Universidad Carlos III
Robótica
Curso 2024-25

Práctica 6 - Trabajo Final

Grupo 81
Cuarto curso - Ingeniería Informática

100472152, Cabrera Nieto, Álvaro (100472152@alumnos.uc3m.es)
100448737, Loor Weir, Iván Sebastián (100448737@alumnos.uc3m.es)

Índice

1. Introducción y planteamiento del problema

En el presente trabajo se debe utilizar el simulador **Webots** y diseñar un **controlador** que explore uno de diez **escenarios**, rescate a dos **personas** en el extremo contrario del mundo y vuelva a la línea de salida sin colisionar con **obstáculos** y en el **menor tiempo posible**. Al final del desarrollo del proyecto, se habrán reforzado ampliamente los conocimientos sobre los **sensores y actuadores** del robot a utilizar (***Pioneer II***), así como sobre la implementación de **algoritmos** de exploración y guía en **entornos semidesconocidos**.

Controlador del robot Pioneer II

Para navegar por cada escenario, se modifica el **controlador *full_controller*** del robot *Pioneer II*. *Pioneer II* puede moverse mediante **dos ruedas fijas** (estabilizadoras o de tracción; similares a las ruedas traseras de un coche) **independientes**, es decir, cada una puede girar en direcciones y con velocidades distintas entre ellas, y una **rueda de soporte** (también conocida como *rueda loca*). Este **modelo**, así como sus **características no pueden ser modificados**. A continuación, se muestra imágenes de este robot:

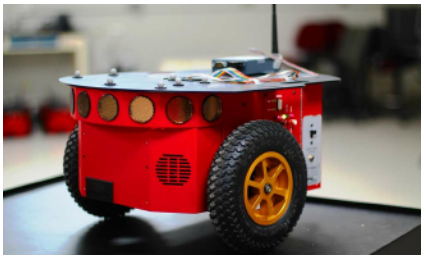

Figura 1.1. Fotografía del robot Pioneer II en la vida real

Figura 1.2. Robot Pioneer II en el simulador Webots

Sensores

En el simulador a utilizar (*Webots*), este robot tiene múltiples sensores: 16 de **distancia** (distribuidos uniformemente a lo largo de la circunferencia superior del robot, permitiendo la medición de distancias 360º), un **sensor de movimiento en cada rueda fija** (utilizado como ***encoder***; convierte el movimiento de la rueda en información para conocer cuánto se ha movido el robot), una **brújula** (funciona de manera precisa y similar entre todos los escenarios), un **sistema *GPS*** (permite conocer la posición aproximada del robot), una **cámara RGB frontal** y una **cámara RGB esférica**.

A continuación, se muestra un ejemplo de cómo funciona el **sistema *GPS***:

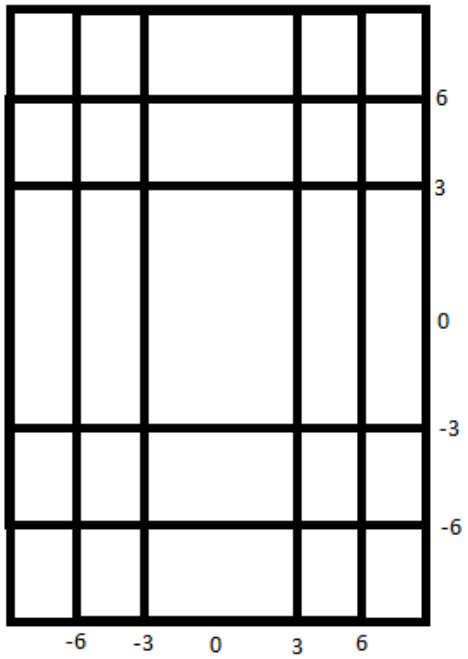

Figura 1.3. Matriz de celdas del escenario mediante GPS

Como se puede observar, al utilizar el sistema *GPS*, el escenario se divide en **celdas de tamaño fijo** para conocer la **posición** actual del robot de manera **aproximada** - como se menciona en el enunciado: '*El GPS tendrá una **resolución de 3 metros**, por lo que se recomienda su uso para **corregir el error acumulado** de otros sistemas de localización como la odometría, no para localizar en tiempo real el robot.*'. Es decir, únicamente se conoce la posición actual del robot **dentro de una celda** - se sabe en qué celda está pero **no la posición exacta dentro de dicha celda**. Sin embargo, cuando el robot **cambia de celda**, se sabe la nueva celda en la que se encuentra actualmente.

De esta manera, por ejemplo, si se cambia de celda se podría saber que **el robot acaba de cruzar la línea** que separa ambas celdas. Asimismo, si una celda cruzase hipotéticamente a través del cruce de varias celdas de manera diagonal, **se podría saber que el robot habría pasado recientemente** por dicha esquina. Sin embargo, la división del escenario en celdas también tiene otro **problema**: en los ejes donde se encuentra el **valor *0***, las celdas son de **mayor tamaño** - simplemente debido a la manera por defecto de dividir el mundo mediante una matriz - por lo que en estos valores (igual a 0) de los ejes, la **precisión** de la localización del robot mediante *GPS* sería **aún menor**.

Actuadores

Respecto a los **actuadores** de *Pioneer II*, únicamente cuenta con los **motores de las dos ruedas fijas**, lo que le permite girar cada una de ellas **de manera independiente** - con direcciones o velocidades distintas.

Escenarios

En este proyecto se presentan **10 escenarios** (también conocidos como **mundos**) distintos, todos ellos de **igual tamaño** (10×20 metros). A pesar de que la **posición de los obstáculos** (también llamados *daños estructurales*), de las **personas a rescatar** y de los **muros puede cambiar** entre cada mundo, así como la **iluminación** o la **niebla**, otros conceptos sí son constantes entre todos ellos. Por ejemplo, la **brújula** (hacia dónde marca el norte) y el **sistema *GPS* no cambian entre escenarios**. De igual manera, la **posición de las personas**, los **obstáculos**, los **muros**, la **iluminación** y la **niebla** siempre son **iguales dentro de un mismo escenario** - no utilizan **ningún factor de aleatoriedad**.

Esto **permite analizar cada escenario**, identificar aspectos únicos de cada uno **para diferenciarlo del resto** (también conocido como ***características del entorno***) y poder seguir una **ruta relativamente prefijada** para cada mundo, pudiendo **optimizar así el tiempo requerido** para rescatar a las personas y regresar a la línea de meta **sin chocarse**.

A continuación, se muestra una imagen con todos los escenarios:

