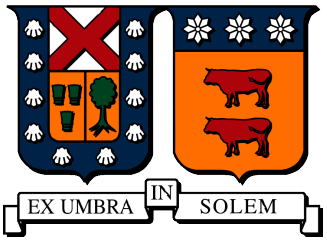

UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Departamento de Matemática
MAT279
Segundo Semestre 2020

Informe Final - Robot Path Planning

Optimización No Lineal

Nombres: Alan Grez - Cristóbal Lobos
Profesores: Julio Deride - Pedro Gajardo

Santiago, 7 de Enero de 2021

Índice

1. Motivación

Moverse de un lugar a otro es una acción trivial, en la mayoría de los casos, para los seres humanos. Nosotros como seres humanos no presentamos mayores en problemas en decidir un camino a recorrer desde un punto a otro, en particular, en menos de la mitad de un segundo el cerebro decide como moverse.

Para los robots esto no es para nada trivial, uno de los mayores problemas para los robots autonomos es tener un sentido de ubicación en un espacio dado. En particular, esto representa al **problema de navegación**, el cual puede resumirse en las siguientes preguntas.

- ¿Dónde estoy?
- ¿A dónde voy?
- ¿Cómo llego ahí?

La primera pregunta corresponde al problema de **localización**, la segunda corresponde al problema de **mapeo**, y la última corresponde al problema de **planificación de caminos**, en inglés, este último corresponde a **path planning**.

Este último consiste en encontrar el camino más corto que debe recorrer el robot desde un punto inicial a un punto final en un ambiente dado, el cual posee obstáculos. [1]

Este problema ha tenido diversas aplicaciones, por mencionar algunas, ha sido utilizado para autos autonómos [2], y vehículos aéreos no tripulados (VANTS) [3].

El problema de path planning, también llamado, **robot path planning**, ha sido estudiado y modelado de varias maneras, entre estas encontramos técnicas de control [4], y de grafos [5]. Y ha sido solucionado mediante varios tipos de algoritmos, tales como los genéticos [6], y evolutivos [7].

En particular, se estudió el problema para el caso de un plano en dos dimensiones con distintas distribuciones de obstáculos, y un modelo para un caso en tres dimensiones.

2. Modelamiento del problema

2.1. Definiciones generales

Matemáticamente, el problema puede ser descrito de la siguiente forma: [1]

Sea $W = \mathbb{R}^2$ ó $\mathbb{R}^3$. Consideramos que $A \subset W$ es un objeto rígido que se traslada a lo largo de W, además, $O \subset W$ denota a los obstáculos, que son objetos rígidos estacionarios.

La geometría, posición, y orientación de A y O son conocidas a priori. Además, la posición de O en W es conocida con precisión.

Por lo tanto, dadas unas posiciones iniciales y finales de $A \subset W$, buscamos un camino $P \subset W$ denotando al conjunto de posiciones tales que para todo $p \in P$, $A(p) \bigcap O = \emptyset$ a lo largo del camino de inicio a fin, por lo que, al terminar de resolver, puede existir el camino P o no, en este caso último caso, $P = \emptyset$. (Cabe destacar que este camino encontrado es el óptimo).

La calidad del camino puede estar sujeto a distintos criterios, como energía, distancia, tiempo de recorrido, entre otros. Es decir, la función objetivo puede estar sujeta a multiples variables, como las antes mencionadas.

2.2. Caso 3D

Planteamos un acercamiento diferente a RPP el cual se encuentra inspirado en viajes en auto a través de diferentes rutas desde un mismo punto inicial hacia un mismo punto final, por lo que en este caso el robot sería un auto. Queremos encontrar una ruta óptima para recorrer en un viaje dadas ciertas restricciones, las cuales serían los obstáculos. En particular, la ruta óptima que buscamos estará dada por un balance entre la distancia recorrida y el gasto de combustible asociado a la ruta. Además, consideramos que el auto puede recorrer toda la superficie excepto en donde se encuentran los obstáculos.

Hay varios factores que pueden afectar al movimiento del auto, lo cuales afectarían el gasto de combustible [8], tales como, el roce con el aire, la aceleración del auto, y el roce como el suelo. Por simplicidad, consideraremos nulo el roce con el aire, y una velocidad constante. Sin embargo, consideraremos el roce con el suelo.

Introduzcamos un poco de notación:

Sea $G : [0,1]^2 \to \mathbb{R}$ una función $C^1([0,1]^2)$, consideremos:

$$S = \{(x, y, G(x, y)) \in \mathbb{R}^3 \ / \ (x, y) \in [0,1]^2\},$$

en el cual definiremos los puntos de inicio y fin respectivamente como:

$$A := (0, 0, G(0, 0)), \quad B := (1, 1, G(1, 1))$$

Y definimos O, el conjunto de obstáculos que veremos más adelante, y $P := \Omega \backslash \Omega_O$, el conjunto de las trayectorias factibles, donde

$$\Omega := \{\gamma : [0,1] \to S \ / \ \gamma(0) = A, \ \gamma(1) = B, \ \gamma \in C^1([0,1])\}$$

$$\Omega_O := \{\gamma \in \Omega \ / \ \exists t \in [0,1], \ \gamma(t) \in O\}$$

Ahora, al momento de presentar el problema general, hablamos de una función de costo a minimizar, esta función va de la mano con la motivación de nuestro problema, i. e., una función que minimice

tanto la distancia recorrida como el gasto de combustible, luego, escribimos nuestro problema de la forma:

$$\min_{\gamma \in \Omega} F(\gamma) \qquad \text{ó} \qquad \min_{\gamma \in P} F(\gamma)$$
$$\textbf{sujeto a } \gamma \notin \Omega_O$$

con

$$F(\gamma) = C_d \int_\gamma dl + W \int_0^1 \left(\frac{\gamma' \cdot e_3}{\sqrt{(\gamma' \cdot e_3)^2 + 1}} + \frac{C_r}{\sqrt{(\gamma' \cdot e_3)^2 + 1}} \right) dt,$$

C_d constante relativa a la distancia y condiciones relativas al sólido que se desplaza y C_r relativas a la gasto de combustible.

El principal problema de querer resolver el problema, es que no se logra asegurar la existencia de un mínimo, puesto que se deben imponer más condiciones sobre el conjunto de obstáculos, tales como exigir cierta regularidad a la frontera y que sea un conjunto abierto, y aún así, no es posible asegurar nada de manera abstracta, se debería estudiar un caso en particular de manera profunda, lo cual escapa de nuestro objetivo, el cual es ver multiples configuraciones de obstáculos y ver de manera gráfica una camino factible.

Otra dificultad que se presenta es la de la implementación, es decir, no se encontraron en la literatura algoritmos que construyeran rutas suaves $3D$ en un ambiente con obstáculos, al menos, de la forma que nosotros necesitabamos.

2.3. Caso 2D

Veamos un modelo de Robot Path Plannning en $\mathbb{R}^2$, donde el robot es un punto en $\mathbb{R}^2$.

