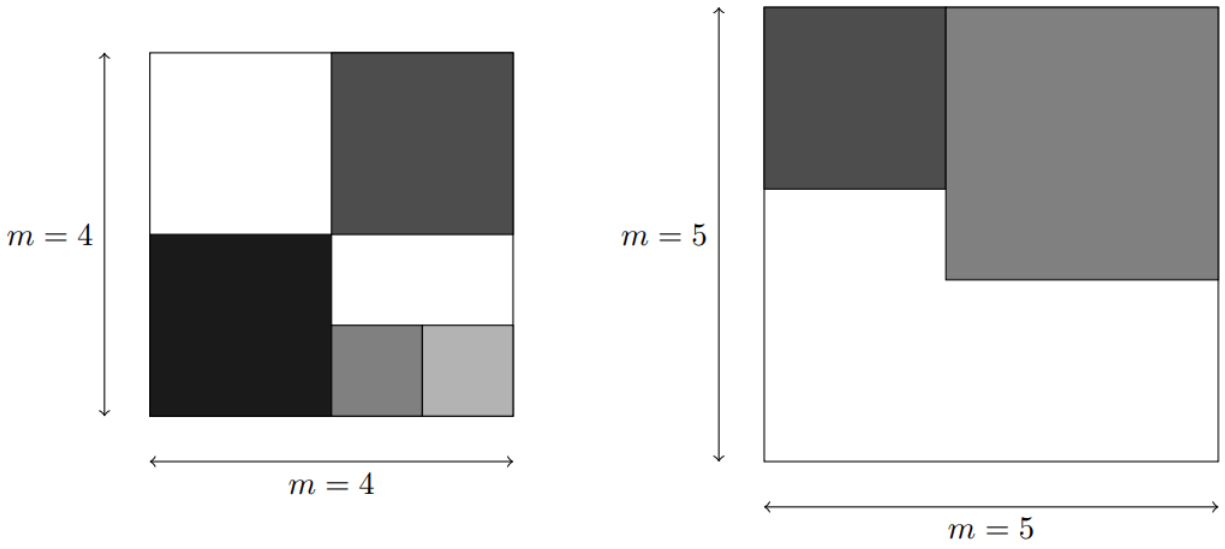
Examen
Grado en Robótica. Curso 2021-2022.
Departamento de Electrónica y Computación

Nombre:.....

1. Problema 1 (5 puntos)

Las zonas de carga de un buque carguero se dividen en regiones estrictamente cuadradas de dimensión $m \times m$ para satisfacer varias restricciones en relación con la distribución de pesos que debe soportar. Dada una cantidad arbitraria N de contenedores siempre cuadrados de dimensiones diferentes $l_i \times l_i$, $0 \leq i < N$, ¿cuál es la dimensión m del cuadrado más pequeño que inscribe los N contenedores?

A continuación se muestran dos ejemplos. En el caso de la izquierda, el cuadrado de menor área que inscribe dos cuadrados de 1×1 y otros dos de 2×2 es de 4×4. El segundo caso muestra que no es posible inscribir un cuadrado de 2×2 y otro de 3×3 en un cuadrado de menos de 5×5.



Se pide responder razonadamente las siguientes preguntas:

1. (**3/2 puntos**) Representar el problema definiendo los estados y las acciones con sus precondiciones, efectos y coste.
2. (**1/2 puntos**) ¿Qué profundidad tendría un árbol de búsqueda desarrollado por un algoritmo de búsqueda no informada para resolver óptimamente este problema?
3. (**1 puntos**) ¿Es posible solucionar el problema con un algoritmo de búsqueda no informada con costes unitarios?, ¿por qué?
4. (**1 punto**) Diseñar una función heurística h(n) que sea admisible y que esté bien informada.
5. (**1 punto**) Habida cuenta que la función h(n) sea admisible, ¿qué algoritmo de búsqueda heurística es el más indicado para resolver este problema óptimamente?.

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.

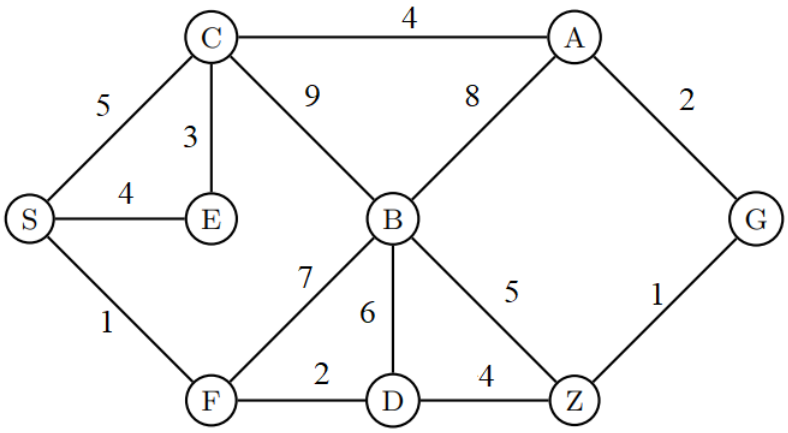


Figura 1: Representación del grafo de búsqueda.

