



Universidad
de Cádiz

Escuela Superior
de Ingeniería

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

DISEÑO Y FABRICACIÓN DE ROBOTS BASADOS EN ARDUINO PARA REALIZAR EL ALGORITMO DE COLONIAS DE HORMIGAS

AUTOR: JUAN MANUEL GÓMEZ
HUTCHISON

Puerto Real, Abril 2021

“Nuestra recompensa se encuentra en el esfuerzo y no en el resultado, un esfuerzo total es una victoria completa.”

Mahatma Gandhi. (1869 - 1948) Pacifista, político, pensador y abogado hinduista indio

Resumen

En el mundo natural, las hormigas, inicialmente, vagan de manera aleatoria en busca de comida y una vez la encuentran, regresan a su colonia dejando un rastro de feromonas.

Existen diferentes variantes de algoritmos probabilísticos que modelan como, una hormiga u otros animales o bichos que funcionan de la misma manera a la hora de buscar alimento, realizan dicha búsqueda, todos estos algoritmos son miembros de la familia de los *Algoritmo de Colonias de Hormigas* (ACO). Este algoritmo se convertirá en la base del proyecto ya que será la que se estudie tanto virtualmente, como físicamente con su implementación en robots reales.

En el siguiente documento estudiaremos el ACO a fondo para su completa comprensión y sus ventajas frente a otros algoritmos conocidos.

Además de su explicación teórica, como se comentó en anteriores párrafos, también se diseñarán y fabricarán robots basados en Arduino los cuáles pondrán en práctica dicho algoritmo, ésto nos permitirá obtener resultados reales sobre el ACO.

Índice general

Índice de Figuras	VII
Índice de tablas	IX
1. Introducción	2
1.1. Objetivos	2
1.2. Alcance del proyecto	3
1.3. Estructura del documento	3
2. Estado del arte y trabajos relacionados	5
2.1. Historia del Algoritmo de Colonia de Hormigas	5
2.1.1. Estimergia	5
2.1.1.1. Feromonas	6
2.1.2. Computación Evolutiva	7
2.1.2.1. Inteligencia de enjambre	8
2.1.3. Optimización por Colonia de hormigas	8
2.1.3.1. Pseudocódigo del Algoritmo de Colonias de Hormigas	10
2.1.3.2. Fórmulas principales del Algoritmo de Colonias de Hormigas	11
2.1.3.3. Ejemplo de una posible ejecución del ACO	12
2.2. Proyectos basados en el Algoritmo de Colonias de Hormigas	21
2.2.1. Optimización de maniobras en barcos	22
2.2.2. Optimización de taladros en placas de circuito impreso	22
2.2.3. Stigmergy: procesos de innovación	23
3. Metodología	24
3.1. Planificación	24
3.2. Implementación del Algoritmo de Colonias de Hormigas en C++	26
3.2.1. Análisis de Requisitos	27
3.2.2. Diseño	28
3.2.3. Implementación	28
3.2.4. Verificación	31
3.2.5. Mantenimiento	34
3.2.6. Ejecución del Algoritmo de Colonias de Hormigas	34
3.3. Fabricación y montaje de la hormiga	35
3.3.1. Finalidad del robot	36
3.3.2. Electrónica necesaria	36
3.3.2.1. Motor de corriente continua	38

3.3.2.2.	Sensor óptico infrarrojo	39
3.3.2.3.	Sensor de colores	40
3.3.2.4.	Arduino Nano	41
3.3.2.5.	Módulo Bluetooth	43
3.3.2.6.	Batería Li-Po	45
3.3.3.	Hojas de características (Datasheets)	47
3.3.4.	Interconexión de componentes electrónicos	48
3.3.4.1.	Placa de circuito impreso para sensores ópticos infrarrojos . .	48
3.3.4.2.	Placa de circuito impreso para interconexión del robot	52
3.3.5.	Estructura física del robot	55
3.3.5.1.	Chasis para el robot	56
3.3.6.	Montaje	57
3.3.6.1.	Preparación de piezas	57
3.3.6.2.	Soldadura a la placa base	59
3.3.6.3.	Montaje y preparación del chasis	60
3.3.7.	Test de piezas individuales	63
3.4.	Estructura del mapa	64
3.4.1.	Estudio previo	64
3.4.2.	Diseño del tablero	65
3.4.2.1.	Tablero Versión 1.0	66
3.4.2.2.	Tablero Versión 2.0	68
3.4.2.3.	Tablero Versión 3.0	69
3.4.2.4.	Tablero Versión 4.0	70
3.4.3.	Impresión del tablero	71
3.5.	Implementación del Algoritmo de Colonia de Hormigas en Arduino.	72
3.5.1.	Especificaciones que debe cumplir el Robot en Arduino	72
3.5.2.	Implementación en Arduino	73
3.5.3.	Problemas durante el desarrollo en Arduino	76
3.6.	Comunicación Cliente Servidor	77
3.6.1.	Interfaz de comunicación	77
3.7.	Implementación de la comunicación	79
3.8.	Aplicación servidor	80
3.8.1.	Antecedentes de la aplicación servidor	80
3.8.2.	Rol de la aplicación servidor	80
3.8.3.	Desarrollo de la aplicación	81
3.8.3.1.	Ejecución de la aplicación servidor	82
3.9.	Integración del robot con el servidor	84
3.10.	Ejecución del proyecto completo	85
4.	Experimentación y resultados	88
4.1.	Experimento teórico del Algoritmo de Colonias de Hormigas	89
4.2.	Experimento del Algoritmo de Colonias de Hormigas en C++	96
4.2.1.	10, 100, 1000, 10000 iteraciones con 2 hormigas	98
4.2.2.	10, 100, 1000, 10000 iteraciones con 6 hormigas	104
4.3.	Contraste de resultado al lanzar dos y seis hormigas	112
4.4.	Experimento del proyecto en Arduino	113
4.5.	Contraste de resultados obtenidos	118

5. Presupuesto	122
5.1. Diagrama de Gantt	122
5.2. Presupuesto	125
6. Conclusiones	129
A. Anexo: Manual de usuario	131
A.1. Introducción	131
A.2. Elementos incluidos	131
A.3. Metodología de uso	132
A.4. Plan de contingencia en caso de error	133
B. Anexo: Código empleado en el proyecto	134
B.1. Introducción	134
B.1.1. Bibliotecas propias del robot Arduino	135
B.1.1.1. PINS	135
B.1.1.2. CNY	136
B.1.1.3. 2xCNY	137
B.1.1.4. TSC3200	140
B.1.1.5. MOTOR	142
B.1.1.6. COMMON_TYPES	146
B.1.1.7. RATIONAL	146
B.1.1.8. PATH	151
B.1.1.9. MAP	153
B.1.2. Bibliotecas propias del servidor	160
B.1.2.1. matrix	161
B.1.2.2. path	162
B.1.2.3. rational	165
B.1.3. Código de testeo del robot	170
B.1.4. Código de testeo de la comunicación Bluetooth	179
B.1.5. Código del robot Arduino	182
B.1.6. Código del servidor	191
B.1.7. Código específico de comunicación	196
B.1.7.1. Comunicación del robot	197
B.1.7.2. Comunicación del servidor	198
B.1.8. Implementación del Código en C++	201
B.1.8.1. Bibliotecas que usa el programa en C++	202
B.1.8.2. Código principal en C++	219
B.1.8.3. Makefile	221
Bibliografía	222

Índice de figuras

1.1. Hormigas buscando alimento.	3
2.1. Hormigas comunicándose con el entorno según marca la definición de la estimeria	6
2.2. Jerarquía de algoritmos evolutivos	7
2.3. Resumen de antecedentes del ACO	9
2.4. Grafo ejemplo ACO	12
2.5. Adyacentes del nodo 1	15
2.6. Camino con feromonas actualizadas $Q = 1$	18
2.7. Camino con feromonas actualizadas $Q = 3$	19
2.8. Camino con feromonas evaporadas $Q = 1$	21
2.9. Camino con feromonas evaporadas $Q = 3$	21
3.1. Diagrama de operaciones del proyecto	26
3.2. Pasos de la metodología de desarrollo en Cascada	27
3.3. Jerarquía de archivos de la implementación del ACO	29
3.4. Especificación de las operaciones más relevantes de la clase Tablero contenidas en el archivo "tablero.hpp"	31
3.5. Ejecución de tests unitarios realizados en Gtest	33
3.6. Ejecución del ACO implementado en C++.	34
3.7. Mapa de calor post-ejecución del ACO.	35
3.8. Motor DC marca SODIAL, 3V a 0.3A y 15RPM con reductora.	38
3.9. Controladora de motor L298N.	39
3.10. Sensor óptico infrarrojo CNY70.	40
3.11. Sensor de colores TSC-3200.	40
3.12. Microcontroladores candidatos, Raspberry Pi Zero y Arduino Nano.	42
3.13. Módulo Bluetooth HC-05.	45
3.14. Medición de la Intensidad del robot.	46
3.15. Batería LiPo 7.4V a 2200mAh.	47
3.16. Esquemático de interconexión para sensores ópticos infrarrojos.	49
3.17. CNY70 internamente.	50
3.18. Diseño de la PCB para acoplar los sensores CNY70.	51
3.19. Cara top del circuito impreso diseñado en la Figura 3.18.	51
3.20. Cara bottom del circuito impreso diseñado en la Figura 3.18.	52
3.21. Esquemático de la PCB de interconexión de todos los componentes.	53
3.22. Diseño de la PCB de interconexión de todos los componentes.	54
3.23. Cara top del circuito impreso diseñado en la Figura 3.22.	55
3.24. Cara bottom del circuito impreso diseñado en la Figura 3.22.	55
3.25. Chasis Pololu robot Zumo.	56

3.26. Dimensión del chasis <i>Pololu Zumo</i>	57
3.27. Chasis <i>Pololu robot Zumo</i> cortado.	58
3.28. Placa base montada.	60
3.29. Placa base montada.	61
3.30. Robot que simula una hormiga en el ACO.	62
3.31. Tablero v1.0.	67
3.32. Versión 2.0 del tablero.	68
3.33. Versión 2.1 del tablero.	69
3.34. Versión 3.0 del tablero.	70
3.35. Versión 4.0 del tablero.	71
3.36. Versión 4.0 del tablero impreso.	71
3.37. Diagrama de flujo de controladores en el programa “ACO.ino”.	75
3.38. Interfaz de comunicación tras cada iteración. Podemos encontrar su implementación en el Anexo B.1.7.	79
3.39. Ventana principal del programa servidor.	82
3.40. Servidor en su estado inicial.	85
3.41. Estado inicial del robot en Arduino.	86
3.42. Estado final del robot en Arduino.	87
 4.1. Mapa ponderado del experimento.	89
4.2. Gráfica referente a los datos obtenidos en la Tabla 4.3.	95
4.3. Ejecución obtenida de lanzar 2 hormigas durante dos períodos de tiempo.	96
4.4. Gráfica obtenida a partir de los resultados de la Tabla 4.4.	98
4.5. Ejecución del ACO de dos hormigas durante diez iteraciones.	99
4.6. Ejecución del ACO de dos hormigas durante cien iteraciones.	100
4.7. Ejecución del ACO de dos hormigas durante mil iteraciones.	101
4.8. Ejecución del ACO de dos hormigas durante diez mil iteraciones.	102
4.9. Gráfica referente a los datos obtenidos en la Tabla 4.5.	103
4.10. Ejecución del ACO de seis hormigas después de diez iteraciones.	105
4.11. Ejecución del ACO de seis hormigas después de cien iteraciones.	106
4.12. Ejecución del ACO de seis hormigas después de mil iteraciones.	107
4.13. Ejecución del ACO de seis hormigas después de diez mil iteraciones.	108
4.14. Gráfica de resultados de ejecutar seis hormigas tras varias iteraciones.	109
4.15. Resultados de ejecutar seis hormigas tras diez mil iteraciones.	110
4.16. Gráfica de resultados recopilados en la Tabla 4.7.	111
4.17. Comparativa entre varias ejecuciones con 2 hormigas y 6 hormigas	112
4.18. Resultados obtenidos de los robot en Arduino tras 1 iteración.	114
4.19. Resultados obtenidos de los robot en Arduino tras 2 iteraciones.	114
4.20. Gráfica recogida de la Tabla 4.8.	115
4.21. Tabla de feromonas tras 10 iteraciones	116
4.22. Tabla de feromonas tras 20 iteraciones.	116
4.23. Gráfica obtenida sobre los resultados captados en la Tabla 4.9.	117
4.24. Gráfica recogida de la Tabla 4.10.	119
4.25. Gráfica recogida de la Tabla 4.11.	121
 5.1. Diagrama de Gantt.	123

Índice de tablas

3.1.	Métodos sobre los que se han realizado pruebas unitarias en la implementación en C++ del ACO.	32
3.2.	Listado de componentes electrónicos necesarios para la construcción del robot.	37
3.3.	Cumplimiento de características.	41
3.4.	Características específicas del microcontrolador.	43
3.5.	Características específicas.	44
3.6.	Hojas de características de cada componente electrónico.	47
3.7.	Paleta de colores utilizada en el tablero.	66
4.1.	Actualización de feromonas, primera iteración.	91
4.2.	Actualización de feromonas, segunda iteración.	94
4.3.	Resultado de feromonas obtenidas de las iteraciones 1, en la Tabla 4.1 y 2, en la Tabla 4.2.	94
4.4.	Resultado obtenido de ejecutar 2 hormigas en 2 iteraciones en C++, datos recogidos de la Figura 4.3.	97
4.5.	Recopilación de resultados de las ejecuciones anteriores.	102
4.6.	Recopilación de resultados de las ejecuciones anteriores.	108
4.7.	Recopilación de resultados tras dos ejecuciones del mismo programa, con seis hormigas y diez mil iteraciones.	111
4.8.	Recopilación de resultados tras dos iteraciones en Arduino.	114
4.9.	Recopilación de resultados tras veinte iteraciones en Arduino.	117
4.10.	Resultados obtenidos tras una iteración.	119
4.11.	Resultados obtenidos tras dos iteraciones.	120
5.1.	Presupuesto del proyecto de final de carrera.	126

Acrónimos

ACO *Algoritmo de Colonias de Hormigas.* [III](#), [VII–IX](#), [2–5](#), [8](#), [9](#), [11](#), [12](#), [21](#), [22](#), [24–26](#), [29](#), [30](#), [32](#), [34–36](#), [42](#), [62](#), [72](#), [73](#), [77](#), [84](#), [88](#), [96](#), [98–108](#), [121](#), [129](#), [201](#)

CAA *Algoritmo de Colonia de Abejas Artificiales.* [9](#)

DC *Direct Current.* [37](#)

IA *Inteligencia Artificial.* [2](#), [7](#)

LiPo *Litio y Polímero.* [VII](#), [45](#), [47](#), [52](#)

PCB *Printed Circuit Board.* [VII](#), [22](#), [25](#), [48](#), [51–55](#)

TAD *Tipo Abstracto de Datos.* [29](#)

TDD *Test-Driven Development.* [31](#)

Capítulo 1

Introducción

El **ACO**, es una técnica probabilística para solucionar problemas computacionales que se reducen en buscar los mejores caminos en grafos.[1]

Dicho algoritmo es miembro de la familia de inteligencia de enjambres, rama de la *Inteligencia Artificial (IA)* que estudia el comportamiento colectivo de los sistemas descentralizados, auto organizados, naturales o artificiales.[2] Como veremos posteriormente en el caso práctico el algoritmo se comportará de manera centralizada, comunicándose con un "hormiguero".

En los siguientes capítulos iremos como nace el **ACO**, y en qué ámbitos gana potencia y por tanto, es recomendable usarlo, además de entenderlo paso a paso.

1.1. Objetivos

Con todo, se propone como objetivo principal entender el **ACO** y demostrar como trabaja físicamente en robots reales. Además del diseño y fabricación de robots en los que se pueda implementar dicho algoritmo.

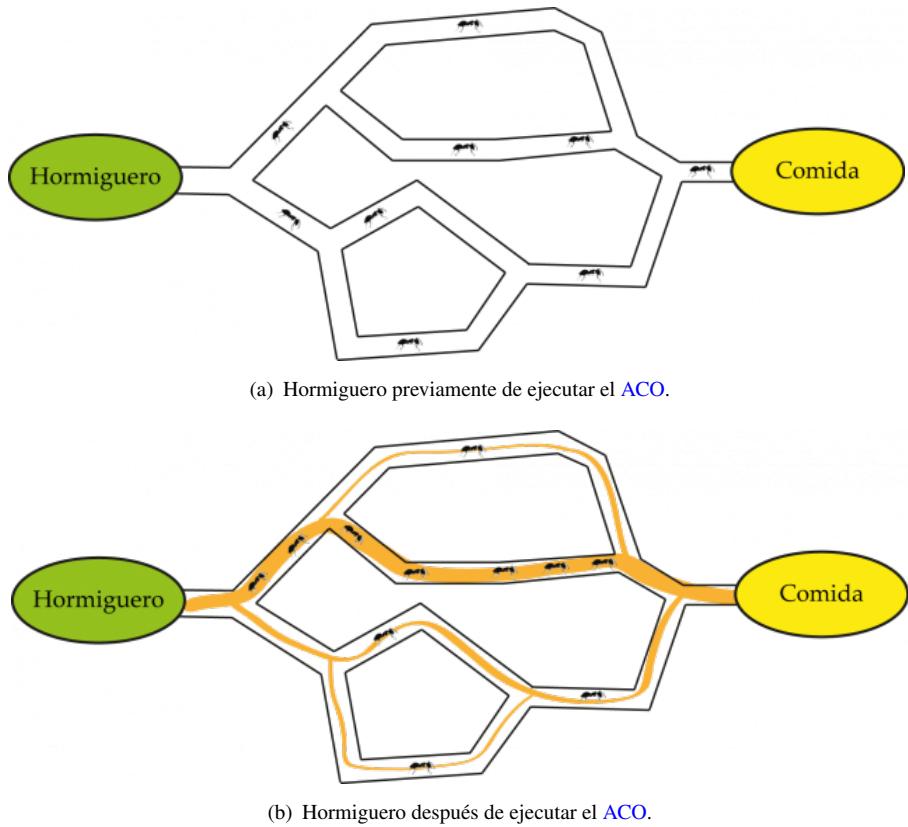


FIGURA 1.1: Hormigas buscando alimento.

1.2. Alcance del proyecto

El alcance del presente proyecto está formado por los siguientes puntos:

- Comprensión del funcionamiento del ACO.
- Ejecución virtual del algoritmo y discusión de resultados.
- Diseño de un robot que simule una hormiga.
- Diseño de un tablero sobre el que los robots realizarán el ACO.
- Ejecución del algoritmo en un entorno real y discusión de resultados.

1.3. Estructura del documento

El documento se compone de los siguiente capítulos:

- Capítulo 1: Introducción. Donde se comentan los objetivos a conseguir al finalizar el proyecto.
- Capítulo 2: Estado del arte y trabajos relacionados (Sección 2). Donde se pretende entender cómo nace el **ACO** hasta su fase de madurez y proyectos reales que lo utilizan.
- Capítulo 3: Metodología (Sección 3). Donde se diseña el robot, y se ensambla además de cómo se ha diseñado y creado el tablero donde el robot correrá, así como el código que hace que el robot realice el algoritmo.
- Capítulo 4: Experimentación y resultados (Sección 4). Se realizan distintas pruebas del **ACO** y se discuten los resultados de dichas pruebas.
- Capítulo 5: Presupuesto (Sección 5). Se muestra desglosado el precio de realizar el proyecto.
- Se ofrece un manual de uso, para cualquier usuario que necesite utilizar los componentes realizados en este proyecto en el Anexo: Manual de usuario.
- Para acabar, se llegan a las conclusiones tomadas en el proyecto en la Sección 6.

Capítulo 2

Estado del arte y trabajos relacionados

En el presente capítulo se procede a exponer la historia del ACO desde el comportamiento de las hormigas, hasta las mejoras que hay de dicho algoritmo. Y por otra parte se mostrarán proyectos reales que utilizan el ACO.

2.1. Historia del Algoritmo de Colonia de Hormigas

En esta sección se explica el nacimiento y evolución del ACO

2.1.1. Estimergia

La *Estimergia* fue introducida por Pierre-Paul Grassé [3], un estudioso de, entre otros, animales e insectos, explicó como las hormigas y muchos insectos que trabajan en grupo, se comunican entre sí sin necesidad de planificación ni de un poder central que dictamine su comportamiento.

Podemos tomar la *Estimergia* como punto de partida de nuestro proyecto, se define como *Estimergia*, a la colaboración de, en nuestro caso las hormigas, con el medio físico. Las hormigas consiguen comunicarse con el medio físico a través de la información que dejan en las feromonas, las cuales sirven también para comunicar a las hormigas entre sí.

