Robótica I

Proyecto Final. Planeación Kinodinámica para uno y dos DDR

2 de junio 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 $\mathcal{C}$.

En este trabajo se muestra la implementación del algoritmo RRT para la planeación de trayectorias en un espacio de dos dimensiones para un robot de control diferencial (DDR) en distintas disposiciones de obstáculos dentro de $\mathcal{C}$. El trabajo se extiende para la implementación de dos DDR simultaneos dentro del mismo espacio. El lenguaje utilizado es *python* y se desarrolla dentro del módulo *pygame*. Se evalúa el desempeño en cada caso.

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 dentro del espacio 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. Robot de Control Diferencial (DDR)

Un robot de control diferencial (Differential Drive Robot), DDR por sus siglas en inglés, es un robot móvil cuyo movimiento se basa en el manejo de dos ruedas puestas a cada lado de la estructura móvil del robot. La dirección del robot puede ser variada de acuerdo a la rotación de cada una de las ruedas por lo que no es necesario el uso adicional de un volante.

Si las dos ruedas tienen la misma dirección y velocidad, el robot se mueve en línea recta . Si las ruedas se mueven con la misma velocidad pero con direcciones opuestas, el robot realiza una rotación en sitio. De otra forma, dependiendo de las velocidades y direcciones de cada una de las llantas, el robot describe un arco de círculo cuyo centro de rotación varia de acuerdo a las velocidades y direcciones de cada llanta.

Considerando que el DDR se mueve en un espacio de dos dimensiones, las ecuaciones en espacio de estados no lineal están dadas por

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{1}$$

Siendo x, y la posición del centro del eje del robot entre las rotaciones de las ruedas relativo al marco global, θ el ángulo relativo a la horizontal del marco de referencia global, v la velocidad lineal del robot, y ω la velocidad angular como lo muestra la figura 1.

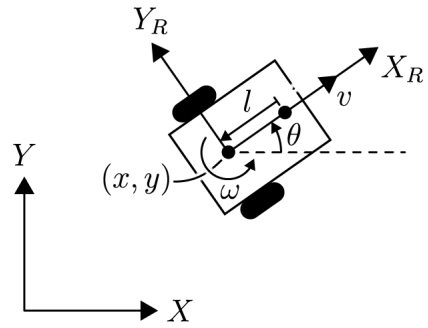

Figura 1. Modelado de un robot de control diferencial (DDR).

IV. Planteamiento del Problema

Se desea generar una ruta libre de colisiones dentro de un espacio bidimensional con obstáculos desde un punto inicial x_{init}, hasta un punto final x_{goal}. Para fines prácticos, se considera como punto meta una vecindad al rededor de x_{goal}. Sea $\mathcal{C}_{free}$ el espacio libre de colisiones, el objetivo del robot es ubicar el centro del robot en el espacio $\mathcal{X}_{goal} \subseteq \mathcal{C}_{free}$. Para este problema, no se considera una orientación final del robot.

El set de controles aplicables al robot consiste en indicar las direcciones de cada una de las ruedas. Es decir,

$$U = \{u_1, u_2\},\ donde\ u_i \in \{-1, 0, 1\}$$

Entonces se considera el set de controles

$$U = \left\{ \begin{array}{cc} 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ -1 & -1 \\ -1 & 0 \\ 0 & -1 \\ -1 & 1 \\ 1 & -1 \end{array} \right\} \tag{2}$$

El control seleccionado de este set es aplicado y mantenido un tiempo Δ_t. Ya que se maneja un tiempo de aplicación y retención del control, se consideran las soluciones a (1) de la forma discretizada

$$\begin{array}{rcl} x_{k+1} & = & x_k + \frac{u_1+u_2}{2} v_{max} cos(\theta_k) \Delta_t \\ y_{k+1} & = & x_k + \frac{u_1+u_2}{2} v_{max} sin(\theta_k) \Delta_t \\ \theta_{k+1} & = & x_k + \frac{u_2-u_1}{2} \omega_{max} \Delta_t \end{array} \tag{3}$$

donde

$$\begin{array}{c} x_0 \\ y_0 \\ \theta_0 \end{array}$$

es la configuración inicial del robot, o en otras palabras, el punto de partida. Las velocidades, tanto la lineal como la angular, del robot están definidas por v_{max} y ω_{max}.

La manera en que se elige y aplica el control en U se hace generando un número entero entre 0 y 8 de forma aleatoria, que corresponde a cada uno de los elementos en U de forma ordenada. El control seleccionado se simula un tiempo Δ_t fijo y se guarda en el árbol generado la trayectoria descrita en $\mathbb{R}^2$ siempre y cuando está se encuentre libre de colisiones.

