Examen
Grado en Robótica. Curso 2022-2023.
Departamento de Electrónica y Computación

Nombre:.....

1. Problema 1 (4 puntos)

La Comunidad de Galicia está interesada en optimizar la gestión de sus embalses. Para ello, considérense todo el conjunto de embalses, para cada uno de los cuales se conoce el volumen de agua que contiene, y la capacidad máxima que puede acoger. Cada embalse da servicio a diferentes poblaciones y, si fuera preciso, es necesario mover agua de unos embalses a otros para garantizar la continuidad del servicio en todas las poblaciones de la Comunidad. Por lo tanto, un *requerimiento de servicio* consiste en una especificación de volumen mínimo de agua que debe llegar a un embalse específico. Con ello, el sistema debe determinar automáticamente los movimientos de agua necesarios desde cada embalse que satisfagan el requerimiento de servicio. Cada embalse sólo tiene compuertas para ceder una parte de su volumen a otros embalses, pero no a todos y, por supuesto, la capacidad de cada uno es finita.

Se pide responder razonadamente las siguientes preguntas:

1. (**1.5 puntos**) Representar el problema como un espacio de estados y conjunto de operadores especificando para cada uno de estos últimos sus precondiciones y efectos.

2. (**0.5 puntos**) Sabiendo que el coste del movimiento de aguas entre embalses es siempre el mismo, independientemente de la cantidad de agua, y los embalses origen y final, ¿qué algoritmo de búsqueda no informada sugerirías para minimizar el coste total para atender un determinado requerimiento de servicio?

3. (**1.5 puntos**) Sugiere una función heurística *h()* que sea admisible e informada para el problema de minimizar el coste total del movimiento de aguas.

4. (**0.5 puntos**) Suponiendo que se dispone de una función heurística *h()* admisible, sugiere un algoritmo de búsqueda heurística que sirva para resolver óptimamente el problema.

2. Problema 2 (2 puntos)

Recorrer el grafo de la Figura 1 utilizando el algoritmo IDA* teniendo en cuenta los costes indicados en cada arco. En todos los casos, indicar en qué orden se visitan los nodos, distinguiendo nodos generados de nodos expandidos. Para cada nodo indicar, además, su valor correspondiente a la función de evaluación, la función de coste y su valor heurístico. Tomar como estado inicial el nodo S y como único estado meta el nodo G. Cada nodo del grafo tiene el valor heurístico descrito en la Figura 2.

3. Problema 3 (4 puntos)

Una empresa de construcción ha decidido incorporar drones en su operativa. Estos drones servirán para monitorizar estructuras de difícil acceso en edificios en construcción. Para ello, al comienzo de un día de trabajo se indica al dron una serie de *landmarks* que debe visitar. Existen dos tipos de *landmarks*: las que requieren tomar imágenes en 2D, y las que requieren utilizar un sensor 3D que permita construir una nube de

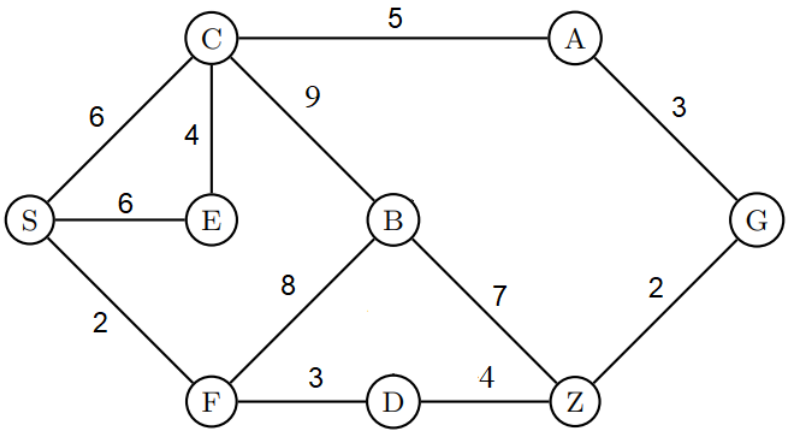

Figura 1: Representación del grafo de búsqueda.

n	S	A	B	C	D	E	F	G	Z
h(n)	7	2	5	6	3	7	7	0	1

Figura 2: Valores heurísticos.

puntos. La información 2D o 3D recogida por el dron en las *landmarks* indicadas será analizada posteriormente con más detalle por los operarios. Por lo tanto, a la hora de tomar una imagen, el dron utilizará un sensor u otro dependiendo del tipo de *landmark* donde se encuentre. El dron recorre el edificio visitando una serie de *waypoints* previamente definidos. Una *landmark* siempre se encuentra en un *waypoint*, pero no todos los *waypoints* están asociados a una *landmark*. De un *waypoint* el dron puede ir a otros, pero no a todos. Una representación gráfica de un problema para este dominio se encuentra en la Figura 3.