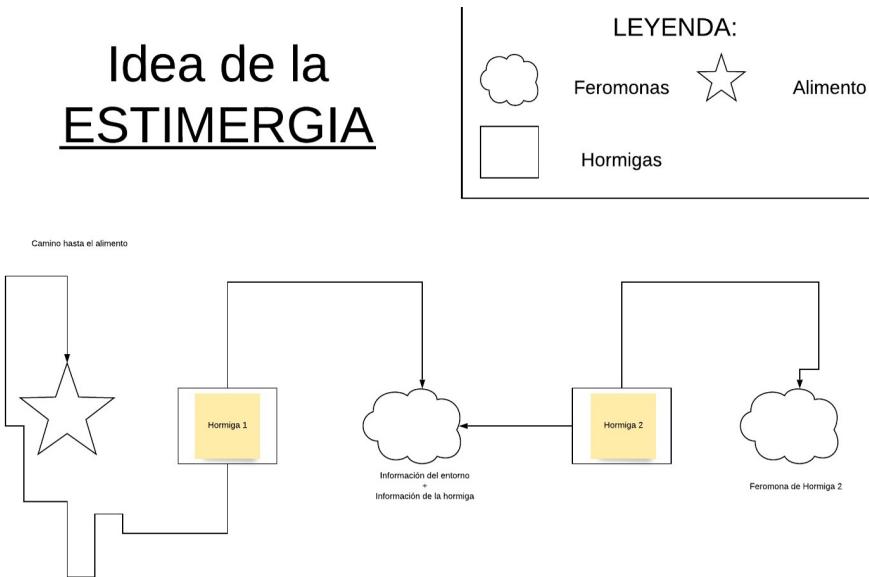


FIGURA 2.1: Hormigas comunicándose con el entorno según marca la definición de la estimergia

2.1.1.1. Feromonas

Las feromonas se producen a través de las glándulas exocrinas. En función de las distintas proporciones de feromonas procedentes de varias glándulas, estas señales químicas tienen distintos significados según el contexto en el que se libere la feromona.

Las feromonas que más nos interesan para nuestro proyecto son:

- Reclutamiento: Para buscar alimento, conseguir nuevos lugares para hacer nido y enfrentar a los enemigos.
- Trofalaxia: intercambio entre individuos en el que uno de ellos proporciona líquidos orales, anales o de otra índole.
- Intercambio de partículas de alimento sólido.
- Efecto grupal: facilitación o inhibición colectiva de una actividad determinada.

Además de estas, hay más tipos de feromonas como se cita en [4].

2.1.2. Computación Evolutiva

La *Computación evolutiva*, que es un paradigma abstracto cuya principal propiedad es la mejora de orden computacional mediante la **IA**, es el nodo raíz de una jerarquía de diversos algoritmos desde algoritmos evolutivos, cuya función es la selección natural, hasta algoritmos basados en enjambres de insectos/animales que trabajan en conjuntos con un fin común.

No se debe confundir, en esta clasificación basada en la *Computación evolutiva* [5] definida en el párrafo anterior con los *Algoritmos evolutivos*, basados en la selección natural biológica y que son una especialización de la *Computación evolutiva* en la jerarquía mostrada en la Figura 2.2.

La finalidad de este párrafo es saber que tenemos una jerarquía en la que el nodo raíz es la *Computación evolutiva* cuya principal propiedad es la búsqueda de un orden computacional mejor, propiedad que heredarán todos los descendientes de esta jerarquía que podemos contemplar en la siguiente Figura:

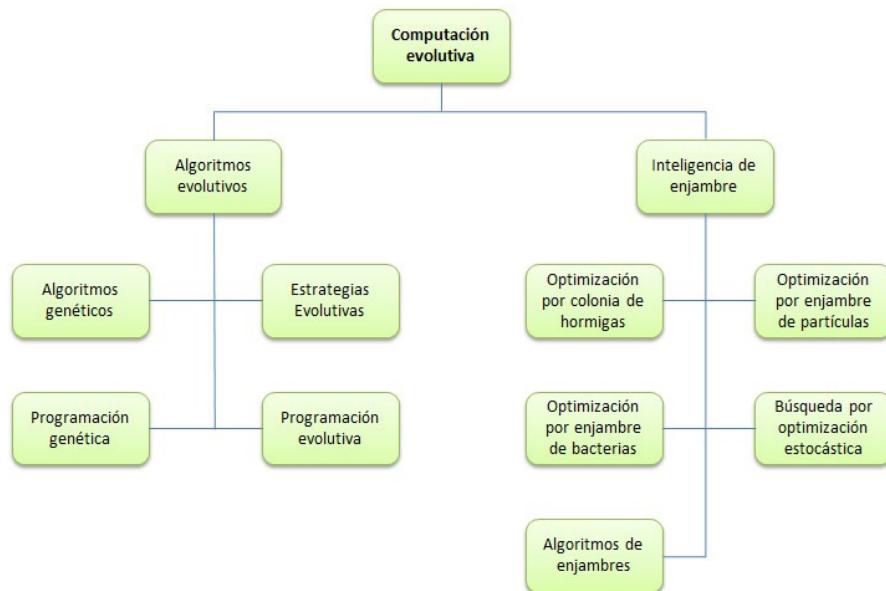


FIGURA 2.2: Jerarquía de algoritmos evolutivos

2.1.2.1. Inteligencia de enjambre

Como siguiente acercamiento al [ACO](#), conviene hablar de la *Inteligencia de Enjambre*, ya que, como podemos observar en la Figura 2.2, es el padre directo del [ACO](#).

Inspirados por la naturaleza, los sistemas de *Inteligencia de enjambre* están formados por agentes computacionales simples que interactúan entre sí y con el medio (*Estimergia* 2.1.1).

Estos agentes computacionales siguen reglas simples y no tienen una estructura de control centralizado. La simpleza de estos agentes locales en conjunto, provoca un comportamiento global complejo.

Todo esto, visto desde un punto de vista algorítmico hace que empecemos a abstraernos de la realidad y nos acerquemos más a nuestro objetivo que es el de explicar el [ACO](#).

2.1.3. Optimización por Colonia de hormigas

Hasta ahora hemos abarcado varios puntos, con el fin de llevarnos al punto cumbre de este capítulo en el cuál, explicaremos el [ACO](#), antes de seguir, en la siguiente Figura, podemos ver un breve resumen de todo lo comentado anteriormente:

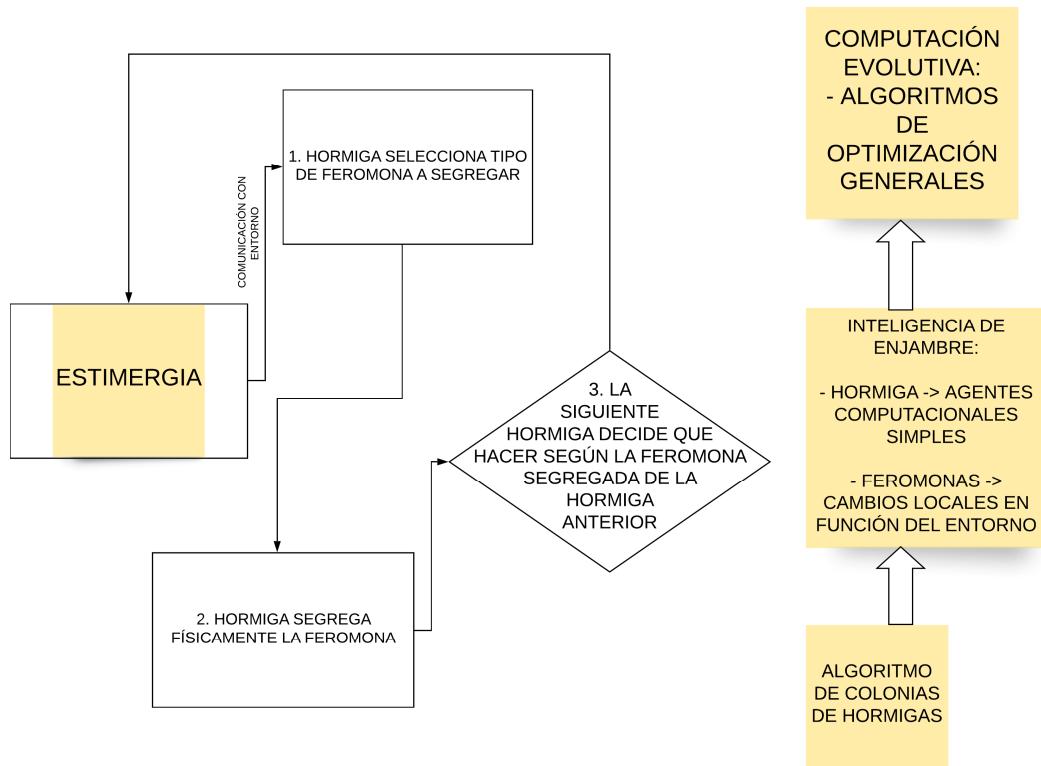


FIGURA 2.3: Resumen de antecedentes del ACO

Partimos de la base de que ya sabemos como una hormiga busca alimento 2.1.1 y sabemos lo que es la *Inteligencia de enjambre* 2.1.2.1, a partir de esto, construiremos el ACO.

La propiedad principal de la *Computación evolutiva* 2.1.2 es la mejora de orden computacional, la cual para cada problema concreto se consigue con un algoritmo distinto, puede que, el *Algoritmo de Colonia de Abejas Artificiales* (CAA) [6] sea más eficiente que el ACO para un problema concreto, luego habría que hacer un estudio detallado antes de elegir qué algoritmo usar. No se abarcará este tema en este proyecto ya que no corresponde, puesto que estoy haciendo un estudio propiamente, del ACO.

El *Algoritmo de Colonias de Hormigas*, o ACO, surgió en 1992, por Marco Dorigo en su tesis doctoral, para buscar un camino óptimo en un grafo, basado en el comportamiento de las hormigas cuando estas buscaban un camino entre alimento y su colonia u hormiguero.

La idea original se ha aplicado para resolver gran cantidad de problemas de carácter numérico.

Si retomamos lo explicado en puntos anteriores, sabemos cómo una hormiga busca alimento superficialmente, pero hay determinadas cosas a tener en cuenta, la primera de ellas es la Evaporación de Feromonas. La principal función de la evaporación de feromonas es condicionar a las hormigas que se dejen seducir por éstas. Cuanto más largo sea el camino, más tiempo tienen las feromonas para evaporarse, lo cuál condicionará menos a una hormiga que una feromona de un camino más corto, y por tanto, más fuerte.

Como segundo punto a tener en cuenta, debemos saber que las hormigas se dejan condicionar por las feromonas que suelta una hormiga al recorrer un camino completo entre alimento y hormiguero, es decir, que cuando regresa al hormiguero es cuando esas feromonas se tienen en cuenta y condicionan el camino de las siguientes hormigas que lo recorran.

Tercero, no se visitarán nodos que ya hayan sido visitados. Esto es una mejora frente a la realidad, ya que, podemos saber que nodos hemos recorrido y cuales no.

2.1.3.1. Pseudocódigo del Algoritmo de Colonias de Hormigas

A continuación, se describe el comportamiento de las hormigas sobre el cuál, se basará el algoritmo:

1. La(s) hormiga(s) vagan de manera aleatoria alrededor de la colonia con el objetivo de buscar comida.
2. Si alguna hormiga encuentra comida, vuelve a la colonia por un camino aleatorio, dejando tras de sí un rastro de feromonas.
3. Estas feromonas son atractivas, es decir, condicionan el camino que vayan a coger las siguientes hormigas, cuanto más fuerte sea la feromona, más probabilidad de tomar ese camino tiene la hormiga actual.
4. Al regresar al hormiguero, estas hormigas habrán fortalecido las feromonas de la ruta tomada, condicionando más a las siguientes hormigas que busquen alimento.
5. Si existen dos rutas que lleguen a la misma fuente de alimento, entonces, la ruta más corta es la que más hormigas hayan recorrido.

6. La ruta más corta habrá aumentado y será la más probable a tomar por las hormigas.
7. La ruta más larga irá desapareciendo ya que, debemos tener en cuenta que las feromonas se evaporan.
8. Finalmente, las hormigas habrán determinado y escogido el camino más corto.

Ahora procedemos a presentar el pseudocódigo del **ACO**, habrá que tener en cuenta dentro de las cajas negras presentadas en el pseudocódigo, el comportamiento explicado anteriormente.

```

begin procedure ACO:
    while(NOT(objetivo()) ) :
        actual := siguienteNodo()

    while(NOT(inicio()) ) :
        actual := siguienteNodo()

        actualizarFeromonas()
end procedure

```

Las funciones anteriormente escritas deben realizar las siguientes tareas:

- Función objetivo(): indica que hemos llegado a la fuente de alimento.
- Función inicio(): indica que hemos llegado al hormiguero, se utiliza para la vuelta, una vez encontrada la comida.
- Función siguienteNodo(): elige el siguiente nodo al que acceder basándose en las feromonas, luego en este punto es donde se define que el **ACO** es probabilístico.
- Función actualizarFeromonas(): se actualizan las feromonas, si las hormigas en este tiempo t han tomado un cierto camino, los nodos de este camino se fortalecerán mientras que, los nodos que no se hayan visitado en este tiempo t , se evaporarán.

2.1.3.2. Fórmulas principales del Algoritmo de Colonias de Hormigas

A continuación se muestran las principales fórmulas en las que se basa el **ACO** y su función:

- $p_{xy}^k = \frac{(\tau_{xy}^\alpha)(v_{xy}^\beta)}{\sum(\tau_{xy}^\alpha)(v_{xy}^\beta)}$: Fórmula para saber qué probabilidad tiene una hormiga de elegir el camino.
- $\sigma_{xy} = \sigma_{xy} + \sum \frac{1}{d_{xy}}$: Ecuación para saber el valor de las feromonas.
- $\frac{1}{d_{xy}}$: Ecuación de distancia.
- $\sigma_{xy} = (1 - \phi)\sigma_{xy} + \phi\sigma_0$: Ecuación de actualización de feromonas.
- $\sigma_{xy} = (1 - \rho)\sigma_{xy}$: Ecuación de Evaporación global.

2.1.3.3. Ejemplo de una posible ejecución del ACO

Seguidamente podemos ver un posible ejemplo, posible porque es probabilístico y puede tomar distintas variantes, no tiene por qué elegir el camino que se determina en el ejemplo, aunque si que podría.

Tomamos de referencia el siguiente grafo:

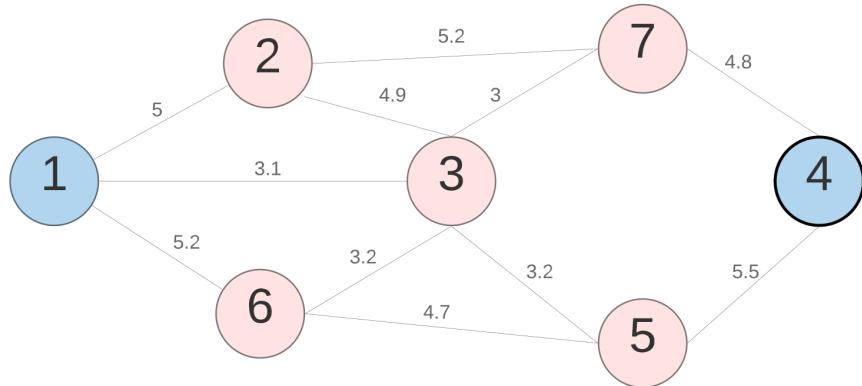


FIGURA 2.4: Grafo ejemplo ACO

Con el fin de buscar una fuente de alimento, un número determinado de hormigas h , sale del hormiguero. De inicio estas hormigas no conocen el mapa con lo cuál no saben qué camino tomar a la hora de encontrar esa fuente de alimento, el ACO, nos solucionará tal problema.

Por tanto tenemos:

- Por un lado un grafo $G \mid G = (V, A)$, siendo V el número de vértices y A el número de aristas que conecta los vértices, ponderado, conexo y bidireccional.
- Por el otro un número $h \mid h \in \mathbb{Z}^+$ de hormigas

Lo primero que debemos hacer es interpretar el grafo que tenemos en [2.4](#). Tenemos dos nodos resaltados en color azul que indican, el nodo 1 el inicio u hormiguero y, el nodo 4 que es donde está situada la comida o fuente de alimento, es decir nuestro destino, se han elegido tales nodos como origen y destino puramente al azar. El resto de nodos son nodos intermediarios que llevarán siempre a la fuente de alimento puesto que el grafo es conexo.

En el grafo [2.4](#) podemos lanzar el número de hormigas que queramos, teniendo en cuenta que cuantas más hormigas, más rápido encontrará un camino óptimo, en este ejemplo hecho a mano, lanzaremos 2 hormigas y veremos los resultados que generan.

Teniendo en cuenta las condiciones anteriormente descritas, vamos a dividir el ejemplo en dos partes:

- El camino que va a tomar la hormiga de ejemplo.
- La actualización de feromonas para condicionar la siguiente iteración.
- Adicionalmente haremos un segundo ejemplo que incluirá otra búsqueda de alimento pero esta con las feromonas actualizadas de la anterior iteración.

Para saber el camino que va a tomar una hormiga, primero, partimos de las siguientes premisas:

- Tenemos una feromona inicial global para todos los nodos igual a 0.01, luego $\text{FIG} = 0.01$.
- Tenemos un valor alpha igual a 1 que sirve para ponderar la feromona en un nodo, luego $\alpha = 1$.
- Tenemos un valor beta igual a 1 que sirve para ponderar la distancia de un nodo a otro, luego $\beta = 1$.
- Tenemos también la siguiente fórmula: $p_{xy}^k = \frac{(\tau_{xy}^\alpha)(v_{xy}^\beta)}{\sum(\tau_{xy}^\alpha)(v_{xy}^\beta)}$ enunciada anteriormente, la cuál nos da la probabilidad que tiene cada hormiga de tomar un camino entre un nodo x, y un nodo y.

- Tenemos la fórmula de la distancia que es inversamente proporcional al peso de la arista introducida en la fórmula de la probabilidad de camino.

Si juntamos todos estos datos usando de base la fórmula podemos observar que (τ_{xy}^{α}) , alpha condiciona a τ que es el valor de la feromonía que tiene un nodo, al elevarlo a α , si lo elevamos a un valor más alto que β estamos dándole más importancia a la variable de feromonía que a la de la distancia, v , si elevamos β a un valor más alto que α sucede lo contrario, damos más importancia a la distancia entre dos nodos, es decir, condicionamos a que ese camino o nodo en cada caso, sea más propicio a tomarlo por la hormiga.

Entonces, el procedimiento para tomar un camino es el siguiente:

1. Partimos del nodo inicial, o nodo en el que actualmente estemos y comprobamos los caminos adyacentes que tenemos, en estos caminos, aplicamos la fórmula de probabilidad de tomar un camino enunciada anteriormente en este mismo punto. Esta fórmula ponderará los los adyacentes condicionando la probabilidad del camino que aleatoriamente se vaya a tomar.
2. Una vez tenemos ponderada la probabilidad de cada nodo adyacente, lanzamos un número aleatorio de manera que según el número que salga estará en el rango de un adyacente o de otro, comprobamos el rango y según el adyacente al que pertenezca, ese camino seguirá la hormiga. Véase la ilustración 2.6 para entenderlo.

Luego aplicamos la fórmula a los adyacentes del nodo 1, que en este caso es nuestro inicio.