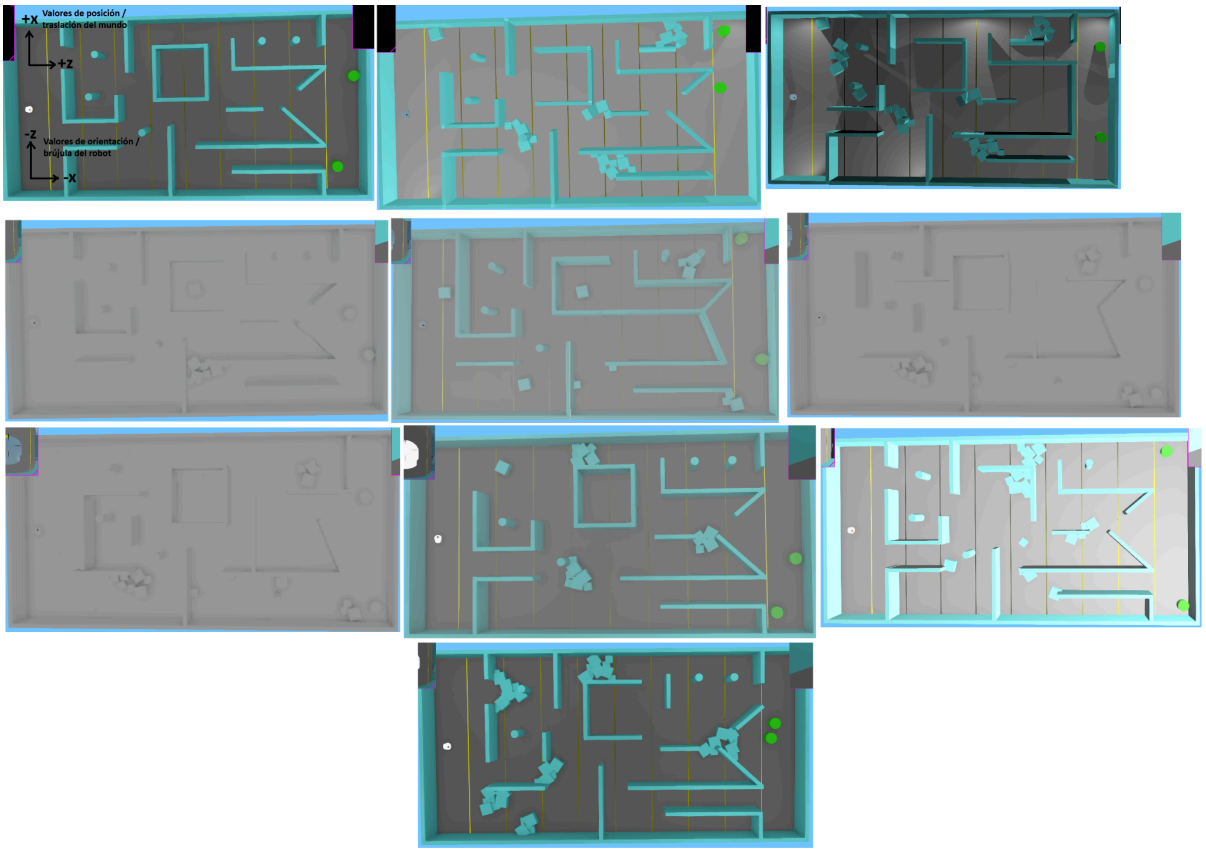

Figura 1.4. Montaje con imágenes de los diez escenarios

Como se puede apreciar, **cada uno** de estos mundos tiene una **configuración** y disposición **diferente** de **obstáculos, iluminación y niebla**.

A continuación, se muestra una imagen con el **primer escenario**:

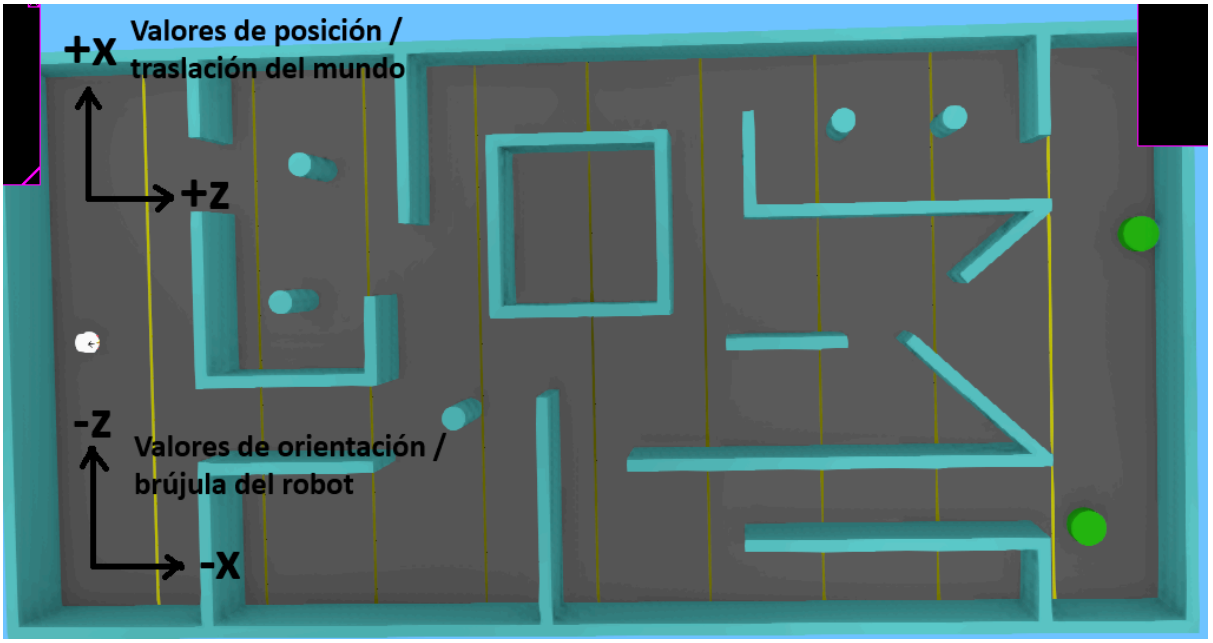

Figura 1.5. Escenario 01

En la figura anterior se puede observar el **robot** (de color blanco) en la parte izquierda, los **muros y obstáculos** (de color azul) presentes a lo largo del escenario y las **dos personas a rescatar** en la parte derecha (representadas por dos cilindros de color verde). El robot siempre **comienza en cualquier punto sobre la línea amarilla inicial** (en la parte izquierda de la imagen), con una **orientación y posición exactas no conocidas**. Esto será realmente **útil** a la hora de poder **identificar el escenario** en el que se encuentre el robot en ese momento, con el fin de **seguir una ruta predefinida y optimizada para cada mundo**.

Cabe destacar que los **ejes *x* y *z* no son consistentes entre los valores de posición** (o traslación) de los escenarios **y los valores de orientación** (o brújula) del robot - el **eje *y*** siempre se considera la **altura** (será constante en este trabajo, con valor igual a *0*). Como se explica en la parte izquierda de la imagen, los valores de los **ejes** (positivos y negativos) ***x* y *z*** no concuerdan entre los **sensores de orientación** (utilizado por el controlador) y los **valores de posición de los objetos en el mundo** (no utilizado por el controlador), lo que puede inducir a errores de interpretación. Es fundamental remarcar que, si **se orienta el robot** desde la posición inicial **hacia la línea** amarilla **de meta** (parte derecha de la imagen), los **valores de la brújula** sí son **consistentes entre los mundos**.

Por todo ello, la **serie de pasos** que el robot debe seguir **en todos los escenarios** es la siguiente: el robot debe **partir desde la línea de salida**, encontrar a la **primera persona**, **dar una vuelta** sobre sí mismo (para indicar que se ha identificado y salvado a la persona), encontrar a la **segunda persona**, **dar otra vuelta** sobre sí mismo, **volver a la línea de salida y parar**.

2. Descripción de la solución adoptada

Teniendo en cuenta lo anterior, se decide crear un **controlador** centrado en **identificar el escenario** actual y, a partir de ahí, **seguir una ruta predefinida** o fija manualmente y **evaluada con anterioridad** de manera exhaustiva y paulatina para asegurar su correcto funcionamiento, en un **tiempo optimizado y sin chocarse**, ajustando los valores de cada giro, espera y avance del robot según **observación directa** y prueba y error.

Esto se puede conseguir mediante los distintos **sensores** que posee el **robot**. Debido a su mayor **utilidad**, se decide emplear la **brújula** (ya que permite conocer la **orientación** de manera **precisa**, **en todo momento** y sin importar el **escenario**), los **sensores de distancia** (para comprobar la **cercanía a muros y obstáculos**) y la **cámara frontal** (con el fin de **monitorizar los códigos de los colores RGB** que observa el robot en cada instante).

