
# Proyecto I:
# Smart Drone

**Integrantes del Grupo:**
**Jean Paul Davalos - 1832375**
**Miguel Angel Escobar - 2159832**
**Juan Manuel Perea Coronado - 1926462**
**Yenny Margot Rivas Tello - 2182527**

**Profesor:**
**Oscar Bedoya**

**Universidad del Valle**
**Escuela de Ingeniería de Sistemas y Computación**
**Inteligencia Artificial**
**2025 - 1**

# INTRODUCCIÓN.

En el presente informe se muestra el desarrollo e implementación del proyecto "Smart Drone", donde diseñamos un agente inteligente (dron) capaz de recorrer un entorno bidimensional para recolectar paquetes, tomando decisiones óptimas de desplazamiento en función de diferentes estrategias de búsqueda.

En el proyecto se implementaron algoritmos de búsqueda no informada como algoritmos de búsqueda informada, entre los que se encuentran: Búsqueda en Amplitud (BFS), Búsqueda de Costo Uniforme (UCS), Búsqueda en Profundidad evitando ciclos (DFS), Búsqueda Avara (GBFS) y Búsqueda A*. La heurística implementada en los algoritmos de búsqueda informada se basa en la distancia de Manhattan.

Este informe presenta los fundamentos que se usaron en la construcción de cada algoritmo y su implementación en el contexto del entorno simulado. Además se profundiza sobre la heurística y su admisibilidad para este caso.

# ÍNDICE.

**INTRODUCCIÓN..... 1**
**ÍNDICE..... 2**
**DESARROLLO DEL PROYECTO.....3**
ALGORITMOS DE BÚSQUEDA NO INFORMADA..... 3
BÚSQUEDA POR AMPLITUD..... 3
BÚSQUEDA POR COSTO UNIFORME.....4
BÚSQUEDA PREFERENTE POR PROFUNDIDAD EVITANDO CICLOS..... 5
ALGORITMOS DE BÚSQUEDA INFORMADA.....6
BÚSQUEDA AVARA.....6
BÚSQUEDA A*..... 7
**HEURÍSTICA..... 8**

# DESARROLLO DEL PROYECTO.

## ALGORITMOS DE BÚSQUEDA NO INFORMADA.

### BÚSQUEDA POR AMPLITUD.

El algoritmo de amplitud (**BFS**) busca en primer lugar la ubicación del dron (**2**) en el mapa, después busca cuántos paquetes (**4**) hay en el mapa, si no encuentra los paquetes da un mensaje que no hay paquetes disponibles en el mapa. Luego se le da al dron sus movimientos que puede realizar (derecha, abajo, arriba, izquierda) y por último para terminar se agrega unas variables para el BFS las cuales son: la cola, conjunto de posiciones visitadas, los nodos expandidos, los paquetes encontrados y el camino .

Después tenemos un **While**, el cual, se acaba cuando haya nodos en la cola y aun no se hayan recogido todos los paquetes del mapa. Dentro del **While** encontramos un **if** que verifica si en la celda llegada hay un paquete, si lo hay se agrega un paquete a la variable de paquetes recogidos, se reinicia la búsqueda en la posición actual con el camino ya recorrido y actualiza la variable visitado, tambien se vera en la interfaz que se le añade un punto por encontrar encontrar y recoger el paquete encontrado.

Lo siguiente que hay en el **While** es un **For** el cual exporta los vecinos en los movimientos dados anteriormente en la variable de direcciones, en este **For** verifica si el movimiento siguiente es válido, es decir, si no tiene ninguna de las siguientes restricciones:

- Que el movimiento siguiente esté dentro de la matriz dada.
- Que el movimiento siguiente no sea un obstáculo (**1**).
- Que el movimiento siguiente no sea una celda ya visitada anteriormente.

Entonces si el siguiente movimiento no tiene ninguna de las restricciones el dron se moverá a esa posición, lo añadirá a la cola y lo marcará como visitado. Después si el siguiente movimiento del dron se encuentra con una trampa (**3**) da un mensaje de que encontró una trampa en la posición actual y en la interfaz aparecerá que se le quita un punto si tiene y si no, se le un punto negativo.

# BÚSQUEDA POR COSTO UNIFORME.

Para el algoritmo de búsqueda de costo uniforme **(UCS)**, buscamos que el dron recorra el mapa recogiendo todos los paquetes que hay en este, siempre recorriendo el camino más barato o de menor costo, evitando a su vez los campos electromagnéticos.

Lo primero que hacemos es determinar el valor de cada casilla, para que así nuestro algoritmo tenga en cuenta el costo de cada una, esto lo hacemos con la función **obtener_costo_casilla(valor_celda)** que define el valor de cada casilla de la siguiente forma:

- Si es 0 (camino libre) o 4 (paquete), cuesta 1.
- Si es 3 (campo electromagnético), cuesta 8.
- Si es 1 (pared u obstáculo), cuesta infinito, o sea que no se puede pasar.