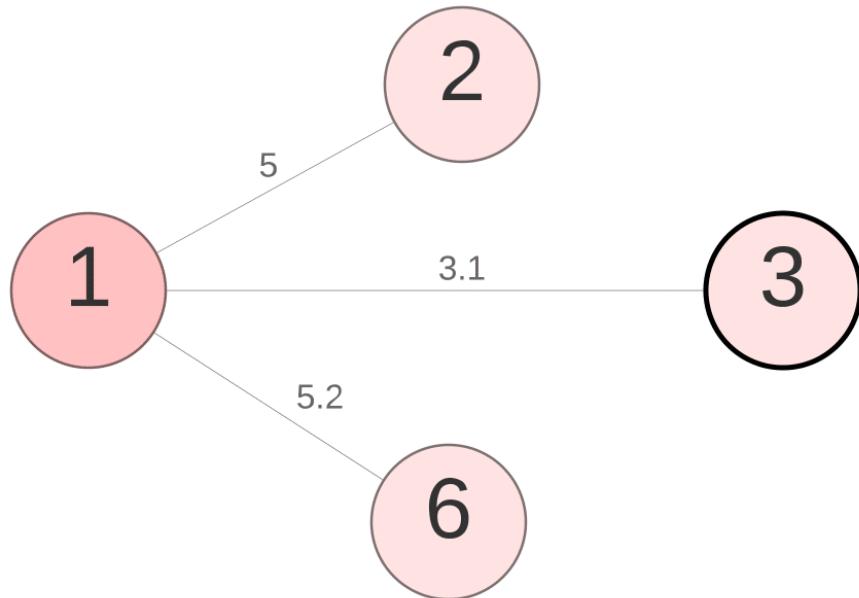


FIGURA 2.5: Adyacentes del nodo 1

En las aristas de [2.5](#) podemos observar la distancia ponderada entre cada par de nodos así que aplicamos la fórmula con una feromona inicial global de 0.01 ya que aún no ha pasado ninguna hormiga por ningún nodo del grafo. Luego quedarían las siguientes fórmulas:

- $p_{1-2}^k = \frac{(0,01)^\alpha * (\frac{1}{5})^\beta}{(0,01)^\alpha * (\frac{1}{5})^\beta + (0,01)^\alpha * (\frac{1}{3,1})^\beta + (0,01)^\alpha * (\frac{1}{5,2})^\beta} = 0.279$
- $p_{1-3}^k = \frac{(0,01)^\alpha * (\frac{1}{3,1})^\beta}{(0,01)^\alpha * (\frac{1}{5})^\beta + (0,01)^\alpha * (\frac{1}{3,1})^\beta + (0,01)^\alpha * (\frac{1}{5,2})^\beta} = 0.4507$
- $p_{1-6}^k = \frac{(0,01)^\alpha * (\frac{1}{5,2})^\beta}{(0,01)^\alpha * (\frac{1}{5})^\beta + (0,01)^\alpha * (\frac{1}{3,1})^\beta + (0,01)^\alpha * (\frac{1}{5,2})^\beta} = 0.2687$

Luego tenemos la probabilidad de tomar cada camino multiplicando estos valores por 100, es decir:

- Probabilidad de tomar el camino 2: $0.279 * 100 = 27.9\%$.
- Probabilidad de tomar el camino 3: $0.4507 * 100 = 45.07\%$.
- Probabilidad de tomar el camino 6: $0.2687 * 100 = 26.87\%$.

Ahora toca lanzar un dado en un rango de valores 0, 100, éste valor será una variable x, que en nuestro ejemplo $x = 18$, puramente elegido al azar.

Creamos nuestros rangos para ver en cuál ha caído nuestro número al azar. Ponemos en orden de lectura los nodos adyacentes, en este caso 1º-2; 2º-3; 3º-6. El camino a tomar será el nodo n si x pertenece al rango de n. Rangos:

- Rango del nodo 2: {0 %, 27.9 %}
- Rango del nodo 3: $\{(27.9 + 0.1) = 28 \%, (27.9 + 45.07) = 72.97 \%\}$
- Rango del nodo 6: $\{72.9 \%, (72.9 + 26.87) = 100 \%\}$ (En este caso redondeamos por el error cometido por la precisión al 100%).

Buscamos el rango de x, la condición es que x sea mayor que el mínimo del rango y menor que el máximo del rango, con lo cuál el rango que cumple dicha condición es:

- Rango del nodo 2: {0, 27.9} y x = 18.

Por tanto la hormiga tomará el nodo 2.

Este proceso se hace sucesivamente hasta llegar al nodo de destino, es decir, a la fuente de alimento que en nuestro ejemplo en particular es el nodo 4. Una vez llega al nodo de destino, la vuelta es exactamente igual hasta llegar al hormiguero o nodo de inicio, una vez llegue al nodo de inicio toca actualizar las feromonas.

Para realizar la actualización de feromonas tenemos que tener en cuenta que usamos la tasa de evaporación de feromonas. De la evaporación de feromonas tenemos que tener en cuenta los siguientes puntos:

- Una hormiga actualiza las feromonas una vez ha completado un camino de búsqueda de comida y vuelta al hormiguero.
- Cuando se actualizan las feromonas la probabilidad del camino tomado se condiciona incrementando la probabilidad de que las hormigas que vengan detrás, o esta misma cuando salga de nuevo a buscar comida, tomen dicho camino.
- Matemáticamente, tenemos las siguientes fórmulas las cuales nos permitirán actualizar las feromonas:

- $\sigma_{xy} = \sigma_{xy} + \sum \frac{1}{d_{xy}}$ donde σ_{xy} representa el valor de la feromona en un tiempo t, y $\sum \frac{1}{d_{xy}} = \frac{Q}{L_k}$ donde Q es el aprendizaje que tiene la hormiga, matemáticamente marcará el peso que se le dará a la feromona en cada iteración, y L_k que es el peso del camino ida-vuelta por comida. Con estos dos coeficientes tendremos el peso que tendrá nuestra nueva feromona, la cuál se suma a la que ya había.

- El punto anterior realizado por cada hormiga es el que va a permitir actualizar las feromonas y, por tanto, condicionar el camino de las hormigas hacia la comida.

A continuación mostramos un caso práctico del punto anteriormente explicado.

Supongamos que la hormiga ha tomado el camino: 1 - 6 - 3 - 7 - 4 de la figura 2.4. Si sumamos todos los pesos de las aristas del camino tomado nos sale un total de 16,2 unidades de distancia. Si tenemos en cuenta que para nuestro caso práctico usamos un aprendizaje de $Q = 1$, se sumará el valor de la feromona actual con el generado por la hormiga en cada par de nodos tomados: $\frac{1}{16,2} = 0,0617$ El valor resultante se sumará a cada par de nodos con la feromona que ya tienen, si suponemos que todas las feromonas tiene el valor inicial que si recordamos es 0,099, los nodos quedarán con las siguientes feromonas:

- 1 - 6: $0,099 + 0,0617 = 0,1607$
- 6 - 3: $0,099 + 0,0617 = 0,1607$
- 3 - 7: $0,099 + 0,0617 = 0,1607$
- 7 - 4: $0,099 + 0,0617 = 0,1607$

Mientras que el resto de feromonas seguirán valiendo 0,099, que si recordamos en el punto 2.1.3.2 el valor de la feromona que depositamos en el par de nodos condiciona la fórmula de búsqueda de camino.

FEROMONAS CON Q = 1

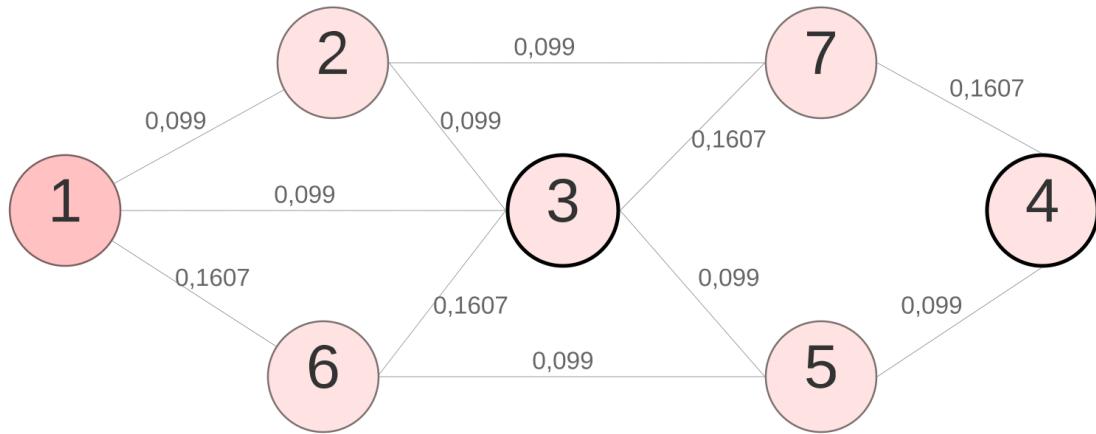


FIGURA 2.6: Camino con feromonas actualizadas Q = 1

Si queremos potenciar aún más la feromona por cada iteración, modificamos el aprendizaje, así que probamos con $Q = 3$, tomando el mismo camino del ejemplo anterior:

$$\frac{3}{16,2} = 0,185$$

Como podemos observar $Q(1) < Q(3) = 0,0617 < 0,185$. Por tanto hay más cantidad de feromonas en cada par de nodos tomados por la hormiga, es decir, por el camino: 1 - 6 - 3 - 7 - 4.

FEROMONAS CON Q = 3

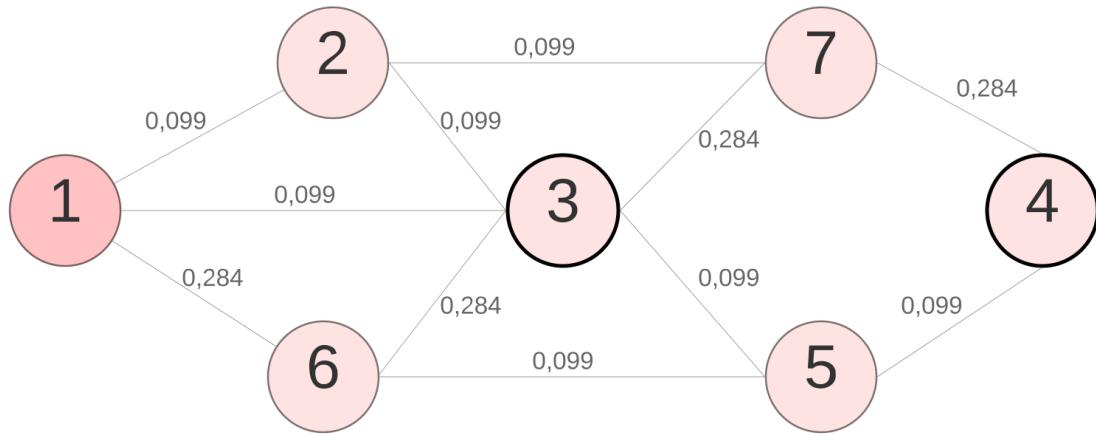


FIGURA 2.7: Camino con feromonas actualizadas Q = 3

Por último, por cada camino completo que una hormiga no tome un par de nodos, el valor de la feromona, se verá reducido mediante la siguiente fórmula:

$$\sigma_{xy} = (1 - \rho)\sigma_{xy}$$

Se deduce que para Q = 1 (Figura 2.8):

- $\sigma_{1-6} = 0,1607 * 0,0617 = 0,009921$
- $\sigma_{1-3} = 0,099 * 0,0617 = 0,0061$
- $\sigma_{1-2} = 0,099 * 0,0617 = 0,0061$
- $\sigma_{2-7} = 0,099 * 0,0617 = 0,0061$
- $\sigma_{3-7} = 0,1607 * 0,0617 = 0,009921$
- $\sigma_{2-3} = 0,099 * 0,0617 = 0,0061$
- $\sigma_{3-6} = 0,1607 * 0,0617 = 0,009921$
- $\sigma_{3-5} = 0,099 * 0,0617 = 0,0061$
- $\sigma_{6-5} = 0,099 * 0,0617 = 0,0061$
- $\sigma_{5-4} = 0,099 * 0,0617 = 0,0061$

- $\sigma_{7-4} = 0,1607 * 0,0617 = 0,009921$

Y para Q = 3 (Figura 2.9):

- $\sigma_{1-6} = 0,284 * 0,185 = 0,05254$

- $\sigma_{1-3} = 0,099 * 0,185 = 0,018315$

- $\sigma_{1-2} = 0,099 * 0,185 = 0,018315$

- $\sigma_{2-7} = 0,099 * 0,185 = 0,018315$

- $\sigma_{3-7} = 0,284 * 0,185 = 0,05254$

- $\sigma_{2-3} = 0,099 * 0,185 = 0,018315$

- $\sigma_{3-6} = 0,284 * 0,185 = 0,05254$

- $\sigma_{3-5} = 0,099 * 0,185 = 0,018315$

- $\sigma_{6-5} = 0,099 * 0,185 = 0,018315$

- $\sigma_{5-4} = 0,099 * 0,185 = 0,018315$

- $\sigma_{7-4} = 0,284 * 0,185 = 0,05254$

Con lo cual si aplicamos dicha fórmula para todos los nodos en los ejemplos de las figuras anteriores (2.6 , 2.7) nos queda el mapa de la siguiente manera:

FEROMONAS EVAPORADAS CON Q = 1

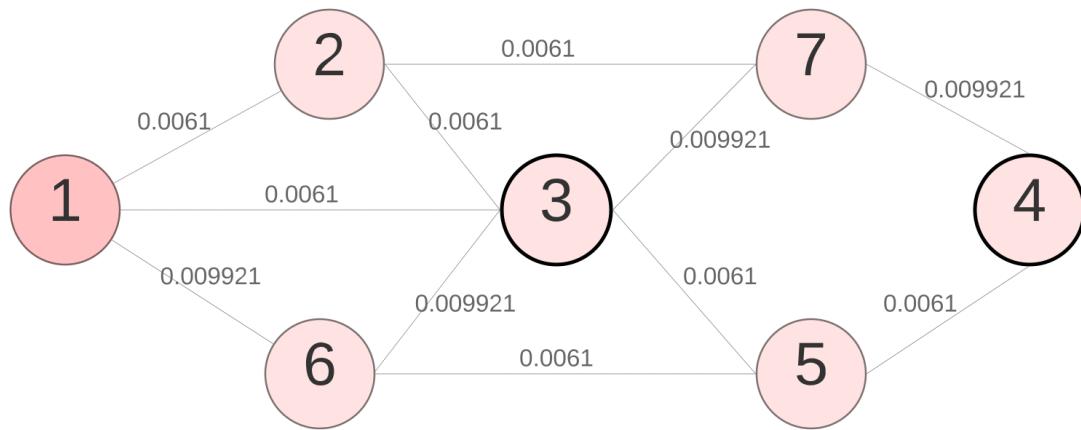


FIGURA 2.8: Camino con feromonas evaporadas Q = 1

FEROMONAS EVAPORADAS CON Q = 3

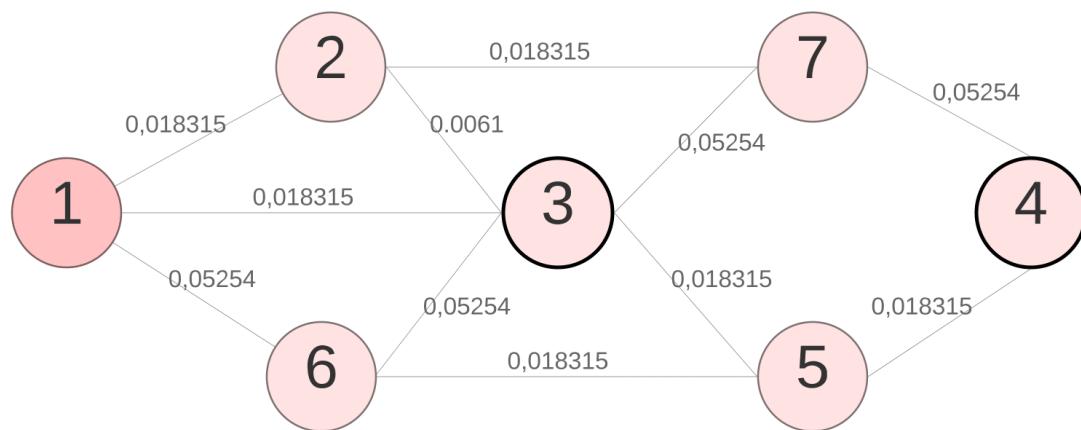


FIGURA 2.9: Camino con feromonas evaporadas Q = 3

2.2. Proyectos basados en el Algoritmo de Colonias de Hormigas

En esta sección se muestran proyectos que han basado su metodología en el ACO.

2.2.1. Optimización de maniobras en barcos

En el artículo [7] se describe la mejora de maniobras en barcos utilizando el ACO, que deriva como podemos apreciar en la Figura 2.2 de la familia de Inteligencia de Enjambre, tal como se cita en el artículo [7].

En éste se hace uso del algoritmo enfocado a la mejora de planificación de recorrido para vehículos autónomos, en este caso, enfocado a los barcos, cuyo “problema” consiste en obtener una secuencia óptima de velocidad y rumbo que permita trazar una maniobra realizable en el menor tiempo posible.

Como se cita en el artículo [7] las hormigas comienzan trazando trayectorias al azar, a medida que van alcanzando el objetivo, marcan con feromonas la trayectoria seguida, así comienzan a aparecer marcas de feromonas que guían la trayectoria para trazar la maniobra.

Las marcas de feromonas se depositan en el mapa a medida que se realiza una trayectoria, por tanto, las trayectorias más cortas recibirán un mayor peso de feromonas. Así, con el tiempo, las trayectorias se van optimizando, hasta que convergen en aquella que permite trazar la maniobra en el menor tiempo posible.

2.2.2. Optimización de taladros en placas de circuito impreso

Las Placas de Circuito Impreso, o en inglés *Printed Circuit Board* (PCB), acrónimo que utilizaremos a partir de ahora para referirnos a las placas de circuito impreso, son tarjetas que interconectan componentes electrónicos a través de pistas de cobre, en la Sección 3.3.4 se entrará en detalle de su definición, para entender este punto, es suficiente con entender esta pequeña definición de PCB.

En las PCB se realizan operaciones de orificios de taladro para acomodar por ejemplo, los componentes que se vayan a soldar. Sabiendo que el número de estos orificios puede llegar a ser de miles, es indispensable optimizar esta ruta de orificios.

En el artículo [8] se utiliza el ACO para optimizar la ruta que tomará el taladro a la hora de realizar los orificios en la PCB, ganando así tiempo, lo cuál se traduce en dinero.

Se cita que se utiliza el comportamiento de las hormigas a la hora de encontrar alimento, para mejorar la trayectoria de sistemas de perforación en circuitos impresos.

Implementa un código basado en MATLAB, el cuál define un número constante de hormigas, feromonas que se liberan en el camino y diferentes reacciones tomadas por las hormigas definiendo así la mejor trayectoria a tomar. Se cita que a mayor cantidad de hormigas, mayor cantidad de feromonas y, por tanto, mejor resultado a la hora de escoger el camino de menor trayectoria de desplazamiento.

Por tanto, se reduce el número de coordenadas por las que el taladro tiene que pasar y, por consiguiente, se mejora la ruta a seguir.

2.2.3. Stigmergy: procesos de innovación

Stigmergy [9] que traducido significa *Estimergia*, concepto que se explica en el punto [2.1.1](#), y que trata desde un punto de vista teórico la forma que tienen las hormigas de encontrar comida.

Stimergy [9], es una empresa que mediante la búsqueda de prioridades optimiza el trabajo de una tercera empresa que solicite sus servicios.

En su web [9] se cita textualmente: "En Stigmergy utilizamos algoritmos para facilitar el proceso de priorización de oportunidades de innovación, de forma que las empresas concentren sus esfuerzos en aquellas con mayor probabilidad de éxito".

Con lo cuál se deduce que esta empresa no aplica una implementación de un algoritmo de hormiga directamente si no que, introduce el concepto de *Estimergia* [2.1.1](#) como filosofía de actuación para abordar un problema abstracto a la hora de optimizar el trabajo que realiza un tercero para que, como se indica en su web [9]: "En Stigmergy utilizamos algoritmos para facilitar el proceso de priorización de oportunidades de innovación, de forma que las empresas concentren sus esfuerzos en aquellas con mayor probabilidad de éxito".

Capítulo 3

Metodología

En el siguiente capítulo se detalla la metodología utilizada para fabricar, montar y programar el robot que ejecutará el [ACO](#), además de la estructura general utilizada en el proyecto.

El capítulo abarca desde el diseño del algoritmo y del robot, hasta que se comprueba que el mismo se ejecuta correctamente.

3.1. Planificación

En este apartado veremos como se estructura el proyecto, que pasos se han seguido y en qué orden.

El proyecto final constará de una aplicación cliente-servidor, donde el servidor podemos simular que es el centro de mando donde se le indicará a la hormiga qué camino ha de tomar entre otros datos, y habrá dos tipos de clientes, virtuales y físicos, lo cual se explicará más adelante.

El cliente físico será el robot desarrollado en el proyecto.

