Robot Control: Integrate path-planning and task-planning

Sergio de la Mata Moratilla
Javier Pastor Moreno
3º GII A

índice

Introducción

En este documento se van a responder a las preguntas planteadas por la práctica a parte de cierta información en relación con lo que se ha hecho para realizar el completo y correcto funcionamiento de la práctica junto con los resultados obtenidos.

Path-planning

Implementación

Con relación a este apartado, se pedía completar las heurísticas: euclídea y de manhattan además de incluir la octil y generar el algoritmo de Theta* basado en su pseudocódigo y el de A*.

Los códigos de las distintas heurísticas se muestran en la siguiente imagen y muestran distintas maneras de obtener el valor de la heurística entre dos posiciones del escenario de estudio.

```
def manhattan(point,point2):
    """
        Function that performs Manhattan heuristic.
    """
    #Absolute value of the difference from the x grid reference of points
    diference_posx = abs(point2.grid_point[0] - point.grid_point[0])
    #Absolute value of the difference from the y grid reference of points
    diference_posy = abs(point2.grid_point[1] - point.grid_point[1])
    #Return the addition of the two absolute differences
    return diference_posx + diference_posy

pp.register_heuristic('manhattan', manhattan)

def naive(point, point2):
    """
        Function that performs a naive heuristic.
    """

    return 1

pp.register_heuristic('naive', naive)

def euclidean(point, point2):
    """
        Function that performs euclidean heuristic.
    """
    #Square value of the difference from the x grid reference of points
    square_diference_posx = (point2.grid_point[0] - point.grid_point[0])**2
    #Square value of the difference from the y grid reference of points
    square_diference_posy = (point2.grid_point[1] - point.grid_point[1])**2
    #Return the square root of the additions of the square differences
    return math.sqrt(square_diference_posx + square_diference_posy)

pp.register_heuristic('euclidean', euclidean)

def octile(point, point2):
    """
        Function that performs octile heuristic.
    """
    #Absolute value of the difference from the x grid reference of points
    diference_posx = abs(point2.grid_point[0] - point.grid_point[0])
    #Absolute value of the difference from the y grid reference of points
    diference_posy = abs(point2.grid_point[1] - point.grid_point[1])
    #Return the estimation of the distance between two positions taking into account rows, columns and diagonals
    #1.414 is the constant which will be used if the two positions can be measured with a diagonal in the grid
    return 1.414 * min(diference_posx, diference_posy) + abs(diference_posx - diference_posy)
pp.register_heuristic('octile', octile)
```

La heurística de Manhattan está basada en la suma de las diferencias absolutas entre las coordenadas x e y de inicio y fin.

La heurística euclídea hace uso de la raíz cuadrada de la suma de los cuadrados de las diferencias entre las coordenadas x e y de inicio y fin.

La heurística octil, a diferencia de las anteriores, dado que tiene en cuenta la diagonal, la columna y la fila, considera la heurística como suma de lo obtenido del producto entre la raíz de dos y el mínimo de las diferencias entre las coordenadas x e y de inicio y fin con la diferencia absoluta de las diferencias previamente mencionadas.

La implementación del algoritmo Theta* cuenta con la reutilización del código proveniente del algoritmo Theta*, pero modificando su función principal para que tenga en cuenta también en cuenta si existe un camino entre un nodo y su padre siendo posible que el padre no tenga que ser su nodo vecino obligatoriamente, y dos funciones más para tener en cuenta si existe algún obstáculo a su alrededor y el camino entre los nodos existe.