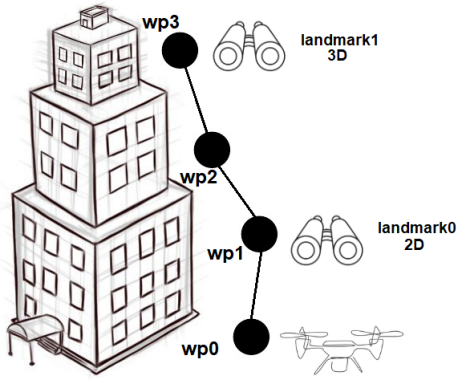

Figura 3: Representación del problema de inspección de edificios por drones.

Dado este dominio, se pide resolverlo utilizando *planificación automática*. Para ello:

1. (1.5 puntos) Empleando la **lógica de predicados**, indicar qué predicados sirven para describir cada estado.
2. (1 punto) Pon un ejemplo de **estado inicial** y **final**.
3. (1.5 puntos) Describir brevemente e **informalmente** los operadores que utilizarías para solucionar el problema y modelar uno de ellos **formalmente** en PDDL.

4. Solución Problema 1

1. El espacio de estados es una formalización que habilita la aplicación de algoritmos de búsqueda (informados o no) para la resolución de problemas. Consiste únicamente, en la definición de estados y operadores que sirven para transitar entre ellos. Relacionando, después, el conjunto de posibles estados con otro de vértices V y el de operadores (convenientemente instanciados) con otro de arcos E, resulta entonces de forma natural la definición de un grafo, el grafo de búsqueda que se recorrerá eficientemente con el uso de árboles de búsqueda. Este ejercicio propone ejercitar todos estos conceptos y, en el primer apartado, el del espacio de estados.

   - **Estados.** Un estado en este problema está constituido por un conjunto de *embalses*, para los cuales se conoce su capacidad, el volumen de agua que albergan y aquellos a los que puede trasvasar agua. Por lo tanto, cada estado vendrá determinado por la información de todos los embalses considerados en el problema, cada uno de los cuales se identifica como sigue:
     - **Embalse.** Cada *embalse* se distingue por un identificador, *id*, que podría ser su nombre, por ejemplo. Además, para cada uno se conoce su *capacidad* máxima, y el *volumen* de agua que alberga en cada estado. Por último, para poder calcular las operaciones legales de trasvasar agua entre estados, cada embalse debe tener un último atributo *adyacentes* con referencias a todos aquellos embalses a los que puede trasvasar, efectivamente, parte de su volumen.
       Además, el algoritmo que se diseñe para optimizar la gestión de todos los embalses, debe considerar el *requerimiento de servicio*:
     - **Servicio.** Para el que sólo es preciso almacenar el identificador, *id*, del embalse objetivo, y el *volumen* de agua que se requiere para él. De esta manera, un estado inicial consistirá en varias instancias de *embalse* y, una de *servicio* para la que el *volumen* que se requiere es estrictamente mayor que el que tiene actualmente. Análogamente, en un estado meta, el *volumen* que se requiere para el embalse destino es menor o igual que el que contiene.
   - **Operadores.** En la definición de operadores es importante describir sus precondiciones/postcondiciones y, además, el coste que tienen. En este problema hay un único operador: *trasvasar* cuyos argumentos son el embalse *origen*, el embalse *destino*, y el *volumen*.
     - **Precondiciones.** Que la cantidad que se desea trasvasar desde el embalse *origen* al *destino* no exceda el volumen actual del primero, ni la capacidad del segundo: $origen.volumen \leq volumen \ \wedge \ destino.volumen + volumen \leq destino.capacidad$
     - **Postcondiciones.** La ejecución del operador, efectivamente decrementa el volumen en el embalse origen, y lo aumenta en la misma cantidad, en el destino: $origen.volumen \ -= volumen$, y $destino.volumen \ += volumen$.
     - **Coste.** Si se considera, como se indica en el segundo apartado, que el coste es siempre el mismo (independientemente del embalse origen, destino, y la cantidad trasvasada) entonces el coste es sencillamente $k = 1$. Si, por el contrario, se considera el coste como el volumen trasvasado, entonces $k = volumen$.