A continuación se sintetiza cada paso que, posteriormente, será redactado en el punto que corresponda:

1. El primer paso que se ha tomado ha sido entender qué es, como funciona, y que aplicación tiene el **ACO**, todo esto ha sido explicado en el Capítulo 2.
2. Partiendo del primer paso (punto 1), se ha diseñado un algoritmo, que posteriormente ha sido implementado en el lenguaje C++ puesto que es compatible con Arduino y a la hora de transportar código de C++ a Arduino, se podrá reciclar en su mayoría ya que, Arduino está basado en C++.
3. Como tercer paso, una vez se entiende el funcionamiento del **ACO** y se ha implementado, vamos a planificar qué material necesitaremos para desarrollar el robot (sensores, actuadores, tablero...).
4. Cuarto, una vez tenemos los materiales procedemos a un montaje-prototipo sobre una protoboard, comprobando que los sensores y actuadores funcionan, previo diseño de esquema de interconexión de componentes.
5. Una vez tenemos claro el funcionamiento y que los sensores y actuadores elegidos para el robot funcionan, se procede a diseñar las **PCB**, para que el robot quede lo más compacto posible.
6. Se procede a la impresión de las **PCB** necesarias y, una vez impresas, el montaje de componentes sobre éstas, su posterior montaje en la estructura y montaje del tablero sobre el que la hormiga física (robot) circulará.
7. Volvemos a probar todos los sensores y actuadores utilizados sobre el robot integrados en el prototipo final.
8. Probamos que el robot circula correctamente a través del mapa.
9. Reciclamos el código desarrollado en el punto número 1 para su integración en Arduino, discriminando qué datos se generarán y procesarán de la propia Arduino y qué datos se enviarán desde la aplicación de escritorio.
10. Se desarrollará la aplicación cliente, para que sirvan de apoyo a la hormiga física, es decir, al robot.
11. Se desarrollará la aplicación servidor para abastecer y comunicar a los clientes.
12. Por último, se procede a la integración de todo el sistema.
13. Pruebas de funcionamiento.

A continuación se muestra un *diagrama de operaciones* de los puntos anteriormente comentados (en esta web [10] se muestra como realizar un diagrama de operaciones):

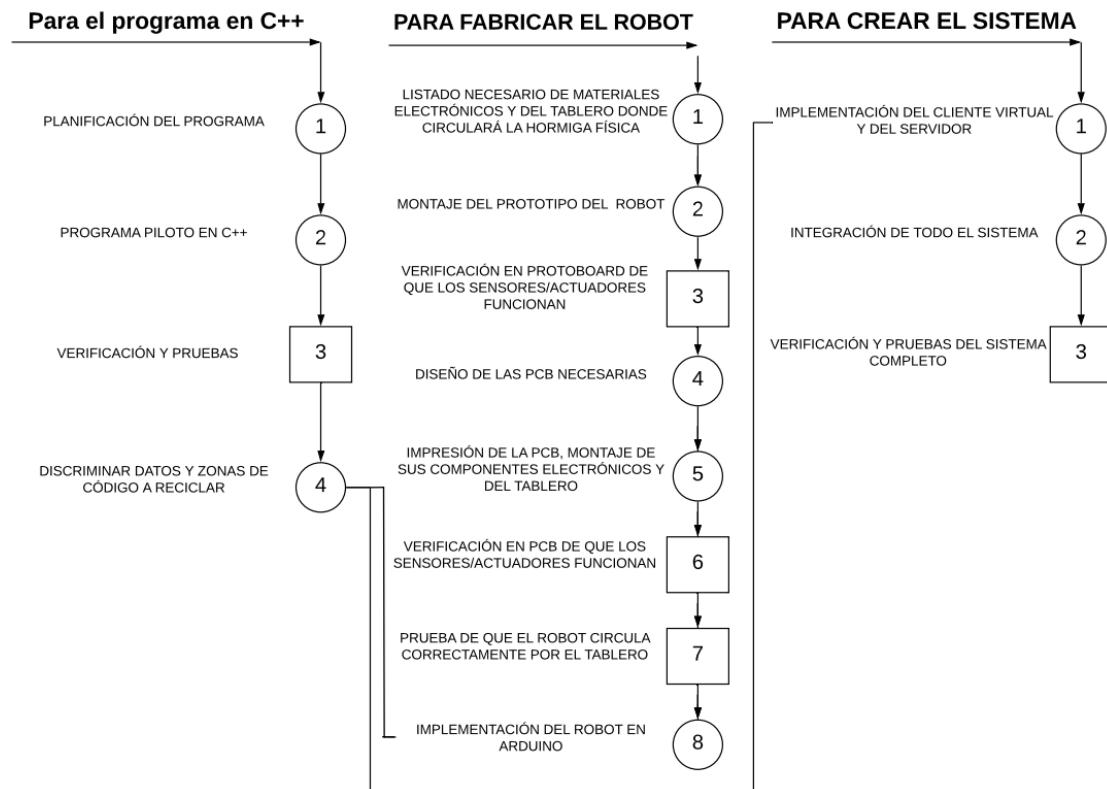


FIGURA 3.1: Diagrama de operaciones del proyecto

Una vez hemos hecho una vista general de la planificación del proyecto procedemos a explicar a fondo cada operación.

3.2. Implementación del Algoritmo de Colonias de Hormigas en C++

A continuación se muestra una posible implementación del ACO implementada en el lenguaje de programación C++, se detallarán todos los pasos seguidos hasta llegar a una ejecución exitosa del código.

Se ha seguido una metodología de desarrollo en Cascada para acabar implementando el código, a continuación se muestran los pasos seguidos que incluye una metodología de dicho tipo:

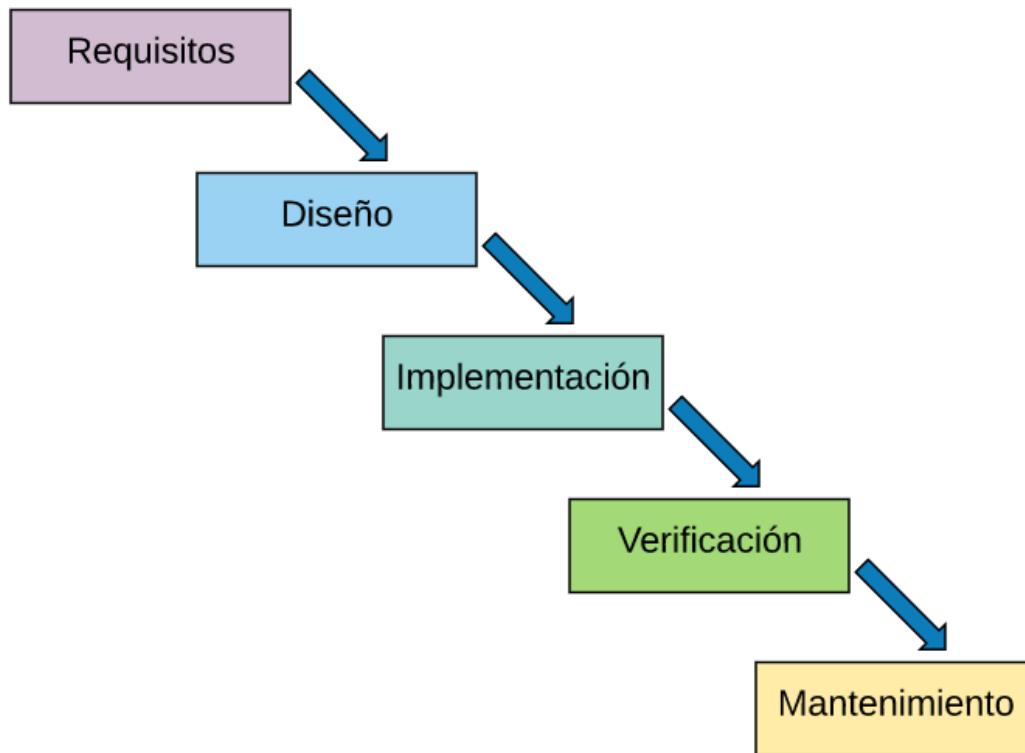


FIGURA 3.2: Pasos de la metodología de desarrollo en Cascada

3.2.1. Análisis de Requisitos

El primer paso para realizar un proyecto siguiendo una metodología de desarrollo en Cascada es realizar un análisis de requisitos, ¿Qué necesita el cliente?, puesto que estamos implementando directamente un algoritmo que ya ha sido desarrollado bastantes veces, simplemente tenemos que cumplir una serie de pautas preestablecidas:

- El algoritmo debe satisfacer las necesidades que se indican en la Sección 2.1.3.
- Debemos basar el algoritmo en las fórmulas indicadas en la Sección 2.1.3.2.
- Con este algoritmo se quiere solventar el problema de búsqueda de caminos en un grafo, simulando como una hormiga busca comida.

- Adicionalmente se busca poder reciclar la mayor parte de código que se implemente en este subproyecto, ya que éste es un prototipo.

Una vez especificado resumidamente qué necesita nuestro prototipo, pasamos a la fase de diseño como se indica en la Figura [3.2](#).

3.2.2. Diseño

El diseño de nuestro algoritmo se encuentra redactado en la Sección [2.1.3.1](#).

3.2.3. Implementación

A continuación, se procede a desarrollar el algoritmo previamente diseñado, el algoritmo se ha dividido en la siguiente jerarquía:

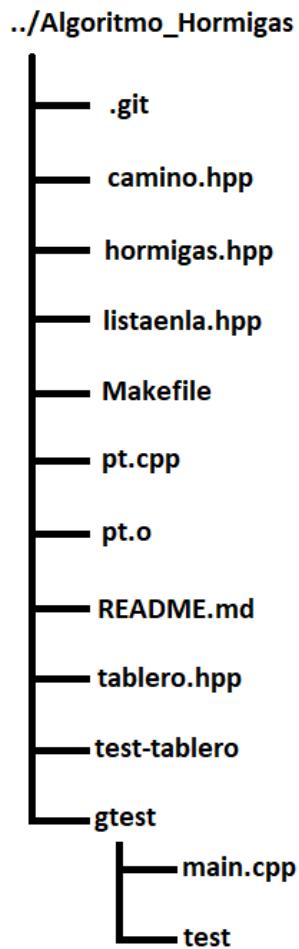


FIGURA 3.3: Jerarquía de archivos de la implementación del ACO

Vamos a detallar que contiene cada directorio o archivo:

- *.git*: Directorio que se genera al subir el archivo a Github, se puede acceder al proyecto mediante el siguiente enlace [11].
- *camino.hpp*: Contiene todo lo referente a los caminos que tome la hormiga así como de crear el mapa (Grafo). Podemos encontrarlo en el Anexo B.1.8.1.
- *hormigas.hpp*: Contiene las operaciones y atributos que definen una hormiga. Podemos encontrarlo en el Anexo B.1.8.1.
- *listaenla.hpp*: Se utiliza el *Tipo Abstracto de Datos*, a partir de ahora TAD, Lista enlazada, en diferentes clases, por ejemplo, para guardar el recorrido que ha tomado una hormiga para así derivar posteriormente su costo, en todos los casos se utiliza la Lista Enlazada

porque no se sabe cuántos nodos se van a almacenar en todas las clases en las que se usa. Podemos encontrarlo en el Anexo [B.1.8.1](#).

- *Makefile*: Script que sirve para compilar el código únicamente introduciendo el comando ”make” se genera automáticamente el archivo ejecutable, además del archivo objeto (”pt.o”) que sirve para enlazarlo al binario final que se ejecutará. En mi caso utilicé el compilador ”clang++”. Podemos encontrarlo en el Anexo [B.1.8.3](#).
- *pt.cpp*: Las iniciales del archivo ”pt” significan ”prueba tablero”, hace referencia al código principal de ejecución (código completo en el Anexo [B.1.8.2](#)), de este fichero podemos destacar como código significativo el siguiente fragmento:

```

while(!tab.esObjetivo(actual))
    actual = tab.siguienteNodo(actual, &h1);

actual = tab.inicial();
while(!tab.esObjetivo(actual))
    actual = tab.siguienteNodo(actual, &h2);

tab.actualizar_feromonas();

h1.volver();
h2.volver();

```

Donde podemos contemplar como la hormiga recorre el camino, el código es una implementación similar del pseudocódigo escrito en la Sección [2.1.3.1](#). Básicamente, indica que las hormigas ”h1” y ”h2” recorren el tablero hasta encontrar el objetivo y, una vez lo hacen, vuelven por el mismo camino exactamente que tomaron de ida, ya que me interesa medir y tomar conclusiones empíricas de la búsqueda de comida. La única diferencia del [ACO](#) con esta implementación es que a la vuelta realiza nuevamente el algoritmo mientras yo guardo el camino tomado en una Lista y para volver al hormiguero sigo dicho camino de manera inversa.

- *pt.o*: Archivo objeto del archivo fuente ”pt.cpp”, sirve como intermediario a la hora de enlazar el binario, este archivo lo genera el propio comando ”make” con las reglas indicadas en su archivo ”Makefile”, concretamente la regla ”test-tablero.o”.
- *README.md*: Archivo de texto plano que se muestra inicialmente en Github para describir tu proyecto.

- *tablero.hpp*: Este archivo es el más importante ya que enlaza casi todas las clases implementadas con la principal (“*pt.cpp*”). Podemos encontrar el código completo en el Anexo B.1.8.1. A continuación se deja escrita la especificación de las operaciones más relevantes de dicha clase:

```

/*Método para hacer avanzar una hormiga de un nodo al siguiente utilizando la fórmula de la probabilidad sobre la cuál
se elige uno arbitráreamente y condicionado según su probabilidad. Se usa la primera fórmula de la Sección Fórmulas.
PRECONDICIÓN: el nodo pasado por parámetro tiene al menos un adyacente. El nodo no es el nodo objetivo. La hormiga h
está recorriendo el tablero.
POSTCONDICIÓN: devuelve el siguiente nodo por el que pasará la hormiga*/
vertice siguienteNodo(vertice x, Hormiga<tCoste>* h);
/*Método para saber la distancia de un punto x a otro y, se utiliza la fórmula de distancia en la Sección Fórmulas
PRECONDICIÓN: ambos parámetros son adyacentes entre sí, puesto que el grafo es no dirigido, con el que el primer
parámetro sea adyacente al segundo nos vale
POSTCONDICIÓN: devuelve la distancia entre dos puntos*/
float distancia(vertice x, vertice y);
/*Método para saber si una hormiga específica ha salido del hormiguero en busca de comida
PRECONDICIÓN: -
POSTCONDICIÓN: devuelve verdadero si la hormiga existe en el tablero o falso en caso contrario*/
bool hormigaBuscaComida(const Hormiga<tCoste>* h) const;
/*Método para mandar a una nueva hormiga a buscar comida
PRECONDICIÓN: -
POSTCONDICIÓN: añade una hormiga nueva al Tablero y marca como visitado el nodo de inicio*/
void nuevaHormiga(Hormiga<tCoste>* h);
/*Método para saber si hemos alcanzado el objetivo
PRECONDICIÓN: -
POSTCONDICIÓN: devuelve true si hemos alcanzado el objetivo, false en caso contrario*/
bool esObjetivo(vertice i) const;
/*Método para saber cuál es el mejor camino en un instante de tiempo t
PRECONDICIÓN: -
POSTCONDICIÓN: devuelve un camino recorrido hasta el instante de tiempo en que se llama a la función y su coste*/
Camino<tCoste> mejorCaminoActual();

```

FIGURA 3.4: Especificación de las operaciones más relevantes de la clase Tablero contenidas en el archivo ”*tablero.hpp*”

- *test-tablero*: Archivo ejecutable que lanzará nuestra aplicación.
- *gtest*: Directorio en el que se encuentra todo lo referente a pruebas de código que se verán en la Sección 3.2.4.
- *gtest/main.cpp*: Archivo de código fuente donde se implementan las pruebas unitarias de código realizadas con el framework Gtest [12].
- *gtest/test*: Archivo ejecutable generado de compilar el archivo ”*gtest/main.cpp*”.

3.2.4. Verificación

Para verificar la mayoría del software, se ha seguido una práctica de desarrollo dirigido por test, o en inglés *Test-Driven Development*, que a partir de ahora llamaremos **TDD**, puede verse más información sobre esta práctica en el siguiente enlace [13].

Se han realizado test unitarios a los siguientes métodos que aparecen a continuación:

Archivo	Clase	Método
tablero.hpp	Tablero<T>	void nuevaHormiga (Hormiga<tCoste>*)
tablero.hpp	Tablero<T>	bool esObjetivo (vertice) const
tablero.hpp	Tablero<T>	vertice siguienteNodo (vertice, Hormiga<tCoste>*)
tablero.hpp	Tablero<T>	vertice inicial ()

TABLA 3.1: Métodos sobre los que se han realizado pruebas unitarias en la implementación en C++ del ACO.

Como podemos apreciar en la Tabla 3.1 todos los métodos pertenecen a la clase ”*Tablero*” que se encuentra en el archivo ”tablero.hpp” ya que, todo el ACO converge en estos métodos descritos que son una implementación directa del pseudocódigo descrito en la Sección 2.1.3.1.

A continuación se muestra la especificación de los métodos indicados en la Tabla 3.1, en las que basaremos nuestras pruebas unitarias.

- *void nuevaHormiga (Hormiga<tCoste>*)*
 - PRECONDICIÓN: -
 - POSTCONDICIÓN: añade una hormiga nueva al Tablero y marca como visitado el nodo de inicio.
- *bool esObjetivo(vertice) const*
 - PRECONDICIÓN: -
 - POSTCONDICIÓN: devuelve true si hemos alcanzado el objetivo, false en caso contrario.
- *vertice siguienteNodo (vertice, Hormiga<tCoste>*)*
 - PRECONDICIÓN: El nodo pasado por parámetro tiene al menos un adyacente. El nodo no es el nodo objetivo. La hormiga h está recorriendo el tablero.
 - POSTCONDICIÓN: devuelve el siguiente nodo por el que pasará la hormiga.
- *vertice inicial()*
 - PRECONDICIÓN: -

- POSTCONDICIÓN: Devuelve el vertice inicial.

Para las pruebas unitarias se ha utilizado el framework Gtest, para más información sobre este framework visite el repositorio oficial en [12].

A continuación vemos la salida mostrada:

```
juanma@buster:~/carrera/TFG/SI/Algoritmo_Hormigas/gtest$ ./test
[=====] Running 4 tests from 1 test case.
[-----] Global test environment set-up.
[-----] 4 tests from Tablero
[RUN    ] Tablero.nuevaHormiga
[OK     ] Tablero.nuevaHormiga (0 ms)
[RUN    ] Tablero.esObjetivo
[OK     ] Tablero.esObjetivo (0 ms)
[RUN    ] Tablero.siguienteNodo
[OK     ] Tablero.siguienteNodo (0 ms)
[RUN    ] Tablero.inicial
[OK     ] Tablero.inicial (0 ms)
[-----] 4 tests from Tablero (1 ms total)

[-----] Global test environment tear-down
[=====] 4 tests from 1 test case ran. (1 ms total)
[PASSED ] 4 tests.
```

FIGURA 3.5: Ejecución de tests unitarios realizados en Gtest

Las pruebas de integración se deducen a partir de las unitarias anteriormente realizadas puesto que como se dijo en párrafos anteriores en esta misma sección, todo el código converge a éstos métodos anteriormente citados.

En el punto 3.2.6 veremos una ejecución de prueba y podremos ver como todo se ejecuta correctamente.

El código fuente ejecutado en la Figura 3.5 se encuentra dentro del proyecto (mencionado en la Sección 3.2.3) en el directorio “*gtest*”, concretamente el archivo “*main.cpp*”.

3.2.5. Mantenimiento

Puesto que la aplicación en sí, es un prototipo su función es la de testear los resultados en las distintas ejecuciones, no hay necesidad de mantener el código, además dicho código será posteriormente diseccionado, reciclando zonas de código que se utilizarán para el robot en Arduino y para la aplicación servidora que hará de intermediaria entre las hormigas físicas y las hormigas virtuales, esto se verá más adelante.

3.2.6. Ejecución del Algoritmo de Colonias de Hormigas

A continuación se muestra una ejecución arbitraria del ACO, diseñado e implementado en la Sección 3.2.6.

```
juanma@buster:~/carrera/TFG/SI/Algoritmo_Hormigas$ ./test-tablero
*****GRAFO*****
  0   1   2   3   4   5   6   7
0 INF  INF  INF  INF  INF  INF  INF
1 INF  INF  5.00 3.10  INF  INF  5.20  INF
2 INF  5.00  INF  4.90  INF  INF  INF  5.20
3 INF  3.10  4.90  INF  INF  3.20  3.20  3.00
4 INF  INF  INF  INF  INF  5.50  INF  INF
5 INF  INF  INF  3.20  5.50  INF  4.70  INF
6 INF  5.20  INF  3.20  INF  4.70  INF  INF
7 INF  INF  5.20  3.00  4.80  INF  INF  INF

*****FEROMONAS*****
  0   1   2   3   4   5   6   7
0 0.095 0.090 0.090 0.090 0.090 0.090 0.090 0.090
1 0.090 0.095 0.090 0.404 0.090 0.090 0.522 0.090
2 0.090 0.090 0.095 0.090 0.090 0.090 0.090 0.090
3 0.090 0.404 0.090 0.095 0.090 0.136 0.090 0.358
4 0.090 0.090 0.090 0.090 0.095 0.568 0.090 0.358
5 0.090 0.090 0.090 0.136 0.568 0.095 0.613 0.090
6 0.090 0.522 0.090 0.090 0.090 0.613 0.095 0.090
7 0.090 0.090 0.090 0.358 0.358 0.090 0.090 0.095

Camino recorrido: 1 -> 6 -> 5 -> 4
Coste del camino: 15.400
```

FIGURA 3.6: Ejecución del ACO implementado en C++.