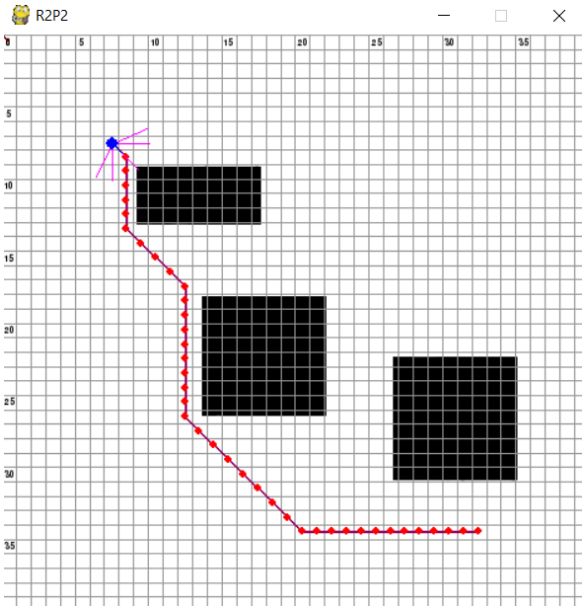
Heurística usada por defecto

Como se pudo ver la primera vez que se abrió el archivo "controller-pathplanning.json" en "*r2p2/conf*", la heurística usada por defecto es la ingenuo (naive) puesto que es la más fácil de implementar y la heurística asignada a cada par de posiciones es siempre uno.

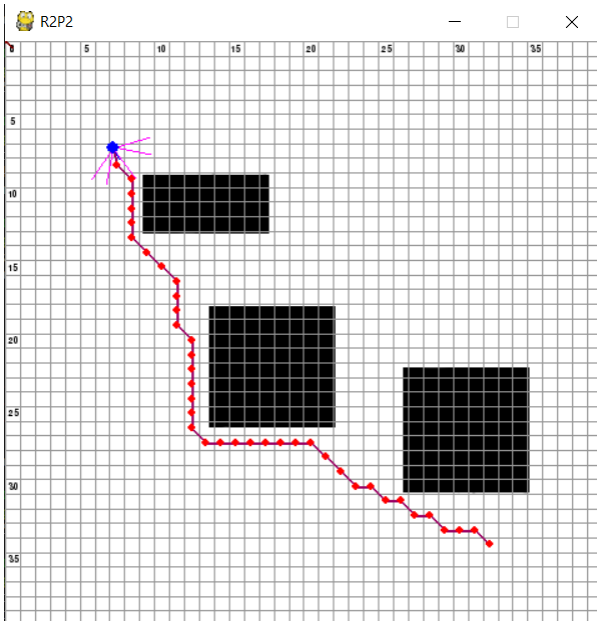
Comparación resultados de los distintos algoritmos con cada heurística

A continuación, se va a ver cómo se realiza el camino en relación con lo obtenido en cada una de las tres heurísticas con el uso del algoritmo A*.

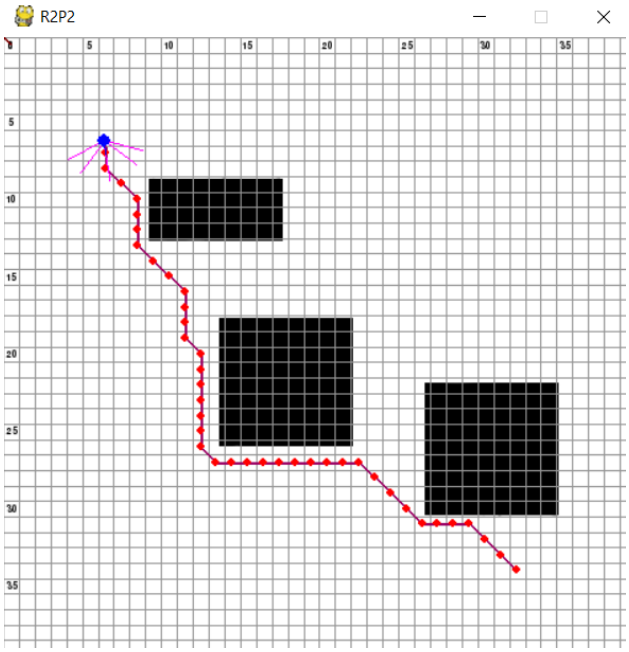
Heurística de Manhattan:



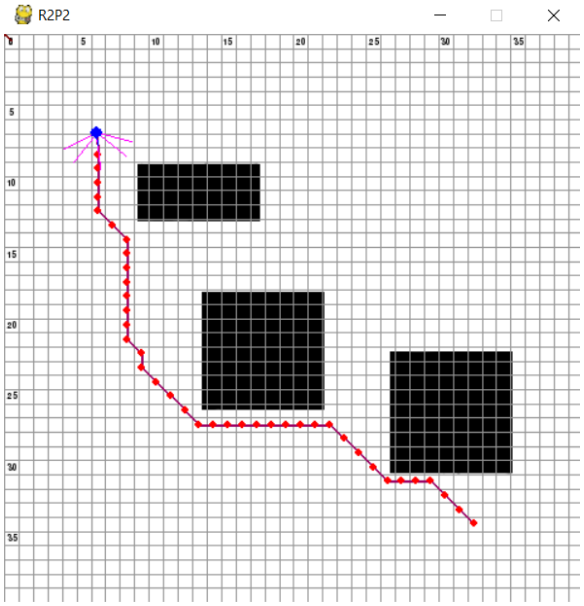
Heurística euclídea:



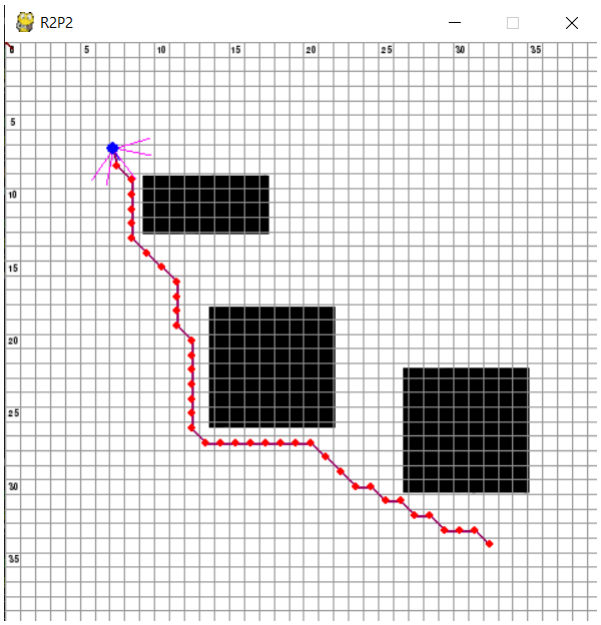
Heurística octil:



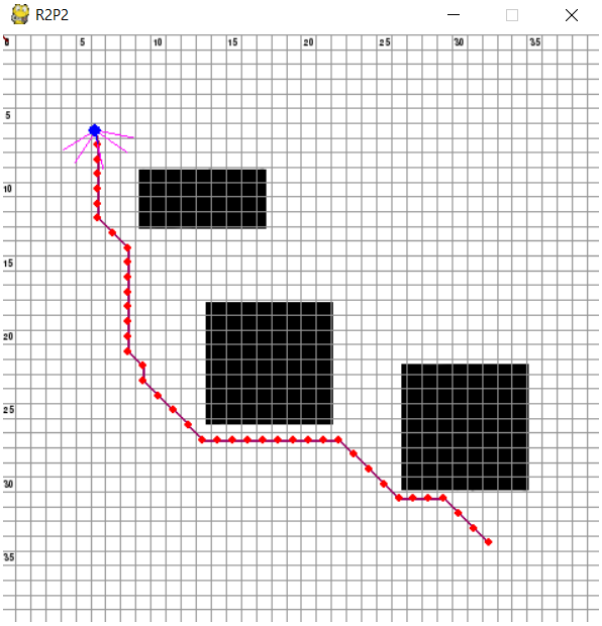
Como se puede observar, al aplicar cada una de las heurísticas a un algoritmo, se obtiene con cada uno un camino óptimo (más corto) distinto desde los puntos de inicio y destino en relación con el valor heurístico que se ha dado para cada par de puntos.
Esto mismo también ocurre si a cada una de las heurísticas antes mencionadas, se les aplica los algoritmos de Dijkstra y Theta*.
Estos serían los resultados obtenidos por parte del algoritmo Dijkstra:
Heurística de Manhattan:



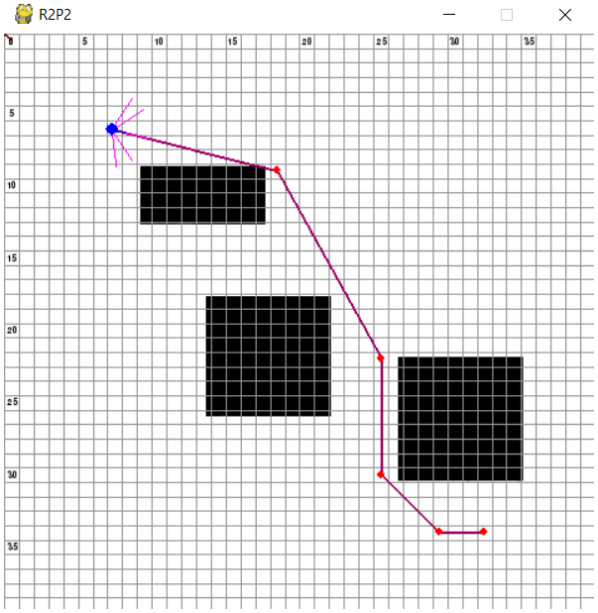
Heurística euclídea:



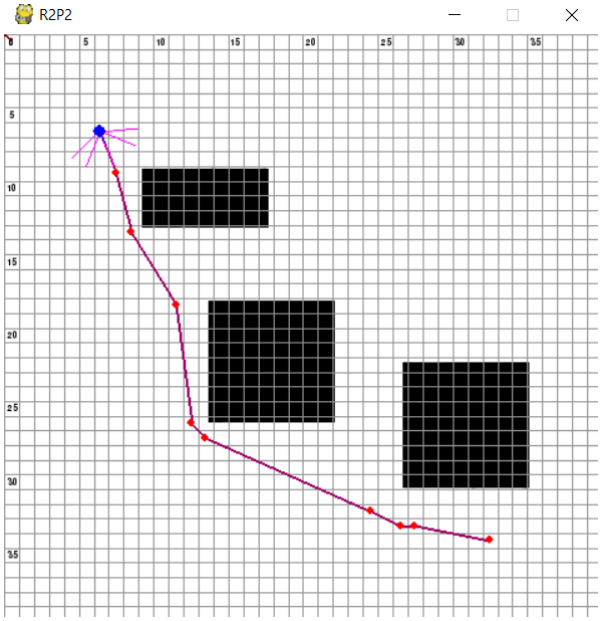
Heurística octil:



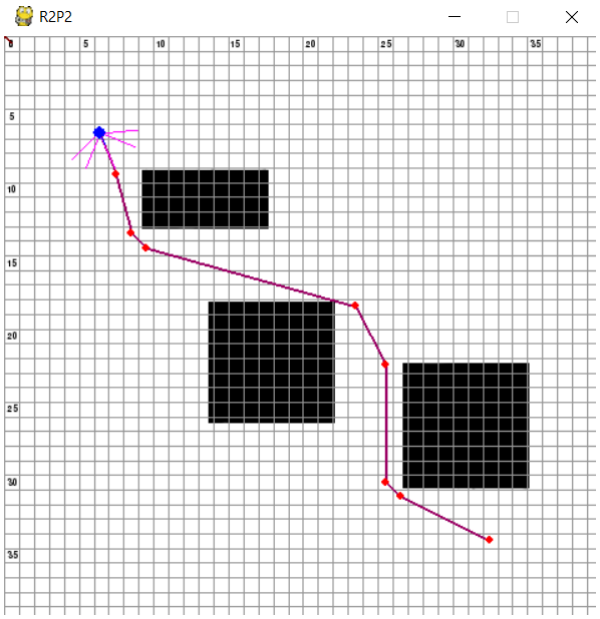
Estos serían los resultados obtenidos por parte del algoritmo Theta*:
Heurística de Manhattan:



Heurística euclídea:



Heurística octil:



Como se puede observar, los resultados obtenidos para cada algoritmo son diferentes. La mayor diferencia se encuentra entre el algoritmo Theta* con respecto a los otros dos. Esto se ve a simplemente por la reducción de puntos marcados en el camino seguido hasta la meta además de que no se desplaza en las distintas posiciones no solo a través de las columnas, filas y diagonales de 45º, sino que hace uso de diagonales con otros tipos de inclinación, los caminos seguidos por todas las heurísticas son completamente distintos entre sí cuando en A* y Dijkstra los caminos son muy parecidos no son entre sus heurísticas, sino entre ellos.

Integración

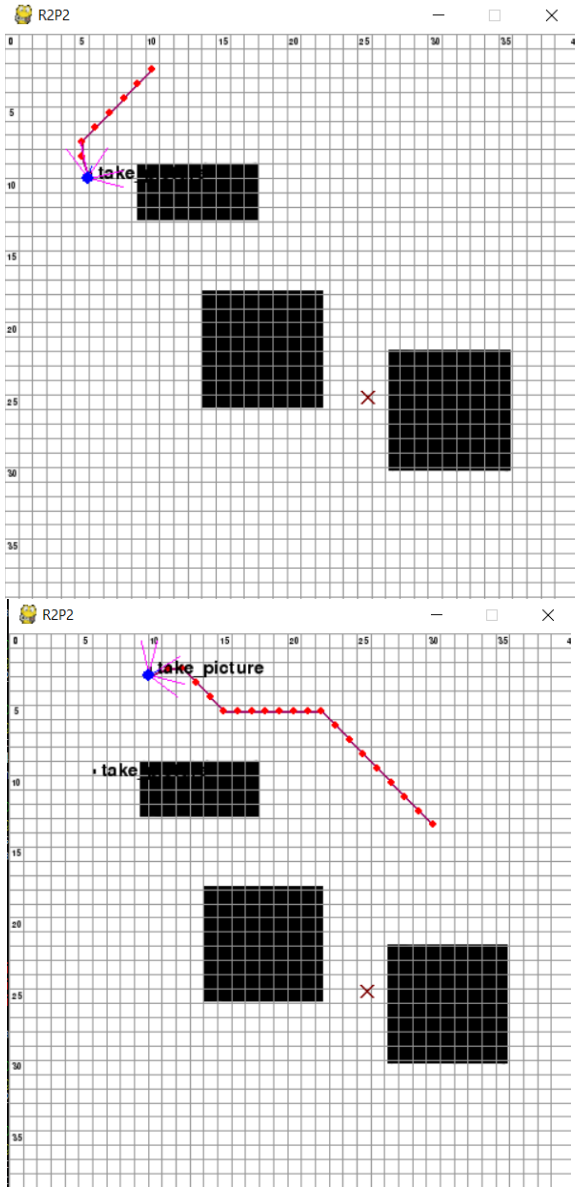
Para la elaboración de este apartado se hizo uso del código de pddl del ejercicio de cooperación. Por un lado, se unieron las coordenadas x e y por un objeto llamado “punto” que contuviera ambas y se modifico el problema de manera que se cumpliera este nuevo cambio además de que solo el robot representara a uno de los vehículos autónomos y que se añadieran más predicados en las metas para ampliar el plan obtenido y ver cómo actúa el robot en cada una de las distintas acciones. También se ha cambiado algunas de las posiciones para que no produjera ningún problema en relación con los obstáculos o dichos puntos se encuentren dentro de los mismos.

