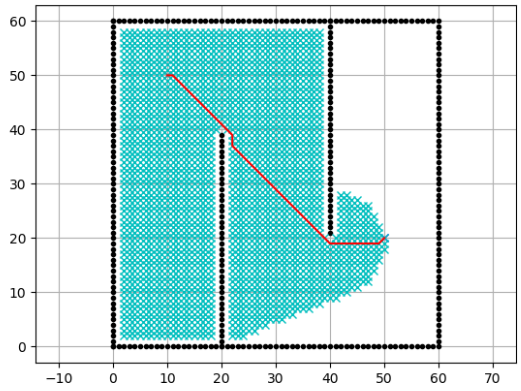
Ignacio Gonzalez Portillo

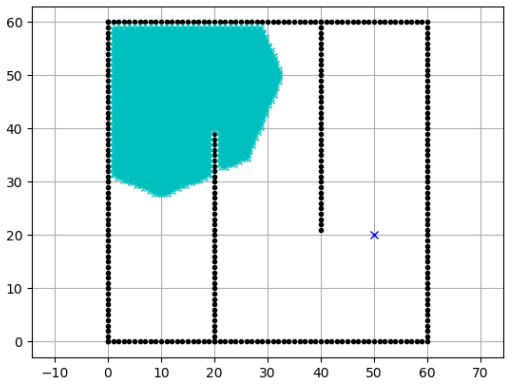
Ejercicio 1: Dijkstra Global Planner

1.2 Capturas

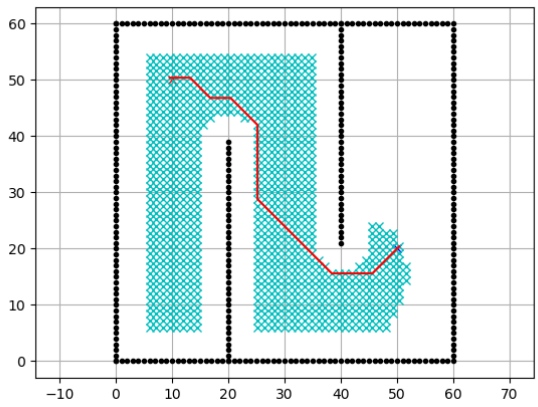
Modificación posición inicial y final



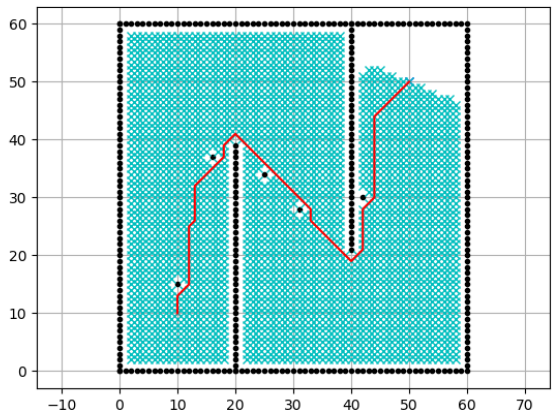
Modificación resolución



Modificación tamaño del robot



Obstaculos añadidos



En el caso en el que se añaden obstáculos podemos ver como la trayectoria generada es similar a la original, pero en este caso se evitan los obstáculos, pues en estas zonas no hay nodos y por lo tanto no hay caminos posibles para recorrer.

1.3 Explicación

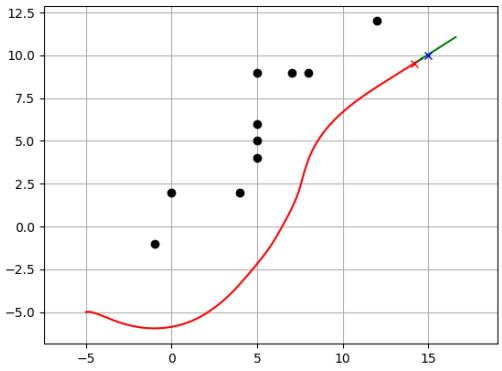
En general Dijkstra es un algoritmo capaz de producir trayectorias libres de colisión entre dos puntos en un grafo de nodos, asegurando el camino más óptimo posible. En un grafo de nodos el camino para ir de un nodo a otro (edge) es pesado según diferentes factores, en este caso la distancia recorrida. El algoritmo se encarga de visitar todos los nodos adyacentes al nodo donde se encuentra y asignar un peso a cada camino según la distancia recorrida para ir de un nodo a otro y de esta manera se expande hasta visitar todos los nodos del grafo. Una vez todos los nodos han sido visitados, y todos los caminos han sido pesados, dado un punto de origen y un punto final (dos nodos) el algoritmo puede dar el camino que conecte ambos nodos recorriendo la menor distancia, es decir, el camino con menor peso. En el caso concreto aquí tratado y en la mayoría de implementaciones, el algoritmo no recorre absolutamente todos los nodos, pues desde el inicio se conoce cual es la meta o nodo al que se desea llegar y el nodo de origen, por lo que el algoritmo se expande partiendo de la salida hacia los nodos adyacentes hasta que llega al nodo de la meta y genera el camino más corto. Esto permite ahorrar coste computacional al no visitar todos los nodos del grafo.