Identificación de escenarios

Para ello, se diseña un **procedimiento inicial igual** para todos los escenarios, a partir del cual se diferenciará cada uno de ellos en función de sus **características del entorno distintivas**:

1. Dado que la **orientación y posición iniciales** del robot son **aleatorias** (comienza en cualquier punto **encima de la línea amarilla inicial**), **se ajusta la orientación** hacia el **muro superior**.
2. Se hace **avanzar** al robot **hasta colisionar con el muro superior** o hasta acercarse lo suficiente como para detectarlo con los sensores de distancia. Debido a que el número de grandes **colisiones permitidas** antes de incurrir en **penalización es tres**, esta primera colisión **no supone un problema**.
3. En este punto, **independientemente del escenario**, el robot se encuentra **frente al muro** y orientado hacia él. Ahora, **se gira** el robot **90º** hacia la izquierda, orientándose hacia el **muro inicial**.
4. Se hace **avanzar** al robot **hasta colisionar con el muro inicial** o hasta acercarse lo suficiente como para detectarlo con los sensores de distancia. Esta es la **segunda colisión**, por lo que **tampoco** es una **penalización**.
5. En este punto, el robot se encuentra frente al **muro inicial** y orientado hacia él, en la **esquina** entre el **muro inicial y el superior**. Ahora, **se gira** el robot **180º**, orientándose hacia la **línea de meta**.
6. Ahora se hace **avanzar** el robot **hasta identificar el escenario** determinado.

Es fundamental mencionar que, si **durante** cualquiera de los **pasos anteriores**, se detecta una **característica del entorno distintiva** de cierto escenario o conjunto de escenarios, el **comportamiento** del controlador del robot **cambia**. Por ejemplo, en el **escenario 09** se utiliza un juego de luces con una **iluminación muy alta**, por lo que, en ese caso, si se detecta cierto **patrón de colores** (más claros; no presentes en el resto de mundos) después de la primera colisión, el **comportamiento** del robot **cambia** y **comienza** a seguir la **ruta predeterminada** para dicho escenario.

A continuación, se muestra una **imagen** para **facilitar la comprensión** de los pasos anteriores:

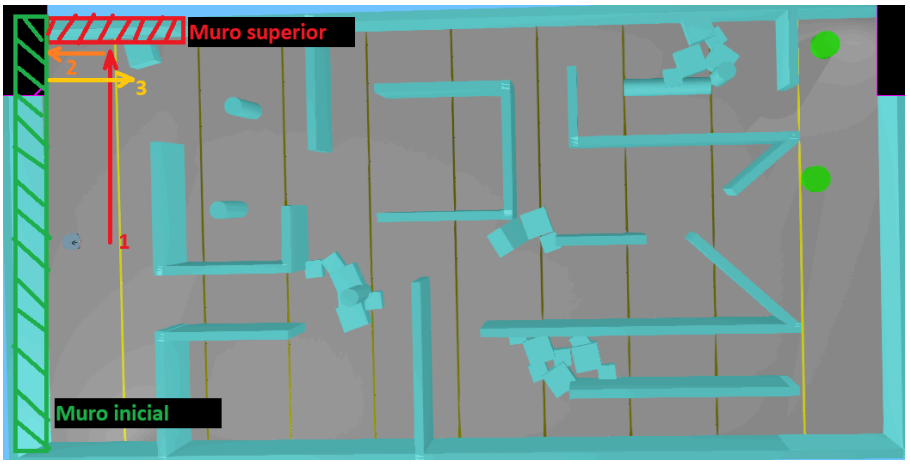

Figura 2.1. Descripción visual de los pasos iniciales

A continuación, se muestra un **diagrama de flujo** de alto nivel que describe este **comportamiento inicial**:

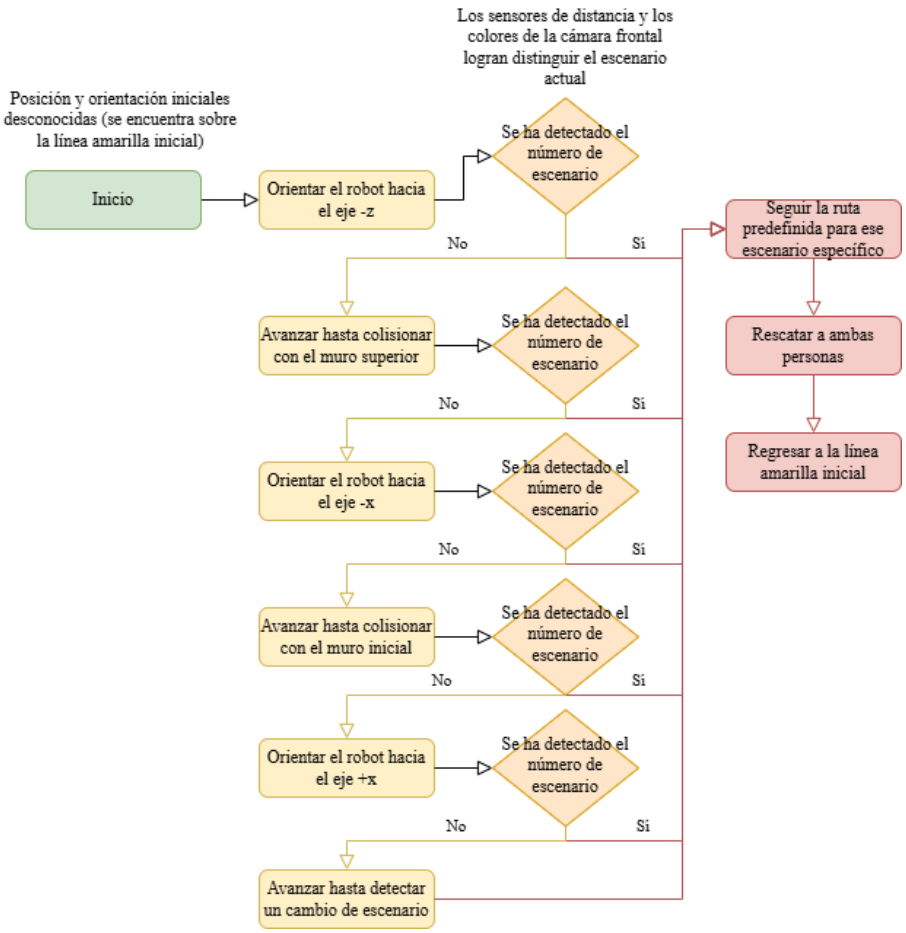

Figura 2.2. Diagrama de flujo de alto nivel de los pasos iniciales

Teniendo todo ello en cuenta, se recogen a continuación los **colores observados** durante la **primera colisión** (con el muro superior) y durante la **segunda colisión** (con el muro inicial):

	Primera colisión (muro superior)	Segunda colisión (muro inicial)
Escenario 01	(40,100,100) [20735 pixels] (40,110,110) [17665 pixels]	(50,120,120) [38400 pixels]
Escenario 02	(80,170,170) [38400 pixels]	(80,180,180) [38400 pixels]
Escenario 03	(30,80,80) [38400 pixels]	(50,120,120) [38400 pixels]
Escenario 04	(80,190,190) [38400 pixels]	(80,190,190) [38400 pixels]
Escenario 05	(80,190,190) [38400 pixels]	(80,190,190) [38400 pixels]
Escenario 06	(80,190,190) [38400 pixels]	(80,190,190) [38400 pixels]
Escenario 07	(80,190,190) [38400 pixels]	(80,190,190) [38400 pixels]
Escenario 08	(40,100,100) [38400 pixels]	(50,120,120) [38400 pixels]
Escenario 09	(160,250,250) [38400 pixels]	(210,250,250) [38400 pixels]
Escenario 10	(40,100,100) [38400 pixels]	(50,120,120) [38400 pixels]

Tabla 2.2. Colores observados después de las colisiones con los muros superior e inicial

Como se puede observar, algunos escenarios tienen los **mismos colores** (como los mundos 04, 05, 06 y 07 - o 01, 08 y 10), mientras que algún escenario tiene **patrones de colores únicos** (como el escenario 09).

