# Robótica. Proyecto Final. Implementación de RRT en DDR

Baruc Samuel Cabrera García

5 de diciembre de 2023

# RRT

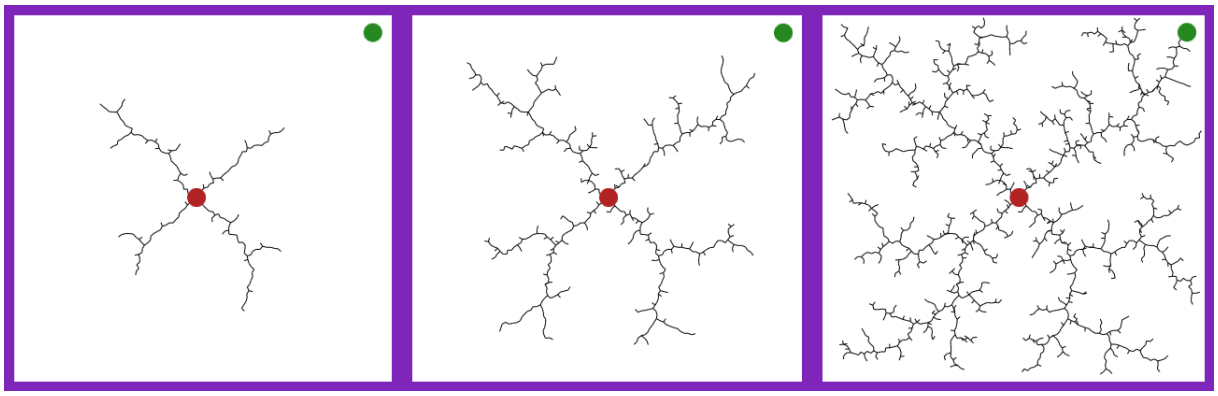
Rapidly-exploring Random Tree, o mas conocido por sus siglas, RRT. Es el algoritmo el cual desarrollaremos en este proyecto sobre un DDR, que es un Differential Drive Robot.

Pero antes de proseguir con el desarrollo de tal algoritmo sobre el robot ya mencionado, es necesario mencionar las características principales sobre el algoritmo RRT, para así implementarlo adecuadamente sobre el robot.

Primero, el concepto de RRT es el de una estructura de datos aleatoria que está diseñada para una amplia clase de problemas de planificación de rutas. Un RRT es expandido iterativamente al aplicar input de control que dirigen al sistema ligeramente hacia puntos seleccionados aleatoriamente.

Ahora, considerando el espacio $X$, un RRT es construido de tal forma que cada vértice suyo sea un estado que se encuentre en $X_{free}$, el complemento de $X_{obs}$. Además, cara arista de RRT corresponderá a un camino que este completamente contenido en $X_{free}$.

Figura 1:



Ahora, para entender mejor como funciona RRT, hay que mirar su pseudocódigo.

Figura 2:

**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** $G = (V, E)$;

Recordemos que el objetivo de RRT es la de construir un grafo, por lo que empezamos el algoritmo con un grafo $G = (V, E)$. Este grafo inicial consta unicamente de un vértice, el cual es el estado inicial $x_{init}$.

Luego, se inicia un proceso de $n$ iteraciones, donde en cada iteración se realizan los pasos siguientes:

1. $x_{rand} \leftarrow SampleFree$: Seleccionamos un punto aleatorio en nuestro espacio.
2. $x_{nearest} \leftarrow Nearest = (G = (V, E), x_{rand})$: Calculamos el vértice del grafo $G$ mas cercano a $x_{rand}$.

3. $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$: Dados estos vértices, podemos generar una dirección de avance según un tamaño de paso $\mu$. Esto es, tomaremos como $x_{new}$ al estado mas cercano a $x_{rand}$, tal que se encuentre en el segmento $\overline{x_{nearest}, x_{rand}}$ y $d(x_{new}, x_{rand}) = \mu$.