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.

3. Problema 3 (3 puntos)

Una agencia espacial dispone de un satélite capaz de tomar imágenes de la Tierra, y transmitirlas a una determinada estación base. Para ello, al inicio del día, cada estación base solicita al satélite las imágenes que necesita que pueden ser de tres tipos: espectografía, termografía y normal. Además, cada imagen está asociada a una determinada posición, y el satélite debe estar apuntando a esa posición antes de tomar la imagen. Para la toma de imágenes, el satélite dispone de una cámara, que debe estar calibrada en el modo adecuado (espectografía, termografía, normal) antes de tomar la imagen requerida.

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

1. (1 punto) Empleando la **lógica de predicados**, indicar qué predicados sirven para describir cada estado.
2. (1 punto) Pon un ejemplo de **estado inicial** y **final** con dos estaciones base que solicitan tres imágenes cada una.
3. (1 punto) 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 caracterizado por una disposición parcial de los contenedores en un cuadrado de dimension $m \times m$:
     - **Contenedor:** El contenedor i-ésimo debe estar caracterizado por su tamaño l_i y su posición en el cuadrado como un par (x_i, y_i). Haciendo que la posición por defecto tome valores imposibles (por ejemplo, $x_i = y_i = -1$) es posible distinguir los contenedores que ya han sido dispuestos en un estado de los que están pendientes por colocarse. Se asume, sin pérdida de generalidad, que el tamaño de los contenedores es siempre un número entero y que, por lo tanto, su disposición se hace también en coordenadas enteras.
     - **Cuadrado:** La única información que es preciso almacenar del cuadrado mínimo que inscribe todos los contenedores es su dimensión, m que representaremos con un campo específico m.
   - **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:
     - **Disponer:** Este operador recibe un contenedor que aún no ha sido colocado (esto es, con $x_i = y_i = -1$) y una posición (x, y) y lo dispone en el cuadrado en el lugar indicado:
       - **Precondiciones:** El área del cuadrado inscrita por los puntos (x, y) y $(x + l_i, y + l_i)$ está completamente libre.
       - **Postcondiciones:** Debe actualizarse la posición del contenedor i-ésimo haciendo $x_i = x$ y $y_i = y$. Además, en caso de que colocando este contenedor se exceda el tamaño del cuadrado actual, es preciso registrar el nuevo tamaño del cuadrado: $m = min\{m, x+l_i, y+l_i\}$.
       - **Coste:** El coste de disponer un nuevo contenedor será igual al incremento en el tamaño del cuadrado y que se calcula fácilmente como $m_{new} - m_{prev}$, donde m_{new} es el tamaño del cuadrado al incluir el nuevo contenedor, y m_{prev} es el tamaño del cuadrado antes de incluir el contenedor. Por lo tanto, se trata de un problema de optimización con costes diferentes que, de hecho, podrían tomar también el valor cero, siempre que el tamaño del nuevo cuadrado sea igual que el anterior. Por ejemplo, en la disposición que se muestra en la figura de la izquierda, si asumimos que inicialmente se añadieron los contenedores de 2×2, cuando se añadieron los de 1×1 se hizo con coste 0, puesto que añadirlos no implicó aumentar el tamaño del cuadrado.

   En otra posible representación, podemos suponer inicialmente que todos los contenedores ya están inscritos en un cuadrado que obviamente no puede ser el óptimo (si lo fuera, ¡ya tendríamos la solución!), y lo que tenemos que hacer es moverlos de una posición a otra hasta conseguir el cuadrado de menor m. Sin embargo, el estudio de los siguientes apartados se va a realizar teniendo en cuenta la representación anterior.

2. La profundidad d de un árbol de búsqueda se define como la profundidad mínima a la que se encuentra la solución. A partir de la definición del espacio de estados del apartado anterior, un algoritmo de búsqueda no informada dispondría un nuevo contenedor en cada nivel y, por lo tanto, todos los nodos a profundidad N (con N el número inicial de contenedores a disponer) son nodos solución. Por lo tanto, $d = N$.