Ignacio Gonzalez Portillo

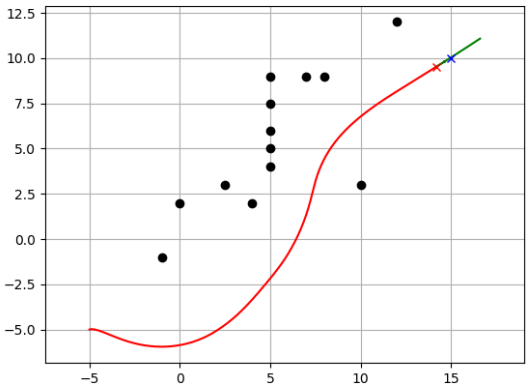
Ejercicio 2: Dynamic Window Approach local planner (Ventana Dinámica)

2.2 Capturas

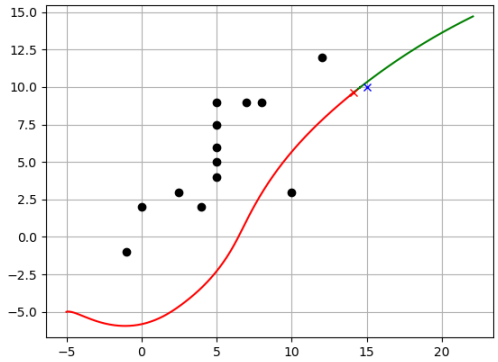
Modificación salida y meta



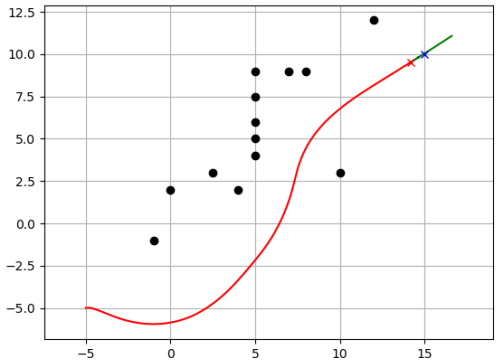
Obstaculos añadidos



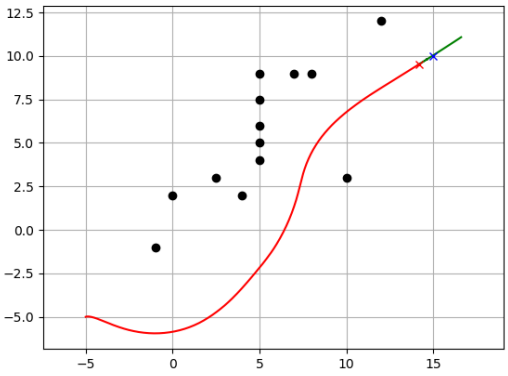
Aumento velocidad máxima



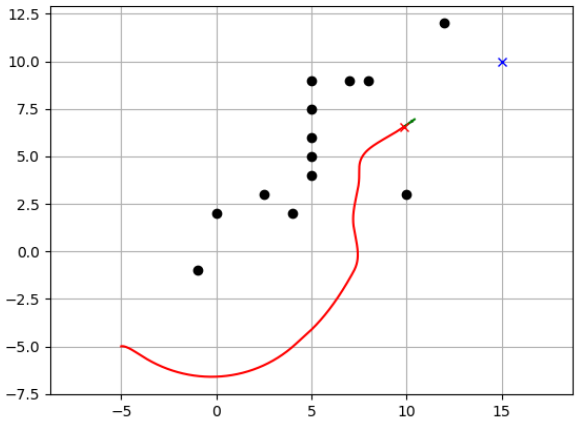
Disminución velocidad mínima



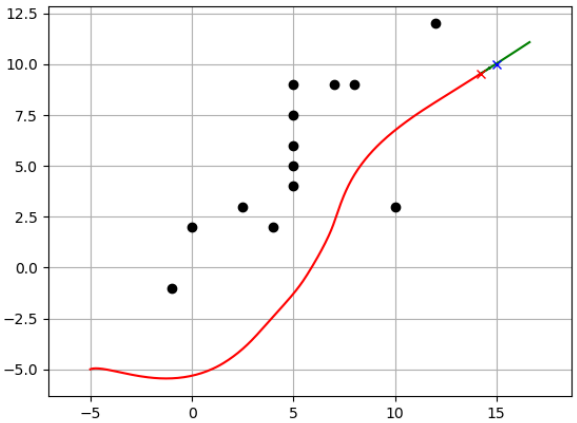
Aumento velocidad angular máxima



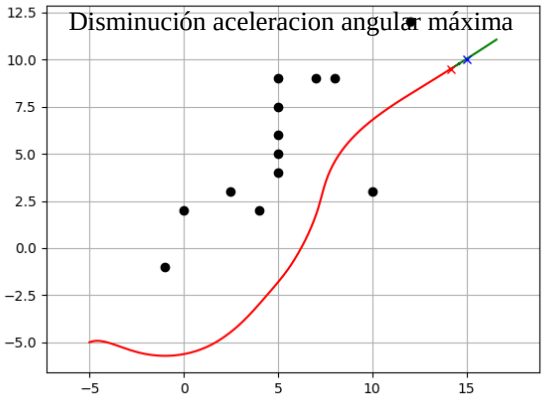
Disminución aceleración máxima



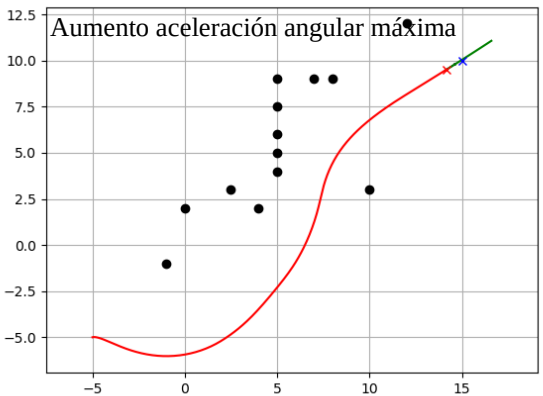
Aumento aceleración máxima



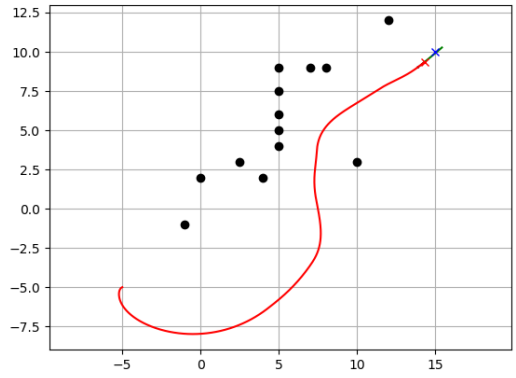
Disminución aceleracion angular máxima



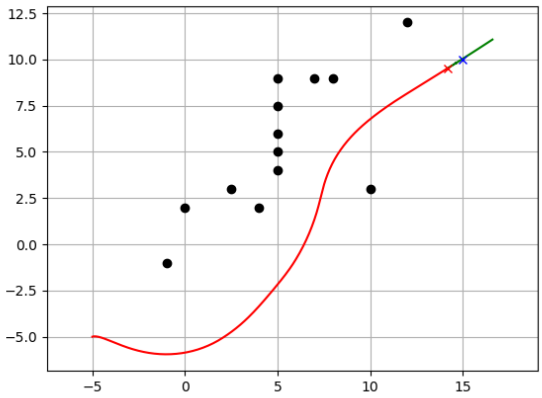
Aumento aceleración angular máxima



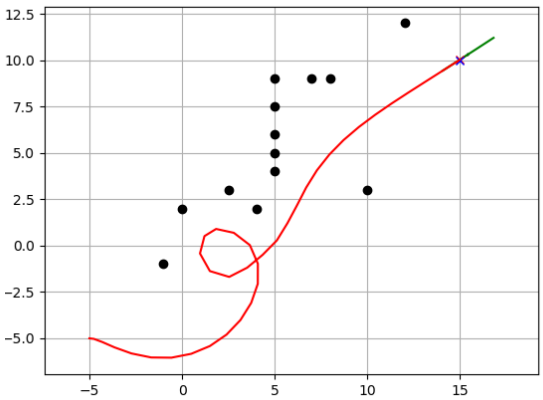
Aumentar resolución velocidad



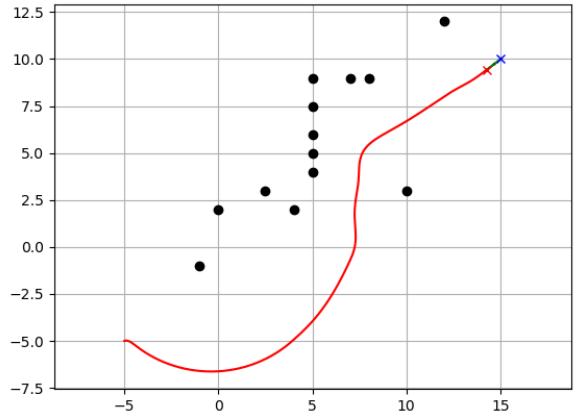
Aumentar coste velocidad



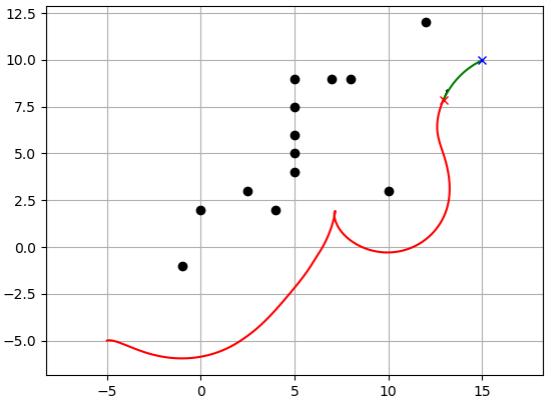
Aumentar diferencial de tiempo



Disminución predicción tiempo



Aumentar radio robot



Se adjuntan algunos de los resultados de introducir variaciones en los parámetros de serie del planificador local.

Primero se han variado los parámetros referentes a la dinámica del movimiento del robot (velocidades y aceraciones máximas), podemos apreciar pocas variaciones en la trayectoria seguida. Estos parámetros amplían o restringen el espectro de velocidades y aceleraciones que serán evaluadas por el algoritmo para proponer soluciones, entoces si no se excluye fuera de los limites ninguna solución valida, las trayectorias no deberían verse afectadas por estos parámetros.