La idea es encontrar un camino óptimo desde un punto A a un punto B en un subconjunto de $\mathbb{R}^2$ que llamaremos mapa, donde el mapa consiste de una sección rectangular M del primer cuadrante del plano cartesiano, junto a un conjunto de obstáculos O que se encuentra adentro de esta sección rectangular.

Luego, de manera análoga al caso $3D$, definimos el conjunto de **caminos factibles** Ω como el conjunto de todas las curvas **simples y continuas** tales que tienen como punto inicial a A y como punto final a B, y que **no pasan por el conjunto obstáculos** O es decir:

$$\Omega = \{\gamma : [0,1] \to M/\ \gamma \textbf{ simple, continua; } ;\ \nexists t \in [0,1] \text{ t.q } \gamma(t) \in O;\ \gamma(0) = A, \gamma(0) = B\}$$

Esto se puede considerar como una relajación del problema en $3D$ debido a que no requeremos la suavidad de la función para calcular su derivada, por lo que le permitimos al robot realizar, en palabras simples, un giro brusco que genere una punta en el camino. En particular, si le pidieramos que fuera diferenciable solo en la tercera coordenada, como nos encontramos en $\mathbb{R}^2$, esta sería nula inmediatamente.

Luego, el problema de minimización corresponde a encontrar el camino factible más corto desde A hasta B, es decir, aquella curva que posea la menor longitud de arco.

Por lo tanto, nuestro problema de minimización es:

$$\min_{\gamma \in \Omega} \int_{\gamma} dl$$

Además, relajamos las condiciones de la curva, ya que la implementación de un algoritmo que encuentre curvas suaves, o, aproximaciones a curvas suaves, es un trabajo que se escapa de nuestros objetivos de la relajación del problema, ya que en particular, queremos encontrar estas curvas de manera gráfica en un mapa dado.

3. Algoritmo RRT

3.1. Motivación para el uso de algoritmos para Path Planning

Consideraremos solo $\mathbb{R}^2$.

Como se puede notar en base al modelo, es bastante complicado demostrar la existencia de mínimo, y si existe, es aún más díficil encontrarlo, esto es debido a que el conjunto de óbstaculos O produce una dificultad al momento de estudiar de manera teórica el problema.

Podemos notar que el problema es viable de estudiar si el mapa M es conexo por caminos, ya que de esta manera tendremos curvas (caminos) que son viables para que el robot llegue desde el punto A al punto B, y encontrar estas curvas de manera análitica no es viable en la gran mayoría de los casos.

Por lo tanto, requerimos estudiar este problema, que consiste en encontrar caminos de longitud mínima de manera gráfica, para obtener al menos un camino factible que el robot pueda recorrer en un ambiente dado, y en caso de que se tenga el robot fisicamente y el mapa dado represente un ambiente real, el camino que recorra el robot puede ser programado a partir del camino encontrado mediante el uso de un algoritmo. Por otra parte, puede ocurrir que viendo una configuración de obstáculos intuitivamente se tenga una idea de cual sea el camino óptimo, pero visto de manera computacional y análitica, no es viable.

Existen varios algoritmos para este fin. Algunos como el algoritmo A* [9] y el algoritmo basado en campos potenciales artificiales [10], consideran el robot no como un punto, si no que tiene dimensiones, es decir, el robot es un subconjunto de puntos de $\mathbb{R}^2$.

En particular, el algoritmo que utilizaremos, llamado Rapidly-exploring random trees [11], no encuentra siempre un camino óptimo, pero encuentra caminos factibles (bajo nuestra definición), y en particular, este camino encontrado será el camino que encuentre más rápido que se acerque a la meta. En la siguiente sección se encuentra una explicación del funcionamiento del algoritmo, pero queremos hacer incapié en que este camino encontrado por el algoritmo **no siempre** es un camino de distancia mínima, pero si es factible en la gran mayoría de las veces, en la sección 4.1.2 veremos un ejemplo donde el camino encontrado no es factible, ya que no es simple, junto a su respectiva explicación de porque sucede.

3.2. Explicación del algoritmo RRT

Dado un mapa generado como una matriz de puntos $n \times m$, el cual tiene un punto de inicio y un punto final, consideraremos puntos iguales aquellos puntos que se encuentren a una distancia menor a una tolerancia denominada τ y en consecuencia de esto último, debemos definir un paso p que sea mayor a esta tolerancia y a su vez, menor a límites permitidos por el mapa.

Este algoritmo crea un árbol de búsqueda que agrega puntos reescalados de puntos admisibles al azar a lo largo del mapa sucesivamente hasta que si un punto se encuentra a una distancia menor a τ del punto final, entonces el algoritmo procede a calcular la ruta más corta recorriendo el árbol en sentido contrario, es decir, partiendo desde el nodo donde se encuentra el punto final, hasta la raíz sumando en cada iteración la distancia entre puntos.

Se dice punto reescalado si:

- Se encuentra en el interior del segmento de recta que une el punto admisible con el nodo más cercano a este.
- La distancia entre el punto reescalado y el nodo del árbol más cercano es igual al valor del paso.

Se dice punto admisible si:

- El segmento de recta que une el punto con el nodo más cercano al árbol se encuentra completamente contenido en el mapa y no colisiona con los obstáculos
- El punto no se encuentra a una distancia menor a la tolerancia τ del nodo más cercano al árbol

A continuación se presenta un ejemplo de los resultados de este algoritmo para un laberinto de resolución 1402×1414 pixeles (que correspondería a la matriz $m \times n$ con $m = 1402$ y $n = 1414$). El paso utilizado fue de 60 pixeles con una tolerancia de 30 pixeles.

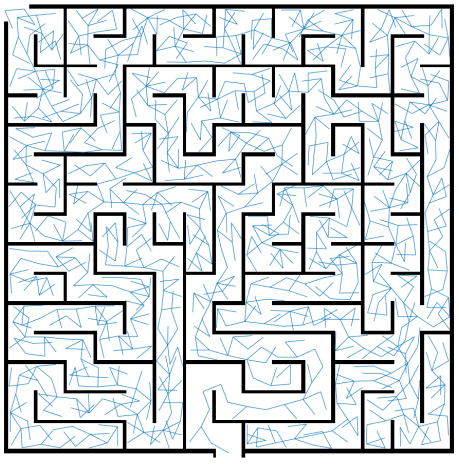

Figura 1: Árbol obtenido con un paso de 60 pixeles y una tolerancia de 30 pixeles para un laberinto de fácil dificultad

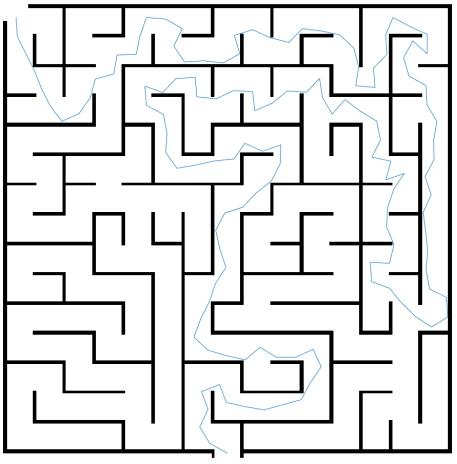

Figura 2: Solución obtenida con un paso de 60 pixeles y una tolerancia de 30 pixeles para un laberinto de fácil dificultad

3.3. Pseudo-código

Un pseudo-código del algoritmo RRT implementado en MATLAB [11] es el siguiente.