3. Obviamente no, precisamente porque este es un problema donde hemos definido costes arbitrarios. Si la adición de un contenedor tuviera siempre un coste de 1, optimizaríamos el número de contenedores que

añadimos, pero no el área del cuadrado que los inscribe. Para hacer ésto, necesariamente necesitamos tener operadores con diferentes costes que dependen del incremento del cuadrado al inscribir un nuevo contenedor tal y como se ha definido en el apartado 1. En definitiva, en cualquier problema de búsqueda el coste de los operadores debe estar relacionado con aquello que queremos minimizar/maximizar.

4. La generación de heurísticas admisibles se sigue 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) son relajaciones factibles las siguientes:

   - Considerar que los contenedores no tienen área. Si los contenedores no tienen área, da igual donde los coloquemos. Por tanto, el problema se reduciría a colocar todos los contenedores, y la heurística sería $h_1(n) = c$, donde c es el número de contenedores restantes que nos quedan por colocar en el nodo n. Es fácil deducir que esta heurísta sería una heurística admisible pero nada informada.
   - Considerar que todos los contenedores pueden superponerse unos sobre otros. De esta forma, el cuadrado mínimo que inscribe a todos sería:

     $$h_2(n) = min(0, \max_{i=1,N}\{l_i\} - m) \tag{1}$$

     Nótese que es preciso restar m (la dimensión del cuadrado exterior en el estado n) para asegurarse que se está estimando el incremento en el tamaño del cuadrado sin sobreestimar el tamaño final.

     Desgraciadamente, esta heurística no está informada en absoluto puesto que estimará que, para cualquier estado, las dimensiones finales del cuadrado serán exactamente $\max_{i=1,N}\{l_i\}\times\max_{i=1,N}\{l_i\}$.
   - También podemos estimar el incremento del cuadrado en cada estado n como:

     $$h_3(n) = min(0, \max_{i=1,n}\{l_i\} - l_{vacio}) \tag{2}$$

     donde $\max_{i=1,n}\{l_i\}$ es la longitud del contenedor de mayor tamaño de entre los que quedan por poner, y l_{vacio} es la longitud del lado del hueco vacio de dimensiones $l_{vacio} \times l_{vacio}$ de mayor tamaño dentro del cuadrado. Lógicamente, si el contenedor más grande de entre los que aún no hemos colocado no coge en ningún hueco disponible, el cuadrado m se incrementará al menos en la diferencia entre $\max_{i=1,n}\{l_i\} - l_{vacio}$. La heurística no sobreestima en ningún caso el coste real, pero sin duda está poco informada.

5. La selección de un algoritmo de búsqueda informada para este caso depende, naturalmente, de la dificultad de los problemas:

   - **A*** 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 como es nuestro caso —véase el primer apartado). Sin embargo, tiene un consumo de memoria exponencial.
   - **IDA*** El algoritmo IDA* reexpande nodos 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.

   Por lo tanto, para la resolución de instancias sencillas se podría sugerir el primer algoritmo pero, para las instancias más complicadas se recomienda el segundo.

5. Solución problema 2

IDA*, a diferencia de A*, consiste en una serie de recorridos en profundidad hasta que $f(n) > \eta$ o hemos 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: $\eta = h(S) = 7$

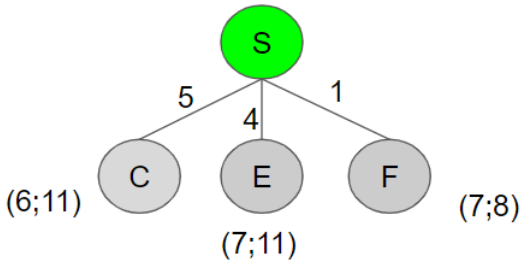


Figura 3: Primera iteración IDA*

En la Figura 3 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, $f(C) = 11 > \eta = 7$, luego hacemos *backtracking* y continuamos nuestro recorrido en profundidad por el nodo E. En E nos encontramos con una situación similar, $f(E) = 11 > \eta = 7$, así que hacemos nuevamente *backtracking* y continuamos nuestro recorrido en profundidad para llegar a F. $f(F) = 8 > \eta = 7$, luego en esta primera iteración únicamente se expande S. En la siguiente iteración, se elige el valor de η entre el menor de $f(C) = 11$, $f(E) = 11$, $f(F) = 8$, es decir, en la siguiente iteración $\eta = 8$.