En cuanto al resto de parámetros podríamos decir que sí tienen una mayor influencia en la geometría de la trayectoria. La resolución de las velocidades afecta en como se realiza el muestreo del aspectro de velocidades a evaluar, a cuanta más resolución menor sera la discretización del aspectro de velocidades y menos soluciones se podrán evaluar y proponer. Disminuir la resolución hará que la trayectoria sea más "precisa" a costa de aumentar el coste computacional. El mismo concepto se aplica para la resolución de la velocidad angular.

El diferencial de tiempo determina el tiempo que se aplica cada solución, és decir cuanto tiempo se recorre cada arco de circunferencia, por este motivo al aumentar este parámetro las trayectorias son cada vez más circulares pues cada familia de velocidades escogidas como solución, se aplican de manera constante durante más tiempo.

Las ganancias del coste de la orientación a la meta y del coste de velocidad, sirven para pesar ambos factores en la *Objective Function* que se optimiza para encontrar las soluciones. Por defecto ambos factores tienen el mismo peso y deben fijarse en función de la geometria del mapa.

El radió del robot tiene un claro efecto en la trayectoria por restricciones geométricas obvias.

Cabe destacar que algunos de los parámetros presentan limitaciones, y al subir por encima de un cierto valor el algoritmo no es capaz de encontrar una trayectoria valida para llegar a la meta, esto sucede por ejemplo con la predicción de tiempo. Este parámetro controla para cuanto tiempo se va a simular la trayectoria de cada familia de velocidades