El algoritmo utilizado es el siguiente:

Algorithm 1: Algoritmo RRT kinodinamico

Data: x_{init}, $\mathcal{X}_{goal}$, Δ_t
for $k = 0$ *to* n **do**
- Escoger un punto x_{rand} en $\mathcal{C}_{free}$
- Encontrar el nodo del árbol más cercano a x_{rand}, x_{near}
- Elegir un control de forma aleatoria u_{rand}
- Aplicar el control por un tiempo Δ_t
- Si se genera un camino libre de colisiones agregar la ruta al árbol T
- Romper si se llega a la meta

return T

El algoritmo se detiene si se llega a la meta o se llega a un máximo de nodos n en el árbol.

El objetivo del problema es, entonces, encontrar la serie de controles U tales que lleven al DDR desde una configuración inicial en x_{init} a una configuración final en $\mathcal{X}_{goal}$ mediante el uso del algoritmo RRT. Evidentemente la solución, de ser encontrada, no protende ser óptima y no se garantiza que se genere una ruta en tiempo finito.

El problema se extiende a encontrar una ruta libre de colisiones para dos DDR dentro del espacio bidimensional, con las restricciones adicionales que debe existir una distancia mínima y máxima entre los centros de cada robot. Esto para asegurar una distancia de seguridad que garantice que ambos DDR no colisionen y al mismo tiempo intentar acortar el cálculo de las posibles rutas manteniendo los robots uno cerca del otro. En este caso, el algoritmo reporta éxito si cualquiera de los dos DDR llega a $\mathcal{X}_{goal}$.
Se proponen las distancias

$$d_{min} = DDR_{diam}$$

$$d_{max} = \frac{3}{2} DDR_{diam}$$

donde DDR_{diam} es el diamétro del espacio de seguridad del DDR, entendiéndose éste como un robot disco virtual dentro del espacio euclideano de dos dimensiones.

V. Resultados

El algoritmo se implementa en el lenguaje de programación *Python* haciendo uso del módulo **pygame**. Se muestran los resultados para un espacio sin obstáculos y otros dos espacios con obstáculos cuyas configuraciones se muestran en las figuras siguientes. Para todos los casos se considera

$$DDR_{diam} = 80\ pixeles$$

$$v_{max} = 100\ pixeles/segundo$$

$$\omega_{max} = 360^o/segundo$$

$$\Delta_t = 0{,}5\ segundos$$

$$n = 50000$$

Debido a la forma de trabajo del módulo *pygame*, se toman las medidas en pixeles. El espacio en el que se desarrolla el algoritmo es un espacio rectangular de dimensiones 1200×800.
Los resultados obtenidos son los siguientes. La nave dentro de las imágenes y las animaciones representa el DDR.

Espacio libre de obstáculos

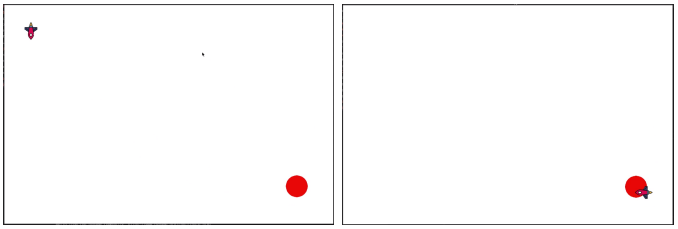

Figura 3. Posición inicial y final en un espacio libre de obstáculos.

$$x_{init} = (97, 95)$$

$$x_{goal} = (1067, 663)$$

Tiempo para calcular la trayectoria = 3.6448 segundos
Tiempo para alcanzar la meta = 33.9026 segundos
Total de nodos generados = 1090.

Primer configuración de obstáculos

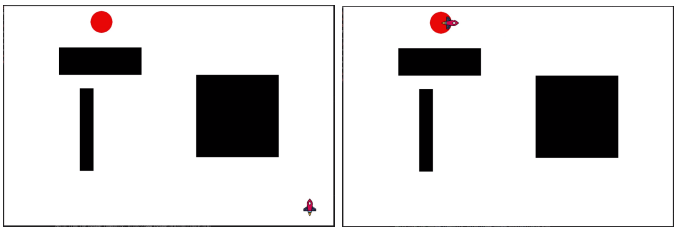

Figura 4. Posición inicial y final en la primer configuación de obstáculos.

$$x_{init} = (1114, 734)$$

$$x_{goal} = (355, 58)$$

Tiempo para calcular la trayectoria = 5.0408 segundos
Tiempo para alcanzar la meta = 40.7216 segundos
Total de nodos generados = 1413.

Segunda configuración de obstáculos

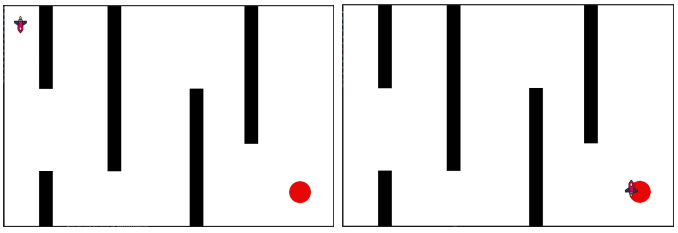

Figura 5. Posición inicial y final en la segunda configuación de obstáculos.

$$x_{init} = (58, 67)$$

$$x_{goal} = (1079, 677)$$

Tiempo para calcular la trayectoria = 6.6721 segundos
Tiempo para alcanzar la meta = 48.4012 segundos
Total de nodos generados = 1814.