4. $ObstacleFree(x_{nearest}, x_{new})$: Si no hay obstáculos entre $x_{nearest}$ y$x_{new}$, añadimos a $x_{new}$ al grafo, es decir:

$$V \leftarrow V \cup \{x_{new}\}, \quad E \leftarrow E \cup \{(x_{nearest}, x_{new})\} \quad .$$

Una vez que tenemos nuestro grafo que se expande, si queremos llegar a una zona en específico ($q_{goal}$), basta con comprobar si el nuevo vértice a añadir esta en tal zona. En tal caso, la ruta estaría definida por las conexiones que unen a tal vértice con con la raíz de grafo.

# DDR.

Un DDR es un robot cuyo movimiento se basa en dos ruedas separadas colocadas en los extremos del robot. Por lo tanto, le permite avanzar y retroceder, ademas de cambiar su dirección cambiando la velocidad relativa de las ruedas, lo que le permite prescindir de movimientos de dirección adicionales. A continuación podemos ver un esquema de como es un DDR.

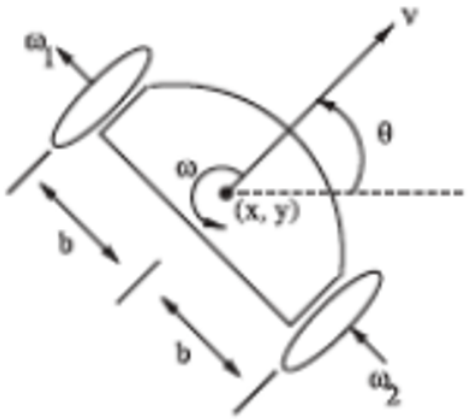



Figura 3:

Como se puede apreciar, el robot tiene un centro $(x, y)$, cada llanta esta separada de dicho centro a una distancia $b$, y $\theta$ es el ángulo de la dirección del robot respecto al eje $x$. Otro aspecto a notar, es el uso de $\omega, v$, los cuales representan la velocidad angular y lineal del robot, los cuales son resultado de las distintas velocidades angulares de las ruedas, las cuales llamaremos $\omega_l, \omega_r$. Esto mismo nos permite generar el siguiente espacio de control.

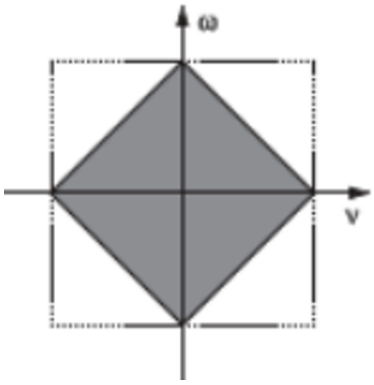



Figura 4:

En el Espacio de Control previo podemos notar que los ejes están representados pos los la velocidad lineal y angular, de. Lo que representa, es que durante la máxima velocidad lineal, no hay velocidad angular, y viceversa. Dicha propiedad se expresa en la desigualdad

$$|\omega^{max}| \leq \frac{1}{b}|V^{max} - |v||,$$

donde $v$ es la velocidad lineal actual, y $V^{max}$ es la velocidad máxima linear.

Podemos obtener las velocidad lineal y angular con base en las velocidades angulares $\omega_l, \omega_r$, ademas de la información sobre las dimensiones del robot, es decir, su longitud $2b$ y el radio de sus llantas $r$, como se ve a continuación.

$$\begin{pmatrix} v \\ \omega \end{pmatrix} := \begin{pmatrix} r\left(\frac{\omega_l+\omega_r}{2}\right) \\ r\left(\frac{\omega_l-\omega_r}{2b}\right) \end{pmatrix} = \begin{pmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{2} & \frac{-r}{2b} \end{pmatrix} \begin{pmatrix} \omega_l \\ \omega_r \end{pmatrix}.$$

Luego, expresamos nuestras ecuaciones de transición de estado de la siguiente forma.