La ejecución y las pruebas que se hagan hasta la fase final se realizarán sobre el tablero de ejemplo de la Figura 2.4, por tanto la ejecución de la Figura 3.6 corresponde con el grafo de la Figura 2.4.

En la sección inferior de la Figura 3.6 podemos ver cómo han quedado las feromonas tras 5 iteraciones (dato sacado del código fuente del propio programa) de donde se deduce el siguiente mapa de calor.

FEROMONAS TRAS 5 ITERACIONES CON 2 HORMIGAS POR ITERACIÓN

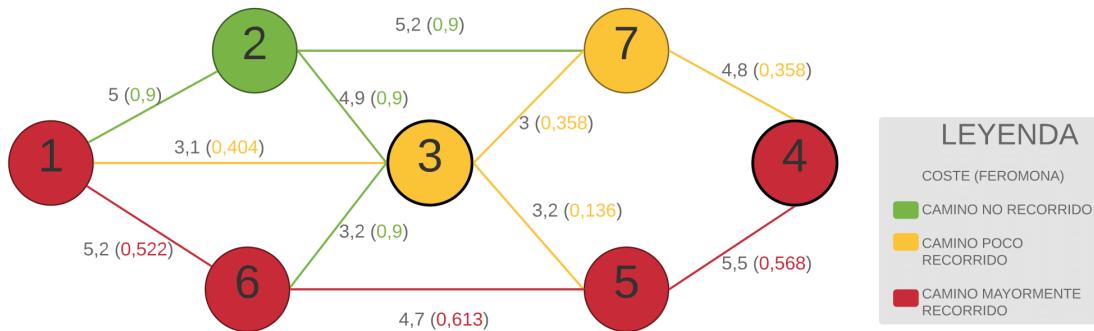


FIGURA 3.7: Mapa de calor post-ejecución del ACO.

Para estudiar el comportamiento del ACO se han contrastado distintos tipos de ejecuciones que se explican detalladamente en la Sección 4.2.

3.3. Fabricación y montaje de la hormiga

En la siguiente sección, veremos el procedimiento de montaje del robot, así como la necesidad de cada sensor o actuador utilizado.

La sección se estructurará en las siguientes subsecciones que marcan cada paso seguido para montar el robot u hormiga física:

1. *Finalidad del robot*, Sección 3.3.1.
2. *Electrónica necesaria*, Sección 3.3.2.
3. *Interconexión de componentes electrónicos*, Sección 3.3.4.
4. *Estructura física del robot*, Sección 3.3.5.
5. *Montaje*, Sección 3.3.6.

6. *Pruebas*, Sección [3.3.7](#).

Una vez enumerados los pasos procedemos a su desarrollo.

3.3.1. Finalidad del robot

Partimos de la necesidad de un robot físico que realice el [ACO](#) como bien se indica en el alcance del proyecto.

La finalidad de este robot es ir de un punto de partida, a un punto de destino, haciendo símil con el ámbito de las hormigas el punto de partida podría ser el hormiguero y el punto de destino la comida, donde el procedimiento del [ACO](#) es la búsqueda de comida.

Por tanto, necesitamos un robot que se mueva de un punto a otro, detecte en qué punto está, inicio, intermedio o destino, procese una información y, en este caso, se comunique con el resto de hormigas.

La comunicación se realizará de manera centralizada y no distribuida como lo hacen las hormigas realmente, y por tanto el [ACO](#), ya que necesitaríamos un hardware más potente, caro y que necesitaría una mayor alimentación.

Esto supone una peculiaridad en el proyecto puesto que se hace una aportación a la ciencia ya que no se han encontrado estudios sobre la realización del [ACO](#) centralizado, es decir, que toda la información de cada hormiga pase por un punto de información compartida.

Creada la necesidad de un robot con las características anteriormente contadas procedemos a satisfacer dichas necesidades mediante el uso de la electrónica, informática y diseño que forman, la robótica.

3.3.2. Electrónica necesaria

A continuación se indican las necesidades indicadas en la Sección [3.3.1](#) y el componente electrónico que satisface dicha necesidad.

Para el movimiento utilizaremos motores de corriente continua, también conocidos como motores **DC** por sus siglas en inglés *Direct Current*, mínimo utilizaremos dos, además de sensores ópticos infrarrojos, dos también, ya que servirán como base de nuestro sigue línea, para esto nos basamos en la Sección 3.4, donde se indica el tipo de tablero sobre el que el robot se desplazará.

Para saber en qué punto del mapa está, si en el hormiguero o punto de inicio, punto intermedio o en la fuente de alimento o destino, se ha utilizado un sensor de color ya que, cada punto se encontrará identificado con un color como se indicará en la Sección 3.4.

Al necesitar comunicarse el robot, podemos discriminar entre una comunicación cableada o inalámbrica, nos decantamos por una inalámbrica como se indicará en la Sección 3.3.2.5, concretamente utilizaremos un módulo Bluetooth.

Obviamente, necesitaremos procesar información, para ello hemos hecho uso de una placa Arduino, concretamente una Arduino Nano, se discutirá también por qué en la Sección 3.3.2.4.

Al elegir una solución inalámbrica no tendría sentido alimentar por cable al robot, con lo cual hemos utilizado una batería Li-Po [14].

Por último, pero no menos importante, para mover los motores necesitaremos una controladora ya que los pines de Arduino no ofrecen la suficiente potencia para mover los dos motores **DC** directamente conectados a ésta.

En la siguiente tabla se sintetiza lo descrito anteriormente en esta sección:

Necesidad	Solución	Justificación	Cantidad
Movimiento	motor DC	Sección 3.3.2.1	2
	Sensor óptico infrarrojo	Sección 3.3.2.2	2
Posición	Sensor de colores	Sección 3.3.2.3	1
Procesamiento Información	Arduino Nano	Sección 3.3.2.4	1
Comunicación	Módulo Bluetooth	Sección 3.3.2.5	1
Alimentación	Batería Li-Po	Sección 3.3.2.6	1

TABLA 3.2: Listado de componentes electrónicos necesarios para la construcción del robot.

A continuación discutiremos por qué hemos utilizado los componentes mencionados en la Tabla 3.2 en detalle y qué modelo concreto del componente se ha usado.

3.3.2.1. Motor de corriente continua

Incuestionablemente, necesitaremos dos motores DC ya que, necesitamos que el robot ande, estos motores lo moverán, concretamente se han utilizado los motores de marca *SODIAL*, "SODIAL(R) DC 3V 0.3A 15RPM con reductora", al tener reductora, el robot se moverá mas lento de lo que lo haría sin reductora pero ganamos en fuerza.

Son motores pequeños pero que tienen mucha fuerza y poco consumo, ideales para mover nuestro robot. A continuación se muestra una imagen de los motores utilizados.



FIGURA 3.8: Motor DC marca SODIAL, 3V a 0.3A y 15RPM con reductora.

Adicionalmente, para mover estos motores, como se comentó anteriormente, necesitamos una controladora ya que nuestro Arduino no tendrá suficiente potencia para realizar dicha acción.

Se ha utilizado concretamente la controladora de motor modelo *L298N* puesto que contiene 2 puentes H (componente electrónico que sirve para mover en ambas direcciones los motores [15]) para controlar ambos motores, a continuación se muestra una imagen:

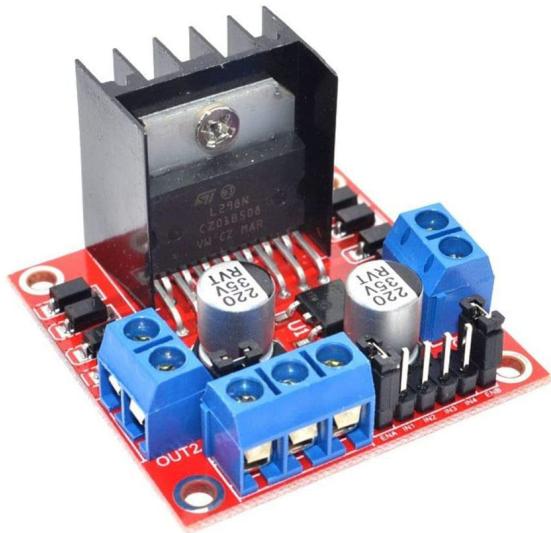


FIGURA 3.9: Controladora de motor L298N.

3.3.2.2. Sensor óptico infrarrojo

Puesto que el mapa que se ha elegido es un grafo físicamente plasmado en un tablero como se detallará en la Sección 3.4, y, como sabemos, un grafo esta compuesto por nodos interconectados a través de aristas, estas aristas crean la necesidad de utilizar el sensor óptico infrarrojo.

Esto se debe a que las aristas serán tiras de color negro sobre un fondo de tablero blanco con lo que nuestro robot seguirá dichas aristas utilizando dos sensores ópticos infrarrojos, los cuales indicarán si leen blanco o negro, para enderezar el robot a seguir dicha arista, coloquialmente a esto se le conoce como un robot "siguelínea [16]" .

Concretamente, hemos utilizados los sensores "CNY70" del fabricante "Red.Planet". A continuación se muestra una foto del sensor óptico infrarrojo "CNY70".



FIGURA 3.10: Sensor óptico infrarrojo CNY70.

3.3.2.3. Sensor de colores

El sensor de color, se utiliza para saber en que nodo concretamente, se encuentra el robot. Puesto que el mapa que se utiliza es un grafo plasmado en un tablero, como se detalla en la Sección 3.4, y como sabemos un grafo son un conjunto de nodos interconectados por aristas, los nodos serán puntos de colores sobre los que el robot se parará y procesará la información en función de qué color ha leído.

Los colores que lee nuestro sensor de colores son el rojo, el verde y el azul, con lo cuál se les dará un significado a cada uno que pueden ser o bien hormiguero o punto de inicio, nodo intermedio, o fuente de alimento o destino, esto se concretará en la Sección 3.4.

El sensor de colores que se utiliza en el proyecto, es el "TSC-3200" de la marca "AZDelivery". A continuación se muestra una imagen de dicho sensor.



FIGURA 3.11: Sensor de colores TSC-3200.

3.3.2.4. Arduino Nano

Para procesar la información recibida de los sensores e interactuar con un actuador para que se realice una acción física, se han contemplado las siguientes opciones:

- Raspberry Pi (Web Oficial [17]).
- Arduino (Web Oficial [18]).

Para discriminar entre una u otra me he basado en las siguientes características:

- Tamaño del microcontrolador.
- Consumo del microcontrolador.
- Capacidad mínima de procesamiento del microcontrolador.
- Precio del microcontrolador.
- Trece pines GPIO[19] o más.

En la siguiente tabla se muestra una comparativa de qué microcontrolador cumple con las características anteriormente enumeradas:

Microcontrolador	Tamaño	Consumo (MÁX.)	Procesamiento	Precio	Trece o mas pines GPIO
<i>Raspberry Pi</i>	No Apto	Apto	Apto	No Apto	Apto
<i>Arduino</i>	Apto	Apto	Apto	Óptimo	Apto

TABLA 3.3: Cumplimiento de características.

Como podemos apreciar en la Tabla 3.3, los microcontroladores de *Raspberry*, aunque de procesamiento son óptimos, en cuanto a consumo se trata necesitarían una batería demasiado grande, aun así vamos a debatir punto a punto los pros y contras de utilizar cada uno.

Partimos de que el tamaño es esencial ya que queremos que el robot sea lo más pequeño posible para no tener que hacer un tablero muy grande, en base a esto, en ambos microcontroladores contemplamos las versiones más pequeñas de estos, a saber:

- Raspberry Pi Zero[20].
- Arduino Nano[21].

A continuación se muestran fotos de ambos microcontroladores:

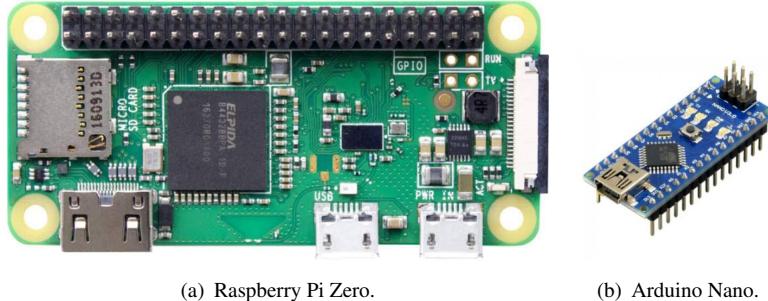


FIGURA 3.12: Microcontroladores candidatos, Raspberry Pi Zero y Arduino Nano.

Procedemos a debatir cada punto para elegir alguno de los dos microcontroladores.

En cuanto a **tamaño**, la *Arduino Nano* mide $4.5\text{cm} \times 1.8\text{cm}$ [22] mientras que la *Raspberry Pi Zero* cuenta con unas dimesniones de $6.5\text{cm} \times 3\text{cm}$ [20] empeorando la marca de la *Arduino Nano*.

El siguiente punto a tratar es el del **consumo**, se busca el mínimo consumo posible, donde podemos decir que en este punto *Raspberry Pi Zero* supera con un consumo máximo de 5V a 0.24A (1.2W)[23] a *Arduino Nano* con un consumo máximo de 5V a 0.5A (2.5W)[24] aunque ambos consumos siguen siendo aptos para nuestro proyecto.

Este punto es bastante importante, y trata del **procesamiento**, siendo el de *Arduino Nano* con el ATmega328P [25], un core a 16MHz [25] tremadamente inferior al de la *Raspberry Pi Zero* que cuenta con un procesador a 1GHz[20]. Concretamente, de la elección del procesador derivamos la estructura del programa, ya que al necesitar más capacidad de procesamiento, que podría darnos una *Raspberry Pi Zero*, debemos apoyarnos en un procesador con más potencia y en este punto del proyecto se decide crear un sistema centralizado, donde el procesador principal es el de un programa de escritorio que se ejecute sobre un PC, haciendo así una aportación a la ciencia ya que, no se ha documentado absolutamente nada sobre un **ACO** que utilice un nodo central para intercambiar información con otros, este sistema es puramente distribuido.

En cuanto al **precio**, ha sido la elección principal puesto que una *Raspberry Pi Zero* cuesta 29.63€[26], pero adicionalmente, la *Raspberry Pi Zero* necesita una tarjeta SD, que actúe como disco duro que cuesta en torno a 5.90€[27], con lo cuál si sumamos ambas cantidades nos queda que la *Raspberry Pi Zero* cuesta unos 35.53€ frente a la *Arduino Nano* que cuesta unos 3.60€ por unidad [28].

Por último, el **número de pines GPIO**, es esencial, pero ambos microcontroladores son aptos como podemos apreciar, la *Raspberry Pi Zero* cuenta con 40 pines GPIO [20] mientras que la *Arduino Nano* tiene 14 pines GPIO [22].

A continuación se sintetizan las especificaciones comentadas anteriormente.

Microcontrolador	Tamaño (cm)	Consumo (MÁX.)	Procesamiento	Precio	Trece o mas pines GPIO
<i>Raspberry Pi Zero</i>	6.5 x 3	0.24A - 5V	1 core - 1GHz	35.53€	40
<i>Arduino Nano</i>	4.5 x 1.8	0.5V - 5V	1 core - 16MHz	3.60€	14

TABLA 3.4: Características específicas del microcontrolador.

3.3.2.5. Módulo Bluetooth

Para realizar la comunicación inalámbrica, necesaria para que todas las hormigas se comuniquen entre sí, tenemos varias posibilidades:

- Wi-Fi (IEEE 802.11) [29].
- Bluetooth (IEEE 802.15.1) [30].
- Zigbee (IEEE 802.15.4) [31].

Para elegir una de las tres opciones anteriormente enumeradas me he basado en las siguientes características:

- Cobertura.
- Consumo.

- Complejidad de uso.
- Precio.

Tecnología Inalámbrica	Cobertura	Consumo	Complejidad de uso	Precio
<i>WiFi</i>	Óptimo	No apto	Viable	Apto
<i>Bluetooth</i>	Apto	Apto	Óptima	Apto
<i>Zigbee</i>	Apto	Óptimo	Muy complejo	No Apto

TABLA 3.5: Características específicas.

Ahora vamos a detallar cada elemento de la Tabla 3.5.

La **cobertura** es un punto esencial ya que, si no hay cobertura el robot no se puede comunicar con el PC que es el siguiente nodo de contacto ya que todo el tráfico pasa por este nodo. Este punto, lo cumplen los tres estándares ya que el robot estará cerca del PC. Se resalta el Wifi ya que tiene un alcance mayor.

El **consumo** es un punto importante puesto que cuanto menos consuman los componentes electrónicos más autonomía tendrá el robot. En este punto tenemos que hacer hincapié en las tres tecnologías:

- *WiFi*: Se utilizaría el módulo Wifi "ESP8266-01", el cual puede llegar a tener un consumo máximo de 3.3 V a 0.2 A lo cuál es excesivo si lo comparamos con las otras dos tecnologías [32].
- *Bluetooth*: Si nos decantamos por el Bluetooth, tendríamos un consumo máximo de 3.3 V a 0.04 A lo cuál es mucho más bajo que si utilizaramos Wifi. [31]
- *Zigbee*: Si utilizamos Zigbee, tendríamos aún un menor consumo que en Bluetooth, 3.3V a 0.03 A de consumo máximo.[31].

La **complejidad de uso** de cada tecnología es un punto importante puesto que no es igual conectar el módulo y comunicarnos a través de una librería directamente, que tener que desarrollar o utilizar comandos. En este punto, Bluetooth es el más cómodo puesto que al comunicarlos por

los pines disponibles para comunicación serie y utilizando la biblioteca que proporciona arduino "SoftwareSerial" podemos comunicarnos mediante dicho protocolo. Si utilizamos Wifi, tenemos que basarnos en el uso de "comandos AT [33]" lo cuál nos dificulta un poco más que el Bluetooth y por último, Zigbee no ofrece una biblioteca en Arduino como tal, tendríamos que crear una y tener un receptor que trabajase a la misma frecuencia para comunicarnos, cosa que un PC común no trae de fábrica mientras que Wifi y Bluetooth, sí.

En cuanto a **precio**, Wifi y Bluetooth son económicos mientras que Zigbee se dispara. Se muestran los precios de cada uno de los módulos:

- *Wifi*: 5.29€[34].
- *Bluetooth*: 9.99€[35].
- *Zigbee*: 59.99€[36].

Como podemos apreciar, los precios hablan por sí solos.

El módulo Bluetooth utilizado es el HC-05 [37], a continuación se muestra una imagen del mismo:



FIGURA 3.13: Módulo Bluetooth HC-05.

3.3.2.6. Batería Li-Po

La batería elegida tiene un voltaje de 7.4V a 2200mAh, se ha utilizado una batería de *Litio y Polímero*, a partir de ahora *LiPo*. Se ha calculado la autonomía de ésta batería en función del consumo del circuito utilizado, el circuito se midió directamente con un multímetro como se muestra en la Figura 3.14.

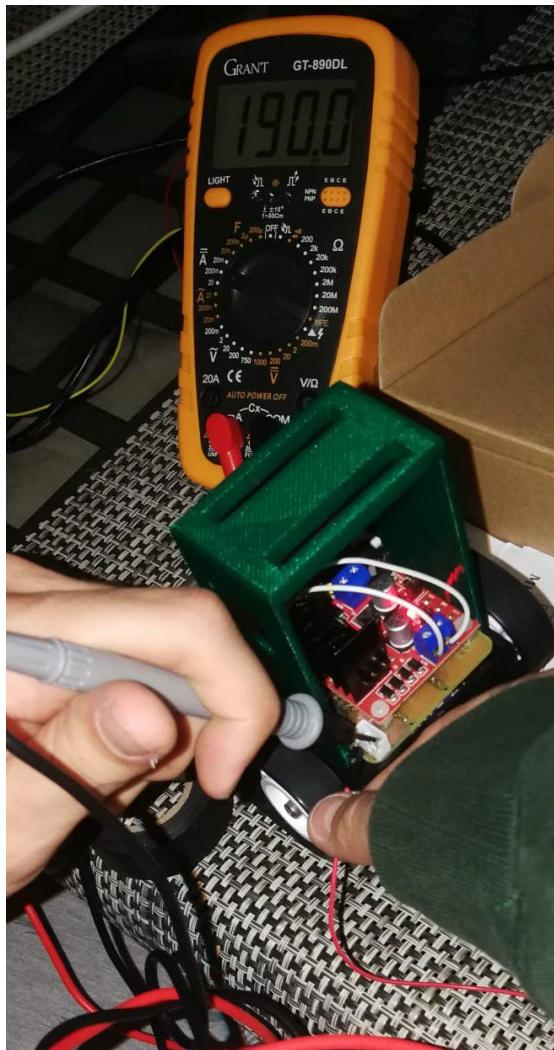


FIGURA 3.14: Medición de la Intensidad del robot.