Algorithm 1: RRT para Robot Path Planning

Input : Mapa
Input : Punto inicio A, punto final B
Input : Paso p, Tolerancia τ
Input : Cantidad Máxima de Errores $MaxAttemp$
begin
  $NTree(A)$ inicializar árbol con raíz A
  $CountAttemp \leftarrow 0$
  $E \leftarrow False$ # *Estado de la ruta*
  while $CountAttemp \leq MaxAttemp$ **do**
    $v_a \leftarrow$ punto aleatoreo del mapa
    $n_a \leftarrow$ nodo más cercano a v_a
    $\theta \leftarrow$ ángulo de separación entre n_a y v_a
    $n_p \leftarrow n_a + p \cdot (\sin(\theta), \cos(\theta))$ # *reescalamiento del vector que se dirige a* n_a
    if $[n_a, n_p] \notin Mapa$ **then**
      $CountAttemp \leftarrow CountAttemp + 1$
      Saltar a la siguiente iteración
    end
    if $dist(n_p, B) < \tau$ **then**
      $NTree \leftarrow B$
      $E \leftarrow True$
      Salir del While
    end
    if $dist(n_p, n_a) < \tau$ **then**
      $CountAttemp \leftarrow CountAttemp + 1$
      Saltar a la siguiente iteración
    end
    $NTree \leftarrow n_p$
    $CountAttemp \leftarrow 0$
  end
  if *E is False* **then**
    return No encuentra ruta
  end
  $R \leftarrow \{B\}$ # *R Ruta óptima*
  $n_p \leftarrow$ Nodo anterior a B
  while n_p *no es* A **do**
    $R \leftarrow \{n_p\} \cup R$ # *Esta inclusión respeta el orden por la izquierda*
    $n_p \leftarrow$ Nodo anterior a n_p
  end
  $L \leftarrow 0$ # *L: Largo ruta*
  $count \leftarrow 0$
  while *count* $\leq$*(cardinalidad(R)-1)* **do**
    $L \leftarrow L+$dist(nodo i-esimo, nodo $(i+1)$-esimo)
    $i \leftarrow i + 1$
  end
  return R, L
end

4. Simulaciones realizadas con algoritmo RRT

Para todas las simulaciones ejecutadas se utilizó un mapa en formato **.bmp** (bitmap). Los puntos iniciales y finales de cada configuración presentada se encuentran indicadas en la sección correspondiente a cada mapa, donde A representa al punto inicial, y B representa al punto final.

En particular, los puntos iniciales y finales de cada mapa son pixeles. Se encuentran mencionados estos pixeles en sus secciones respectivas.

Cada mapa puede verse como el plano XY, donde x e y se encuentran en el rango de pixeles dado por la resolución de cada mapa.

Se utilizó el algoritmo para encontrar caminos en mapas con formas determinadas, en particular, uno posee la forma del pokémon Pikachu, y el otro posee la forma del logo de Unreal Engine 4. Las explicaciones de porque se utilizaron estas formas se encuentran en sus respectivas secciones.

Además de esto, se realizaron simulaciones para dos laberintos de diferentes dificultades, pero estas se encuentran en el repositorio

`https://github.com/CristobalLobos/Robot_Path_Planning_MAT279_Optimizacion`

junto a los códigos utilizados para todas las simulaciones encontradas en este informe.

Debido a la naturaleza de este algoritmo, se utilizó para resolver un laberinto circular, esto es para someter al robot a moverse por caminos con muchas curvas, y además, para ver la capacidad del algoritmo para casos donde no puede realizar un paso muy grande.

Para cada configuración de obstáculos se realizó una comparación del camino obtenido con distintas distancias de paso y de distancias de igualdad, además, se reescaló el mapa a una mayor resolución y se realizó la misma comparación. La finalidad de esto es ver cuales caminos son más suaves, cuales producen que el robot de menos giros bruscos, y cual resolución nos permite tener un mejor estudio del mapa. Además, una razón de estudio de laberintos para Path Planning, es que los planos de los departamentos se pueden ver como laberintos, y así se obtiene la mejor ruta para llegar de un punto a otro dentro del laberinto.

Por otra parte, mediante el uso del algoritmo podemos encontrar un posible camino que el robot recorra en el laberinto sin necesidad de resolver uno mismo el laberinto y progamarlo para que lo recorra.

Finalmente, se realizó un aumento de resolución es una escala del 200 % para poder realizar pasos de mayores pixeles y ver si hay alguna diferencia notable entre los caminos encontrados. Esto se hizo en todos los mapas excepto el logo de Unreal Engine 4, ya que a este se le hizo un reescalamiento arbitrario.

Los códigos utilizados para las simulaciones realizadas vienen inspirados por los códigos realizados por Rahul Kala [11].

Para mantener el formato utilizado en los códigos, los pixeles de partida y de llegada se encuentran en el formato (y, x).

4.1. Mapa con forma de logo de Unreal Engine

A medida que la humanidad avanza, la tecnología lo hace con ella. El motor gráfico Unreal Engine 4 fue publicado en el año 2014 por la compañia Epic Games, y es uno de lo más estables que han realizado.

Su logo, presentado a continuación, presenta un buen objeto de estudio para este problema, debido a que intuitivamente hay dos caminos posibles que pueden ser óptimos, los cuales serían irse en sentido horario o antihorario rodeando el borde del círculo. Pero como mencionamos anteriormente, es bastante complejo encontrar el cámino óptimo, por lo que mediante el uso de RRT veremos caminos posibles usando diferentes pasos, y realizaremos una comparación de cada uno.

El mapa utilizado junto con el punto inicial y final utilizados se encuentran en la siguiente figura.

Figura 3: Punto inicial y final en el logo de Unreal Engine 4

4.1.1. Resolución Normal

Resolución: 900×900 pixeles

Pixel de partida: (200,267)

Pixel de llegada: (687,549)

Para un paso de 200 pixeles y una tolerancia de 50 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 1046 pixeles y un tiempo de ejecución de $17,65$ segundos.

Figura 4: Árbol obtenido con un paso de 200 pixeles y una tolerancia de 50 pixeles para el logo de Unreal Engine 4 de resolución normal

Figura 5: Camino obtenido con un paso de 200 pixeles y una tolerancia de 50 pixeles para el logo de Unreal Engine 4 de resolución normal

Para un paso de 100 pixeles y una tolerancia de 50 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 871 pixeles y un tiempo de ejecución de $16,04$ segundos.

Figura 6: Árbol obtenido con un paso de 100 pixeles y una tolerancia de 50 pixeles para el logo de Unreal Engine 4 de resolución normal

Figura 7: Camino obtenido con un paso de 100 pixeles y una tolerancia de 50 pixeles para el logo de Unreal Engine 4 de resolución normal

Para un paso de 70 pixeles y una tolerancia de 35 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 822 pixeles y un tiempo de ejecución de $23,24$ segundos.