A continuación, se recogen los **cinco colores más comunes** observados en cada escenario **después de completar los pasos iniciales** (orientar el robot hacia el muro superior, avanzar hasta chocarse, girar hacia el muro inicial, avanzar hasta chocarse y girar 180º):

E01	(40,110,110) [11922 pixels]	(30,90,90) [8793 pixels]	(80,80,80) [7840 pixels]	(30,80,80) [3894 pixels]	(40,90,90) [3136 pixels]
E02	(80,180,180) [9914 pixels]	(50,110,110) [8623 pixels]	(120,120,120) [7662 pixels]	(80,190,190) [3550 pixels]	(60,140,140) [2124 pixels]
E03	(60,140,140) [16006 pixels]		(50,130,130) [2125 pixels]	(120,120,120) [2121 pixels]	(60,150,150) [1964 pixels]
E04	(90,190,190) [14173 pixels]	(40,100,100) [13939 pixels]	(100,100,100) [8726 pixels]	(60,140,140) [497 pixels]	(190,190,0) [213 pixels]
E05	(40,100,100) [15383 pixels]	(90,190,190) [14066 pixels]	(100,100,100) [8432 pixels]	(190,190,0) [213 pixels]	(200,200,0) [85 pixels]
E06	(90,190,190) [15329 pixels]	(100,100,100) [9849 pixels]	(80,140,140) [4422 pixels]	(70,140,140) [3139 pixels]	(40,100,100) [1590 pixels]
E07	(90,190,190) [15081 pixels]	(100,100,100) [9924 pixels]	(100,170,250) [2823 pixels]	(110,150,150) [1692 pixels]	(110,140,140) [1479 pixels]
E08	(40,110,110) [11280 pixels]	(80,80,80) [9557 pixels]	(40,100,100) [3672 pixels]	(100,170,250) [3336 pixels]	(50,120,120) [2197 pixels]
E09	(160,250,250) [12981 pixels]	(20,40,40) [12516 pixels]	(170,170,170) [8650 pixels]	(20,50,50) [1409 pixels]	(170,250,250) [1240 pixels]
E10	(40,110,110) [10809 pixels]	(80,80,80) [8127 pixels]	(30,90,90) [7752 pixels]	(40,100,100) [4183 pixels]	(30,80,80) [3128 pixels]

Tabla 2.3. Colores más comunes observados después de los pasos iniciales

Como se puede apreciar, cada escenario tiene una **configuración de colores observada distinta** a la del resto de mundos. Por ejemplo, el escenario 09 se puede **identificar muy fácilmente** ya que tiene **colores mucho más claros** que los demás. De igual manera, en los escenarios 04 y 05 existen **tonalidades de amarillo** debido a que se observa la **línea amarilla inicial**.

Ruta predeterminada para cada escenario

Una vez se ha **identificado** correctamente el **escenario actual**, se define en el controlador la **ruta exacta** que debe seguir el robot. Para ello, se diseñan **cuatro acciones principales** que permiten la **creación de rutas precisas** en cualquier escenario:

1. *MOVE_FORWARD*: Moverse **hacia delante durante un tiempo** determinado.
2. *TURN_LEFT*: **Girar a la izquierda un ángulo** determinado.
3. *TURN_RIGHT*: **Girar a la derecha un ángulo** determinado.
4. *WAIT*: **Esperar durante un tiempo** determinado; utilizado después de avanzar o realizar un giro para permitir la **estabilización de los motores y de las ruedas**, ya que, **de lo contrario**, las nuevas señales de dirección y velocidad enviadas a las ruedas **no darían tiempo** a las anteriores señales **para terminar**. Por ejemplo, si se realiza un giro y automáticamente después el robot comienza a avanzar recto, el resultado será que el robot **girará un poco más de lo deseado**, debido a que las ruedas tenían cierta velocidad y dirección orientadas hacia el giro cuando, de repente, en lugar de parar **cierto tiempo para ayudar a estabilizarse**, han continuado recto, continuando el movimiento anterior (giro) antes de estabilizarse poco tiempo después (y avanzar correctamente sin girar).

En esta **ruta predeterminada** también se define el r**escate de las personas** y el **giro sobre sí mismo** (equivalente a cuatro giros seguidos de 90º en el mismo sentido). El robot **no los reconoce como personas** (ni su forma cilíndrica ni su color); debido a su **posición estática** en todos los escenarios, estas personas son simplemente **un paso más** en la ruta a seguir, aunque **si su posición fuera variable o dinámica** el robot podría reconocerlos fácilmente dado que su color **los identificaría** (ningún escenario ni elemento del entorno tiene el mismo color).

Por último, al llegar a la **línea de salida después** de haber **rescatado** a las personas, simplemente hay que **parar el robot** (no hace falta quitar los ***prints*** de la consola ni provocar un ***crasheo*** del programa).

También cabe destacar que, como se ha mencionado anteriormente, el **robot** siempre **empieza sobre** cualquier punto de la **línea amarilla inicial** - por lo que se ha **asumido** que se encuentra en cualquier punto de las **coordenadas** (*x*, *y=0*, ***z=8***), donde ***x*** puede tomar **cualquier valor** que posicione el robot **dentro del escenario** (sin atravesar un muro al mismo tiempo). Esta asunción fue aprobada y confirmada por el profesor.

Cabe destacar que algunos **escenarios** tienen **propiedades únicas**. Por ejemplo, en el escenario 05 hay un **cubo** encima de la **línea amarilla inicial**, obstaculizando el paso del robot - por lo que, para este mundo, se ha ideado una manera para **diferenciar** este **obstáculo** del **muro superior** al detectar un objeto en frente mediante los **sensores de distancia**. De igual manera, en el **escenario 10** las dos personas a rescatar se encuentran **muy próximas** entre sí, por lo que se podría pensar que sería suficiente con colocar el robot a una **distancia adecuada de ambas** (un metro máximo) y dar **una sola vuelta** - ahorrando tiempo. Sin embargo, para indicar de manera efectiva que se ha salvado a **ambas personas** y evitar confusiones, se decide dar **dos vueltas** - ya que, además, la ventaja de ahorrar una vuelta **no** supone una **gran diferencia**. Además, en el **escenario 03** la **ejecución** del programa tarda mucho **más tiempo** en comparación al resto de mundos, ya que se utilizan **luces**, **sombras** y juegos de iluminación en distintos lugares del mundo - **ralentizando la aplicación**.

Por tanto, teniendo en cuenta el **comportamiento inicial** descrito anteriormente, a continuación se explica con **más detalle** la manera de **diferenciar** cada uno de los **escenarios**.

Escenario 01

Para reconocer el escenario 1, se implementó una estrategia dividida en dos fases. Durante la primera fase, el robot utiliza una función de detección de escenarios basada en el análisis de colores dominantes en la imagen capturada por la cámara frontal. Esta función agrupa inicialmente los escenarios 1, 8 y 10 bajo una misma categoría denominada "01y08y10", ya que todos comparten características cromáticas similares, específicamente cuando se detectan los tonos RGB (50, 120, 120) y (40, 100, 100) como colores predominantes en la escena. Al identificar esta combinación de colores, el programa activa la bandera isScenario01y08y10Detected y cambia el estado de navegación.