Se ha conectado el "GND" de la batería directamente a la placa base en su pin correspondiente y la línea de 7.4V de la batería se ha conectado a través del multímetro, se ha ejecutado un programa de prueba donde se han activado todos los sensores de manera concurrente y se ha realizado la medición la cuál da una media de 190mA.

Teniendo en cuenta que el circuito tiene de consumo máximo una media de 190mA, y que nuestra batería es de 2200mAh, quiere decir que la batería funcionando constantemente ofrece 2200mA en una hora antes de agotarse, por tanto para saber la autonomía del robot conectada a la batería dividimos los 2200mAh entre los 190mA:

$$\blacksquare \frac{2200}{190} = 11,57h$$

Que si convertimos los 0.57 h a minutos con una simple regla de tres, obtenemos 34 minutos aproximadamente. Por tanto, el robot conectado a la batería totalmente cargada y a pleno rendimiento tendría una autonomía de **11 horas y 34 minutos**.

El modelo de batería utilizado concretamente lo podemos encontrar en [38] y se muestra en la Figura 3.15 con un precio de 25.99€.



FIGURA 3.15: Batería LiPo 7.4V a 2200mAh.

3.3.3. Hojas de características (Datasheets)

A continuación se muestran las hojas de características de cada uno de los componentes utilizados.

Componente	Datasheet
Motor DC	[39]
L298N	[40]
CNY70	[41]
TSC3200	[42]
Arduino Nano	[43]
HC-05 Bluetooth	[44]
Batería LiPo	[38]

TABLA 3.6: Hojas de características de cada componente electrónico.

3.3.4. Interconexión de componentes electrónicos

A continuación se describe la manera en que se han interconectado todos los componentes anteriormente descritos, puesto que se busca que el tamaño del robot sea lo más pequeño posible, se han diseñado concretamente dos **PCB** para que todo quede conectado entre sí.

Antes de entrar en detalle en el diseño de cada una de las **PCB** daremos una definición citada en [45].

"En electrónica, una placa de circuito impreso es una superficie constituida por caminos, pistas o buses de material conductor laminadas sobre una base no conductora. El circuito impreso se utiliza para conectar eléctricamente a través de las pistas conductoras, y sostener mecánicamente, por medio de la base, un conjunto de componentes electrónicos. Las pistas son generalmente de cobre, mientras que la base se fabrica generalmente de resinas de fibra de vidrio reforzada, cerámica, plástico [...]"

En las siguientes Secciones se entra en detalle en cada una de las **PCB** las cuáles se enumeran y comentan a continuación:

- La primera **PCB** que se describe, se utiliza para interconectar los sensores *CNY70* descritos en la Sección 3.3.2.2.
- La segunda **PCB** se utiliza para interconectar todos los componentes del robot, podríamos denominar que es una placa base haciendo símil con las placas base de los ordenadores que interconectan todos los periféricos y componentes.

3.3.4.1. Placa de circuito impreso para sensores ópticos infrarrojos

Para interconectar los sensores ópticos infrarrojos (*CNY70*) 3.3.2.2, se ha diseñado y fabricado una **PCB** que los interconecta a la **PCB** descrita en 3.3.4.2 y a su vez con la unidad central de procesamiento que en este caso es una *Arduino* descrita en 3.3.2.4. A continuación se muestra el esquemático de esta **PCB**:

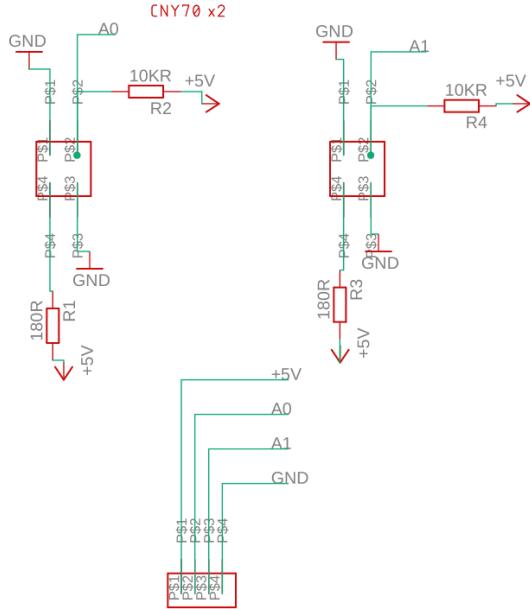


FIGURA 3.16: Esquemático de interconexión para sensores ópticos infrarrojos.

Como podemos observar los pines analógicos de Arduino *A0* y *A1* serán finalmente quien capturen la información emitida de estos sensores, se adelanta que, posteriormente en el circuito impreso de interconexión de todos los componentes hubo varias erratas, y una de estas fue que no se conectó la salida del pin *A0* y *A1* a dichos sensores sino que en su lugar, éstos fueron conectados a los pines *A1* y *A2*.

Para la elección de las resistencias nos hemos basado en la *ley de Ohm* [46] cuya fórmula es $Voltaje(V) = Intensidad(I) \times Resistencia(R)$, luego si tenemos el voltaje y la intensidad, podemos deducir la resistencia que necesitamos así que vamos a ver cuál es el voltaje e intensidad de cada patilla del *CNY70*, a continuación se muestra una imagen sacada de la hoja de características de este sensor [41].

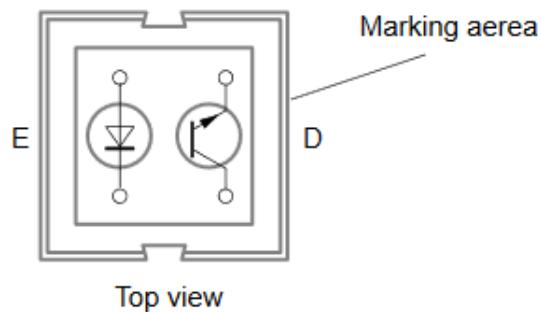


FIGURA 3.17: CNY70 internamente.

Como podemos observar en la Figura 3.17 tenemos un lado marcado con una "E", que significa *Emisor* donde se emite luz, la cuál será receptada por el *Detector* el cuál se identifica con la letra "D".

Para conseguir una distancia de reflexión aceptable debemos conseguir que el diodo led alumbe lo máximo sin dañarlo, para esto, como bien especifica la hoja de características del "CNY70" [41], podemos dar una intensidad máxima de 50mA, la cuál podría dañar el diodo si nos pasamos, por tanto, le daremos una de 20mA, donde tendremos una distancia de reflexión bastante buena, de donde se deduce que aplicando la *ley de Ohm* [46]:

- $I = 0,02A$
- $V = 5V - 1,25V = 3,75V$ de donde 1,25V es el voltaje de caída producido por el diodo del sensor "CNY70" y 5V el voltaje de entrada.

$$R = \frac{V}{I} = \frac{3,75V}{0,02A} = 187,5 \Omega$$

Para proteger el foto transistor colocaremos una resistencia de $10k\Omega$ teniendo esto en cuenta y que Arduino ofrece un voltaje de 5V, que restando el voltaje de caída producido por el foto transistor de 0,3V nos queda una entrada de 4,7V, aplicando la *ley de Ohm*:

$$I = \frac{V}{R} = \frac{4,7V}{10k\Omega} = 47mA$$

47mA está por debajo de los 100mA de corriente máxima permitida por el foto transistor como podemos observar en la hoja de características del sensor "CNY70" [41].

Finalmente nos queda el siguiente diseño para acoplar ambos sensores *CNY70*:

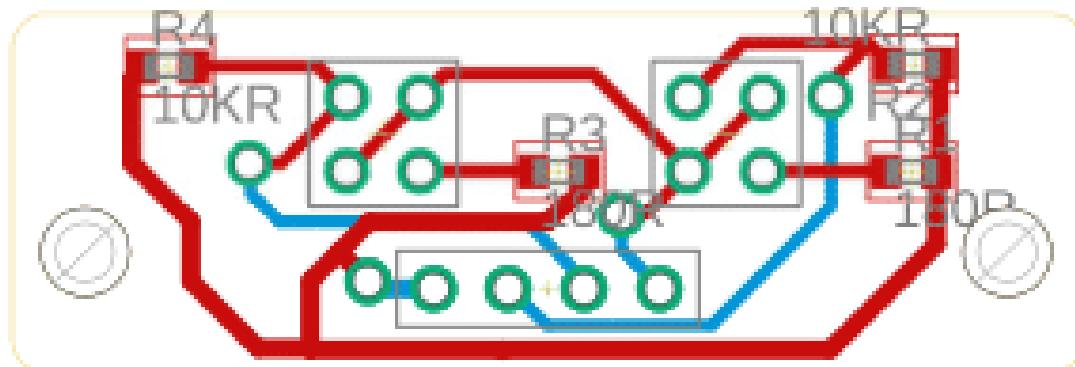
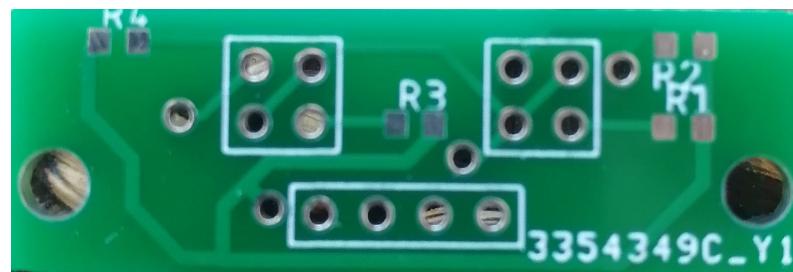


FIGURA 3.18: Diseño de la **PCB** para acoplar los sensores *CNY70*.

La **PCB** de la Figura 3.18 tiene unas dimensiones de 36mm x 12mm (largo x ancho). Aquí podemos encontrar el circuito ya impreso, y posteriormente montado.



(a) Cara top del diseño de la **PCB** de la Figura 3.18.



(b) Cara top del diseño de la **PCB** de la Figura 3.18 montada.

FIGURA 3.19: Cara top del circuito impreso diseñado en la Figura 3.18.

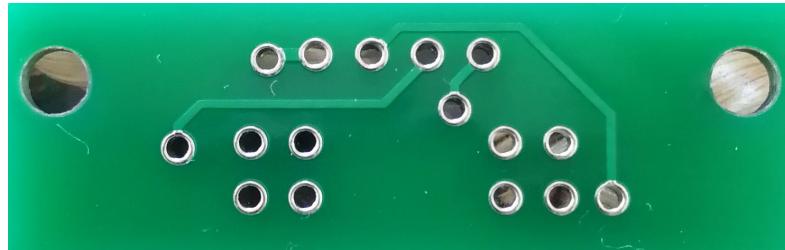
(a) Cara bottom del diseño de la [PCB](#) de la Figura 3.18.(b) Cara bottom del diseño de la [PCB](#) de la Figura 3.18 montada.

FIGURA 3.20: Cara bottom del circuito impreso diseñado en la Figura 3.18.

3.3.4.2. Placa de circuito impreso para interconexión del robot

En segundo lugar, tenemos la [PCB](#) que interconecta todo los componentes que si recordamos son los que se listan a continuación:

- Arduino nano.
- L298N.
- Módulo Bluetooth.
- Batería [LiPo](#).
- Sensor *TSC3200*.
- [PCB](#) descrita en la Sección 3.3.4.1, la cuál contiene ambos *CNY70*.

A continuación se muestra el esquemático seguido para interconectar todos los componentes:

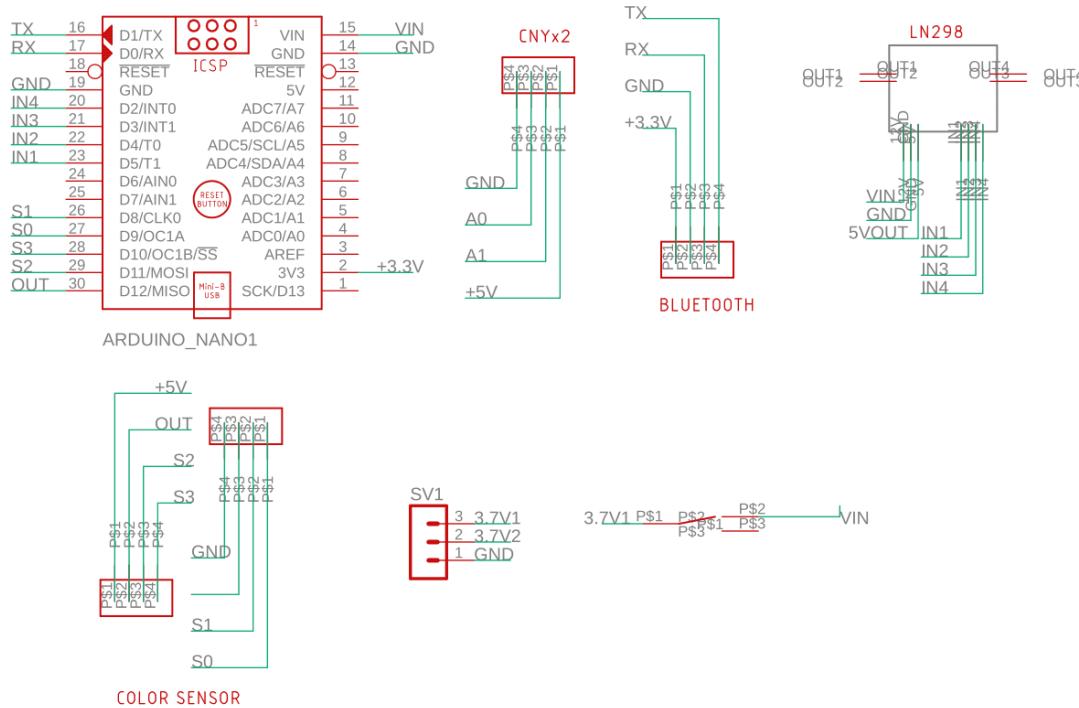


FIGURA 3.21: Esquemático de la [PCB](#) de interconexión de todos los componentes.

Como podemos apreciar en la Figura 3.21 todos los componentes están debidamente etiquetados, se sabe cuál es cada sensor y a qué pin de Arduino está conectado, exceptuando la errata comentada en la Sección 3.3.4.1, donde los pines analógicos utilizados son el "A1" y "A2" en lugar del "A0" y "A1" que si podemos apreciar no están interconectados a la Arduino, esto se ha solventado soldando dos cables de las líneas correspondientes "A1" y "A2" a los pines del conector "CNYx2" mostrado en la Figura 3.21. Como corrección que se ha realizado, hay que unir las líneas comentadas en este párrafo.

También podemos observar en la Figura 3.21 hay un pin cuyo nombre es "SV1", puesto que el nombre no es muy orientativo, este pin corresponde a la entrada de la batería, por donde se alimentará el sistema.

Y por último la línea que no está etiquetada en la Figura 3.21 corresponde a un interruptor colocado para alimentar a través de la batería el sistema o por el contrario que no pase dicha corriente. La corriente cuando el interruptor esté activo entrará a la Arduino a través del pin "VIN" el cuál internamente, como podemos apreciar en el esquemático de Arduino [43], contiene su propio regulador de tensión a 5V ya que alimentamos con una batería de 7,4V.

Dicho esto, pasamos a mostrar el diseño del circuito descrito en el esquemático de la Figura 3.21:

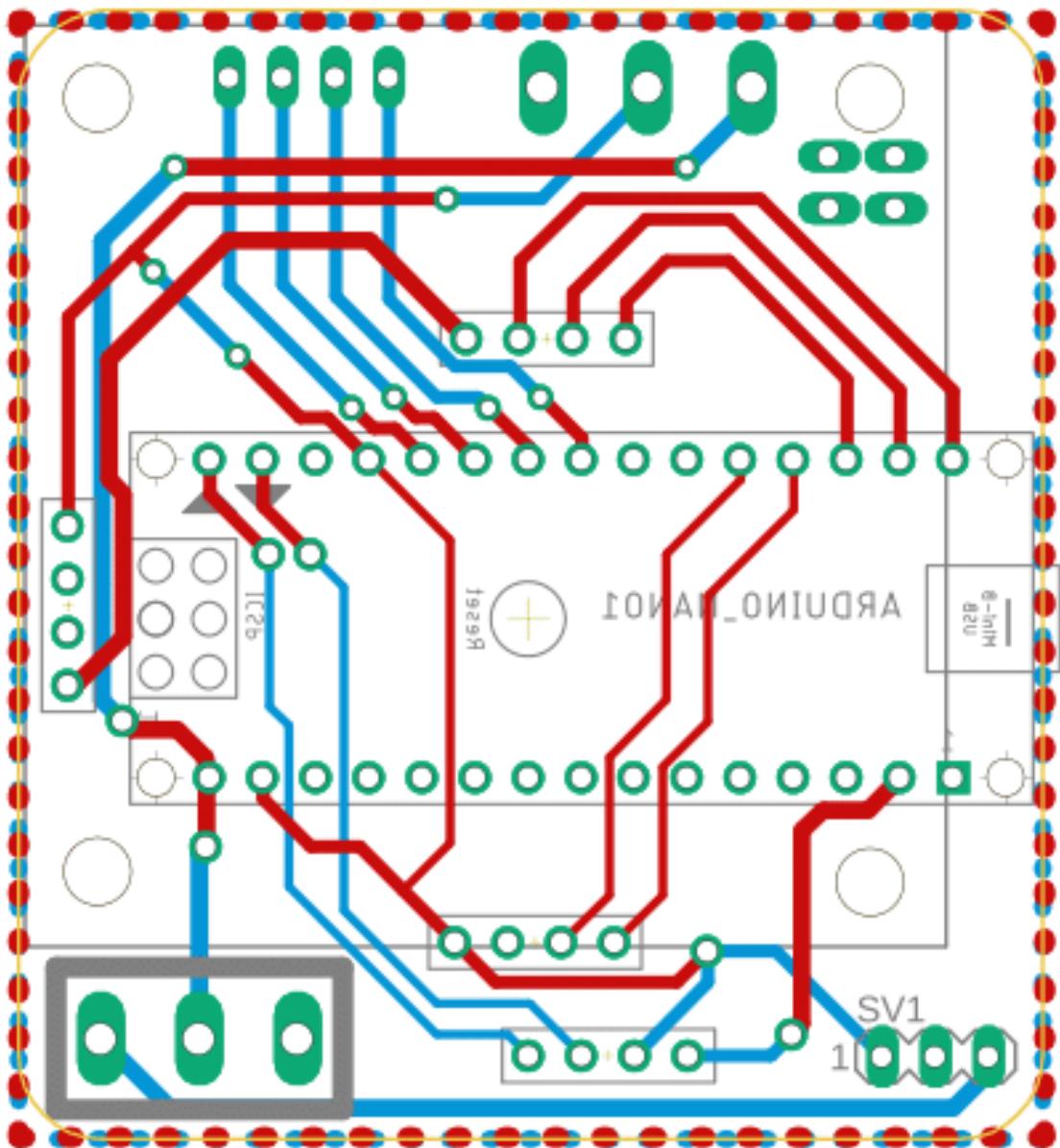
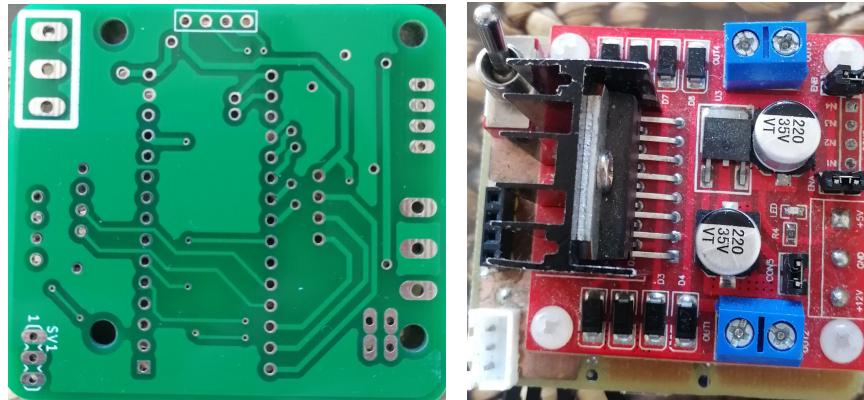


FIGURA 3.22: Diseño de la PCB de interconexión de todos los componentes.