Figura 8: Árbol obtenido con un paso de 70 pixeles y una tolerancia de 35 pixeles para el logo de Unreal Engine 4 de resolución normal

Figura 9: Camino obtenido con un paso de 70 pixeles y una tolerancia de 35 pixeles para el logo de Unreal Engine 4 de resolución normal

Para un paso de 30 pixeles y una tolerancia de 15 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 1188 pixeles y un tiempo de ejecución de $38,52$ segundos.

Figura 10: Árbol obtenido con un paso de 30 pixeles y una tolerancia de 15 pixeles para el logo de Unreal Engine 4 de resolución normal

Figura 11: Camino obtenido con un paso de 30 pixeles y una tolerancia de 15 pixeles para el logo de Unreal Engine 4 de resolución normal

Análisis de los resultados:

Podemos notar que el algoritmo tiende a ejecutarse en un mayor tiempo si el paso decrece.

La menor longitud de curva obtenida fue de 822 pixeles utilizando un paso de 70 pixeles, mientras que la mayor longitud de curva obtenida fue de 1188 pixeles utilizando un paso de 30 pixeles.

Notemos que en ambos casos, la tolerancia utilizada fue la mitad del paso realizado.

Sin embargo, utilizando un paso de 100 pixeles y una tolerancia de 50 pixeles, se obtuvo una longitud de curva de 871 pixeles, la cual no es mucho mayor a la obtenida con el paso de 70 pixeles, y esta última fue obtenida en un tiempo de ejecución 10 segundos mayor.

Además, notemos que ambas curvas obtenidas que fueron de menor longitud, recorren el obstáculo en sentido antihorario, mientras que las de mayor longitud, la recorrieron en sentido horario, lo que nos indica una tendencia a que la curva más corta se encuentra rodeando el obstáculo en sentido antihorario.

4.1.2. Resolución Aumentada

Resolución: 1500×1500 pixeles

Pixel de partida: (333,447)

Pixel de llegada: (1146,915)

Para un paso de 300 pixeles y una tolerancia de 100 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 2339 pixeles y un tiempo de ejecución de $48,25$ segundos.

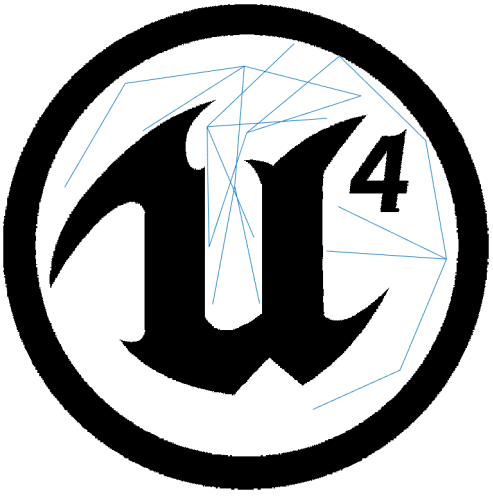

Figura 12: Árbol obtenido con un paso de 300 pixeles y una tolerancia de 100 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Figura 13: Camino obtenido con un paso de 300 pixeles y una tolerancia de 100 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Para un paso de 250 pixeles y una tolerancia de 100 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 1570 y un tiempo de ejecución de $19,13$ segundos.

Figura 14: Árbol obtenido con un paso de 250 pixeles y una tolerancia de 100 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Figura 15: Camino obtenido con un paso de 250 pixeles y una tolerancia de 100 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Para un paso de 150 pixeles y una tolerancia de 50 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 1910 pixeles y un tiempo de ejecución de $26,97$ segundos.

Figura 16: Árbol obtenido con un paso de 150 pixeles y una tolerancia de 50 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Figura 17: Camino obtenido con un paso de 150 pixeles y una tolerancia de 50 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Para un paso de 60 pixeles y una tolerancia de 30 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 1493 y un tiempo de ejecución de $18,87$ segundos.

Figura 18: Árbol obtenido con un paso de 60 pixeles y una tolerancia de 30 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Figura 19: Camino obtenido con un paso de 60 pixeles y una tolerancia de 30 pixeles para el logo de Unreal Engine 4 de resolución aumentada

Análisis de los resultados:

Observamos que de las curvas obtenidas, solo dos son factibles, ya que la curva obtenida con un paso de 300 pixeles y una tolerancia de 100 pixeles no es simple, y la curva obtenida con un paso de 150 pixeles y una tolerancia de 50 pixeles tampoco lo es.

Esto es debido a que al tomar una tolerancia baja en con respecto al paso, se generan más nodos, por lo que al unirlos se puede llegar a perder la simplicidad de la curva, tal como ocurrió en estos casos.

Con respecto a las otras dos curvas, observamos que la longitud de curva menor fue obtenida mediante el uso de un paso de 60 pixeles y una tolerancia de 30 pixeles, la cual fue de 1493 pixeles, mientras que la obtenida mediante el uso de 250 pixeles y una tolerancia de 100 pixeles tuvo una longitud de 1570 pixeles.

Notemos que la curva de menor longitud recorre el obstáculo en sentido antihorario, mientras que la otra la recorre en sentido horario.

Además, al tener una tolerancia menor que la mitad del paso, el algoritmo tuvo un mayor tiempo de ejecución que al ejecutarlo con un paso menor con una tolerancia de la mitad de este.

4.2. Mapa con forma de Pikachu

Pikachu es una de las figuras más reconocibles que existen actualmente, su fama se ha extendido y una gran parte de la humanidad sabe quien es este personaje.

Hemos escogido su figura para este proyecto debido a que su geometría nos permite realizar pasos de pixeles no menores, y además podemos encontrar un camino interesante al recorrer desde su oreja izquierda a su cola debido a la forma zig-zag que esta última posee. Por otra parte, de manera intuitiva, el camino será de menor longitud a medida que el camino se acerque al borde izquierdo de la figura.

El punto inicial y final se encuentran en la siguiente figura.

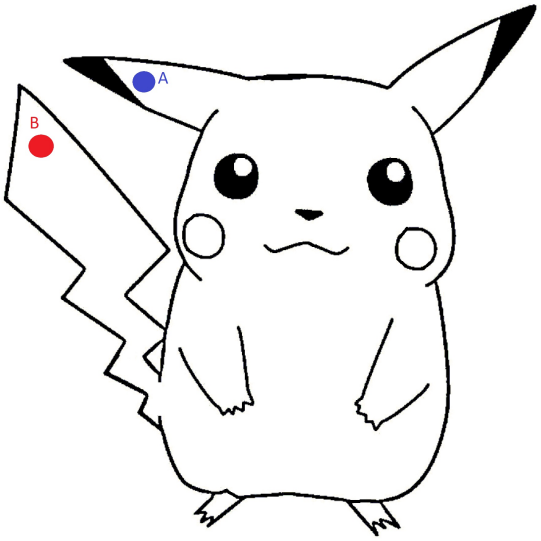

Figura 20: Punto inicial y final en el mapa con forma de Pikachu

4.2.1. Resolución Normal

Resolución: 900×981 pixeles

Pixel de partida: (140,200)

Pixel de llegada: (220,50)

Para un paso de 230 pixeles y una tolerancia de 60 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 2493 pixeles y un tiempo de ejecución de $26,39$ segundos.