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

Luego, si expresamos dichas ecuaciones en términos de las velocidades angulares de las llantas, tenemos que

$$\begin{aligned} \dot{x} &= r\left(\frac{\omega_l+\omega_r}{2}\right)\cos(\theta), \\ \dot{y} &= r\left(\frac{\omega_l+\omega_r}{2}\right)\sin(\theta), \\ \dot{\theta} &= r\left(\frac{\omega_l-\omega_r}{2b}\right). \end{aligned}$$

Con base en lo anterior, proponemos el sistema siguiente para el cálculo de los estados:

$$\begin{aligned} x_{k+1} &= x_k + h\cdot\dot{x}_k(\omega_l,\omega_r) = x_k + h\left(r\left(\frac{\omega_l+\omega_r}{2}\right)\cos(\theta_k)\right), \\ y_{k+1} &= y_k + h\cdot\dot{y}_k(\omega_l,\omega_r) = y_k + h\left(r\left(\frac{\omega_l+\omega_r}{2}\right)\sin(\theta)\right), \\ \theta_{k+1} &= \theta_k + h\cdot\dot{\theta}_k(\omega_l,\omega_r) = \theta_k + h\left(r\left(\frac{\omega_l-\omega_r}{2b}\right)\right). \end{aligned}$$

Donde los casos iniciales están dados están dados por la posición inicial del robot, y $h$ es el tamaño de salto de tiempo con un límite de $\Delta t$.

# RRT en DDR.

Primero, para este ejercicio tomaremos los valores de las velocidades angulares de las llantas en un entorno discreto $\{-1, 1\}$, es decir, que consideraremos unicamente movimientos en línea recta o rotaciones en sitio.

Ahora, recordemos que en RRT, para el proceso de construcción del grafo se requería de generar un $x_{new}$ en base a un $x_{nearest}$ y un $x_{rand}$.

Para la generación de un $x_{rand}$ podemos proseguir normalmente tomando los valores $(x, y, \theta)$ de manera uniforme según los límites del escenario y el intervalo $[0, 2\pi)$.

Para encontrar $x_{near}$, utilizamos la métrica entre estados

$$d(X, X') = \sqrt{(x - x')^2 + (y - y')^2 + \alpha^2},$$
$$\alpha = \min\left\{|\theta - \theta'|, 2\pi - |\theta - \theta'|\right\}.$$

Finalmente, para generar un nuevo $x_{new}$, procederemos en base al siguiente pseudocódigo:

1. Para $\omega_l, \omega_r \in \{-1, 1\}^2$:
   - *a*) Tomamos $(x_0, y_0, \theta_0) = x_{nearest}$.
   - *b*) Tomamos count_time $= 0$
   - *c*) While count_time $\leq \Delta t$:
     1) $x_{k+1} = x_k + h\left(r\left(\frac{\omega_l + \omega_r}{2}\right)\cos(\theta_k)\right)$.
     2) $y_{k+1} = y_k + h\left(r\left(\frac{\omega_l + \omega_r}{2}\right)\sin(\theta)\right)$.
     3) $\theta_{k+1} = \theta_k + h\left(r\left(\frac{\omega_l - \omega_r}{2b}\right)\right)$.
     4) Si $(x_{k+1}, y_{k+1}, \theta_{k+1})$ choca con un obstáculo, volver al paso 1.
     5) count_time $=$ count_time $+ h$
   - *d*) Agregar a $(x_{k+1}, y_{k+1}, \theta_{k+1})$ a *Candidatos*.
2. Si $|Candidatos| \neq 0$, tomamos como $x_{new} = \arg\min_X d(X, x_{nearest})$.
3. Si $|Candidatos| = 0$, no hay $x_{new}$.

Este pseudocódigo contiene lo que serían las líneas 6 y 7 del siguiente pseudocódigo:

**Algorithm 1:** BuildRRT($x_{\text{init}}, \mathcal{X}_{\text{goal}}$)

```
1  V ← {x_init};
2  E ← ∅;
3  while V ∩ X_goal = ∅ do
4  |   x_rand ← SampleState();
5  |   x_near ← NearestNeighbor(V, x_rand);
6  |   (x_new, u_new, Δt) ← NewState(x_near, x_rand);
7  |   if CollisionFree(x_near, x_new, u_new, Δt) then
8  |   |   V ← V ∪ {x_new};
9  |   |   E ← E ∪ {(x_near, x_new, u_new, Δt)};
10 return (V, E);
```

Figura 5:

## Implementación

Con base en lo anterior, tenemos todo lo necesario para implementar el código de interés.

