Robótica I

Tarea 7. Implementación de RRT*
14 de mayo de 2020

Marco Antonio Esquivel Basaldua

I. Introducción

La planeación de trayectorias en robótica es un problema clásico. Un robot de ciertas dimensiones pretende navegar desde un punto A hasta un punto B, evitando un conjunto de obstáculos en el ambiente, $\mathcal{C}_{obs}$.
Existen numerosos algoritmos para la planificación de trayectorias. El algoritmo *Rapidly-Exploring Random Tree* (RRT) es una opción común en la que se genera un grafo y una trayectoria dentro de éste. Esta trayectoria no es necesariamente óptima, en la métrica que se utilice para medir la "distancia" para conectar dos puntos en el espacio de configuraciones $\mathcal{C}$. El RRT* es una modificación optimizada al algoritmo que resulta en encontrar la trayectoria más corta entre dos puntos de $\mathcal{C}$. En la figura 1 se muestran formas típicas de la generación de trayectorias con estos algoritmos dentro de un espacio de dos dimensiones.

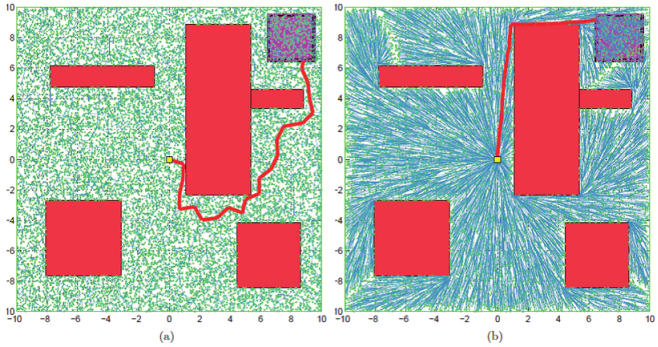

Figura 1. Comparación de una trayectoria trazada con RRT y RRT*.

En este trabajo se muestra la implementación del algoritmo RRT* en un espacio de dos dimensiones para distintas disposiciones de obstáculos dentro de $\mathcal{C}$. El lenguaje utilizado es *python* y se desarrolla dentro del módulo *pygame*. Se evalúa el desempaño en cada caso.
A manera de comparación, se describen brevemente los algoritmos RRT y RRT*.

II. RRT

La premisa de este algoritmo es generar puntos de manera aleatoria y conectarlos al nodo más cercano posible. Cada vez que se crea un vértice, se debe revisar que esté en un espacio libre de colisiones y que el arista que lo conecte con su respectivo nodo también esté en un lugar libre de colisiones. El algoritmo termina cuando se genera un vértice dentro de la región de la meta, o cuando un máximo de nodos es generado. El algoritmo es muy sencillo de implentar, se presenta a continuación.

Algorithm 3: RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}$; $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
return $G = (V, E)$;

Este método requiere menor poder de cómputo, es más simple de implementar y es más rápido que otros métodos de planificación de trayectorias, sin embargo require más nodos para ser almacenados. Su parte más costosa es encontrar el vecino más cercano a una nueva muestra ya que este proceso crece conforme aumenta el número de vértices generados.

III. RRT*

RRT* es una versión optimizada del RRT. Cuando el número de nodos tiende a infinito, este algortimo va encontrando la ruta más óptima hacia la meta. Los principios básicos del RRT* son los mismos que en el RRT, pero dos aspectos extra clave resultan en diferencias significativas.
Primero, RRT* registra la distancia que cada vértice ha viajado relativa a su vértice padre. Esto se refiere como el costo del vértice. Despueés de que el nodo más cercano a la muestra generada en el árbol es encontrado, un vecindario de vértices en un radio fijo desde el nuevo nodo es examinado. Si un nodo con un menor costo es encontrado, éste reemplaza al "vecino más cercano".
La segunda diferencia en el RRT* es un recableado del árbol. Después de que un vértice ha sido conectado al vecino de menor costo, los vecinos son examinados nuevamente. Se revisa si los nodos, al ser reconectados al nuevo vértice añadido, disminuyen su costo. Si esto ocurre, los nodos son reconectados al nuevo vértice. Este proceso da su aspecto característico al árbol.
El precio de este algoritmo optimizado es una reducción en rendimiento. El mayor esfuerzo de cómputo viene de la revisión en espacio libre de colisión, ya que se debe revisar cuando un nodo es añadido, cuando se conecta a un vecino y para cada nodo que es reconectado.
El algoritmo de RRT* se muestra a continuación.

Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

IV. Implementación

La implementación del algoritmo RRT* se realiza en el lenguaje de programación *python* utilizando el módulo *pygame* para la visualización de resultados. El funcionamiento del programa es el siguiente:

- Al ejecutar el código aparece una ventana en blanco de tamaño 800×600.
- Los obstáculos pueden ser dispuestos como dibujos en trazo libre presionando el botón izquierdo del mouse. Éstos aparecen en color negro.
- Una vez dispuestos los obstáculos deseados se debe presionar la tecla "Enter".
- A continuación, se presiona el botón izquierdo del mouse para ubicar el nodo inicial en el árbol (x_{init}). Éste se muestra de color púrpura.
- Al presionar el botón derecho del mouse se ubica la meta de la trayectoria a buscar (x_{goal}). Ésta se muestra de color rojo con un círculo más grande que x_{init}.
- La construcción del árbol comienza de forma automática (trazos verdes).
- Una vez encontrada la meta, se reporta éxito y se visualiza la trayectoria generada (en azul). Se observan los nodos generados y el tiempo de ejecución. De no encontrarse la meta, de acuerdo al máximo de nodos especificado, se reporta fracaso.

A continuación se muestra un ejemplo en la disposición de obstáculos, inicio y meta.

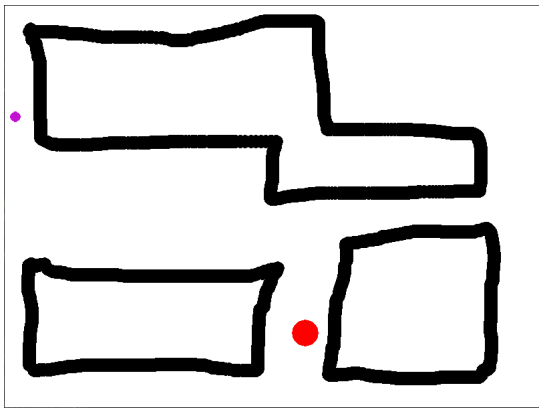

Figura 2. Ejemplo de disposición de obstáculos (negro), inicio (púrpura) y meta (rojo).

En seguida se muestran ejemplos del funcionamiento del algoritmo. En los primeros tres casos se realizan simulaciones sin obstáculos pero variando la distancia máxima inicial de los aristas y el radio de vecindad (en pixeles).

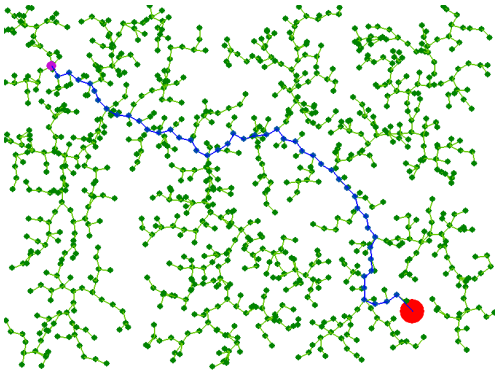

Figura 3. Ejemplo sin obstáculos con distancia máxima inicial entre aristas de 20 y vecindad de 20. Se arrojan resultados similares a costrurir un RRT.

```
Time needed to biuld the Tree  2.0688047409057617  seconds
Goal Reached
Cost to the goal  907.1987161284842
Nodes expantion  983
```

Figura 4. Resultados reportados.

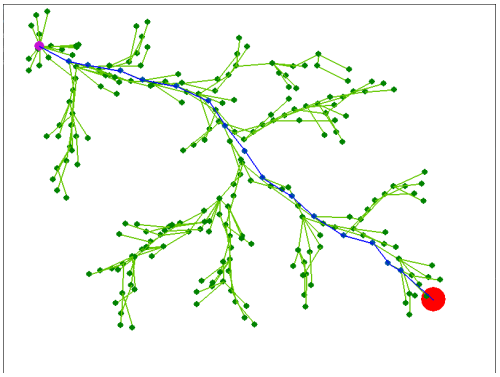

Figura 5. Ejemplo sin obstáculos con distancia máxima inicial entre aristas de 20 y vecindad de 60.

```
Time needed to biuld the Tree  0.1551225185394287  seconds
Goal Reached
Cost to the goal  781.331490816082
Nodes expantion  235
```

Figura 6. Resultados reportados.