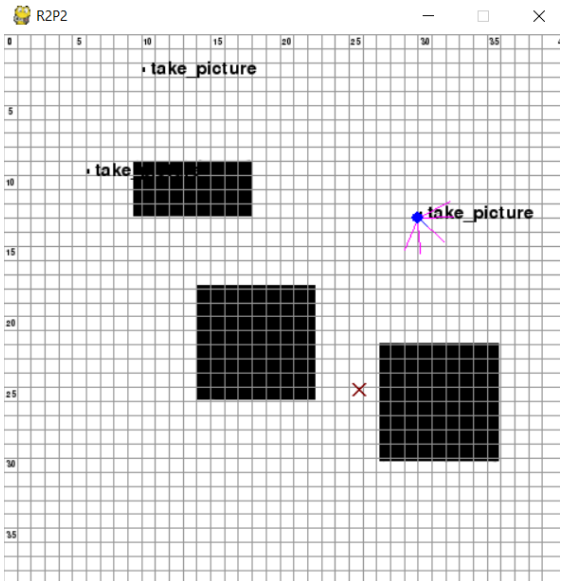
Como se ha podido observar, las últimas acciones que afectan en algo actualmente en el programa son de tomar una foto (“take_picture”) y la de moverse (“move”). Esto se debe a que el resto de las acciones que se plantean en el plan obtenido del dominio y el problema de pddl del ejercicio de cooperación no han sido implementadas o consideradas en el proyecto y se encuentran otras implementadas que no afectan al plan que se tiene.

La acción “move” se encarga de mover al robot entre dos posiciones del grid que se tiene para el escenario con relación al camino que se le está marcando entre las dos posiciones.

La acción “take_picture” simplemente muestra el nombre de esta acción justo al lado de posición en la que en principio se tomaría la fotografía.

Además, se puede observar que se hace una cierta distinción entre los momentos en los que se mueve y los que realiza cualquier otra acción. Esto se puede ver simplemente con la presencia o no del camino entre dos puntos.
Este ha sido el plan estudiado de acuerdo con el problema elaborado para el ejercicio de cooperation:
(move leader p0610 p0509 n0)
(take_picture leader p0509 p_0 t_0)
(rotation-tilt leader t_0 t_90)
(rotation-pan leader p_0 p_90)
(move leader p0509 p1002 n0)
(take_picture leader p1002 p_90 t_90)
(rotation-tilt leader t_90 t_315)
(rotation-pan leader p_90 p_45)
(move leader p1002 p3013 n0)
(take_picture leader p3013 p_45 t_315)
Esto ha sido el resultado obtenido y realizado por el robot aplicado con el algoritmo Theta* y la heurística octil:





Como se puede observar, se van alternando las acciones de movimiento con las de tomas de fotografía y el resto de las acciones, aunque estas últimas no se van reflejadas aquí.

¿Qué más?

Todo lo que se le ha implementado a este robot no es suficiente para controlarlo de manera autónoma.
Para que pudiera tener un control autónomo y ajustado al ejercicio de cooperación, debería poder:

- Conocer sus posiciones de las variables tilt y pan para saber que rotación tiene de manera que se ajuste si es necesario para ir en la dirección correspondiente en cada momento.
- Conocer qué tipo de modo de navegación tiene habilitado para poder ir a una velocidad distinta en relación con ello.
- Realizar el envío de las imágenes que ha realizado en una posición en un momento dado.
- Realizar el aterrizaje o salida a una base.

Como añadido a todas las características previamente mencionadas, pero fuera de lo que se indicaba en el dominio del ejercicio de cooperación, también podría ser necesario:

- Conocer la altitud a la que se encuentra de manera que pueda evitarse obstáculos también subiendo o bajando y no simplemente a partir de una vista desde arriba.
- Poder realizar una parada en una ubicación concreta si conoce que se está quedando sin batería antes de terminar la acción que trataba de realizar.
- Poder realizar distintos tipos de imagen dependiendo de las condiciones atmosféricas en las que se encuentra y de acuerdo con si es de día o de noche.
- Poder evitar obstáculos en movimiento.