Se adjuntaran dos códigos en Python de nombre "RRT_on_DDR_routes.py" y "RRT_on_DDR_move.py"con un ejemplo para los valores siguientes:

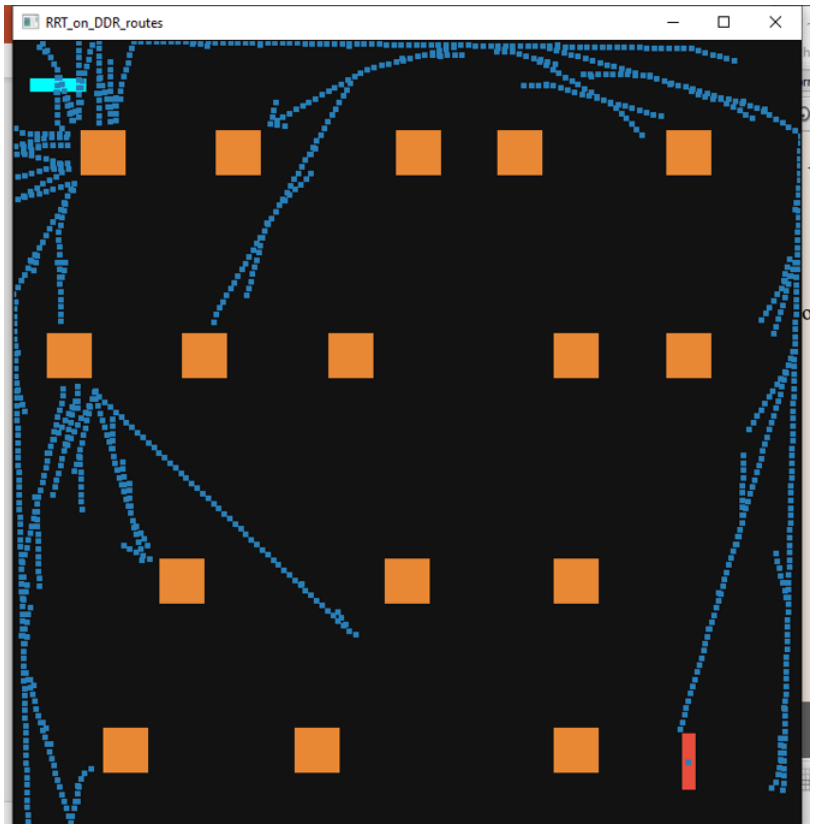
- max_iter: 5000,10000,15000,20000
- tol_error: 30 (para detectar a $x_{goal}$)
- Ancho de escena: 700
- Altura de escena: 700
- $b$: 25
- $r$: 6
- $\Delta t$: 1.0
- $h$: 0.1

Los demás valores como $x_{start}$, $x_{goal}$ y obstáculos se declaran en los códigos. El primer código, gráfica los obstáculos, ademas de los estados $x_{goal}$ y $x_{start}$. Luego, repite el proceso de buscar añadir vértices al grafo a lo mas 15000, o hasta encontrar a $x_{goal}$ a una distancia menor o igual a $tol\_error$ de algún vértice del grafo.

A continuación, mostraremos algunos ejemplos del código "RRT_on_DDR_routes.py" variando el valor de max_iter, es necesario mencionar que no necesariamente el tamaño de grafo es proporcional al valor de max_iter, ya que aun cabe la posibilidad de que se encuentre el objetivo a una cantidad baja de iteraciones.

**5000)**

Figura 6:



**10000)**

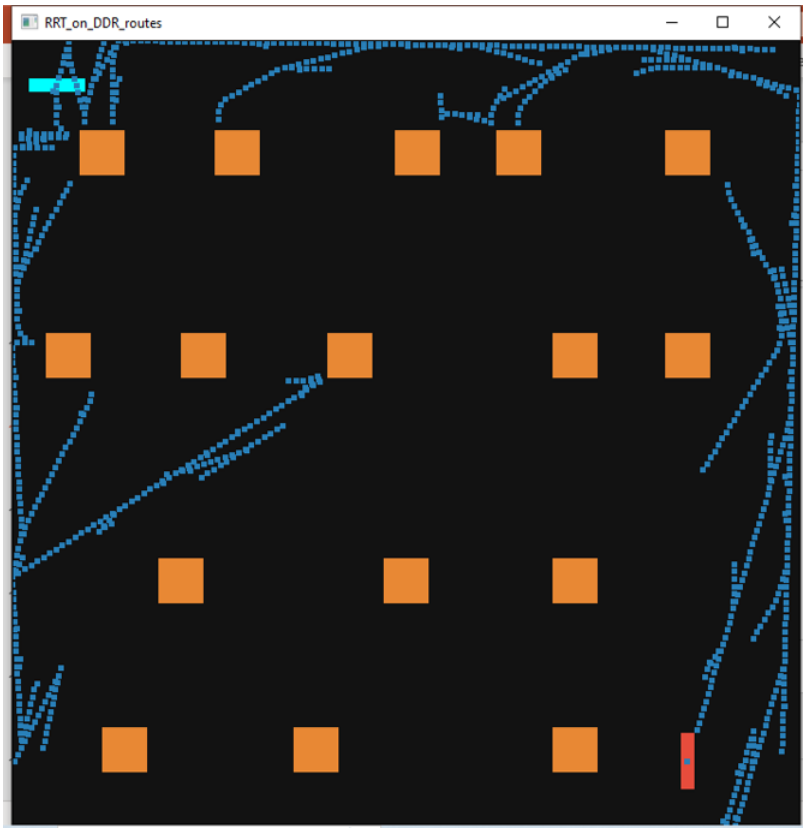
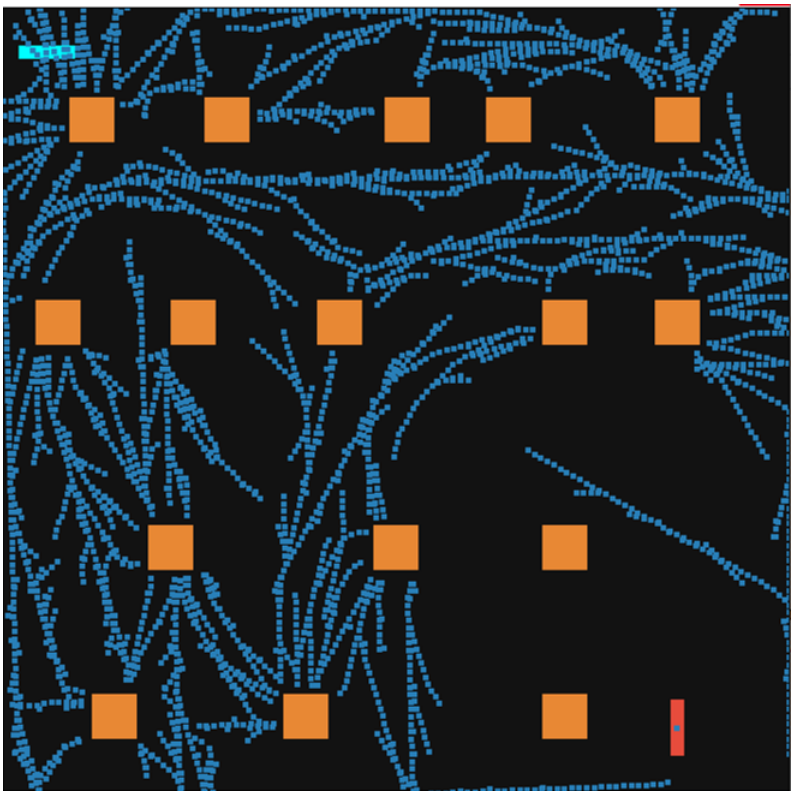



Figura 7:

**15000)**

Figura 8: 