En general cada uno de estos parámetros tiene una influencia particular sobre el comportamiento del algoritmo, pero es difícil dar una explicación exacta e unívoca de como afectan individualmente cada uno de ellos a la trayectoria final. En mi opinión, para cada aplicación se debería realizar un "tunning" exhaustivo y exclusivo de estos parámetros para ajustarlos al comportamiento deseado en cada caso y a la geometría del entorno.

2.3 Explicación

Este algoritmo funciona proponiendo soluciones en el espacio de velocidades (angular y lineal), estas soluciones cumplen por un lado con las restricciones del modelo dinámico del robot y por otro lado solo se incluirán aquellas que no den lugar a colisión o bien que el robot pueda frenar antes de esta. Las soluciones propuestas son trayectorias circulares que se caracterizan por una velocidad lineal y angular únicas. Estas velocidades se aplicarán durante un intervalo de tiempo determinado recorriendo solo una parte del arco. A cada intervalo de tiempo se proponen nuevas soluciones. A cada vuelta del loop, el algoritmo solo tiene en cuenta las velocidades del primer intervalo, asumiendo que la velocidad es constante en el resto. Entre las velocidades clasificadas como admisibles, la ventana dinámica restringe el espacio de la solución a aquellas a las que el robot pueda llegar en el siguiente intervalo de tiempo con su aceleración actual es por este motivo que para cada intervalo de tiempo se proponen nuevas soluciones.