Vistos los resultados obtenidos en las animaciones, se propone la eliminación de los controles que generan los movimientos en reversa del DDR para lograr un movimiento más "natural". Los resultados se muestran a continuación (se omiten las imágenes ya que éstas son muy similares a las figuras 3-5).

Espacio libre de obstáculos

$$x_{init} = (160, 128)$$

$$x_{goal} = (1071, 699)$$

Tiempo para calcular la trayectoria = 2.8852 segundos
Tiempo para alcanzar la meta = 26.4933 segundos
Total de nodos generados = 883.

Primer Configuración de obstáculos

$$x_{init} = (1055, 698)$$

$$x_{goal} = (381, 75)$$

Tiempo para calcular la trayectoria = 7.5808 segundos
Tiempo para alcanzar la meta = 26.6094 segundos
Total de nodos generados = 1990.

Segunda Configuración de obstáculos

$$x_{init} = (57, 90)$$

$$x_{goal} = (1047, 651)$$

Tiempo para calcular la trayectoria = 64.9712 segundos
Tiempo para alcanzar la meta = 35.6841 segundos
Total de nodos generados = 9245.

Implementación de dos DDR en el ambiente

Como se menciona en la sección IV, se añade la versión en la que se incluye un DDR adicional en el ambiente. Los resultados se muestran a continuación.

Espacio libre de obstáculos

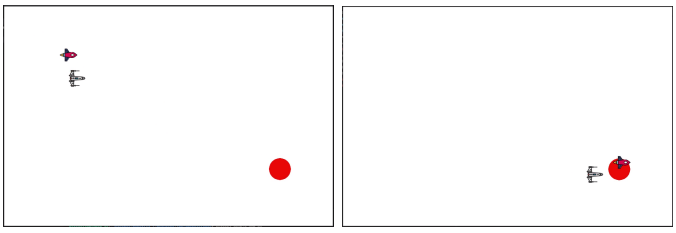

Figura 6. Posición inicial y final en un espacio libre de obstáculos para dos DDR's.

$$x1_{init} = (235, 177)$$

$$x2_{init} = (266, 262)$$

$$x_{goal} = (1007, 593)$$

Tiempo para calcular la trayectoria = 22.3014 segundos
Tiempo para alcanzar la meta = 23.3179 segundos
Total de nodos generados = 3577.

Primer configuración de obstáculos

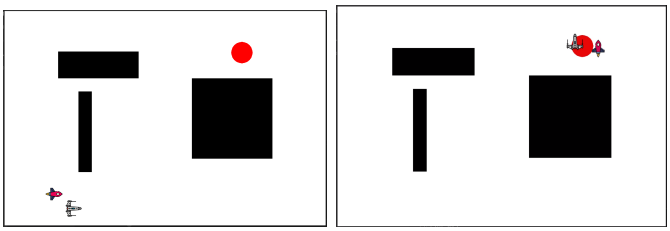

Figura 7. Posición inicial y final en la primer configuación de obstáculos para dos DDR's.

$$x1_{init} = (235, 177)$$

$$x2_{init} = (266, 262)$$

$$x_{goal} = (1007, 593)$$

Tiempo para calcular la trayectoria = 544.5362 segundos
Tiempo para alcanzar la meta = 41.2351 segundos
Total de nodos generados = 31935.

Segunda configuración de obstáculos En la mayoría de las implementaciones con la segunda configuración de obstáculos no se logra convergencia. Por tanto no se incluyen resultados. Una opción para buscar la convergencia es incrementar el total de nodos permitidos en el árbol, sin embargo esto también significa aumentar el tiempo de cómputo. Para $n = 50000$ se genera un tiempo de 584 segundos, cerca de 10 minutos.

VI. Conclusiones

El uso del algoritmo RRT es uno de los más utilizados en el área de robótica para la planificación de trayectorias. En este trabajo, este algoritmo es usado para encontrar el set de controles para el manejo de uno y dos DDR en un espacio de dos dimensiones con y sin obstáculos.

Se incluyen videos que muestran el funcionamiento del algoritmo desarrollado conn ayuda del módulo *pygame* de *Python*.

Aunque se logra el objetivo de llevar a los robots desde un punto inicial x_{init} a un espacio final $\mathcal{X}_{goal}$, no se genera una trayectoria óptima.
El programa tiene puntos de mejora de los que se puede destacar:

- Generar trayectorias más suaves
- Proponer el uso del RRT*
- Producir una animación más fluida de los resultados
- Delinear la trayectoria generada
- Mostrar el espacio explorado

Los documentos que sirvieron como motivación para este trabajo son:

- *Kinodynamic RRTs with Fixed Time Step and Best-Input Extension Are Not Probabilistically Complete* de Tobias Kunz and Mike Stilman.
- *Randomized Kinodynamic Planning*, de Steven M. LaValle y James J. Kuffner Jr.