Iteración 2: $\eta = h(S) = 8$

De la misma forma que en profundidad iterativa, en esta nueva iteración se vuelve a regenerar el árbol por completo utilizando búsqueda en profundidad y comenzando con el nodo inicial S (Figura 4). Teniendo en cuenta que ahora $\eta = 8$, expandimos el nodo S puesto que $f(S) = 7 < \eta = 8$. La expansión genera los nodos C, E y F. Descartaríamos expandir C y E puesto que en ambos casos su $f(n) > \eta$, no siendo así con el nodo F. Ahora sí podemos expandir el nodo F, puesto que $f(F)$ coincide con el valor de $\eta = 8$, La expansión de F genera los nodos B y D. Siguiendo el recorrido en profundidad, trataríamos de expandir B, pero es posible puesto que $f(B) = 13 > \eta = 8$, por lo tanto, hacemos *backtracking* y tratamos de expandir el nodo D. En este caso sí es posible puesto que $f(D) = 6 < \eta = 8$. La expansión de D produce los nodos B y Z. Siguiendo nuestro recorrido en profundidad, y teniendo en cuenta que $\eta = 8$, procedemos a expandir Z que genera el nodo G. Hemos llegado al nodo G, con un coste igual menor o igual al umbral η y por lo tanto podemos asegurar que el camino encontrado de coste 8, que es el mismo que se obtendría con el algoritmo A*, es óptimo.

6. Solución problema 3

1. Nos piden representar el problema utilizando lógica de predicados, y, como siempre en estos casos, no tiene por qué existir una única forma válida de representación. Por lo tanto, lo que se plantea en esta solución es una posible representación que modela todos los conceptos y relaciones que se describen en el enunciado. Un posible conjunto de predicados que permite modelar el problema serían:

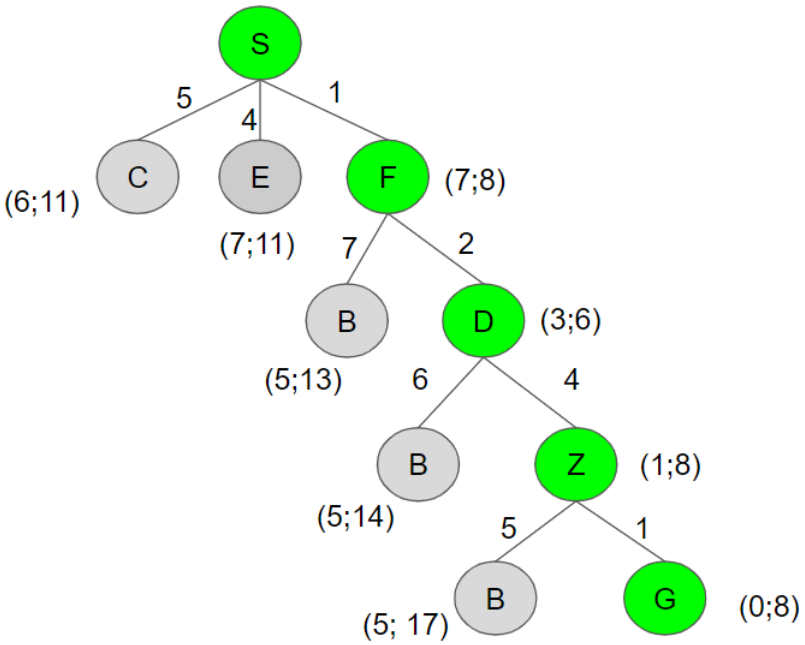


Figura 4: Segunda iteración IDA*

- `(peticion ?base ?imagen)`: Sirve para modelar el concepto petición que relaciona a su vez dos conceptos diferentes: la base que solicita dicha petición, y la imagen en particular que se solicita. Realmente faltarían indicar otros dos conceptos más en la relación: los conceptos *posición* y *modo*, puesto que el enunciado nos dice que cada imagen tiene asociada una determinada posición y puede ser tomada en tres *modos* diferentes. Sin embargo, para no complicar en exceso el predicado, modelaremos esta información en un predicado aparte.
- `(descripcion ?imagen ?pos ?tipo)`: Permite relacionar la imagen `?imagen` con una determinada posición `?pos` y el tipo `?tipo`.
- `(posicion ?sat ?pos)`: Sirve para indicar que el satélite `?sat` está apuntando a una determinada posición `?pos`.
- `(camara ?sat ?cam)`: Permite indicar que el satélite `?sat` está equipado con la cámara `?cam`. Realmente este predicado se podría omitir, puesto que el enunciado habla de un único satélite que dispone de una única cámara. No obstante, para obtener una representación más general que permita que existan varios satélites con varias cámaras, se opta por esta representación.
- `(calibrada ?sat ?cam ?modo)`: Para indicar que la cámara `?cam` del satélite `?sat` está calibrada en el modo `?modo`.
- `(img-tomada ?base ?imagen)`: Permite representar que la imagen `?imagen` solicitada por la base `?base` ha sido tomada.
- `(peticion-servida ?base ?imagen)`: Para indicar que la imagen `?imagen` se ha transmitido correctamente la base `?base`.