Una vez que se ha determinado que el robot se encuentra en uno de estos tres escenarios posibles, se inicia la segunda fase de detección para diferenciarlos con más precisión. El robot primero se reorienta hacia el eje negativo X mediante la función turnToOrientation, permitiendo una observación más detallada del entorno. Tras completar esta maniobra, el controlador cambia al estado DET_01_08_10, donde el robot avanza para recopilar datos adicionales que permitan una clasificación más precisa.

La diferenciación final entre los escenarios se realiza mediante un análisis combinado de datos visuales y sensoriales. El programa verifica la cantidad de píxeles del color específico (40, 110, 110), definido como Color01_08_10. Si este valor supera los 14,000 píxeles, se identifica como escenario 8. En caso contrario, el sistema analiza las lecturas del sensor de distancia frontal (DS0) ya que se trata del escenario 1 o 10. El escenario 1 se confirma cuando se cumplen simultáneamente tres condiciones: el conteo de píxeles del color característico es menor a 14,000, el sensor DS0 detecta un objeto a más de 900 unidades de distancia, y además el sensor lateral DS6 está activo (con valor superior a cero).

Una vez identificado el escenario 1, el robot detiene su movimiento mediante la función stopRobot y actualiza su estado a SCENARIO_01, permitiendo la ejecución de la ruta diseñada para este escenario.

Escenario 02

El escenario 2 presenta características distintivas en esta implementación. El sistema utiliza la función *detect_scenario()* que analiza los colores dominantes capturados por los sensores del robot. A diferencia de otros escenarios que requieren múltiples colores o patrones complejos, el escenario 2 se caracteriza por presentar un único color dominante con valores RGB (80, 180, 180). Esto fue fácil si comparamos con otros escenarios con colores dominantes comunes así como 01_08_10 y 04_05_06_07. Esta singularidad cromática facilita su identificación precisa, ya que la función verifica explícitamente que solo exista este color en la escena mediante la comprobación de colors.size() == 1.

El programa implementa una variable booleana específica, isScenario02Detected, que se activa cuando el sistema confirma la detección del escenario 2. Esta variable sirve para indicar al robot de navegación que debe cambiar su comportamiento y adaptarse a las particularidades de este escenario.

La transición al modo de navegación del escenario 2 ocurre durante la fase TO_NEG_X del estado predeterminado. Antes de realizar esta transición, se tiene que asegurar que el robot esté correctamente orientado hacia el eje negativo X, garantizando así una posición inicial adecuada para comenzar la secuencia de movimientos específica del escenario 2. Después de activar el modo del escenario dicho, el sistema implementa la secuencia de movimientos propia para poner a correr al robot.

Escenario 03

Para el escenario 3 se realiza su detección mediante la función *detect_scenario()*, que implementa una lógica específica para identificar este escenario a través de combinaciones de colores distintivas. A diferencia del escenario 2 que se identifica por un único color, el escenario 3 se caracteriza por presentar múltiples tonalidades en combinación con elementos de color negro. Específicamente, la función busca colores con valores RGB (70, 160, 160) o (60, 140, 140) y verifica la presencia de pixeles negros (0, 0, 0) entre los primeros cinco colores dominantes capturados por los sensores del robot.

El sistema implementa una variable booleana específica, isScenario03Detected, que actúa como indicador cuando se confirma la detección de este escenario particular. Esta bandera permite al sistema de navegación adaptar su comportamiento a las características del escenario 3.

Adicionalmente, el sistema realiza una verificación específica para detectar píxeles negros durante la fase TO_NEG_X del estado predeterminado. Cuando se detecta una cantidad significativa de píxeles negros (más de 1000), el sistema determina que se encuentra en el escenario 3 y realiza la transición al estado correspondiente, el cual trata la ruta que debe seguir el robot cuando comience a avanzar.

Escenario 04

La identificación del escenario 4 en el sistema de detección sigue un proceso específico basado en características cromáticas y sensoriales. Inicialmente, durante el análisis de la imagen de la cámara frontal, el sistema detecta el color dominante RGB (80, 190, 190), que activa la identificación "preliminar" del grupo "04y05y06y07". Una vez que la función *detect_scenario()* identifica el grupo general "04y05y06y07", el sistema activa la bandera isScenario04y05y06y07Detected

Posteriormente, cuando el robot completa su secuencia de navegación inicial (orientación hacia -Z, colisión, orientación hacia +X, colisión, orientación hacia -X), el sistema verifica esta bandera en el caso TO_NEG_X para determinar si debe cambiar al estado SCENARIO_04_05_06_07.

En el estado SCENARIO_04_05_06_07 el robot se reorienta hacia el eje negativo X mediante la función turnToOrientation, que utiliza los datos del compás (_compass) para alcanzar la orientación deseada con una tolerancia predefinida. Una vez completada esta reorientación, el sistema transita al estado DET_04_05_06_07 y registra el tiempo actual como punto de referencia en la variable turn_start_time.

El estado DET_04_05_06_07 implementa la distinción final entre los escenarios 4, 5, 6 y 7. Después de una pausa de un segundo (para estabilización), el robot avanza utilizando la función moveForward con una velocidad de 10.0 unidades. Durante este avance, el sistema va monitoreando continuamente el conteo de píxeles del color predefinido Color04_05_06_07 (RGB 40, 100, 100).

La identificación específica del escenario 4 ocurre cuando se cumplen dos condiciones simultáneamente: El conteo de píxeles del color Color04_05_06_07 supera el umbral de 38000 píxeles y el sensor de distancia DS2 reporta un valor MENOR a 740 unidades. Al confirmarse esto, el sistema detiene el robot mediante la función stopRobot y actualiza la variable scenarioState al valor SCENARIO_04, donde el robot ejecutará la ruta diseñada para su entorno.

Escenario 05

El sistema de detección del escenario 5 presenta una característica distintiva dentro del programa, ya que incorpora dos variantes identificadas como escenario 5 (caso 0) y escenario 5_1 (caso 1), cada una con sus propias identificaciones. El tema está en que en el mundo 5 hay un cubo en medio de la línea de salida, que si partimos de la derecha del mismo no deja continuar con la mini ruta preestablecida para el reconocimiento que hacemos normalmente.

Entonces, decidimos utilizar dos “modus operandi” para el robot en este escenario, uno en el caso de estar a la izquierda del cubo y otro cuando está a la derecha del cubo (no hay más ya que no se puede estar solapando con el cubo). Conforme a esto, el momento de detección identifica el caso 0 tras completar la navegación inicial (cuando está a la izquierda del cubo), mientras que el caso 1 se identifica tempranamente durante el primer movimiento hacia el eje negativo Z (cuando está a la derecha del cubo y no detecta la pared habitual, sino los colores que ofrece el cubo).

Para el escenario 5 (caso 0), el proceso de identificación comienza durante el análisis cromático realizado por la función *detect_scenario()*. Esta función detecta inicialmente el color dominante RGB (80, 190, 190) en la imagen capturada por la cámara frontal del robot, lo que resulta en el identificador "04y05y06y07", agrupando este escenario con otros tres que comparten un patrón cromático similar. Esta identificación preliminar activa la bandera booleana isScenario04y05y06y07Detected.