2. Si el coste de trasvasar agua es siempre el mismo, entonces la solución que minimiza el coste de la operación total para trasvasar agua entre embalses hasta que por fin se consigue el volumen objetivo en el embalse de destino será aquella que ejecute el operador *trasvasar* el menor número de veces.

   En el caso de problemas de optimización con costes unitarios, se considerarán los algoritmos de el primero en amplitud y de el primero en profundización iterativa:

   - El algoritmo de amplitud es completo y, además, admisible cuando el coste de los operadores es siempre el mismo, como en este caso. Es decir, el coste será igual a la profundidad mínima a la que se genere un estado que contenga al embalse de destino con un volumen mayor o igual que el indicado en el requerimiento de servicio.

Este algoritmo, sin embargo, tienen un consumo de memoria exponencial, aunque es particularmente bueno en el caso de que haya transposiciones.

- El algoritmo de el primero en profundización iterativa realiza varias búsquedas de el primero en profundidad secuencialmente, incrementando la profundidad máxima en cada iteración. Por lo tanto, se trata de un algoritmo completo. Además, si el incremento de la profundidad máxima entre iteraciones es igual a la unidad, entonces la soluciones encontradas serán necesariamente óptimas en la profundidad de las soluciones —que, como ya se ha mencionado, resultan ser iguales al coste de las soluciones. Aunque este algoritmo tiene un consumo de memoria que es lineal en la profundidad a la que se encuentra la solución óptima, tiene dificultades en los espacios de estados con transposiciones, puesto que recorre todos los caminos que pasan por cada transposición.

De hecho, este problema tiene una cantidad alta de transposiciones (esto es, es muy fácil generar el mismo estado con la ejecución de diferentes operaciones de *trasvasar*). Por lo tanto, el algoritmo recomendado es el algoritmo de amplitud.

3. Para obtener una función heurística, se sugiere el uso de la técnica de *relajación de restricciones*. En su aplicación, se observan las restricciones del problema y se relajan todas o un subconjunto de ellas hasta que es posible resolver el problema resultante de forma óptima. Como quiera que las restricciones del problema se encuentran típicamente en las *precondiciones* de los operadores del problema (estudiadas en el primer apartado), una relajación factible consiste en hacer que cada embalse pueda estar conectado a todos los demás, en vez de sólo aquellos que aparecen en su atributo *adyacentes*. Considerando únicamente que es posible pasar agua desde cualquier embalse a cualquier otro, entonces para conseguir que llegue un determinado volumen a un embalse de destino, basta con seguir el siguiente procedimiento:

   - Ordenar todos los embalses de mayor a menor volumen de agua contenida.
   - Trasvasar todo el agua desde el primer embalse (el que mayor capacidad tenga) al de destino.
   - Si el embalse de destino consigue así el volumen requerido, parar. En otro caso, volver al paso anterior considerando el siguiente embalse en la ordenación calculada en el primer paso.

   Este procedimiento podría emplearse para calcular el valor heurístico de un determinado estado en el problema original, estando seguros de que nunca se sobreestimará el coste real de llegar a la meta.

4. La selección de un algoritmo de búsqueda informada para este caso depende, como en el caso de los algoritmos de búsqueda no informada, del número de transposiciones y también, naturalmente, de la dificultad de los problemas:

   - El algoritmo A* es admisible y garantiza, por lo tanto, que encontrará soluciones óptimas si la función heurística que lo guía también es admisible. Además, es un algoritmo rápido puesto que no reexpande nodos (y, con frecuencia, las ordenaciones de la lista abierta se pueden hacer en $O(1)$ con las estructuras de datos adecuadas si la función objetivo sólo toma valores enteros). Sin embargo, tiene un consumo de memoria exponencial.
   - El algoritmo IDA* reexpande nodos en caso de que haya transpociones pero no ordena nodos y, mucho más importante aún, tiene un consumo de memoria lineal en la profundidad de la solución (que en nuestro caso es siempre igual a N). Además, también es un algoritmo de búsqueda admisible.

   Como ya se indicó anteriormente, este problema tiene un gran número de transposiciones y, por ello, se sugiere el algoritmo A*.