Una vez planteado esto y pasando a lo que es al algoritmo, lo primero que hacemos es dar con la ubicación del dron,buscando el número 2 en la matriz y guardando su posición. Luego, buscamos donde están los paquetes a recoger , es decir, todas las posiciones con el número 4.

Conociendo nuestro punto de inicio y dónde están los paquetes, el algoritmo comienza a trabajar. Además creamos la variable **final_path** la cual nos guardará todo el camino que sigue el dron, y los paquetes restantes serán guardados en **remaining_targets.**

Donde mientras haya paquetes por recoger, se utiliza una cola de prioridad que empieza desde la posición actual del dron. El algoritmo explora todas las casillas vecinas (arriba, derecha, abajo, izquierda), calcula cuánto cuesta llegar hasta allí. Esas casillas se van agregando a la cola con su respectivo costo y si la casilla tiene un campo electromagnético, aparece una advertencia, ya que es un camino más costoso.

Cuando se encuentra un paquete, se guarda el camino hasta él, además se actualiza la posición del dron, se elimina ese paquete de la lista de pendientes y continuamos para el siguiente paquete.

Si no se puede encontrar un camino hacia algún paquete, se detiene y se devuelve por el camino recorrido hasta ese punto.

Al final, si logra recoger todos los paquetes, se devuelve el camino recorrido por el dron.

# BÚSQUEDA PREFERENTE POR PROFUNDIDAD EVITANDO CICLOS.

El algoritmo de búsqueda preferente por profundidad (**DFS**, por sus siglas en inglés) implementado en este proyecto fue desarrollado de tal forma que el dron realice el recorrido por el tablero mencionado anteriormente (Con las mismas características) teniendo en cuenta una prioridad preestablecida en los movimientos o acciones disponibles para el dron en un determinado estado.

La lógica del algoritmo se estructura sobre una pila o **stack**, que se utiliza para almacenar los estados a explorar. Cada estado en la pila contiene la posición actual del dron, el camino recorrido hasta el momento, el conjunto de paquetes que ya han sido recogidos y un indicador booleano que determina si el dron puede devolverse (Esto se estableció para el caso en que recoja algún paquete).

También se crearon algunas funciones auxiliares para modular un poco las distintas tareas o cálculos que se necesitan para el desarrollo del algoritmo. Una de ellas es la función **obtener_vecinos()**, que se encarga de generar los posibles movimientos a partir de la posición actual, respetando la prioridad establecida (arriba, derecha, abajo, izquierda) y descartando movimientos hacia celdas fuera de los límites o que sean muros. Hay qué tener en cuenta que, por la naturaleza de la pila, donde los primeros elementos agregados son los últimos en salir, el orden de la prioridad establecida inicialmente, se invierte al momento de ejecutar la aplicación.

Para evitar ciclos, el algoritmo utiliza un conjunto llamado **visitados**, donde se guarda el estado con la posición actual y el conjunto de paquetes que han sido recogidos hasta ese punto, de tal forma que el dron no visite de nuevo una misma posición con la misma cantidad de paquetes recogidos, y así mismo evitar bucles infinitos. Sin embargo, existe la excepción para justo después de que el dron recoge un paquete, donde se permite retroceder a celdas anteriores durante un único paso adicional. Esto se logra usando el booleano mencionado anteriormente, **se_puede_regresar**, que se pone como true solo cuando un nuevo paquete es recogido.

Este proceso se repite hasta que todos los paquetes han sido recogidos o hasta que se detecta que no haya un camino posible hacia los paquetes restantes, en ese caso el algoritmo termina retornando **None**.

# ALGORITMOS DE BÚSQUEDA INFORMADA.

## BÚSQUEDA AVARA.

El algoritmo búsqueda avara (Greedy Best-First Search, **GBFS**) implementado en este proyecto utiliza la distancia de Manhattan como función heurística para estimar la proximidad del dron a los paquetes. El algoritmo se desarrolló buscando siempre expandir el nodo que parezca más prometedor; es decir, el que tenga la menor distancia estimada a un paquete.

Al igual que con el algoritmo de búsqueda preferente por profundidad, la implementación propuesta de búsqueda avara cuenta con algunas funciones auxiliares; además que permite evitar ciclos a través del uso de un conjunto **visitados** que almacena todas las posiciones que ya han sido exploradas. Así mismo, solo se permite devolver bajo el caso de que el dron haya recogido un paquete, mediante el booleano **se_puede_regresar**.

El algoritmo se ejecuta de forma iterativa mientras aún existan paquetes por recoger. En cada iteración, se ejecuta una nueva búsqueda avara desde la posición actual del dron hacia el paquete más cercano, utilizando una cola de prioridad o **heap**, que ordena las posibles rutas de exploración en función de la heurística calculada.

