Redes Neuronales y Algoritmos de Planificación
Robótica

Luis Ramón Guajardo
Jafet Castañeda
Andrés Alejandro Suro
Miguel Ángel Paz

Universidad de Guanajuato - CIMAT
7 de agosto de 2023

---

1. Resumen

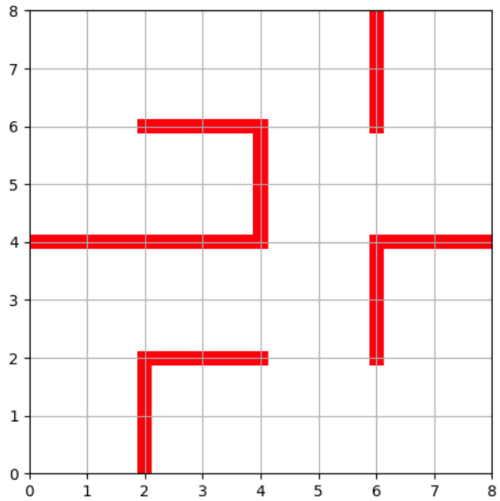
En el presente documento se describirá de forma sintética parte de las actividades realizadas y los resultados obtenidos durante un proyecto de verano llevado a cabo del 1 de junio al 31 de julio de 2023, con enfoque en algoritmos de muestreo para robótica.

Nuestro equipo, conformado por 4 integrantes y bajo la supervisión del Dr. Israel Becerra Durán, se encargó de investigar e implementar mediante simulaciones 4 algoritmos de muestreo para planificación de movimiento en robótica, los cuales son PRM*, RRT, RRG y RRT*. Como base para nuestro proyecto, usamos el documento *Sampling-based algorithms for optimal motion planning (2011)*, de Sertac Karaman y Emilio Frazzoli, así como el libro *Planning Algorithms (2006)*, de Steven M. LaValle.

En lo subsecuente, describimos el proceso de implementación y entrenamiento de los algoritmos y redes neuronales involucradas, además de los resultados en tiempo y costo computacional que se obtuvo de cada uno, lo cual nos permitirá concluir cualidades, ventajas y desventajas que pueden resultar de interés en una aplicación práctica (con un robot o un sistema físico).

2. Planteamiento de Escenario

El objetivo del proyecto a grandes rasgos es probar el desempeño de los algoritmos en un entorno controlado e igual para cada uno. Por dicha razón, para las simulaciones (realizadas en Python), se considero un laberinto en 2D conformado por 16 casillas, 4 de alto y 4 de ancho, de 2 unidades cada una, como se muestra en la siguiente imagen.



Además, se considera que el robot corresponde a un objeto circular de radio 0.25.

3. Implementación de los algoritmos

Siguiendo la descripción de los algoritmos provista en *Sampling-based algorithms for optimal motion planning (2011)*, cada uno de los integrantes del equipo se encargó de implementar uno de los algoritmos de interés. Inicialmente, se consideraron puntos iniciales del robot fijos y una cantidad fija de iteraciones para generar las trayectorias. A continuación, se muestran visualizaciones de las trayectorias generadas por dichos algoritmos.