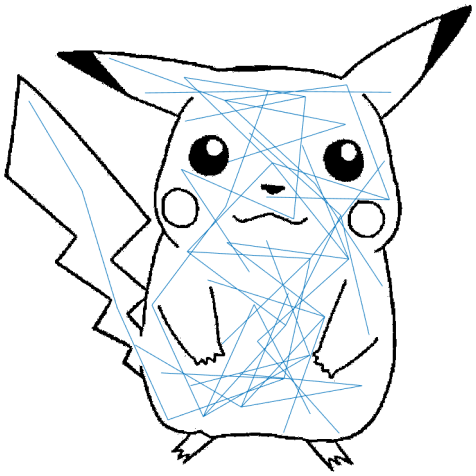

Figura 21: Árbol obtenido con un paso de 230 pixeles y una tolerancia de 60 pixeles para el mapa con forma de Pikachu de resolución normal

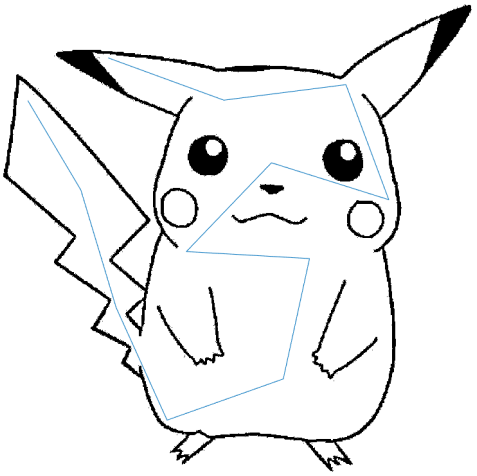

Figura 22: Camino obtenido con un paso de 230 pixeles y una tolerancia de 60 pixeles para el mapa con forma de Pikachu de resolución normal

Para un paso de 150 pixeles y una tolerancia de 70 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 2007 pixeles y un tiempo de ejecución de $25,01$ segundos.

Figura 23: Árbol obtenido con un paso de 150 pixeles y una tolerancia de 70 pixeles para el mapa con forma de Pikachu de resolución normal

Figura 24: Camino obtenido con un paso de 150 pixeles y una tolerancia de 70 pixeles para el mapa con forma de Pikachu de resolución normal

Para un paso de 90 pixeles y una tolerancia de 45 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 1679 pixeles y un tiempo de ejecución de $22,68$ segundos.

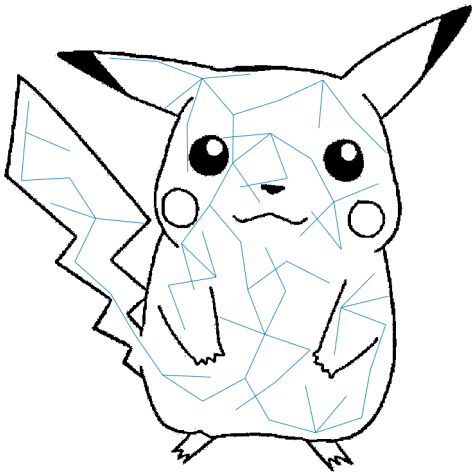

Figura 25: Árbol obtenido con un paso de 90 pixeles y una tolerancia de 45 pixeles para el mapa con forma de Pikachu de resolución normal

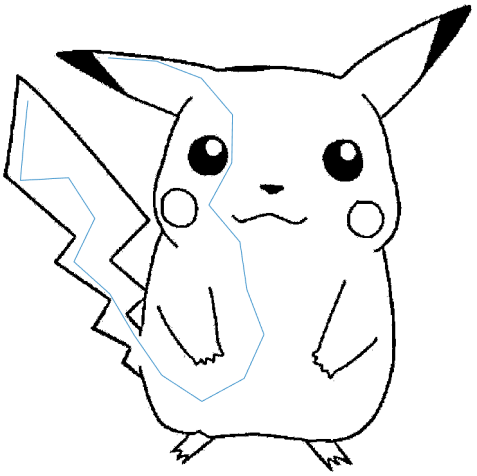

Figura 26: Camino obtenido con un paso de 90 pixeles y una tolerancia de 45 pixeles para el mapa con forma de Pikachu de resolución normal

Para un paso de 25 pixeles y una tolerancia de 12 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 1617 y un tiempo de ejecución de $506,87$ segundos.

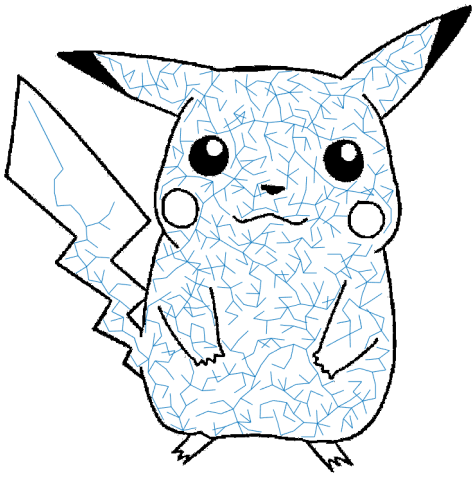

Figura 27: Árbol obtenido con un paso de 25 pixeles y una tolerancia de 12 pixeles para el mapa con forma de Pikachu de resolución normal

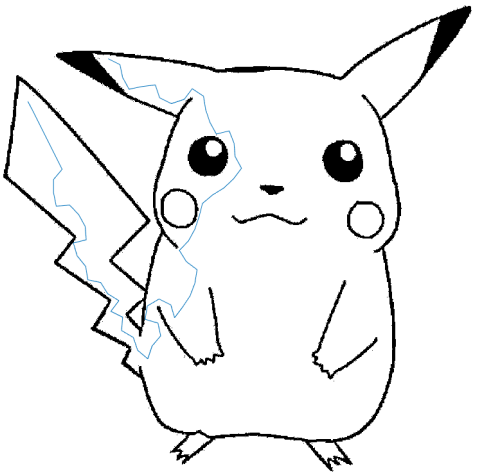

Figura 28: Camino obtenido con un paso de 25 pixeles y una tolerancia de 12 pixeles para el mapa con forma de Pikachu de resolución normal

Análisis de los resultados:

Notemos que, como se mencionó al principio, el camino que más se acerca al borde, que corresponde al obtenido usando un paso de 25 pixeles y una tolerancia de 12 pixeles, tuvo la menor longitud, la cual es de 1617 pixeles. Cabe destacar que este algoritmo tuvo un tiempo de ejecución 506,87 segundos, el cual es notablemente mayor que los demás.

En base a lo último mencionado, se observa una tendencia a tener mayor tiempo de ejecución a medida que el paso utilizado es menor.

Además, para esta forma, se observa que a medida que aumenta el paso de los pixeles, aumenta la longitud de la curva, y como se puede observar,la curva se aleja del borde a medida que el paso utilizado es mayor.