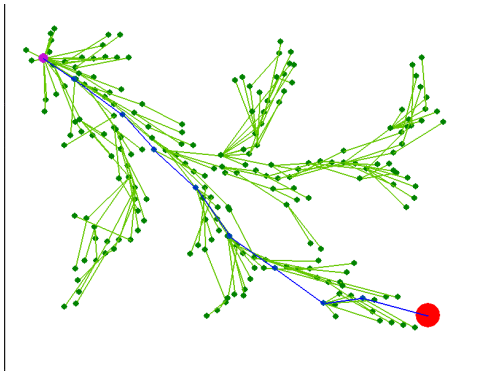

Figura 7. Ejemplo sin obstáculos con distancia máxima inicial entre aristas de 20 y vecindad de 100.

```
Time needed to biuld the Tree  0.13445425033569336  seconds
Goal Reached
Cost to the goal  776.3143766114545
Nodes expantion  215
```

Figura 8. Resultados reportados.

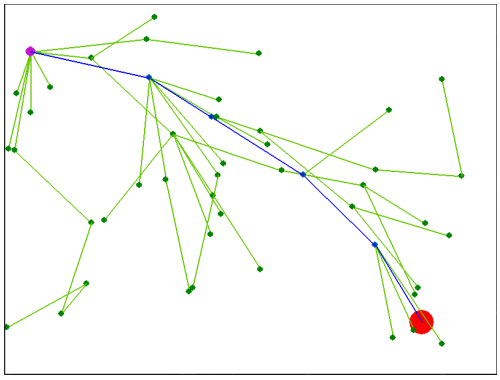

Figura 9. Ejemplo sin obstáculos con distancia máxima inicial entre aristas de 100 y vecindad de 200.

```
Time needed to biuld the Tree  0.010442733764648438  seconds
Goal Reached
Cost to the goal  810.2075602756593
Nodes expantion  46
```

Figura 10. Resultados reportados.

Ahora se muestran los resultados para ambientes con obstáculos, fijando la distancia máxima inicial entre nodos a 20 y la vencindad en 60.

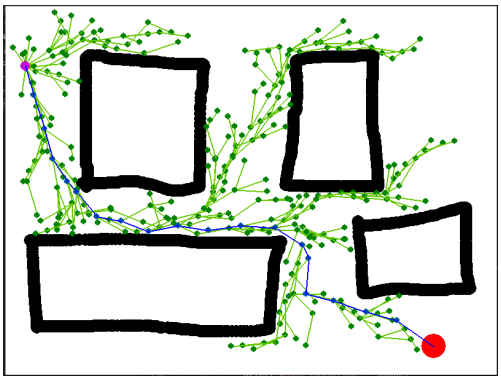

Figura 11. Primer ejemplo con obstáculos.

```
Time needed to biuld the Tree  0.3261735439300537  seconds
Goal Reached
Cost to the goal  951.5539287435679
Nodes expantion  528
```

Figura 12. Resultados reportados.

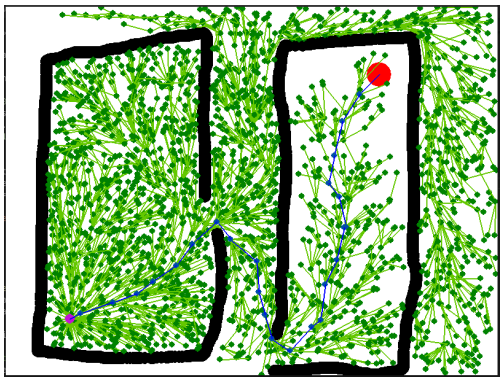

Figura 13. Segundo ejemplo con obstáculos.

```
Time needed to biuld the Tree  14.857017517089844  seconds
Goal Reached
Cost to the goal  1069.957569523758
Nodes expantion  3631
```

Figura 14. Resultados reportados.

V. Modificación para encontrar la ruta más óptima

En los ejemplos presentados anteriormente, el algortimo se detiene al encontrar la primer trayectoria generada a la meta. En esta modificación se generan tantos nodos como se quiera y de entre las trayectorias genradas se elige la más óptima. Se presentan a continuación los resultados sin obstáculos para distintos máximos en los nodos generados y un ejemplo para un ambiente con obstáculos.