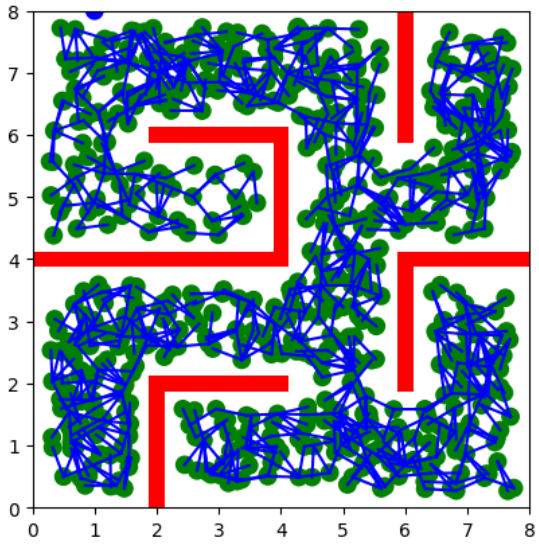


Figura 3.1: Trayectorias generadas por RRT

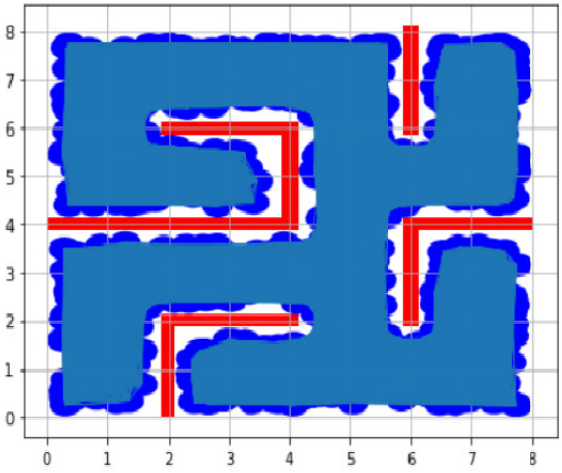


Figura 3.2: Trayectorias generadas por PRM*

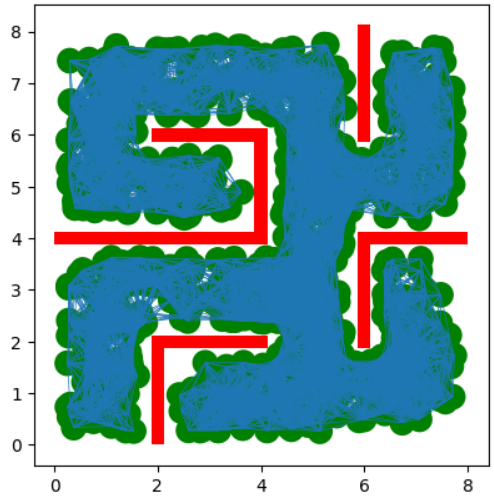


Figura 3.3: Trayectorias generadas por RRG

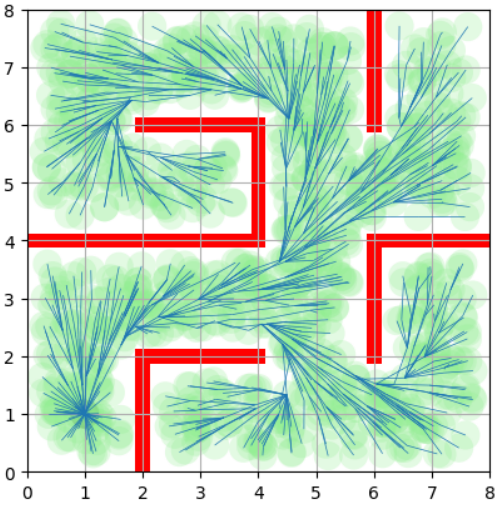


Figura 3.4: Trayectorias generadas por RRT*

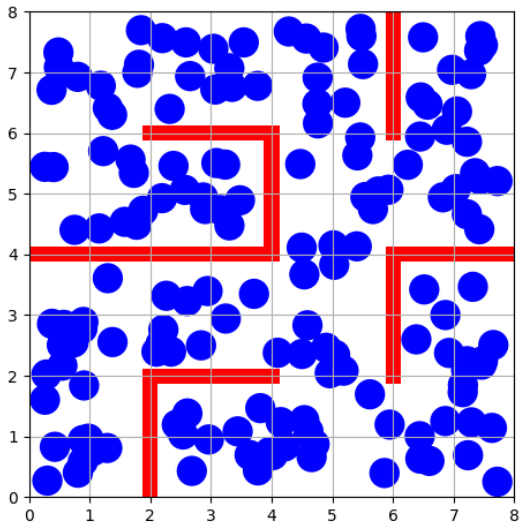
A partir de esta primera implementación de los algoritmos, pudimos darnos cuenta de varios aspectos particulares que nos ayudan a distinguirlos, por ejemplo:

- En los algoritmos RRT y RRT* se puede identificar fácilmente una estructura de árbol, en donde las aristas conectan los nodos entre sí de forma que no se forman ciclos. Por su parte, los algoritmos PRM* y RRG reflejan bastante el estar basados en grafos, ya que la densidad en aristas es significativamente mayor y hay presencia de ciclos en estas.
- El RRT* es la versión óptima del algoritmo RRT, lo cual se puede visualizar al notar que las trayectorias del RRT* son mucho más rectas y el arbol se nota un poco más "peinado" en sus ramas. Esto se debe al proceso de *re-cableado* que hace el algoritmo al establecer nuevas aristas. A pesar de la optimalidad del RRT*, este genera trayectorias mucho mas cercanas a los bordes del laberinto, el cual es un problema que abordaremos más adelante al analizar los resultados.
- Los algoritmos basados en grafos (PRM* y RRG) quizás no puedan producir taryectorias tan directas u óptimas como aquellos basados en árboles, sin embargo, permiten llegar a muchas más ubicaciones en el mapa de diferentes maneras.

4. Entrenamiento de Redes Neuronales

Una vez implementados los algoritmos, se decidió utilizar una red neuronal para cada uno, con el fin de que, a partir de la información generada por el algoritmo, dicha red pueda predecir una trayectoria desde un punto inicial en el laberinto hacia otro punto objetivo.

Para entrenar a las redes neuronales generamos un conjunto de 10 puntos en cada una de las 16 casillas, dando así un total de 160 puntos aleatorios, como se muestran a continuación.



Ahora, para cada punto aleatorio en su casilla determinada, generamos una trayectoria a otros puntos aleatorios ubicados en las 15 casillas restantes del plano, esto usando los algoritmos de planificación. Es así, que en total se obtuvieron 2400 trayectorias para entrenar a las redes neuronales.