Notemos lo siguiente con respecto al camino obtenido mediante el uso de 230 pixeles. Se puede observar que este en particular es el único que rodeó el ojo derecho de la figura, en particular, esto es debido a que el paso al ser muy largo, tiende a ir a lugares más alejados a partir del nodo inicial.

4.2.2. Resolución Aumentada

Resolución: 1800×1962 pixeles

Pixel de partida: (200,267)

Pixel de llegada: (687,549)

Para un paso de 400 pixeles y una tolerancia de 100 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 3283 y un tiempo de ejecución de 5,52 segundos.

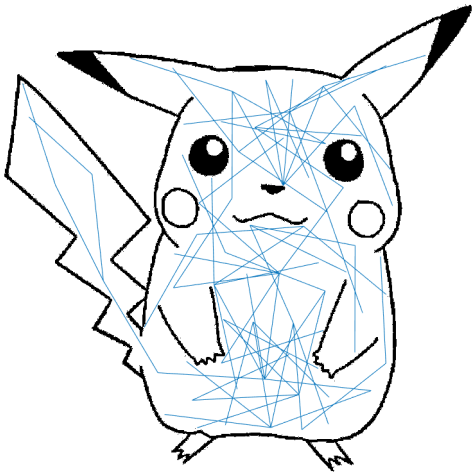

Figura 29: Árbol obtenido con un paso de 400 pixeles y una tolerancia de 100 pixeles para el mapa con forma de Pikachu de resolución aumentada

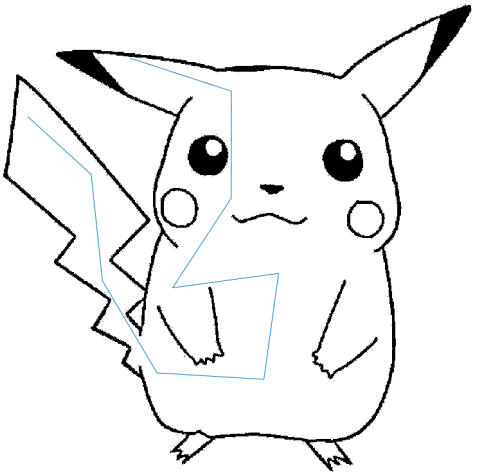

Figura 30: Camino obtenido con un paso de 400 pixeles y una tolerancia de 100 pixeles para el mapa con forma de Pikachu de resolución aumentada

Para un paso de 300 pixeles y una tolerancia de 100 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 3290 pixeles y un tiempo de ejecución de $20,21$ segundos.

Figura 31: Árbol obtenido con un paso de 300 pixeles y una tolerancia de 100 pixeles para el mapa con forma de Pikachu de resolución aumentada

Figura 32: Camino obtenido con un paso de 300 pixeles y una tolerancia de 100 pixeles para el mapa con forma de Pikachu de resolución aumentada

Para un paso de 150 pixeles y una tolerancia de 75 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 3664 pixeles y un tiempo de ejecución de $22,61$ segundos.

Figura 33: Árbol obtenido con un paso de 150 pixeles y una tolerancia de 75 pixeles para el mapa con forma de Pikachu de resolución aumentada

Figura 34: Camino obtenido con un paso de 150 pixeles y una tolerancia de 75 pixeles para el mapa con forma de Pikachu de resolución aumentada

Para un paso de 70 pixeles y una tolerancia de 35 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 2919 pixeles y un tiempo de ejecución de $32,57$ segundos.

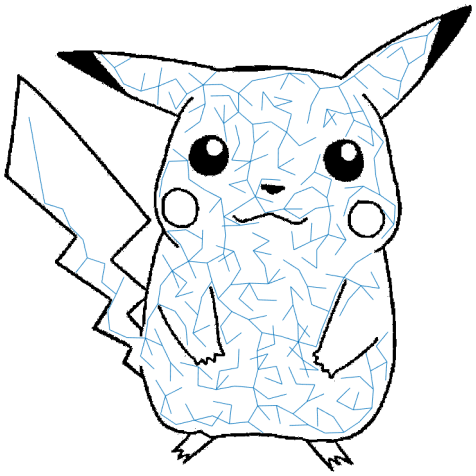

Figura 35: Árbol obtenido con un paso de 70 pixeles y una tolerancia de 35 pixeles para el mapa con forma de Pikachu de resolución aumentada

Figura 36: Camino obtenido con un paso de 70 pixeles y una tolerancia de 35 pixeles para el mapa con forma de Pikachu de resolución aumentada

Análisis de los resultados:

Notemos que tal como se ha mostrado en casos anteriores, la simulación realizada con el mayor paso, tuvo el menor tiempo de ejecución, esta corresponde a la realizada con un paso de 400 pixeles y 100 pixeles, donde se obtuvo un tiempo de ejecución de 5,52 segundos.

Observamos que la menor longitud de curva, al igual que en la resolución normal de la figura, se obtuvo la menor longitud de curva en el camino más apegado al borde izquierdo, esta corresponde a la curva obtenida con un paso de 70 pixeles y 35 pixeles de tolerancia, con una longitud de 2919 pixeles.

Sin embargo, se observa que de las tres curvas restantes, la mayor con el segundo menor paso realizado, correspondiente a 150 pixeles junto con una tolerancia de 75 pixeles. En particular, la longitud de curva obtenida fue de 3664 pixeles. Esto se debe a que, como se puede observar en su respectiva figura, el camino comienza a zig-zagear, lo que provoca una mayor longitud, a pesar de que esté más apegado al borde izquierdo que los caminos obtenidos con pasos mayores.

Por otra parte, la segunda menor longitud de curva obtenida corresponde a 3283 pixeles, correspondiendo al camino obtenido con un paso de 400 pixeles y una tolerancia de 100 pixeles, esto se debe a que, como se puede observar en su correspondiente figura, el camino se acerca al borde izquierdo al principio, pero luego se aleja, sin embargo, no zig-zagea, lo que provoca que la longitud de curva no aumente.

4.3. Laberinto Circular

Los laberintos han intrigado al ser humano desde tiempos de antes de cristo. En particular, este laberinto circular es una configuración de obstáculos interesante, debido a que sabemos que existe una solución al laberinto.

Sin embargo, nos gustaría que nuestro robot pudiera recorrerlo sin problemas, y para esto, utilizamos el algoritmo RRT para encontrar un camino que nos entregue la solución del laberinto.

Queremos hacer incapié que acá sabemos la solución del laberinto, veremos diferencias en la longitud de curva de los caminos encontrados, a pesar de que todos sigan el camino dado por la solución del laberinto.

Notar que en este caso no se pueden realizar pasos muy largos, debido a la complejidad del laberinto, ya que si lo hiciera, no podría hacer giros para rodear el círculo.

El punto inicial y final del laberinto se encuentran en la siguiente figura.