**20000)**

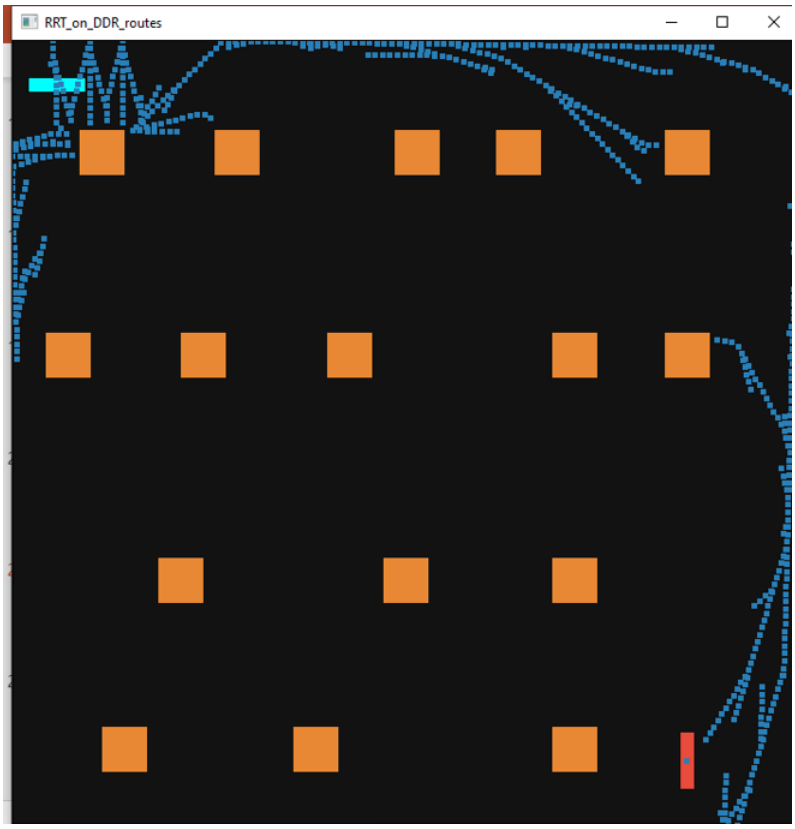
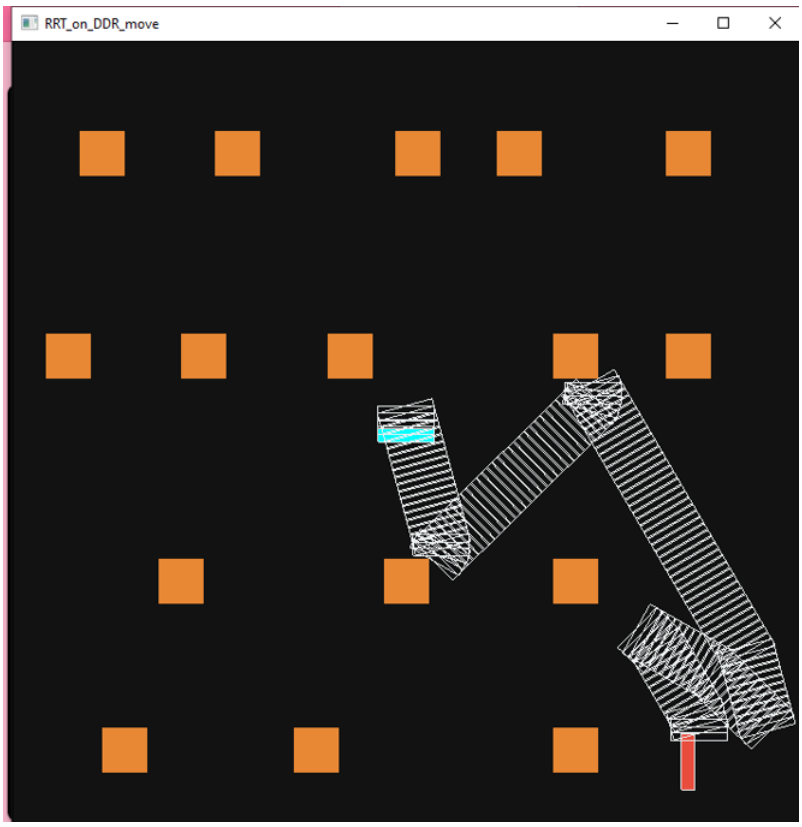



Figura 9:

Luego, el código "RRT_on_DDR_move.py" gráfica unicamente a los obstáculos y los estados $x_{start}$ y $x_{goal}$, para luego visualizar los pasos que requiere el robot para llegar hacia el objetivo en caso de encontrar de que el grafo haya encontrado a $x_{goal}$. Dicha visualización se representa graficando las siluetas del robot en tales estados, lo que nos permite apreciar de mejor manera los estados requeridos.

Figura 10: 


Como se puede apreciar, el código es capaz de generar un camino libre de colisiones desde $x_{start}$ hacia $x_{goal}$.