La estructura que queremos que la red neuronal use es la siguiente, dado un punto de inicio p y un punto final q, nosotros le pasaremos a la red neuronal la tupla y esperamos que nos devuelva un punto, $[p_t, q] \rightarrow p_{t+1}$, idealmente más cercano a q.

Para entrenar una red neuronal necesitamos dos cosas: el conjunto X, que es la entrada y el conjunto Y, la variable de respuesta. De esta manera, de nuestras 2400 trayectorias (donde cada trayectoria es un conjunto de puntos que conducen a un punto final), formaremos tuplas de puntos con la estructura, $[p_t, q] \in X$, que pertenecen a la variable de entrada y como variable de respuesta pondremos $p_{t+1} \in Y$. Dependiendo de las trayectorias que haya generado cada algoritmo tendremos diferentes conjuntos X y Y.

Ahora bien, se utilizaron dos librerías diferentes en Python para construir las redes neuronales, las cuales son TensorFlow y Sklearn. Para la primera red, de Tensor Flow, consideramos el caso donde ponemos un *dropout* de 0.25 % entre cada capa, con el fin de evitar que se sobreajuste el entrenamiento a los datos. Por su parte, la segunda red, de Sklearn no usará drop out. A las 3 redes neuronales le pondremos un máximo de 4,000 iteraciones. En la siguiente sección, se presentan los resultados en tiempo y costo (longitud de las trayectorias) obtenidos por cada una de las redes y algoritmos.

5. Resultados

Predicción de trayectorias con SKLEARN					
Algoritmo	% Trayectorias exitosas	Promedio de tiempo	Promedio de costo (longitud)	Desv. estándar de tiempo	Desv. estándar de costo
PRM*	30	0.0013700167	3.724212942	0.00133056088	3.046009998
RRT	82	0.00153	201.9352	0.000993	152.547739
RRG	32	0.001964845	3.444875249	0.001271727	2.280110342
RRT*	42	0.00391353312	4.122652893387	0.002448143773	2.544622493661

Predicción de trayectorias con TensorFlow					
Algoritmo	% Trayectorias exitosas	Promedio de tiempo	Promedio de costo (longitud)	Desv. estándar de tiempo	Desv. estándar de costo
PRM*	36	0.46614464	4.353401	0.614400	3.12347137
RRT	99	0.99341	202.9878	0.52269	178.91039
RRG	23	0.444068857	2.647175413	0.34019592	2.065953349
RRT*	59	0.28754670337	4.809411815147	0.167524925153	2.796955002065

Predicción de trayectorias con TensorFlow (Dropout)					
Algoritmo	% Trayectorias exitosas	Promedio de tiempo	Promedio de costo (longitud)	Desv. estándar de tiempo	Desv. estándar de costo
PRM*	30	0.6948717	3.40119457	0.875233	2.4536014
RRT	86	2.05982	348.3773	2.25250	427.91487
RRG	24	0.437407156	2.788748024	0.353575907	2.247000569
RRT*	37	0.23231169984	3.47860038676	0.1386413602	2.1533899589

A partir de las tablas anteriores obtuvimos las siguientes conclusiones. En primer lugar, notemos que el algoritmo con un mayor porcentaje de Trayectorias Exitosas en todas las redes neuronales fue **RRT**, donde su mejor caso fue con la red neuronal de Tensorflow. Se sabe que dicho algoritmo no es optimal, es decir, no calcula las rutas óptimas para llegar al destino. Esto es importante dado que al implementar los algoritmos óptimos en un laberinto llegamos a que las rutas óptimas se apegan demasiado a las paredes u obstáculos del laberinto para obtener los menores costos posibles. Por lo tanto, **RRT** produce rutas que no se apegan a las paredes de los laberintos y por ende, sus trayectorias no producen colisiones de forma recurrente, obteniendo un porcentaje alto de trayectorias exitosas.

Con lo anterior establecido, podemos dirigir nuestra atención al algoritmo **RRT***, que produce trayectorias óptimas, con bastante menos costo y en mucho menor tiempo que el RRT, pero que, desafortunadamente, la gran mayoría terminan en colisiones dada su cercanía con las paredes. El mismo caso ocurrió con el algoritmo **RRG**.

Por ultimo, consideremos el algoritmo **PRM***. Se cumple que el algoritmo genera un grafo no dirigido de todas las trayectorias posibles considerando un conjunto de datos fijos. Al obtener dicho grafo, realizamos Dijkstra para obtener los caminos más cortos de una región meta a otra. Lo anterior implica que las trayectorias obtenidas para el conjunto de datos consisten de pocos puntos intermedios para definir la trayectoria (a comparación del resto de los algoritmos) y por lo tanto, tenemos un conjunto de entrenamiento mucho menor a los demás algoritmos. Lo anterior influye directamente en el entrenamiento de la red neuronal y por ende **PRM*** obtiene un porcentaje muy bajo de trayectorias exitosas.