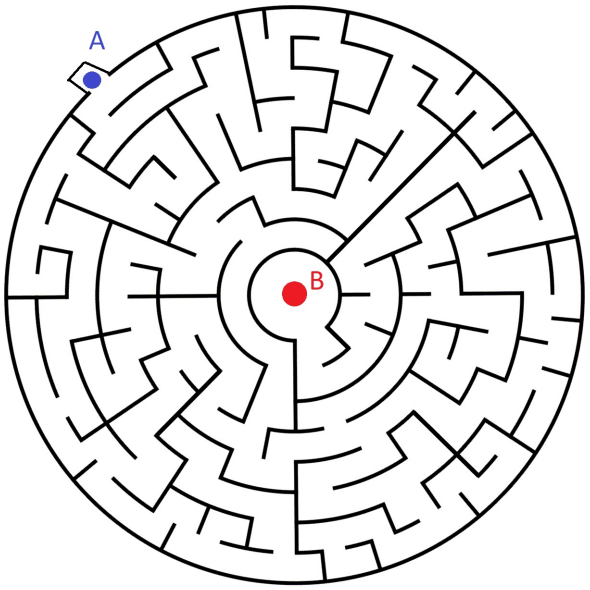

Figura 37: Punto inicial y final en el laberinto circular

4.3.1. Resolución Normal

Resolución: 1258×1251 pixeles

Pixel de partida: (165 190)

Pixel de llegada: (615 630)

Para un paso de 100 pixeles y una tolerancia de 50 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 4690 y un tiempo de ejecución de $61,44$ segundos.

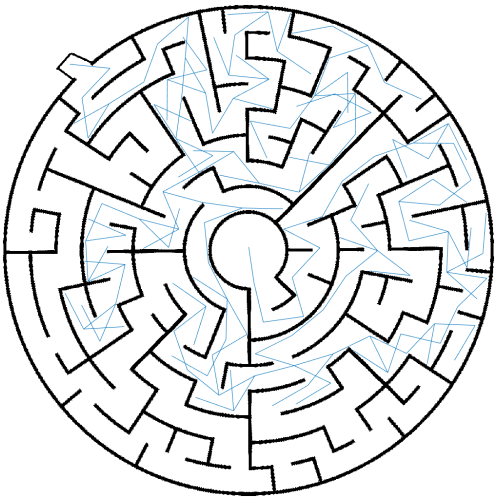

Figura 38: Árbol obtenido con un paso de 100 pixeles y una tolerancia de 50 pixeles para el laberinto circular de resolución normal

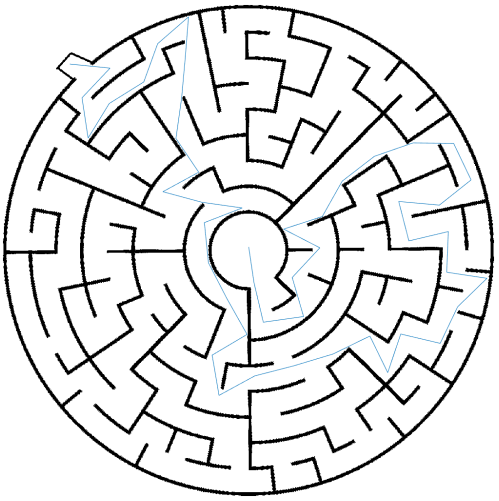

Figura 39: Camino obtenido con un paso de 100 pixeles y una tolerancia de 50 pixeles para el laberinto circular de resolución normal

Para un paso de 50 pixeles y una tolerancia de 25 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 4722 y un tiempo de ejecución de $60,47$ segundos.

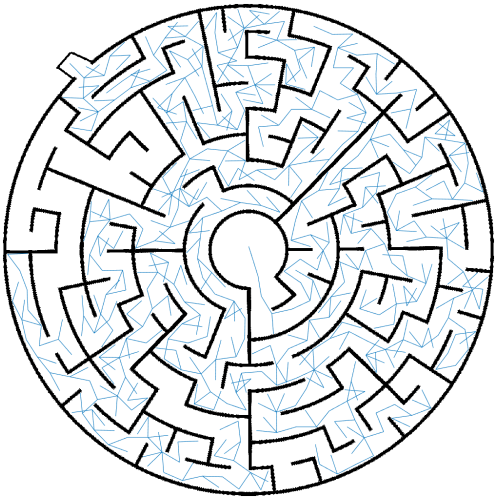

Figura 40: Árbol obtenido con un paso de 50 pixeles y una tolerancia de 25 pixeles para el laberinto circular de resolución normal

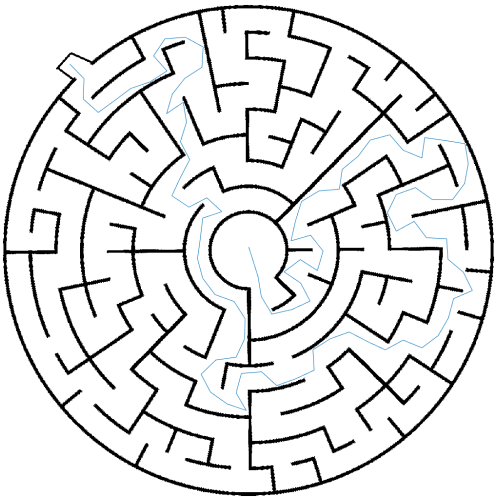

Figura 41: Camino obtenido con un paso de 50 pixeles y una tolerancia de 25 pixeles para el laberinto circular de resolución normal

Análisis de los resultados:

Notemos que ambos caminos encontrados siguen el camino dado por la solución del laberinto, sin embargo, el camino obtenido por el paso 100 pixeles con una tolerancia de 50 pixeles, no realiza curvas tan zig-zageantes, esto se puede notar en el lado derecho del laberinto. De esta manera, tenemos que la longitud de esta curva es de 4690 pixeles, mientras que el otro camino obtenido posee una longitud de curva de 4722 pixeles, donde esta curva zig-zagea más al realizar movimientos más "finos".

4.3.2. Resolución Aumentada

Resolución: 2516×2502 pixeles

Pixel de partida: (330,380)

Pixel de llegada: (1261,1230)

Para un paso de 150 pixeles y una distancia 50 de igualdad de pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 9892 y un tiempo de ejecución de $109,38$ segundos.

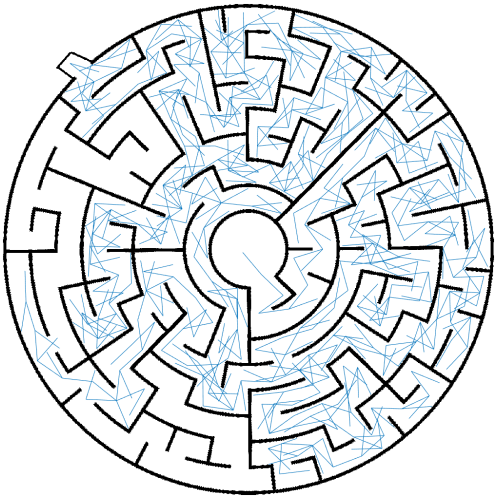

Figura 42: Árbol obtenido con un paso de 150 pixeles y una tolerancia de 50 pixeles para el laberinto circular de resolución aumentada

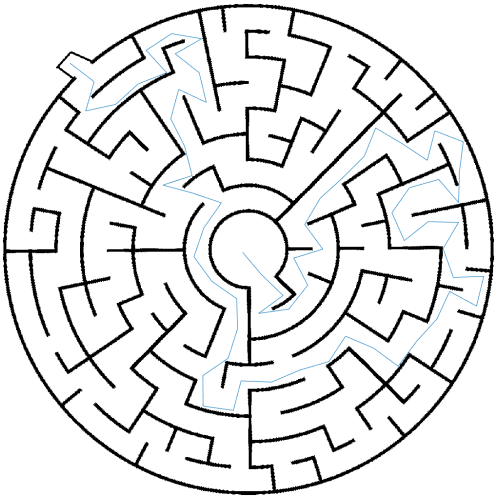

Figura 43: Camino obtenido con un paso de 150 pixeles y una tolerancia de 50 pixeles para el laberinto circular de resolución aumentada

Para un paso de 70 pixeles y una tolerancia de 35 pixeles se obtuvo el siguiente árbol y camino, con una longitud de curva de 9443 y un tiempo de ejecución de $258,36$ segundos.