De esta forma tenemos representados todos los predicados necesarios para representar el problema que describe el enunciado.

2. Utilizando los predicados definidos anteriormente, un estado inicial con dos estaciones base que solicitan tres imágenes cada una sería:

```
(peticion base1 img1)    (camara sat1 cam1)
(peticion base1 img2)    (camara sat1 cam2)
```

```
(peticion base1 img3)    (posicion sat1 pos1)
(peticion base2 img4)
(peticion base2 img5)
(peticion base2 img6)
(descripcion img1 pos1 esp)
(descripcion img2 pos3 term)
(descripcion img3 pos6 norm)
(descripcion img4 pos6 norm)
(descripcion img5 pos2 norm)
(descripcion img6 pos4 term)
```

En este estado inicial, tenemos dos estaciones base diferentes, `base1` y `base2`, y cada una solicita tres imágenes cada una asociadas a una determinada posición y modo. Además, la agencia espacial dispone del satélite `sat1` con dos cámaras `cam1` y `cam2` que no han sido calibradas, y que está apuntando a la posición `pos1`. Se puede ver como, aprovechando la representación propuesta, hemos hecho que el satélite disponga de dos cámaras. Para ajustarse al problema del enunciado, simplemente habría que eliminar una de ellas. En cuanto al estado final, es necesario que aparezcan los predicados:

```
(peticion-servida base1 img1)
(peticion-servida base1 img2)
(peticion-servida base1 img3)
(peticion-servida base2 img4)
(peticion-servida base2 img5)
(peticion-servida base2 img6)
```

que van a sustituir a los predicados `peticion` anteriores una vez las imágenes hayan sido transmitidas a las estaciones base.

3. En cuanto a los operadores tendríamos:

   - `(mover ?sat ?from ?to)`: Permite mover el satélite `?sat` de la posición `?from` a la posición `?to`. Las precondiciones serían que el satélite se encuentra apuntando efectivamente a la posición `?from`, y los efectos serían que pasaría a estar apuntando a la posición `?to`.
   - `(calibrar ?sat ?cam ?modo)`: Permite calibrar la cámara `?cam` del satélite `?sat` al modo `?modo`. Para ello simplemente debemos comprobar que la cámera `?cam` pertenece al satélite `?sat`, antes de poner su modo a `?modo`. En el caso de que hubiesemos optado por no modelar de forma explícita que puede haber varios satélites, entonces esta acción no tendría precondiciones puesto que el enunciado no nos impone ninguna restricción antes de calibrar una cámara a un modo en concreto.
   - `(tomar ?bas ?sat ?cam ?img ?pos ?modo)`: El operador más complicado, puesto que relaciona todos los conceptos que aparecen en el enunciado, y que sirve para tomar con la cámara `?cam` del satélite `?sat` una determinada imagen `?img` en la posición `?pos` y el modo `?modo`, que ha sido solicitada por la estación base `?base`. Las precondiciones serían la imagen `?img` se encuentra en `?pos` y que el satélite está apuntando a esa determinada posición, y que la base `?bas` ha solicitado la imagen `?img` con el modo `?modo`. El efecto sería que la imagen ha sido tomada, y lo indicaríamos con el predicado `(img-tomada ?bas ?img)`.
   - `(transmitir ?bas ?img)`: Permite transmitir la imagen `?img` a la estación base `?bas`. Las precondiciones serían que la imagen ha sido tomada, y los efectos que la petición de la base ha sido servida.

   Como el enunciado nos dice que debemos modelar una de estas acciones de manera formal utilizando PDDL, la siguiente sería la modelización en PDDL correspondiente a la acción `tomar`:

```
(:action tomar
:parameters (?b - bas ?s - sat ?c - cam ?i - img ?p - pos ?m - modo)
:precondition (and (peticion ?b ?i) (descripcion ?i ?p ?m) (posicion ?s ?p)
(camara ?s ?c) (calibrada ?c ?m))
:effect
(and (img-tomada ?b ?i)))
```