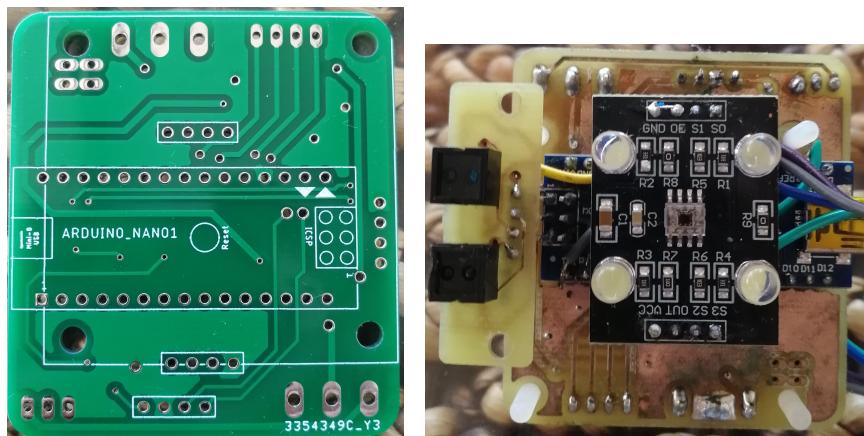
En la Figura 3.22 podemos apreciar otra errata donde la línea de 5V que va del conector del sensor de color al conector del sensor óptico infrarrojo no va conectada a la línea de 5V de la Arduino y por tanto no se estaba alimentando correctamente a los componentes de ahí su funcionamiento anómalo. Esto se solucionó soldando un cable del conector del sensor de color al pin de 5V de Arduino.

Las dimensiones de la **PCB** de la Figura 3.22 son 49mm x 54mm (largo x ancho). A continuación podemos ver el resultado al imprimir y montar el diseño de la Figura 3.22.



(a) Cara top del diseño de la **PCB** de la Figura 3.22. (b) Cara top del diseño de la **PCB** de la Figura 3.22 montada.

FIGURA 3.23: Cara top del circuito impreso diseñado en la Figura 3.22.



(a) Cara bottom del diseño de la **PCB** de la Figura 3.22. (b) Cara bottom del diseño de la **PCB** de la Figura 3.22 montada.

FIGURA 3.24: Cara bottom del circuito impreso diseñado en la Figura 3.22.

3.3.5. Estructura física del robot

En esta Sección se muestran las estructuras seleccionadas donde introducir la electrónica utilizada en el proyecto, el montaje de dicha estructura se describe en la Sección 3.3.6.

Como estructura física del robot se ha elegido un chasis prefabricado, combinado con una pieza impresa en 3D como soporte para la batería.

Dividiremos esta Sección en dos:

- Chasis utilizado como base para interactuar con el Hardware.
- Soporte impreso en 3D para la batería.

3.3.5.1. Chasis para el robot

Como chasis, se ha elegido uno prefabricado debido a la complejidad que conlleva fabricar uno, concretamente podemos encontrarlo en [47].

A continuación se muestra una imagen de como es el chasis montado:

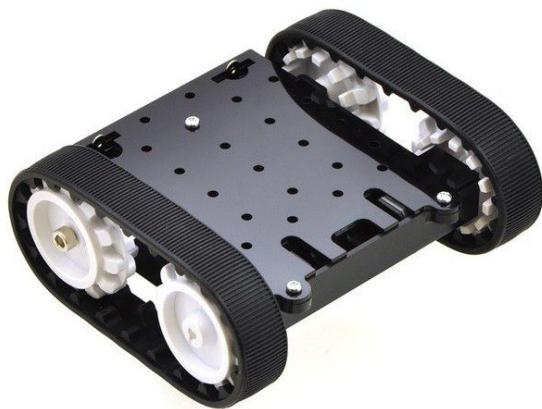
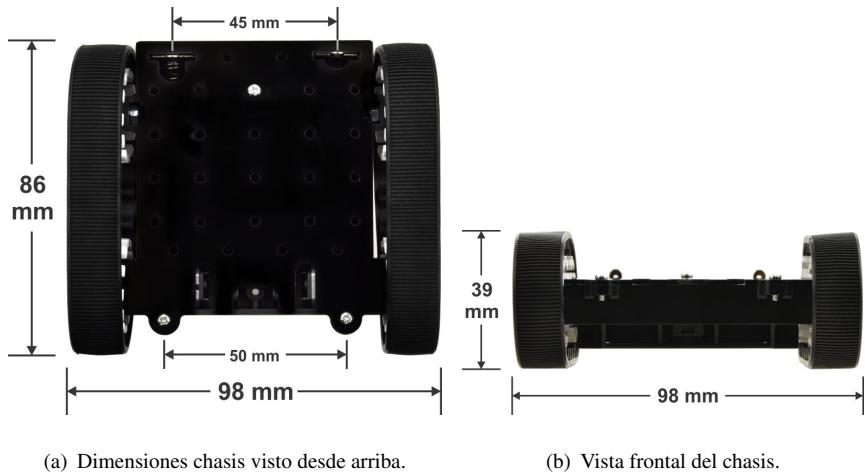


FIGURA 3.25: Chasis *Pololu robot Zumo*.

Las dimensiones del robot son las que se muestran en la Figura 3.26:



(a) Dimensiones chasis visto desde arriba.

(b) Vista frontal del chasis.

FIGURA 3.26: Dimensión del chasis *Pololu Zumo*.

3.3.6. Montaje

En esta Sección se describe como montar el robot utilizado en el proyecto paso a paso. A continuación se sintetizan los pasos que vamos a seguir:

1. Preparación de todas las piezas.
2. Soldadura a la placa base.
3. Montaje y preparación del chasis.
4. Integración del Hardware al chasis y acabado.

3.3.6.1. Preparación de piezas

En primer lugar, colocaremos todas las piezas necesarias para nuestro robot sobre el área de trabajo. Seguiremos estos pasos para la preparación de las piezas:

1. Cortaremos con una Dremel el chasis dando hueco al sensor de color, y a los CNY70 para que sobresalgan por debajo y por detrás para poder insertar el cable a nuestro Arduino. Deberá quedar como en la Figura 3.27.

2. Haremos un alargador para el cable de nuestra batería, crimpando un conector macho JST-XH y soldando uno hembra por la otra parte del cable. Importante respetar la polaridad ya que podemos crear un cortocircuito.
3. Soldaremos los cables sobre los motores de corriente continua, les pondremos goma termorretráctil aplicaremos calor y dejaremos ambos motores en segundo plano.
4. Desoldaremos los 4 pines de entrada de la controladora de motor y la borna triple por la cual se alimenta ésta. Volvemos a soldar sobre la otra cara, los 4 pines que desoldamos, y en el lugar de las bornas soldamos pines macho individuales. Además atornillaremos sobre las tuercas de separación hexagonales por la parte superior de la placa.
5. Aplicaremos silicona en la parte posterior de nuestro módulo Bluetooth en la parte de los pines para aislarlos.
6. A continuación, soldamos nuestra PCB para el circuito de nuestros sensores CNY70 siguiendo el esquemático mostrado en la Figura 3.18.

Una vez preparados los componentes anteriores pasamos al siguiente punto.

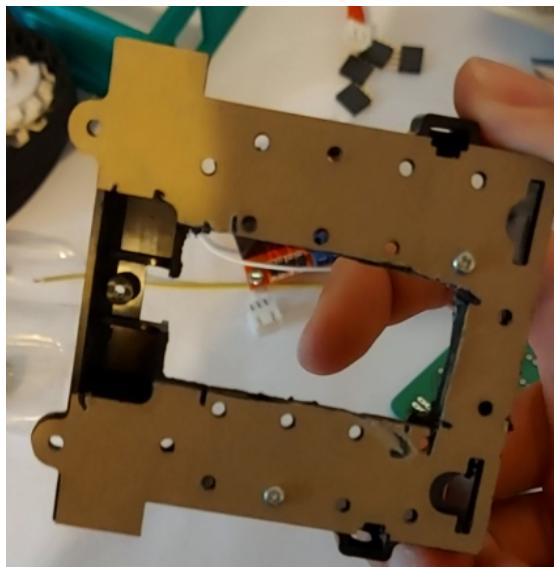
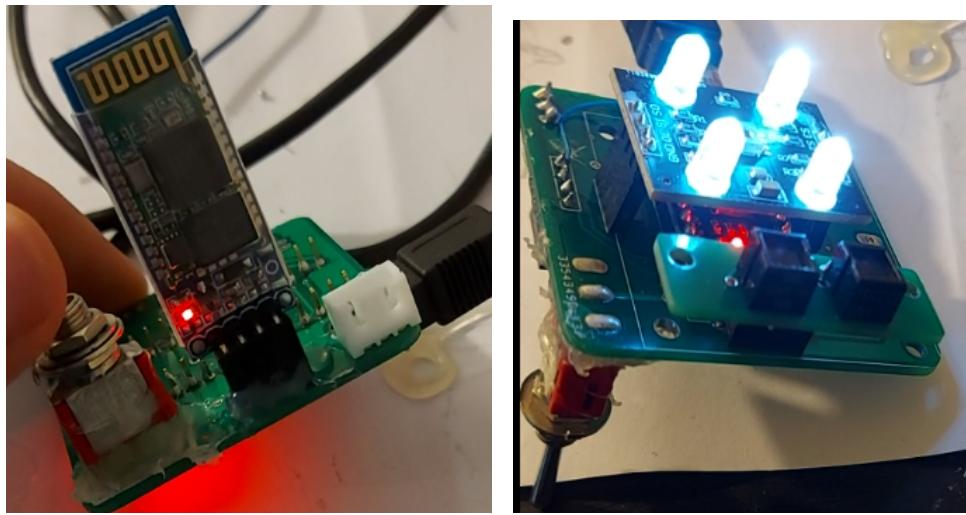


FIGURA 3.27: Chasis *Pololu robot Zumo* cortado.

3.3.6.2. Soldadura a la placa base

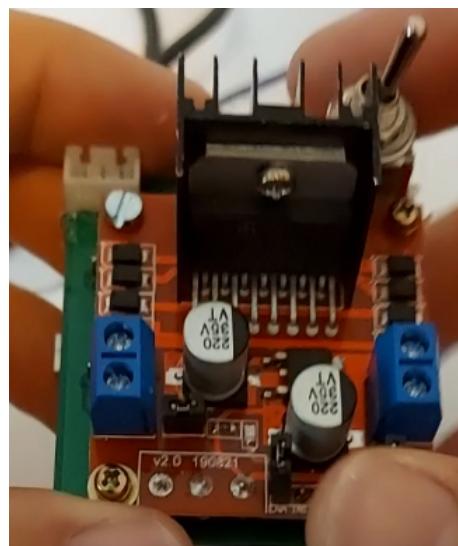
Procedemos a soldar nuestra placa base siguiendo el esquemático mostrado en la Figura 3.22, para ello, seguiremos los siguientes puntos:

1. Tomamos nuestra PCB y soldamos nuestra Arduino por la parte de la serigrafía de la PCB donde se ve la Arduino.
2. Soldamos nuestro interruptor de encendido/apagado por la parte de la serigrafía en la PCB. Una vez soldado, fijamos con silicona caliente.
3. Soldamos el conector JST-XH hembra en la misma cara que el interruptor de encendido/apagado. Nos debe quedar algo parecido a la Figura 3.28(a).
4. Soldamos los conectores hemrba tanto del Bluetooth, como del sensor de color y de nuestra PCB de los sensores CNY70. Nos quedará algo parecido a la Figura 3.28(b).
5. Ahora procedemos a soldar nuestra controladora de motor, en la cara contraria a la Arduino, posamos nuestra controladora de motor previamente preparada siguiendo el esquemático de la Figura 3.22. Nos quedará algo como en la Figura 3.28(c).



(a) Bluetooth, interruptor y JST-XH montado, además de la Arduino.

(b) CNY70 y TCS3200 montado.



(c) Controladora de motor montada.

FIGURA 3.28: Placa base montada.

Una vez terminados estos pasos procedemos al montaje del robot.

3.3.6.3. Montaje y preparación del chasis

Para montar nuestro robot conectamos debidamente el sensor de color, modulo Bluetooth y PCB de sensores CNY70 en sus correspondientes conectores hembra.

Una vez hecho esto, procedemos al montaje del chasis indicado en [48].

Cuando tengamos ambos montados, conectamos los motores a las bornas, motor derecho a la borna trasera, motor izquierdo a la delantera. Concretamente no sabemos que cable de los de un motor va a la borna derecha o izquierda correspondiente, debemos probarlo haciendo que el robot se mueva hacia delante si alguna de las dos orugas se mueve hacia detrás es porque los cables están inversamente conectados.

Posamos la estructura electrónica sobre el chasis y fijamos con silicona al chasis, por la parte trasera sin forzar la estructura electrónica.

Nos debe quedar algo como en la siguiente Figura:

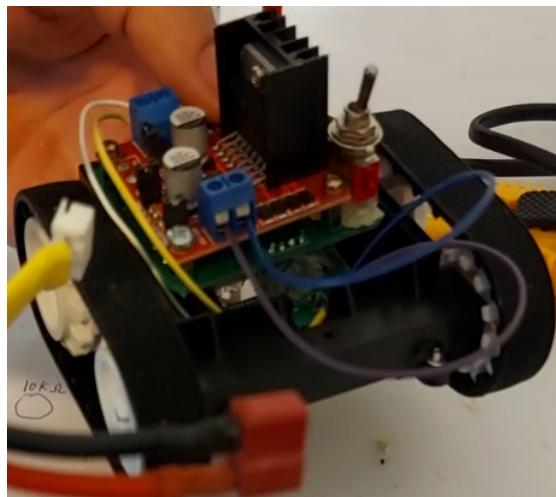


FIGURA 3.29: Placa base montada.

Hecho esto atornillamos por la parte inferior el soporte 3D que aguantará la batería, posamos la batería encima y le ponemos una brida para que quede fija y conectamos la alimentación al robot.

Al terminar nos debe quedar el robot de la siguiente manera.

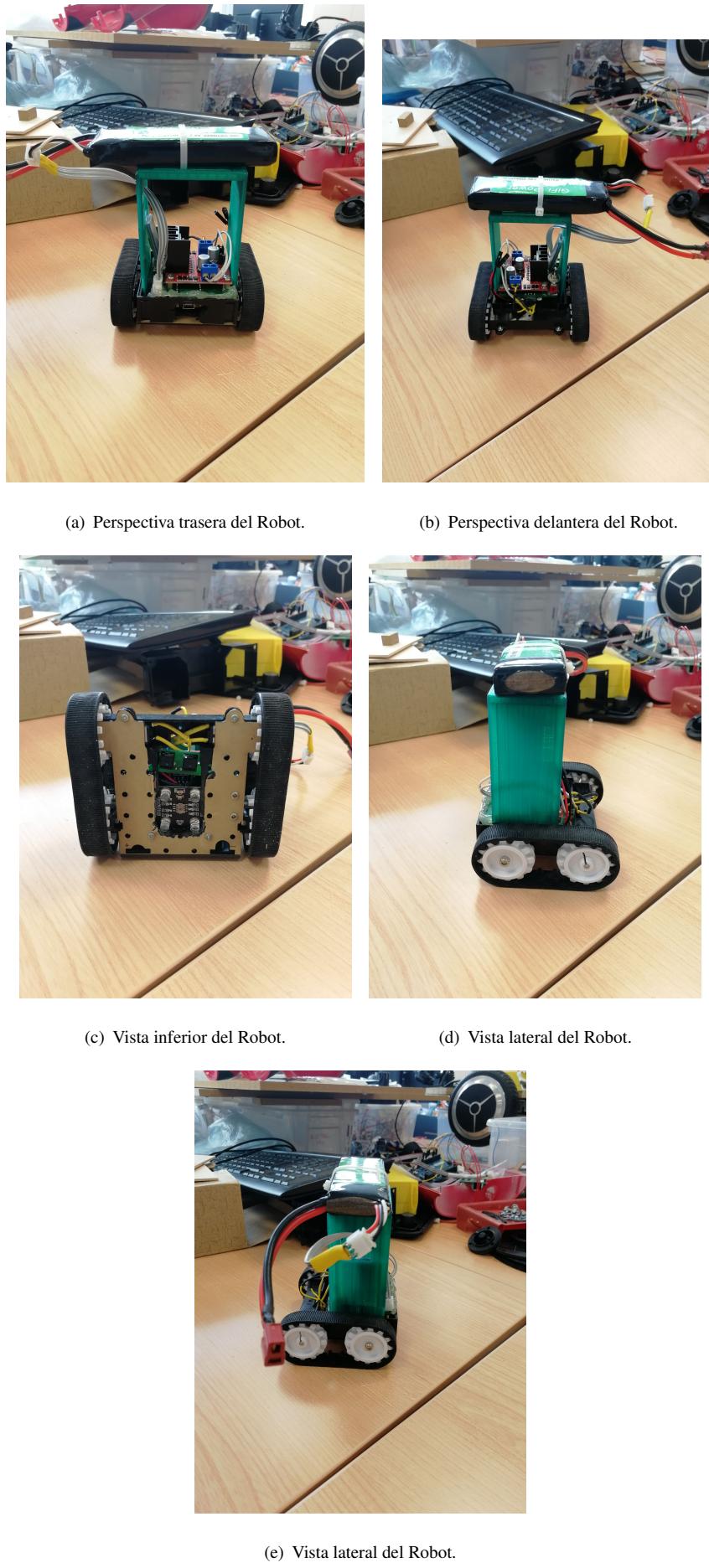


FIGURA 3.30: Robot que simula una hormiga en el ACO.

3.3.7. Test de piezas individuales

Previamente a cargar el programa final deberíamos comprobar individualmente algunos módulos, para ello, disponemos de un programa en Arduino llamado “TEST_PIEZAS_HORMIGA.ino” (cuyo código completo podemos encontrar en el Anexo B.1.3), cuando abrimos el programa vemos definidas las siguientes directivas de preprocesador:

```
#define SERIAL_WIRED_ONLY_
#define STRAIGHT_FOWARD_
#define RUN_WHITE_PATH_ROBOT_
#define PRINT_CNY_READ_
#define PROBAR_COLORES_
#define NODE_OPERATIONS_
#define FOLLOW_BLACK_LINES_
#define GRADES_
```

Hay más directivas definidas, pero no las descomentaremos puesto que fueron pruebas de desarrollo realizadas y puede que su funcionamiento no sea correcto.

Definimos para que sirve cada directiva a continuación, para probarlas, se debe tener descomentada solo dos al mismo tiempo, siendo una de ellas obligatoriamente “SERIAL_WIRED_ONLY_”.

Procedemos a comentar cada una de ellas:

- “SERIAL_WIRED_ONLY_”: Se utiliza para transmitir la información por el monitor serie de Arduino, al tenerla activa podremos monitorizar desde una consola la información que cualquier otra directiva transmita.
- “STRAIGHT_FOWARD_”: Se descomentará para hacer que el robot avance o vaya marcha atrás.
- “RUN_WHITE_PATH_ROBOT_”: El robot realizará un sigue línea pero de camino blanco.
- “PRINT_CNY_READ_”: Monitorizaremos los valores que están devolviendo los sensores CNY70, tanto el derecho como el izquierdo.
- “PROBAR_COLORES_”: Monitorizaremos la lectura de qué color está leyendo el sensor de colores TCS3200.

- “NODE_OPERATIONS_”: El robot realizará un sigue línea por camino negro, se parará cuando ambos CNY70 lean blanco, continuará hasta que estos vuelvan a ser negro, y realizará la lectura de color mostrándola por consola.
- “FOLLOW_BLACK_LINES_”: El robot realizará un sigue línea de camino negro.
- “GRADES_”: Para calibrar los giros del robot, dando más o menos segundos en los “delay”, podremos calibrar los giros de 90°, 180° y 270°.

Debemos probar también nuestro módulo Bluetooth para ello disponemos del código Arduino “BLUETOOTH_PRUEBA.ino”, el cuál cargamos en nuestro robot directamente, luego ejecutamos nuestro servidor, conectamos con el Bluetooth correspondiente y comprobamos que nos tiene que haber simulado un camino recorrido que pasa por los nodos 0,2,5,3. Previamente debemos tener enlazado el módulo Bluetooth HC-05 con nuestro ordenador.

Realizadas dichas pruebas podemos proceder a cargar el programa final, que se explica en la Sección [3.5](#).

3.4. Estructura del mapa

El robot correrá sobre un tablero que plasma un Grafo, comentado en la Sección [2.1.3.2](#), que se detallará en esta Sección. También se expondrán los errores y versiones de mapa plasmadas y probadas.

3.4.1. Estudio previo

Tenemos pues, la necesidad de que nuestro robot se ejecute sobre un mapa, puesto que virtualmente existe un Grafo $G \mid G = (V, A)$, siendo V el numero de vértices o nodos, y A el número de aristas que conecta dichos nodos, y siendo este Grafo ponderado, conexo y bidireccional creamos la necesidad de plasmar dicha idea sobre un entorno real.

Para ello simularemos los componentes de este grafo, vértices y aristas de la siguiente manera:

- Vértices: Los vértices o nodos se simularán en recuadros de colores. Los colores disponibles son el Rojo, Verde y Azul puesto que son los colores que lee nuestro sensor de colores TCS3200.
- Aristas: Las aristas se simularán con líneas de color negro con un ancho de 20mm puesto que nuestro sensor CNY70 interactuará con dichas líneas con un programa de sigue línea.