Después de completar la secuencia de navegación inicial, cuando la bandera está activa, el controlador cambia al estado SCENARIO_04_05_06_07, donde el robot se reorienta hacia el eje negativo X mediante la función turnToOrientation. Una vez reorientado, el sistema transita al estado DET_04_05_06_07 para realizar una discriminación más detallada.

En el estado DET_04_05_06_07, después de una pausa de estabilización, el robot avanza mientras el sistema monitorea el conteo de píxeles del color Color04_05_06_07 (RGB 40, 100, 100). La identificación específica del escenario 5 (caso 0) ocurre cuando se cumple que el conteo de píxeles del color Color04_05_06_07 supera el umbral de 38000 píxeles y el sensor de distancia DS2 reporta un valor MAYOR a 740 unidades. Es justo ahí cuando se muda al SCENARIO_05 y realiza su ruta predeterminada.

Para el escenario 5_1 (caso 1), el proceso de identificación es notablemente diferente y ocurre mucho más temprano en la secuencia de navegación. Durante el análisis cromático inicial, la función *detect_scenario()* reconoce directamente el color dominante RGB (70, 170, 170), característico exclusivamente del escenario 5 en su caso 1 (por ser del cubo). Esta identificación activa la bandera isScenario05Detected, que luego se evalúa durante el estado MOVE_TO_CRASH_Z de la secuencia de navegación inicial. Si se detecta una colisión mientras esta bandera está activa, el sistema transita directamente al estado DET_SCENARIO_05. Ya en este estado, el robot se reorienta nuevamente y luego transita al estado especializado SCENARIO_05_1, que continuará con la nueva ruta preestablecida que tiene el escenario 5 (esta vez a la derecha del cubo).

Escenario 06

El escenario 6 se identifica inicialmente como parte de un grupo de escenarios similares (04, 05, 06 y 07) que comparten características comunes. En el código, cuando el robot detecta que podría estar en uno de estos escenarios, cambia al estado SCENARIO_04_05_06_07 y luego al estado DET_04_05_06_07 para realizar una discriminación más precisa. Cabe recalcar que el escenario 6 se relaciona con el color constante Color06 definido como RGB (90, 190, 190) y con un color para el grupo de escenarios al que pertenece, Color04_05_06_07, definido como RGB (40, 100, 100).

Se analiza primero si colorCount[Color04_05_06_07] debe tener menos de 38000 píxeles del color del grupo, tal y como colorCount[Color06] debiendo tener más de 14600 píxeles del color específico del escenario 06. Hecho esto, se comienza por ver si el sensor de distancia DS6 no detecta obstáculos y si colorCount[Color06] ha cambiado con el tiempo, debiendo tener más de 15367 píxeles del color específico.

Cuando el programa identifica de manera concluyente que está en el escenario 6, cambia su estado a SCENARIO_06 y detiene el robot con la función stopRobot. Ya luego se proseguirá con su ruta característica.

Escenario 07

En el caso del escenario 7, lo identificamos inicialmente como parte del grupo de escenarios similares (04, 05, 06 y 07) que comparten las mismas características ya mencionadas anteriormente. Para cuando el robot cree que podría estar en uno de estos escenarios, cambia al estado SCENARIO_04_05_06_07 y después al estado DET_04_05_06_07 para identificar mejor el mundo.

Se vuelve a utilizar la misma condición que en el escenario anterior, es decir, se usa Color04_05_06_07 para comprobar que debe tener menos de 38000 píxeles del color. También, se debe evaluar que el color constante Color06 definido como RGB (90, 190, 190) debe tener más de 14600 píxeles de color, para decidir si se está en el escenario 6 o 7. Al comparar nuestro escenario con el 6, se nota que la diferencia radica en que la cantidad de píxeles de Color06 va disminuyendo hasta que cumple la condición de tener menos 15050 píxeles. Además, si analizamos el código se podría pensar que existe otra diferencia en que no se utiliza el sensor DS6, pero no puede ser el caso porque cuando el robot avanza detecta algunas posibles colisiones que no importan ya que el objetivo se logra cuando llegan al rango de colores específico que se diseñó.

Una vez el programa concluye que está en el escenario 7, se cambia al estado SCENARIO_07 y detiene el robot con la función stopRobot, esperando a continuar con la ruta que le pertenece.

Escenario 08

Para tratar el escenario 8, utilizamos la misma estrategia que en el caso del escenario 1. Se agrupan inicialmente los escenarios 1, 8 y 10 bajo la categoría "01y08y10", que contiene los tonos característicos RGB (50, 120, 120) y (40, 100, 100) como colores predominantes en la escena. Se activa la variable booleana isScenario01y08y10Detected y cambia el estado de navegación. Luego, el robot se gira hacia el eje negativo X con la ayuda de la función turnToOrientation y se traslada al estado DET_01_08_10, donde el robot avanzará a través del tiempo con el fin de poder distinguir mejor los tres escenarios.

Revisamos que si el valor (40, 110, 110) del Color01_08_10 no supera los 14,000 píxeles, se trata del escenario 1 o 10, pero si se supera estamos en el escenario 8. El robot detiene su avance de identificación y la condición se mueve a SCENARIO_08 para evaluar la ruta que debe obedecer.

Escenario 09

La detección del escenario 9 comienza dentificándose principalmente por la presencia del color específico definido como SCENARIO_09_COLOR con valores RGB (210, 250, 250). El sistema utiliza una variable constante declarada al inicio del programa para almacenar estos valores cromáticos, facilitando su referencia durante el proceso de detección.

Para rastrear la identificación de este escenario particular, el programa implementa una variable booleana específica, isScenario09Detected, que actúa como indicador cuando se confirma la presencia del escenario 9. A diferencia de otros escenarios donde la detección se procesa principalmente a través de la función *detect_scenario()*, el escenario 9 utiliza un enfoque directo donde el sistema busca el color específico en los datos capturados por los sensores.

Una característica distintiva en la detección del escenario 9 es que se verifica continuamente durante la segunda fase de colisión del robot, específicamente durante el estado MOVE_TO_CRASH_X. Cuando se detecta este escenario durante esta fase, el sistema interrumpe inmediatamente el proceso de navegación predeterminado y cambia al modo específico del escenario 9, deteniendo el robot para iniciar la secuencia especializada.

Escenario 10

Por último, en el escenario 10 se prosigue con la estrategia comentada anteriormente en el escenario 1 y 8. Se detecta que estamos en el escenario 01y08y10 debido a la función *detect_scenario()* que pone en común sus colores RGB (50, 120, 120) y (40, 100, 100), poniendo a true la condición isScenario01y08y10Detected.

Después, se mueve al robot para que esté orientado en dirección al eje negativo X, se traslada al estado DET_01_08_10 y comienza a avanzar para identificar mejor las características del mundo. Para clarificar sus diferencias con los otros dos escenarios, definimos Color01_08_10 con (40, 110, 110) como parámetro para observar que si se superan los 14,000 píxeles del mismo, se trata del escenario 8, pero si no lo logran, estamos en el escenario 1 o 10.

Este escenario solo se puede confirmar si el conteo de píxeles de Color01_08_10 característico es menor a 14,000, no se detecta el sensor DS6 y el sensor DS0 no detecta ningún objeto a más de 900 unidades de distancia. Esta diferencia con el escenario 1 se realiza porque al ir avanzando el robot, ambos escenarios detectan algo con el escenario DS0, pero en tiempos distintos y es por eso que recurrimos a la distancia recorrida para esta diferenciación.

Finalmente, el escenario 10 detiene al robot y actualiza su estado a SCENARIO_10, pasando a ejecutar la ruta específica.