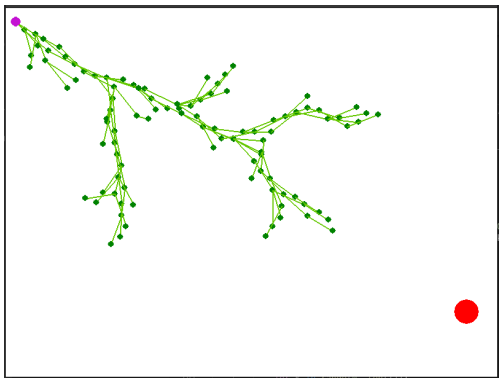

Figura 15. Para 100 nodos solicitados no se llega a la meta.

```
Time needed to biuld the Tree  0.2215893268585205  seconds
Goal NOT Reached
```

Figura 16. Resultados reportados.

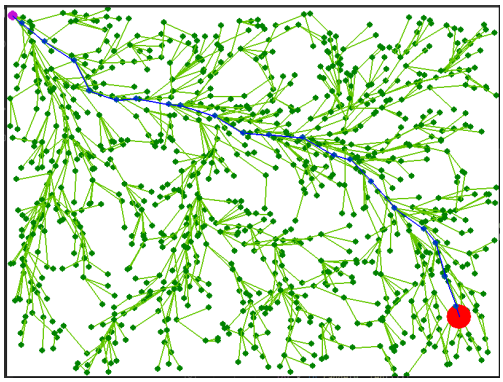

Figura 17. Ruta óptima generada con 1000 nodos.

```
Time needed to biuld the Tree  4.080264091491699  seconds
Goal Reached
Cost to the goal  934.368388177743
Nodes expantion  1000
```

Figura 18. Resultados reportados.

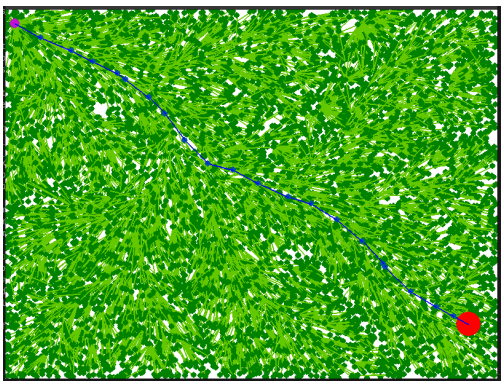

Figura 19. Ruta óptima generada con 10000 nodos.

```
Time needed to biuld the Tree  288.7909164428711  seconds
Goal Reached
Cost to the goal  875.5260285680928
Nodes expantion  10000
```

Figura 20. Resultados reportados.

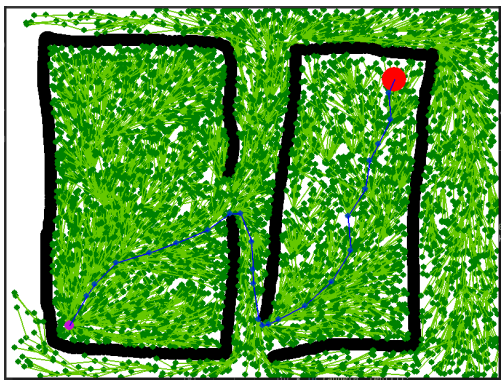

Figura 21. Ruta óptima generada en un ambiente con obstáculos para 10000 nodos.

```
Time needed to biuld the Tree  116.22886633872986  seconds
Goal Reached
Cost to the goal  1023.2500342121785
Nodes expantion  10000
```

Figura 22. Resultados reportados.

VI. Conclusiones

El uso del algoritmo RRT* resulta en rutas más cortas y suaves hacia la meta, sin embargo requiere más operaciones y tiempo de cómputo. En los ejemplos de los resultados obtenidos, las trayectorias generadas no son las óptimas globalmente ya que el algoritmo se detiene al encontrar la primer forma de ir de x_init a x_{goal}. En el caso de seguir generando más muestras en el espacio, se lograrían rutas cada vez más cortas. El ejemplo en la figura 3, muestra un resultado muy parecido al que se obtendría con el algoritmo RRT. Esto es debido a que la vecindad es igual al paso máximo de un nodo a otro, es decir, en la vecindad solo existe el nodo más cercano en distancia euclideana (para este caso). Después de varias corridas se determinó usar un tamaño de paso entre nodos de 20 pixeles y una vecindad tres veces mayor a este valor.

Al utilizar el máximo de nodos como criterio de para del algortimo y no el haber encontrado la primer ruta a la meta, se obtienen trayectorias más suaves y óptimas. Sin embargo, el árbol construido es mucho más denso y el tiempo de ejecución crece de manera considerable.