5. Solución problema 2

IDA*, a diferencia de A*, consiste en una serie de recorridos en profundidad hasta que $f(n) > \eta$ o se ha encontrado la solución, incrementando η en cada iteración al menor exceso cometido. Por lo tanto, además, al

contrario que en profundidad iterativa, el valor de η no se incrementa en un valor k en cada iteración, sino al menor exceso cometido como veremos a continuación. Inicialmente, y para garantizar la admisibilidad, η se inicializa con $\eta = h(S) = 7$. Teniendo en cuenta todo esto, procedemos a aplicar el algoritmo al grafo propuesto.

Iteración 1: $\eta = h(S) = 7$

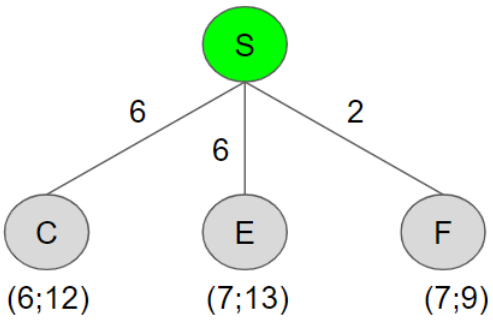

Figura 4: Primera iteración de IDA*.

En la Figura 4 se muestra la primera iteración del algoritmo. Se muestran en verde los nodos expandidos, y en gris los nodos generados. En los arcos del árbol se muestra el coste de transitar de un nodo a otro, y al lado de los nodos se muestra entre paréntesis en primer lugar el valor de $h(n)$ seguido por el valor de $f(n)$ de ese nodo en particular. Teniendo todo esto en cuenta, la expansión de S genera tres nodos: C, E y F. Siguiendo nuestro recorrido en profundidad, deberíamos tratar de expandir el nodo C. Sin embargo, $f(C) = 12 > \eta = 7$, luego hacemos *backtracking* y continuamos nuestro recorrido en profundidad por el nodo E. En E nos encontramos con una situación similar, $f(E) = 13 > \eta = 7$, así que hacemos nuevamente *backtracking* y continuamos nuestro recorrido en profundidad para llegar a F. $f(F) = 9 > \eta = 7$, luego en esta primera iteración únicamente se expande S. En la siguiente iteración, se elige el valor de η entre el menor de $f(C) = 12$, $f(E) = 13$, $f(F) = 9$, es decir, en la siguiente iteración $\eta = 9$.

Iteración 2: $\eta = 9$

En la segunda iteración de IDA* con $\eta = 9$, se generan en primer lugar los mismos nodos generados en la iteración anterior, solo que esta vez sí que podremos expandir el nodo F puesto que $f(F) = 9 = \eta$[1]. La expansión del nodo F genera los nodos B y D. El primero con una función de evaluación $f(B) = 15$ y el segundo con una $f(D) = 8$[2]. Procederíamos en un recorrido en profundidad: tratamos de expandir B, pero no es posible puesto que $f(B) = 15 \geq \eta = 9$. El siguiente nodo que expandimos es D puesto que $f(D) = 8 \leq \eta = 9$. La expansión de D genera el nodo Z, que trataríamos de expandir comprobando que no es posible puesto que $f(Z) = 10 \geq \eta = 9$, cerrando así la segunda iteración de IDA*. Incrementamos η al menor de los excesos cometidos entre las funciones de evaluación de los nodos que aún no han sido expandidos, es decir, $\eta = \text{mín}\{f(C) = 12, f(E) = 13, f(B) = 15, f(Z) = 10\}$. Por lo tanto, en la nueva iteración $\eta = 10$.

Iteración 3: $\eta = 10$

La Figura 6 muestra como quedaría la tercera iteración de IDA*. En esta ocasión, cuando lleguemos al nodo al nodo Z sí que podremos expandirlo, puesto que $f(Z) = 10 \leq \eta = 10$. La expansión de este nuevo nodo

[1]Es importante recordar que en cada nueva iteración de IDA*, se comienza con un nuevo proceso de búsqueda en profundidad desde el nodo de inicio.

[2]En el cálculo de las funciones de evaluación de este par de nodos, hay que tener en cuenta el coste de llegar desde el nodo de inicio al nodo en el que nos encontramos. Por lo tanto, $f(B)$ es igual a la suma de llegar del nodo S al nodo F (2), más el coste de llegar del nodo F al nodo B (8), más el valor heurístico del nodo B (5), obteniendo así $f(B) = 15$. De igual forma se procedería con el cálculo de la $f(D)$.

