectas), en el caso de que la variable tenga valor 0 deberas encontrar el camino más corto (distancia manhattan), para ambos casos puedes usar una representacion interna previa del mapa y los valores de los sensores, puedes ocupar cualquer algoritmo (A, DFS, BFS, Dijkstra, etc) pero debes justificar la eleccion de este para decidir cual entrega el camino mas corto y el mas rapido. En el caso de que MODO tenga valor 2, debes encontrar el camino de A hasta B sin usar la representacion interna o mapa, se recomienda utilizar A* (**ejemplo**).

Puedes obtener los puntos A y B de la funcion entregada *get_points* (n:int), esta funcion retorna un "numpy array" de dimensiones [n, 2] con n posiciones validas por lo que deberas utilizar n=2 (caso base).

*Se espera que desde cualquier posición primero llegue hasta A y luego hasta B.
* Tener en consideracion 0.5 metros (distancia manhattan) de tolerancia para la meta (punto B)

2.1.5. Planificación de Trayectorias con obstaculos (1.25 ptos.)

En la carpeta REDACTED debes implementar un controlador capaz de moverse desde un punto A a un punto B (obtenidos de la misma manera que la pregunta anterior) en el *World file* "maze-obstacle2.wbt", en este caso no importa que tan rapido o corto sea el camino, pero contaras con obstaculos en el mapa que debes esquivar, la mayoria de estos no estan en el rango visible del lidar por lo que deberas hacer uso del modulo **"camera"** integrado en el robot, este cuenta con una funcion de reconocimiento de objetos y distancia de la camara, si el objeto detectado es un **conejo** deberas deveras evitarlo y cambiar de camino, pero si es una **botella**, deberas cambiar objetivo para llegar a un nuevo punto C (deberas hacer otra peticion a la funcion *get_points*)(en caso de encontrarte con otra botella no debes hacer nada), si es cualquier otro objeto puedes ignorarlo o esquivarlo en caso de no permitirte el paso.

* En caso de que alguno de los puntos sea un conejo se considera como una ruta no valida y debes reiniciar el proceso considerando nuevos puntos.
* Tener en consideracion 1 metro (distancia manhattan) de tolerancia para la meta (punto B o C)

2.2. Visión por Computador (YOLO) (1 pto.)

Haciendo uso del modelo YOLO de la libreria **ultralytics**, deberas entrenar un modelo de IA que logre jugar el juego del archivo "select_game.py", este muestra 4 imagenes al azar donde 1 pertenece a la carpeta "game/correct_images" y las otras 3 a "game/incorrect_images", al hacer click en la imagen de la carpeta correcta obtienes 1 punto, al obtener 10 puntos ganas el juego.

Debes generar un data set con imagenes (.png) de dos categorias (por ejemplo perros y gatos), con este entrenar al modelo y usando una libreria como **pyautogui** deberas hacer que un codigo implementado en el archivo "player.py" detecte los elementos en las imagenes y seleccione la opción correcta.

*Con el data set generado debes llenar las carpetas de "game/correct_images" y "game/incorrect_images" para que el juego funcione, por lo que depende de ti cual sera la categoria correcta e incorrecta (Por ejemplo las imagenes correctas son perros y las incorrectas gatos).
*Para generar el data set se recomienda el uso de herramientas como **googleapis** y **cvat**, aunque tambien se acepta descargar un dataset listo de paginas como **kaggle**, por el contrario no esta permitido el uso de un modelo pre-entrenado.

HINT: Puedes entrenar el modelo en colab dado que para utilizar el modelo para deteccion solo necesitas el archivo ".pt"

3. Parte teorica (1 pto.)

Responda las siguientes preguntas:

1. En la vida real los sensores no son ideales y funcionan un poco distinto al simulador, investigue sobre el funcionamiento y aplicacion de los sensores LiDAR en la vida real, de al menos 3 ejemplos de uso practicos.
2. Investigue que tipo de sensores se utilizan en la realidad para realizar odometria y mencione cuales serian necesarios para implementar este robot en la vida real (obviando la camara y el LiDAR).
3. Los metodos para resolver laberintos no solo sirven para encontrar caminos en 2D, explique 2 aplicaciones de estos fuera de las 2 dimensiones y comente cuales serian los pasos para mapear estos problemas.
4. Explique por que en la realidad para un mismo punto puede existir un camino mas corto y otro que sea mas largo pero mas rapido para un mismo robot.