Una vez el espacio de soluciones (velocidades) esta definido y delimitado, se utiliza una función llamada "*Objective Function"* que realiza una suma ponderada de los siguientes factores; "target *heading*" (progreso/ orientación del sistema respecto a la meta, mayor como mejor sea la orientación respecto a la meta), "*clearance*" (distancia al objeto de colisión más próximo de la trayectoria, mayor como mayor sea la distancia a la colisión), y "*velocity*" (medida del avance del robot dentro de la trayectoria circular evaluada).

El objetivo final es optimizar esta función de manera que se encuentre la velocidad (lineal y angular) que maximize esta función.

Objective Function

$$G(v; !) = \sigma * (\alpha * heading(v; w) + \beta * dist(v; w) + \gamma * velocity(v; w))$$

En el código trabajado en el ejercicio, la "*Objective Function*" se encuentra implementada de la siguiente forma:

$$to\_goal\_cost = calc\_to\_goal\_cost(traj, goal, config)$$

$$speed\_cost = config.speed\_cost\_gain * /(config.max\_speed - traj[-1, 3])$$

$$ob\_cost = calc\_obstacle\_cost(traj, ob, config)$$

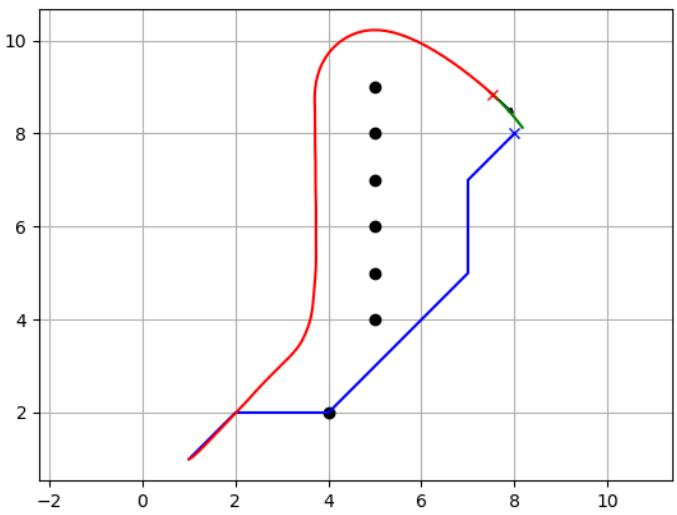
$$final\_cost = to\_goal\_cost + speed\_cost + ob\_cost$$

Podemos ver como la expresión que encontramos implementada es idéntica a la original, excepto porque no aparece un termino análogo a la beta que se utiliza para pesar el factor de distancia de colisión, y además tampoco aparece el término sigma (σ) multiplicando la suma de todos los factores.

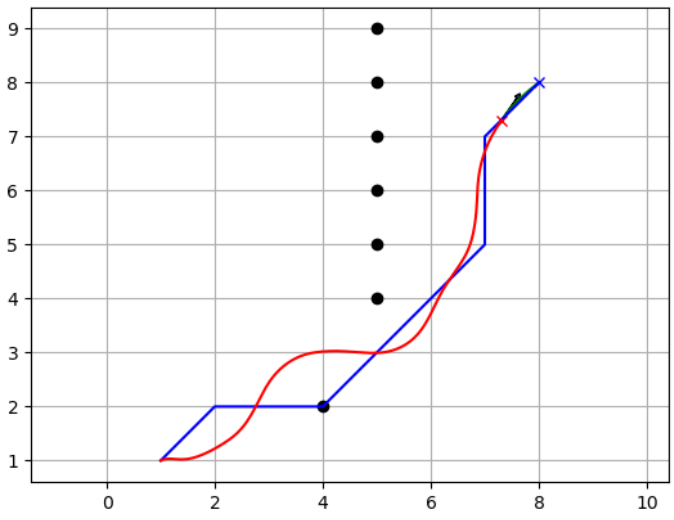
Ejercicio 3:

El script de python se encuentra adjunto junto con este pdf, o bien puede encontrarse en el siguiente repositorio:
https://github.com/nachogo23/Path_Planning.git (python_scripts/dijkstra.py)

DWA sin seguir el path de Dijkstra



DWA siguiendo el path de Dijkstra



Ignacio Gonzalez Portillo