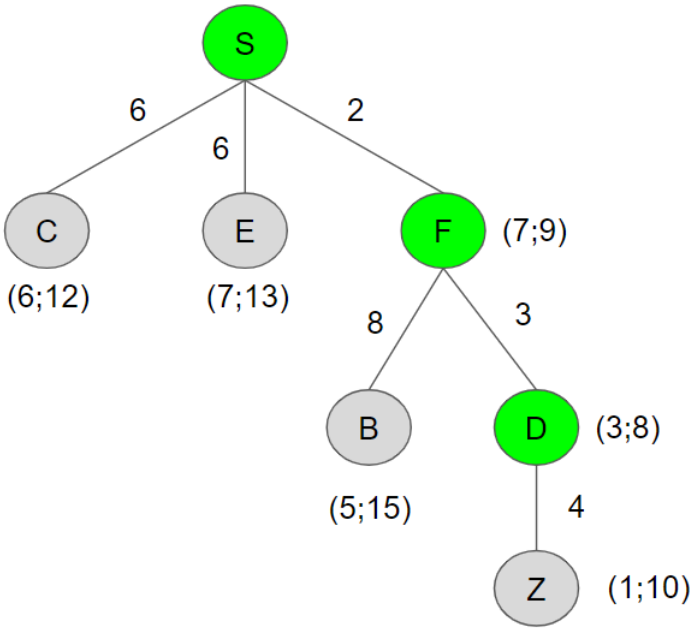

Figura 5: Segunda iteración de IDA*.

genera los nodos B con una $f(B) = 21$ y el nodo meta G con una $f(G) = 11$. Hemos encontrado un camino hasta el nodo G de coste 11, pero como $f(G) = 11 \geq \eta = 10$, no podemos garantizar que sea la ruta óptima. Debemos ejecutar necesariamente una nueva iteración incrementando η al menor de los excesos cometidos, es decir, $\eta = 11$.

Iteración 4: $\eta = 11$

En el nuevo árbol generado mediante un recorrido en profundidad del grafo con un valor $\eta = 11$, volvería a generar el mismo árbol que se muestra en la Figura 6, solo que en este caso generando el nodo G con una función de evaluación $f(G) = 11 \leq \eta = 11$, luego podemos garantizar que el camino óptimo para llegar del nodo S al nodo G es $S \rightarrow F \rightarrow D \rightarrow Z \rightarrow G$ con coste 11.

6. Solución problema 3

1. Como ocurre en todos los problemas que nos piden representar los conceptos que aparecen en el enunciado, no tiene por qué existir una única representación válida. En este caso se pide representar el problema utilizando lógica de predicados, y un conjunto válido de predicados podría ser:

   - `(at-dron ?dron ?wp)`: Un determinado dron `?dron` está en un determinado *waypoint* `?wp`. Este predicado es necesario puesto que dependiendo de donde se encuentre el dron podrá hacer unas cosas u otras. Para este predicado se ha decidido utilizar un parámetro `?dron` que realmente no sería necesario puesto que el enunciado nos habla de un único dron. Este predicado, sin embargo, nos permite generalizar el enunciado para tener en cuenta más de un robot.
   - `(at-lm ?lm ?wp)`: Una determinada *landmark* `?lm` está asociada a un determinado *waypoint* `?wp`.
   - `(inspected ?dron ?lm)`: Este predicado sirve para indicar que una determinada *landmark* `?lm` ha sido inspeccionada ya por el dron `?dron`.
   - `(lm-2d ?lm)`: Sirve para indicar que la *landmark* `?lm` es de un tipo 2d.
   - `(lm-3d ?lm)`: Sirve para indicar que la *landmark* `?lm` es de un tipo 3d.
   - `(connected ?wp1 ?wp2)`: El *waypoint* `?wp1` está conectado al *waypoint* `?wp2`.