3. Metodología y resultados

Cabe mencionar que, durante las **fases iniciales** del desarrollo y programación del controlador, se descubrió un ***bug* o error** del programa, confirmado también por el profesor. Esto ocurre al **modificar la orientación o posición** del robot mediante la **interfaz gráfica** (utilizando las **flechas axiales** o dimensionales), dando lugar a **comportamientos inesperados** o **incorrectos**. Sin embargo, este problema parece **no existir** al modificar dichos parámetros **manualmente** mediante el **menú de la izquierda** (llamado árbol de escena).

Tabla 3.1: Métricas de rendimiento del sistema robótico en distintos escenarios

Escenario	Personas rescatadas	Número de colisiones	Promedio de tiempos	Distancia recorrida
1	2	2	0:02:08:15	56.5991 metros
2	2	2	0:02:14:68	66.7763 metros
3	2	2	0:02:09:08	59.216 metros
4	2	2	0:02:00:22	50.8064 metros
5 (caso 0)	2	2	0:02:14:20	62.7479 metros
5 (caso 1)	2	1	0:01:58:04	55.3945 metros
6	2	2	0:02:09:31	57.9491 metros
7	2	2	0:01:55:87	54.6613 metros
8	2	2	0:01:48:54	50.2953 metros
9	2	1	0:01:48:99	51.5106 metros
10	2	2	0:02:19:13	61.893 metros

Tabla 3.1: Métricas de rendimiento del sistema robótico en distintos escenarios

En la **tabla** se puede observar cómo **rescatamos a todas las víctimas**, sin excepción alguna. Partimos de que al tratarse de la distancia recorrida con las líneas negras atravesadas, nuestro robot las atraviesa todas debido a que llega hasta la otra punta del mapa, salva a las personas, y regresa al punto de origen (recorriendo la **misma cantidad de líneas negras** que ya había atravesado).

Respecto a las colisiones, se observa que en la mayoría de escenarios (9 de 11 casos) el robot sufre exactamente **dos colisiones**. Estas ocurren principalmente durante la fase inicial de identificación del entorno. **La primera colisión** se produce generalmente con la pared ubicada en dirección -Z desde la perspectiva inicial del robot, excepto en el escenario 5 (caso 1) donde la colisión inicial ocurre con un cubo situado justo en esa dirección. Solo en los

escenarios 05 (caso 1) y 09 el sistema consigue identificar completamente el mundo tras la primera colisión, lo que explica que sean los únicos con **una sola colisión** registrada.

Los **tiempos de rescate** presentan variaciones entre los distintos escenarios, oscilando entre 1:48:54 (escenario 8) y 2:19:13 (escenario 10). En ningún caso se **superan los 4 minutos** especificados en el enunciado, aunque dependiendo del computador que corra el programa, el robot puede tardar más en la vida real por **ralentizarse mucho Webots**.

También se detalla la **distancia total recorrida**, calculada mediante los ***encoders*** de las **ruedas**. Las distancias van desde los **50 metros** hasta los **67 metros** de media, variando mucho dependiendo del escenario. Esta diferencia de más de 16 metros (aproximadamente un **33% adicional**) entre el recorrido más corto y el más largo refleja la variabilidad en la complejidad de los entornos y en las rutas seleccionadas por el sistema.

El siguiente **enlace** contiene la carpeta de *Google Drive* con **imágenes y vídeos** de cada escenario solucionado. Se ha otorgado a jgamboa@ing.uc3m.es acceso de lector a esta carpeta:

https://drive.google.com/drive/u/0/folders/150ycpwULFLQwNxb68UjKvmoejK9mr7B8

Adicionalmente, **creamos** una carpeta en el drive llamada **Pruebas** dentro de Fotos y Videos que tiene **grabaciones** de los escenarios con **distintas orientaciones** iniciales. Hicimos varias pruebas exhaustivas, pero para las grabaciones decidimos coger estos **parámetros**, por ser bastante **diferenciales**:

Escenario 10

Caso 1 → Rotation 0 1 0 3.66519

Caso 2 → Rotation 0 1 0 2.48709

Caso 3 → Rotation 0 1 0 1.17809

Caso 4 → Rotation 0 1 0 0.26179

Escenario 09

Caso 1 →4.18879

Caso 2 →1.4399

Caso 3 →2.3562

Escenario 08

Caso 1 →4.05789

Caso 2 →0.5236

Caso 3 →1.309

Escenario 07

Caso 1 →2.2253

Caso 2 →

Caso 3 →

Caso 4 →

Escenario 06

Caso 1 →

Caso 2 →

Caso 3 →

Caso 4 →

Escenario 05

Caso 1 →

Caso 2 →

Caso 3 →

Caso 4 →

Escenario 04

Caso 1 →3.93535

Caso 2 →2.09439

Caso 3 →1.43989

Caso 4 →0.26179

Escenario 03

Caso 1 →2.8798

Escenario 02

Caso 1 →2.61799

Caso 2 →0.3927

Caso 3 →1.309

Escenario 01

Caso 1 →0.2618

Caso 2 →1.0472

Caso 3 →2.2253

Caso 4 →4.05789

4. Discusión

Tras haber expuesto de manera objetiva los **resultados obtenidos** en el apartado anterior, en el **capítulo actual** se discute razonadamente y se **especula** de forma **subjetiva** acerca de ellos.

Los **datos de la tabla** muestran una tasa de **éxito del 100%** en el rescate de víctimas, ya que en todos los escenarios se consiguió rescatar a las dos personas objetivo. El simple hecho de conseguir el objetivo demuestra la **robustez** del algoritmo de navegación y rescate implementado. Sin embargo, analizaremos en **profundidad** el comportamiento del robot en las diferentes situaciones para identificar tanto **fortalezas** como **debilidades**.

Principalmente, la **cantidad de colisiones** que maneja el robot ocurren al principio del programa, cuando se **trata de identificar el mundo**. Esta **fase inicial** resulta crítica para el rendimiento global del sistema, ya que determina la estrategia de navegación que se utilizará posteriormente. El proceso de identificación **sigue un patrón sistemático**:

1. El robot tiene una **colisión controlada con la pared en dirección -Z** para obtener información inicial (se da en todos los escenarios, excepto en el 05 (caso 1) por tratarse del escenario con un cubo en medio de la línea de salida.

2. Basándose en esta información, el robot **tiene la oportunidad de reconocer un mundo muy diferenciado** o de creer que aún le falta más información para hacerlo y terminar **agrupando ese mundo junto a otros** con resultados similares.

3. **En la mayoría de los casos** (9 de 11), el sistema **requiere una segunda colisión**, esta vez **avanzando en dirección +X** hasta colisionar de nuevo con la pared del eje indicado (muro inicial).

4. **Tras la segunda colisión**, el robot gira y recoge datos adicionales mientras **se desplaza por el eje -X**. Es este pequeño recorrido el responsable de hacer que muchos escenarios detecten de primeras el mundo al que pertenecen, como es en el caso del mundo 02, 03 y 06. Otros escenarios avanzan un poco más y lo detectan en el camino como el escenario 07 y el 08. Y por último, los restantes avanzan hasta detectar mayor información de la pared más próxima del eje -X, siendo los escenarios 01, 04, 05 (caso 0) y 10 sus representantes.

Este **método de exploración**, aunque genera **colisiones controladas**, permite al robot construir un **mapa mental** del entorno que resulta crucial para la planificación de rutas eficientes. Creemos que las colisiones durante esta fase son un compromiso aceptable, dado que son limitadas (**máximo 2**) y ocurren a velocidad controlada, minimizando cualquier daño potencial al robot o al entorno.