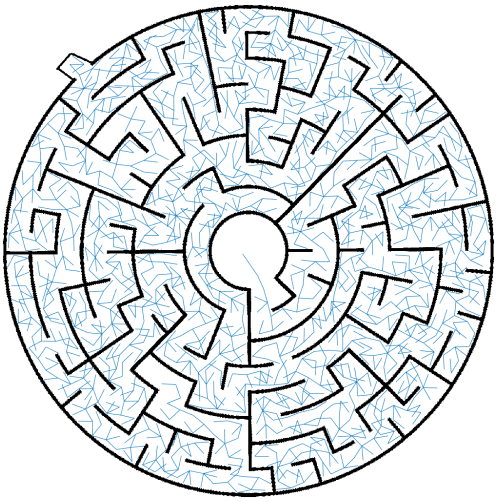

Figura 44: Árbol obtenido con un paso de 70 pixeles y una tolerancia de 35 pixeles para el laberinto circular de resolución aumentada

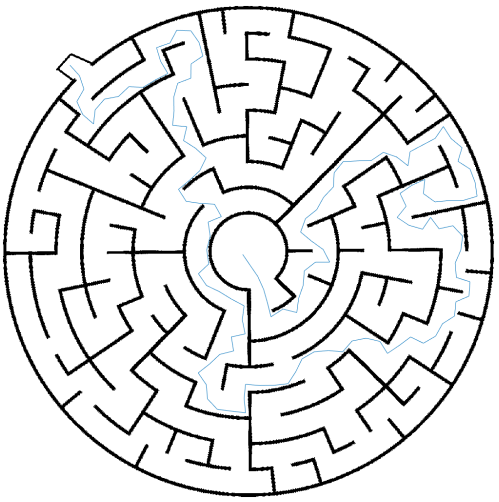

Figura 45: Camino obtenido con un paso de 70 pixeles y una tolerancia de 35 pixeles para el laberinto circular de resolución aumentada

Análisis de los resultados:

Observando los caminos obtenidos, notamos que el camino obtenido con un paso 70 pixeles y una tolerancia de 35 pixeles tiene una longitud de curva menor que la del otro camino obtenido, esta corresponde a una longitud de 9443 pixeles, mientras que la otra corresponde a una de 9892 pixeles. Esto es debido a que el camino mencionado rodea más el borde del laberinto que el otro, lo cual produce una menor longitud de curva, además, como se puede observar en su respectiva figura, el otro camino encontrado tiende a alejarse del borde al girar, lo cual produce una mayor longitud de curva de 9892 pixeles.

5. Conclusiones y discusión

Concluimos que mediante el uso del algoritmo RRT, si bien no se obtienen caminos que sean el óptimo de nuestro problema de minimización, nos entrega caminos factibles para que el robot recorra en los ambientes entregados.

De manera general, mientras mayor sea el paso realizado, con una tolerancia no tan menor con respecto al paso realizado, el algoritmo no toma mucho tiempo en encontrar un camino, y en contraparte, si el paso realizado es pequeño, el algoritmo toma un tiempo no menor en encontrar un camino.

A modo de comentario, un problema de optimización que podría surgir a partir de este problema, sería escoger el paso óptimo para el algoritmo RRT de tal manera de que nos entregue la curva con la menor longitud de curva para una configuración de obstáculos dada.

Por otra parte, los caminos encontrados no son suaves en ningún caso, sin embargo, en la literatura, se ha trabajado en encontrar curvas suaves para problemas de path planning [12] [13] en dos dimensiones.

6. Referencias

[1] Allen Mcintosh. *Robot Path Planning and Cooperation - Foundations, Algorithms and Experimentations*, volume 772 of *Studies in Computational Intelligence*. Springer, 2018.

[2] K. Chu, Kichun Jo, and Myoungho Sunwoo. Real-time path planning of autonomous vehicles for unstructured road navigation. *International Journal of Automotive Technology*, 16, 08 2015.

[3] Brian Geiger, Joseph Horn, Anthony DeLullo, and Albert Niessner. Optimal path planning of uavs using direct collocation with nonlinear programming. *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2006*, 2, 08 2006.

[4] Diego Pardo, Lukas Moller, Michael Neunert, Alexander W. Winkler, and Jonas Buchli. Evaluating direct transcription and nonlinear optimization methods for robot motion planning. *IEEE Robotics and Automation Letters*, 1(2):946-953, Jul 2016.

[5] Christos Papadimitriou, Prabhakar Raghavan, and Hisao Tamaki. Motion planning on a graph. *Proc. 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, 09 2001.

[6] Payam varshovi jaghargh and Davood Naderi. Path planning of wheeled mobile robot using a new objective function. 07 2011.

[7] Masoud Fetanat, Sajjad Haghzad, and Saeed Bagheri Shouraki. Optimization of dynamic mobile robot path planning based on evolutionary methods. *2015 AI Robotics (IRANOPEN)*, Apr 2015.

[8] Carlos Federico Brunner Parra. *Impacto de considerar pendientes en el consumo de combustible en una planificación de rutas de vehículos*. PhD thesis, Pontificia Universidad Católica de Chile, Agosto 2018.

[9] R. Kala. *Code for Robot Path Planning using A* algorithm*. Indian Institute of Information Technology Allahabad. Available at: http://rkala.in/codes.html.

[10] R. Kala. *Code for Robot Path Planning using Artificial Potential Fields*. Indian Institute of Information Technology Allahabad. Available at: http://rkala.in/codes.html.

[11] R. Kala. *Code for Robot Path Planning using Rapidly-exploring Random Trees*. Indian Institute of Information Technology Allahabad. Available at: http://rkala.in/codes.html.

[12] Masatomo Kanehara, Satoshi Kagami, James Kuffner, Simon Thompson, and Hiroshi Mizoguchi. Path shortening and smoothing of grid-based path planning with consideration of obstacles. pages 991–996, 10 2007.

[13] Baoye Song, Guohui Tian, and F. Zhou. A comparison study on path smoothing algorithms for laser robot navigated mobile robot path planning in intelligent space. 7:2943–2950, 12 2010.