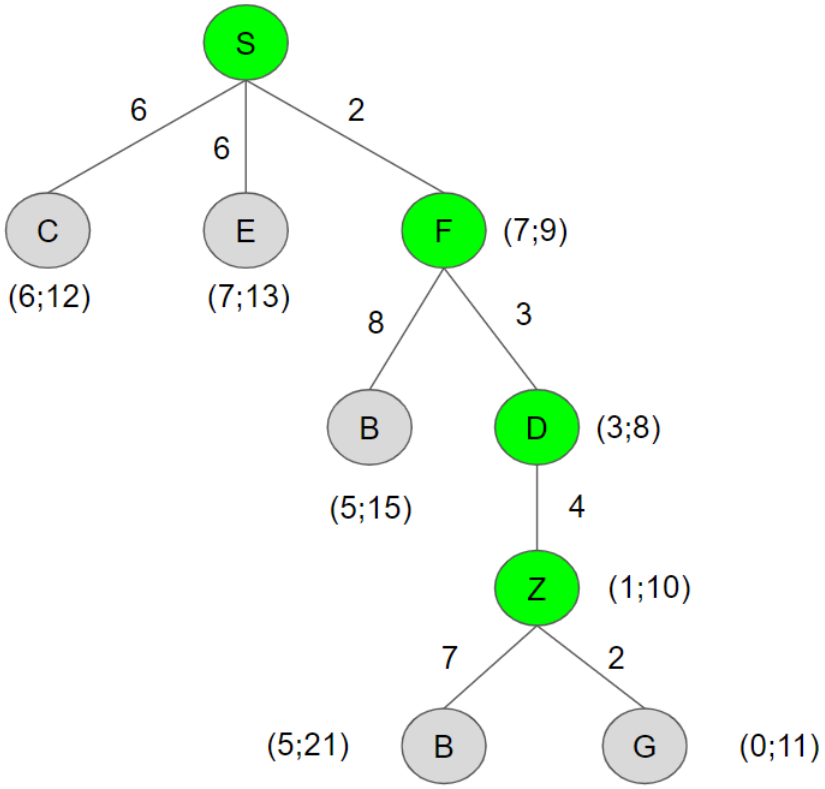

Figura 6: Tercera iteración de IDA*.

Hasta aquí tenemos todos los predicados necesarios para modelar el problema.

2. En este caso nos piden un estado inicial y uno final. En cuando al estado inicial, podríamos modelarlo de la siguiente manera:

```
(at-dron drone wp0)
(at-lm lm0 wp1)
(at-lm lm1 wp3)
(connected wp0 wp1)
(connected wp1 wp2)
(connected wp2 wp3)
(lm-2d lm0)
(lm-3d lm1)
```

Con este estado inicial, estamos indicando que tenemos a nuestro dron en el *waypoint wp0*, y que debe revisar dos *landmarks*: *lm0* y *lm1* que se encuentran en los *waypoints wp1* y *wp3*, respectivamente. La primera *landmark* es de tipo *2d*, y la segunda de tipo *3d*. Además, se establecen conexiones entre el *waypoint wp0* y *wp1*, el *wp1* y *wp2* y el *wp2* y *wp3*.

En cuanto al estado final, sería igual al anterior, pero indicando explícitamente que el dron ha inspeccionado, es decir, ha tomado las imágenes, de los *landmarks* requeridos:

`(inspected drone lm0)` `(inspected drone lm1)`

3. En cuanto a los operadores, con la representación utilizada, tendríamos 3 operadores diferentes:

   - `(move ?dron ?from ?to)`: Para indicar que el dron se mueve de una determinada localización de partida `?from` a una localización de destino `?to`. Para ello, el dron se debe encontrar en la

localización de partida y, una vez acabada la acción, el dron dejará de estar en la localización de partida y pasará a la de destino.

- `(take-2d ?dron ?wp ?lm)`: El dron `?dron` toma la imagen 2d de una de las *landmarks*. Para ello, el `?dron` y la *landmark* `?lm` se deben encontrar en el *waypoint* `?wp`, y además la *landmark* debe ser de tipo 2d. En los efectos indicaríamos que la *landmark* `?lm` ha sido inspeccionada.
- `(take-3d ?dron ?wp ?lm)`: El dron `?dron` toma la imagen 3d de una de las *landmarks*. Para ello, el `?dron` y la *landmark* `?lm` se deben encontrar en el *waypoint* `?wp`, y además la *landmark* debe ser de tipo 3d. En los efectos indicaríamos que la *landmark* `?lm` ha sido inspeccionada.

Por último, como el enunciado nos pide además modelizar uno de ellos de manera formal mediante sintaxis PDDL:

```
(:action take-2d
:parameters (?d - dron ?wp - waypoint ?lm - landmark)
:precondition (and (at-dron ?d ?wp) (at-lm ?lm ?wp) (lm-2d ?lm))
:effect
(and (inspected ?dron ?lm)))
```