Los **tiempos de rescate** varían dependiendo de los escenarios. Hemos representado las tres variables que influyen en éstos de la siguiente manera:

1. **Gráfica del entorno**: Es importante notar que los escenarios con mejores gráficos (como el 02 y el 03) muestran tiempos de rescate más prolongados. Esto probablemente se debe a la mayor carga computacional que estos entornos suponen para el sistema, lo que ralentiza el procesamiento de la información sensorial y la toma de decisiones.

2. **Densidad de obstáculos**: Los escenarios con mayor número de obstáculos (como el 02, 03 y 06) muestran tiempos ligeramente superiores, aunque sin ser tan diferencial como se esperaba. Esto nos deja pensando que nuestro algoritmo de planificación de rutas maneja relativamente bien los entornos complejos, pero aún existe margen de mejora en este aspecto.

3. **Tiempo de identificación del entorno**: Los escenarios donde el robot tarda más en identificar completamente el mundo (como el 01, 05 (caso 0) y 10) muestran tiempos totales mayores. Esto es lógico, ya que el robot no puede iniciar su ruta ideada hasta haber completado esta fase de reconocimiento.

Es interesante observar que los escenarios 05 (caso 1) y 09, donde el robot logra identificar el mundo con **una sola colisión**, no son necesariamente los más rápidos en términos de tiempo total. Podemos creer que, aunque la fase de identificación es importante, otros factores como la **complejidad del entorno** y la **ruta de rescate** tienen un impacto más significativo en el tiempo total de la misión.

Para las **distancias** recorridas se puede ver que existe una correlación positiva entre las mismas y el tiempo empleado en la misión. Los escenarios con **menores distancias** (08 y 04, con aproximadamente 50 metros) presentan **tiempos más reducidos** (alrededor de 1:48-2:00 minutos), mientras que aquellos con **mayores recorridos** (02 y 10, con más de 60 metros) muestran **tiempos más prolongados** (superiores a 2:14 minutos).

Si recapitulamos con los escenarios de **una sola colisión**, éstos no son necesariamente los que presentan las distancias más cortas. Aun así, muestran una buena relación tiempo-distancia, lo que sugiere que la **temprana identificación** del entorno permite planificar **rutas más eficientes**.

Es destacable notar que el escenario 02 presenta la **mayor distancia** recorrida (66.77 metros) y uno de los **tiempos más elevados** (2:14:68), mientras que el escenario 08 muestra la **menor distancia** (50.30 metros) y el **menor tiempo** (1:48:54). Lo cual nos podría llevar a pensar en una especie de **proporción**, pero esta **no se mantiene constante** en todos los casos, lo que indica que otros **factores**, como la **velocidad variable** del robot y los mencionados anteriormente, influyen más en el **rendimiento**.

Además, los escenarios donde el robot requiere **dos colisiones** para identificar el entorno tienden a presentar **distancias mayores**, ya que la fase inicial de exploración implica un **recorrido adicional** antes de **iniciar la ruta óptima** hacia los objetivos.

A continuación, comentamos lo que creemos son las fortalezas y debilidades de nuestro sistema.

Fortalezas:

1. **Fiable en el rescate**: El sistema cubre el 100% de las víctimas, en cualquier escenario, lo que confirma la robustez del algoritmo de navegación y rescate.

2. **Método sistemático de exploración**: Aunque implica colisiones controladas, el método de exploración implementado permite una identificación eficaz del entorno en todos los escenarios probados.

3. **Eficiencia razonable**: En la mayoría de los escenarios, la relación entre distancia recorrida y tiempo empleado muestra una eficiencia aceptable, con velocidades medias que oscilan entre 0.4 y 0.5 metros por segundo.

Debilidades:

1. **Dependencia de colisiones para la identificación**: el método actual requiere colisiones controladas para la identificación del entorno. Si bien estas son limitadas y controladas, representan un enfoque que podría optimizarse para eliminar completamente las colisiones.

2. **Variabilidad en los tiempos de rescate**: existe una variación significativa en los tiempos de rescate entre diferentes escenarios, lo que sugiere que el algoritmo de planificación de rutas podría optimizarse para ser más consistente.

3. **Sensibilidad a la calidad gráfica**: la ralentización observada en entornos con mejores gráficos indica una limitación en la capacidad de procesamiento que podría abordarse mediante optimizaciones en el código o mejoras en las computadoras utilizadas.

4. **Tiempo de identificación variable**: el tiempo que tarda el robot en identificar completamente el entorno varía considerablemente entre escenarios, lo que impacta en el tiempo total de la misión.

5. **Adaptabilidad**: el sistema no denota gran capacidad para adaptarse a entornos más reales, debido a nuestro mayor enfoque en planificación de rutas, antes que en brindarle libertad y aprendizaje propio al robot.

6. **Variabilidad de las condiciones iniciales**: el robot puede obtener resultados distintos en función de la orientación y posición iniciales. Esto se debe a la rigidez de las rutas y la precisión mejorable de las funciones de orientación y movimiento. Por ejemplo, si los giros del robot fueran prácticamente infalibles y de muy alta precisión, estas condiciones iniciales no afectarían al resultado final.

En conjunto, nuestro sistema robótico de rescate demuestra un **rendimiento sólido** en términos de fiabilidad para completar la **misión principal** de rescate. Sin embargo, reconocemos las **limitaciones actuales**, particularmente en lo relativo al método de **identificación del entorno**, la optimización de los **tiempos de rescate** y la poca veracidad para probarse en un **escenario real**.

5. Conclusiones

Ciertamente, el robot podría utilizar más **sensores** u otros distintos. Sin embargo, estos **no son aparentemente necesarios** al utilizar la **aproximación desarrollada** ni serían especialmente útiles a la hora de **optimizar el tiempo** de rescate. Si bien se podría utilizar el sistema ***GPS*** para intentar **localizar** el robot (aún teniendo en cuenta su baja precisión) o para **corregir los errores** acumulados de la odometría, seguiría siendo **menos útil** que utilizar una **ruta fija** para cada mundo. Además, dado que únicamente son **10 escenarios** (no 100 o 1000), el **enfoque utilizado** obtiene **mejores resultados** y rendimiento. Por ejemplo, un **controlador genérico** podría funcionar mejor para **escenarios** aún no vistos o **desconocidos**, como un algoritmo ***sigue-paredes*** sencillo, pero podría quedarse atrapado con un obstáculo, sería **menos eficiente** y tardaría más tiempo.

También se consideró utilizar la **cámara esférica (360º)** para mostrar por consola cuándo se **detectaba una línea amarilla** (inicial o final). De esta manera, se estarían utilizando **más sensores**, por lo que se estaría haciendo un mayor uso de las **capacidades del robot**. Sin embargo, se decidió **no llevarlo a cabo** ya que no suponía un gran **beneficio** en el presente estudio.

Siguiendo la **lista de comprobación** (también conocida como *checklist*) a seguir según el **enunciado**, se cumplen todos los **objetivos propuestos**:

- ✔ Se rescatan adecuadamente a las **dos víctimas**, realizando un **giro de 360º** a menos de un metro de distancia de cada persona
- ✔ Ida y vuelta del robot en menos de **cuatro minutos**
- ✔ **Sin modificaciones** del modelo del robot
- ✔ Se **optimiza el tiempo** requerido para cada escenario
- ✔ Con **menos de tres colisiones**

Respecto a **futuros estudios** que desearan proseguir con el presente trabajo, se **recomienda** tener en cuenta las **fortalezas y debilidades** recogidas en el anterior apartado de discusiones, haciendo también especial hincapié en la **mejora de las funciones de movimiento y orientación** del robot - ya que podrían obtener **resultados más precisos y predecibles**.