Cada entrada en la cola contiene tres elementos: el valor heurístico, la posición actual y el camino recorrido hasta ese punto. La posición más prometedora se obtiene primero de la cola y, si no ha sido visitada antes (a menos que haya recogido algún paquete), se marca como visitada y se revisa si hay un paquete en ella. En caso afirmativo, se elimina de la lista de paquetes restantes, se extiende el camino acumulado y se vuelve a planear una nueva búsqueda desde esa posición, volviendo a reiniciar el conjunto de visitados. Si no es un paquete, se siguen expandiendo los vecinos accesibles desde esa posición, siempre agregando a la cola aquellas rutas o nodos que llevan a nuevas posiciones viables con su correspondiente heurística.

Este proceso se repite hasta que todos los paquetes han sido recogidos o hasta que se detecta que no haya un camino posible hacia los paquetes restantes, en ese caso el algoritmo termina retornando **None**.

## BÚSQUEDA A*.

En este algoritmo utilizamos una heurística la cual es, distancia de Manhattan, definida en la función **heuristica_manhattan(pos1, pos2)**. Esta función calcula cuántos movimientos horizontales y verticales que hay que hacer para ir de una casilla a otra. Una vez definida la heurística. Lo primero será ver nuestro algoritmo y determinar la posición del dron, el cual en la matriz es un 2. Luego, busca todas las casillas donde hay paquetes, o número 4. Si no encuentra paquetes, termina el proceso.

Luego de saber dónde están el dron y los paquetes. El algoritmo empieza desde la posición actual del dron, y guarda esa posición como punto de partida. Crea una lista llamada **final_path** el cual almacena el camino recorrido hasta recoger todos los paquetes además, mientras queden paquetes por recoger, el algoritmo elige el paquete más cercano al dron, usando la distancia de Manhattan como criterio. Para llegar a ese paquete, se utiliza una cola de prioridad, que permite ordenar las opciones de movimiento por el costo total estimado (f = g + h), donde g es el costo real del camino recorrido hasta ese punto, y h es la estimación de lo que falta para llegar al objetivo.

Para determinar el costo de los caminos, el algoritmo evalúa las casillas vecinas y calcula el costo de moverse por las casillas casillas con la función **obtener_costo_casilla(valor_celda)**, donde:

- 1 si la casilla es un camino normal (0) o un paquete (4).
- 8 si es un campo electromagnético (3).
- infinito si es un obstáculo (1).

Si encuentra un camino hacia el paquete más cercano, lo guarda, lo agrega al recorrido final (final_path), mueve al dron a la nueva posición, y elimina ese paquete de la lista de pendientes. Si no puede encontrar un camino hacia algún paquete, lo indica y termina con el recorrido que haya podido completar. Al final, se retorna todo el camino que hizo el dron para recoger los paquetes.

# HEURÍSTICA.

La heurística usada para el desarrollo de los algoritmos de búsqueda informados, es la **distancia de Manhattan** la cuál es útil en contextos donde el entorno se puede representar mediante una matriz o tablero como el caso que se está estudiando.

La distancia de Manhattan se define como la suma de las diferencias absolutas entre las coordenadas horizontales y verticales de dos puntos; algo así como como una distancia en L desde el un punto inicial hasta un punto objetivo. Teniendo dos posiciones $A = (x_1, y_1)$ y $B = (x_2, y_2)$, la distancia de Manhattan entre ellos está dada por la expresión:

$$distancia\ de\ Manhattan\ =\ h(n)\ =\ \left|x_1 - x_2\right| + \left|y_1 - y_2\right|$$

Este valor representa el número mínimo de movimientos requeridos para llegar de un punto a otro cuando solo se pueden realizar desplazamientos ortogonales; es decir, hacia arriba, abajo, izquierda o derecha, tal y como se mueve el dron en el tablero.

Se sabe que una heurística es admisible si nunca sobreestima el costo real mínimo desde un estado específico hasta el objetivo o meta; es decir que $h(n)$ es admisible si $h(n) \leq CR(n)$; para todo estado $n$. Teniendo lo anterior en cuenta, la heurística $h(n) = distancia\ de\ Manhattan$ es una **heurística admisible**, pues siempre proporciona una subestimación o una estimación exacta del número real de pasos requeridos para que el dron alcance un paquete.

Dado que el dron no puede atravesar muros y solo se desplaza en las cuatro direcciones cardinales, la distancia de Manhattan representa un límite inferior exacto al coste real si el camino está libre de obstáculos (Escenario positivo u optimista), y sigue siendo una subestimación válida si hay obstáculos, ya que los que existen hacen que el coste sea mayor, por lo que el dron tendría que realizar al menos ese número de movimientos o más para llegar a algún paquete.

La admisibilidad de esta heurística garantiza que las decisiones tomadas están bien fundamentadas en cuanto a proximidad relativa.

Para acceder al repositorio de GitHub con el proyecto, de click en el siguiente enlace:

Repositorio del Proyecto.