En las siguientes secciones se seguirá detallando como se ha llegado a implementar esta idea que parte de la base descrita en esta Sección.

3.4.2. Diseño del tablero

Procedemos a en este paso a diseñar nuestro tablero con medidas concretas.

Puesto que el sensor CNY70 tiene una lectura de color blanca al leer los tres colores, aprovechamos esto para enlazar nuestro programa, el cuál realiza el sigue línea negro y luego el blanco, esto para posarse encima del color para dar pie al sensor de color para que realice la lectura, el sigue línea blanco, no lo realiza en la versión final sobre un fondo blanco, sino uno de color. Los nodos finalmente tienen un tamaño de 39mm.

Para realizar el diseño del tablero se ha utilizado el software “Inkscape” [49] en su versión 0.92.

A continuación se comentan las distintas versiones por las que se han pasado hasta llegar al mapa final, enumeradas serían las siguientes:

- Versión 1.0
- Versión 2.0
- Versión 3.0
- Versión 4.0

3.4.2.1. Tablero Versión 1.0

La primera versión del mapa estable, con la cuál se empezaron a hacer pruebas constaba de unos recuadros más pequeños y líneas un poco más gruesas, además de curvas redondeadas las cuales daban muchos problemas, por lo que en la versión final se pusieron con líneas rectas, puede que en una futura versión estas curvas se pudieran retomar en el mapa.

Cada nodo consta también de un embudo el cuál actúa una vez el robot sale del mapa por si tiene algún error en el giro o lo acarrea de la entrada al nodo que se rectifique gracias a los CNY70.

La paleta utiliza el modelo cromático RGB que representa los colores Rojo, Verde y Azul en una escala de intensidad de 0 a 255 la cuál combina los colores para formar otros nuevos. Los valores que se han utilizado para representar los nodos son los siguientes.

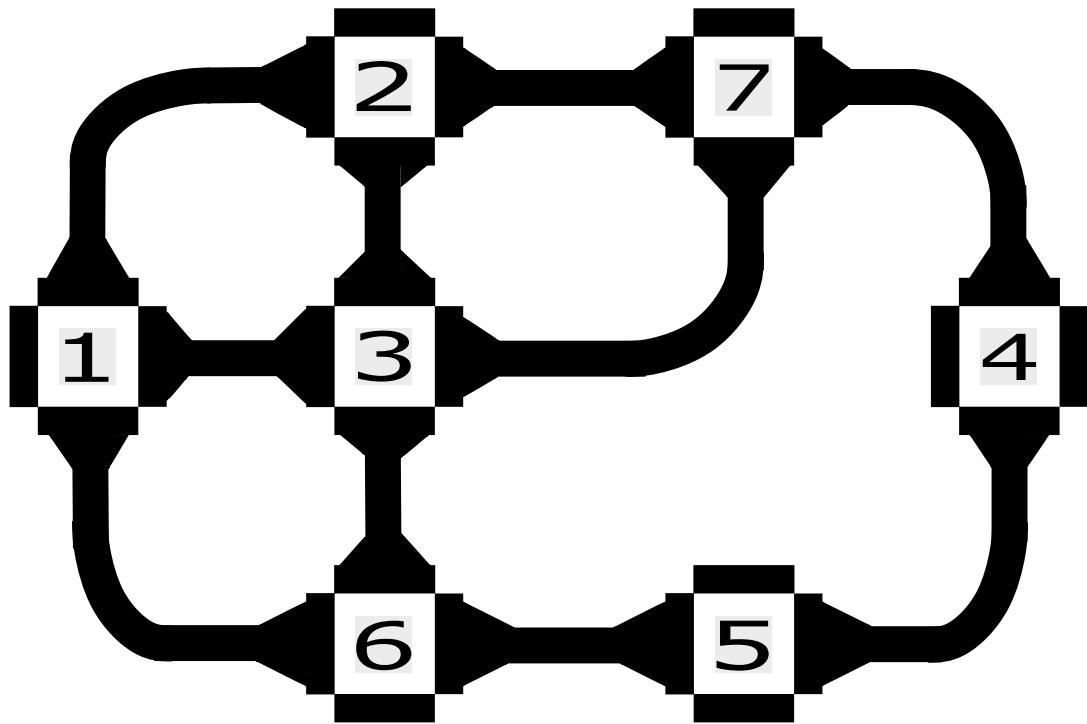
	R	G	B
Rojo	255	0	0
Verde	0	141	0
Azul	0	0	255

TABLA 3.7: Paleta de colores utilizada en el tablero.

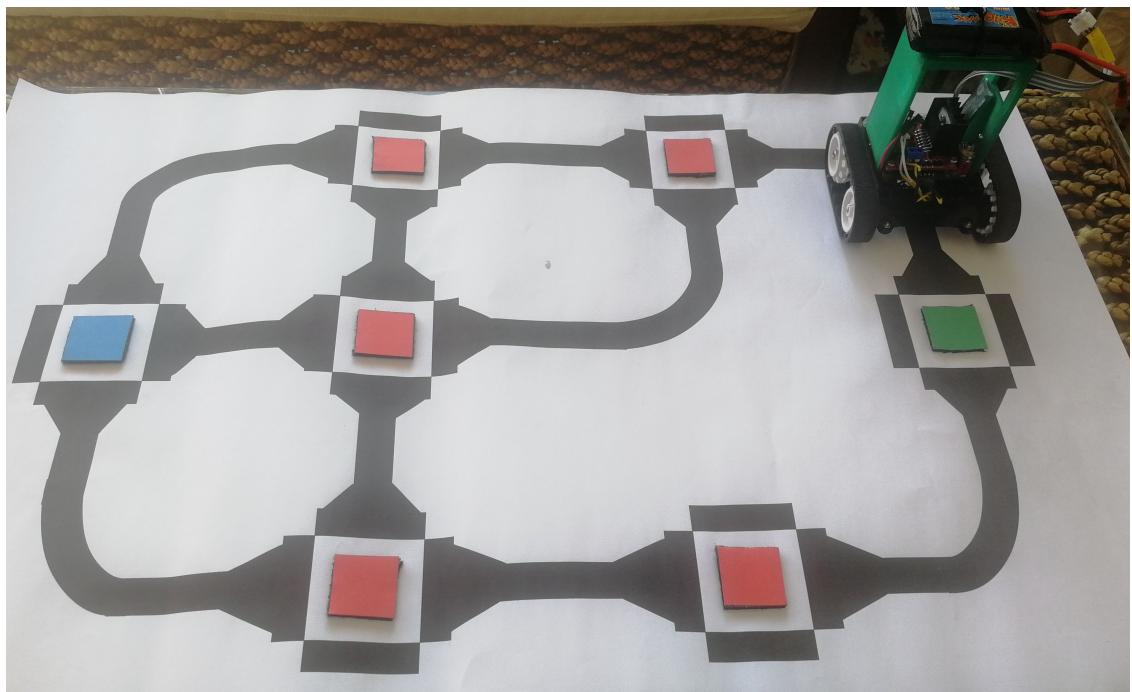
Resalta que los nodos están numerados y no tienen color, esto es porque en una primera idea se recortó velcro adhesivo de doble cara, una cara iba pegada al tablero y la otra cara tenía una cartulina del color que correspondiera, esto se pensó para poder cambiar en plena ejecución la fuente de alimento para poder ver mejor el algoritmo, pero se rectificó en esta idea puesto que el velcro, al estar por encima del nivel del mapa, el robot cuando giraba, era impreciso ya que golpeaba con el velcro llegando incluso a caerse en alguna que otra ejecución.

Resalta también que el número de nodos es mayor, esto se comentará más adelante, pero Arduino al tener poca memoria, no permitía cargar grandes matrices así que hubo que recortar, en esta versión vemos 7 nodos, que van desde el 1 al 7.

A continuación se muestra la primera versión del tablero.



(a) Diseño de la versión 1.0 del tablero.



(b) Tablero versión 1.0 impreso.

FIGURA 3.31: Tablero v1.0.

3.4.2.2. Tablero Versión 2.0

En la segunda versión del mapa, se optó por corregir el problema de los velcro principalmente, se plasmo en el mismo diseño el color de los nodos, y se dio espacio en los recuadros que engloban los nodos pensando que esto sería una mejora, posteriormente se volvería atrás en esta última idea. Además se acortó el número de nodos en 1, habiendo 6 nodos en total, y estando estos numerados del 0 al 5 para que fuera natural en la matriz del programa ya que en C/C++ el primer elemento de una matriz ocupa la posición (0,0).

A continuación se muestra el diseño de la versión 2.0 del mapa.

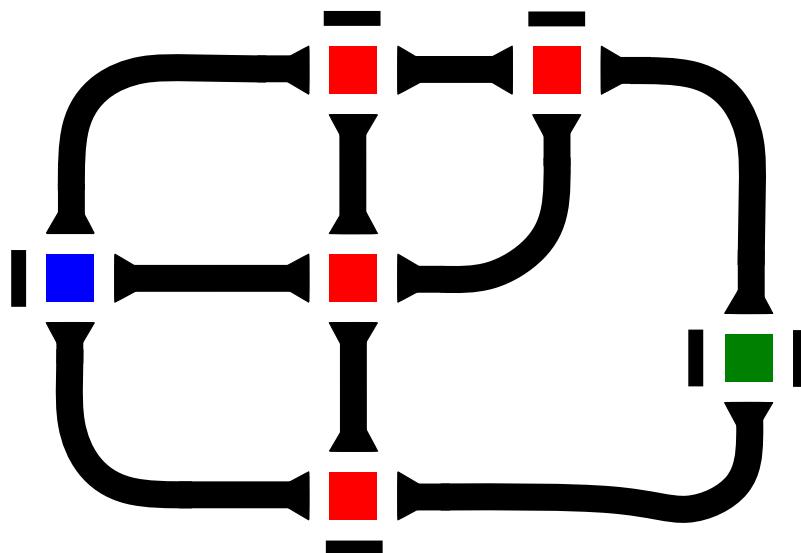


FIGURA 3.32: Versión 2.0 del tablero.

Se hizo una pequeña modificación dentro de este mismo mapa considerada como la Versión 2.1 en la que se mejora el ángulo de las curvas así como se ponen líneas rectas en vez de curvas y se ponen barreras negras alrededor de los nodos mejorando así que el robot no se cuele por el blanco y se salga del mapa.

A continuación se muestra el diseño de la versión 2.1 del mapa.

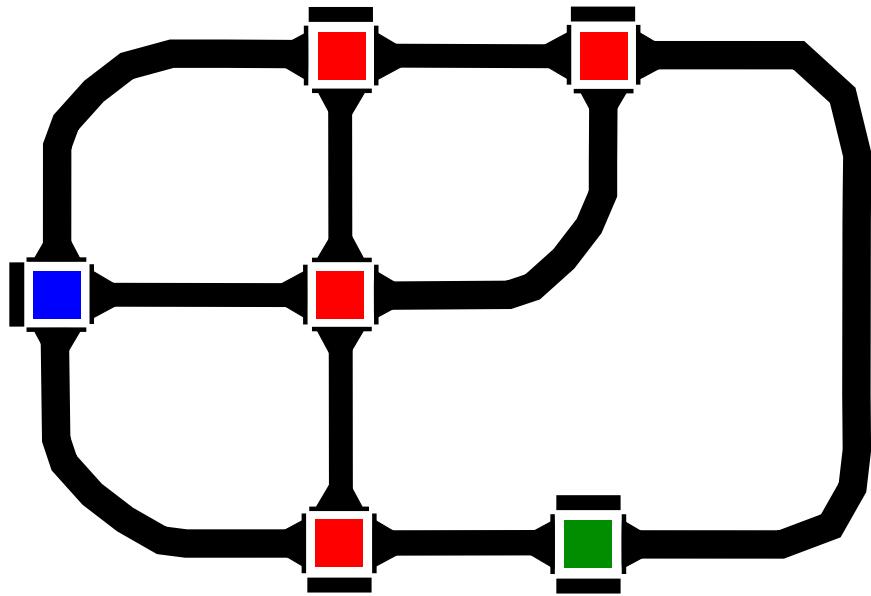


FIGURA 3.33: Versión 2.1 del tablero.

3.4.2.3. Tablero Versión 3.0

En la versión 3.0 hay una pequeña mejora en la cuál se deja menos espacio en blanco en los nodos además se reduce el grosor de las aristas. Ésta podría ser considerada incluso una versión 2.2.

A continuación se muestra la versión 3.0 del mapa.

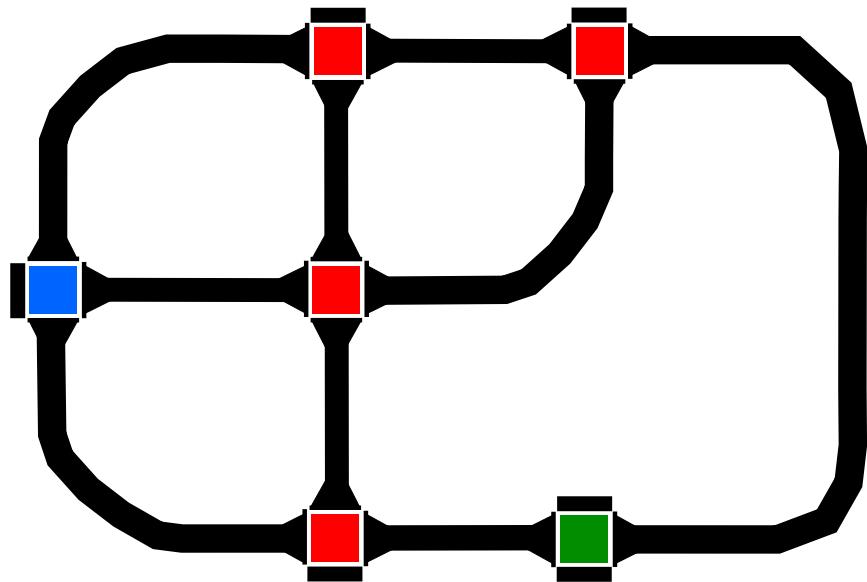


FIGURA 3.34: Versión 3.0 del tablero.

3.4.2.4. Tablero Versión 4.0

Finalmente llegamos a la versión actual del mapa en la cuál se rellenan los nodos y se cierran los marcos negros, además el embudo que tienen al salir de un nodo se expande un poco más.

A continuación se muestra el diseño de la versión 4.0 del mapa:

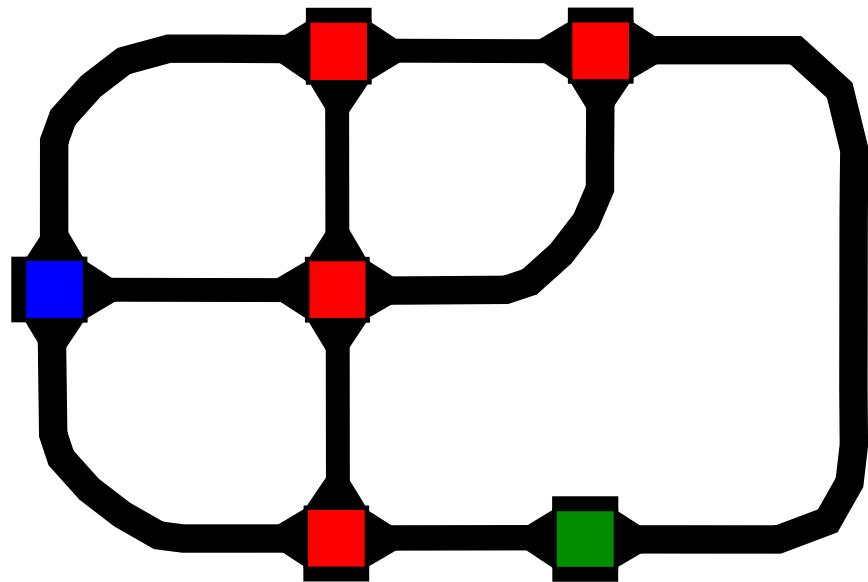


FIGURA 3.35: Versión 4.0 del tablero.

3.4.3. Impresión del tablero

Una vez realizado el diseño se pasa a la impresión, la cuál se realiza a través de una impresora que permita impresión en A2. Como se comenta aquí mismo el tamaño de impresión del mapa es en A2 (420 x 594 mm).

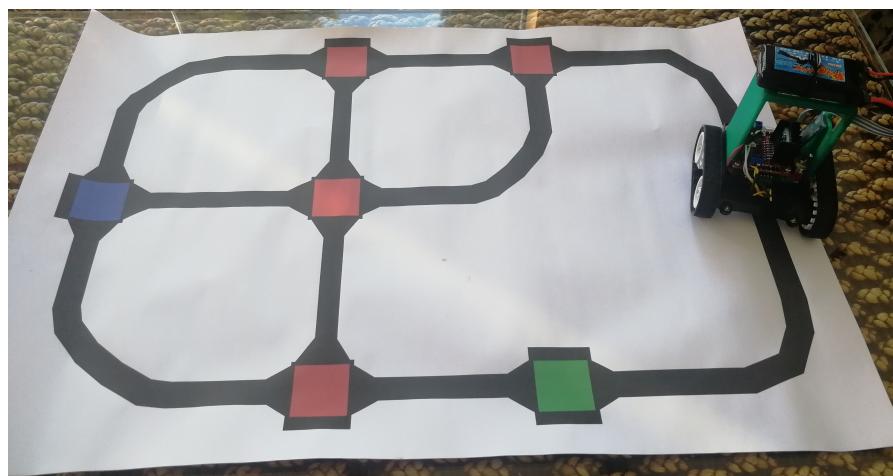


FIGURA 3.36: Versión 4.0 del tablero impreso.

3.5. Implementación del Algoritmo de Colonia de Hormigas en Arduino.

En esta Sección se comenta la implementación realizada en el robot, qué se ha utilizado y como se ha realizado dicha implementación desde que partimos con la electrónica testeada hasta que se realice una primera ejecución.

El IDE utilizado es el propio de Arduino en su versión 1.8.12.

Para ello dividiremos esta Sección en:

- Especificaciones del ACO que debe cumplir el robot.
- Explicación de la implementación en Arduino.
- Problemas encontrados durante el desarrollo.

3.5.1. Especificaciones que debe cumplir el Robot en Arduino

Puesto que la estructura del proyecto es un Cliente/Servidor, donde el cliente será el robot, e inherentemente el ACO es un sistema distribuido donde las hormigas se comunican entre sí, debemos discriminar qué parte del algoritmo va al servidor y qué parte al cliente, en esta Sección discriminaremos qué va al Robot, es decir, a nuestro cliente.

Como se comentó en la Sección 2.1.3.1 el ACO busca un recurso, inicialmente de manera aleatoria, aleatoriedad que se irá condicionando a medida que se vayan realizando ejecuciones, y al cabo de un número indefinido de iteraciones las hormigas se moverán por el camino que más se haya recorrido, que no tiene por qué ser el mejor, pero si que será un muy buen camino resuelto en tiempo óptimo.

El cliente o robot en este caso, debe encargarse de recorrer el camino, esto implica elegir qué nodo se va a elegir y pasar a él sucesivamente hasta llegar al recurso. El servidor como veremos en la Sección 3.8 se encargará de realizar el resto de tareas.

El robot contendrá una matriz de feromonas que será la que nos indique entre otros parámetros la probabilidad de tomar un camino u otro, esta matriz es compartida y en el robot es una matriz de solo lectura que se irá actualizando por cada iteración.

El robot utilizará del [ACO](#) la fórmula de elección de siguiente nodo descrita en la Sección [2.1.3.2](#). La fórmula es la que sigue:

$$\blacksquare \quad p_{xy}^k = \frac{(\tau_{xy}^\alpha)(v_{xy}^\beta)}{\sum(\tau_{xy}^\alpha)(v_{xy}^\beta)}$$

En cuanto a especificaciones técnicas de Arduino Nano tenemos de memoria de almacenamiento de programa 30720 bytes y de almacenamiento de variables 2048 bytes. Esto es un problema puesto que el [ACO](#) requiere de grandes prestaciones y si queremos implementarlo en un robot, necesitamos aún más.

El programa final ocupa 11228 bytes, en variables locales utiliza 702 bytes de memoria dinámica y 1346 bytes de almacenamiento para variables locales. Donde el robot aun podría dar más de sí en cuanto a incrementar el número de nodos.

3.5.2. Implementación en Arduino

El código que implementa el robot está separado por controladores y librerías para cada uno de los sensores o actuadores, los controladores hacen que interactúen entre sí las librerías además de con la lógica.

A continuación se van a enumerar las clases que contiene el código en Arduino llamado “ACO.ino” además de detallar que función tiene cada una de ellas.

- “ACO.ino”: Programa principal donde se llama al controlador principal “MAIN_CONTROLLER.hpp” y a la clase “PINS.hpp” además de inicializar la comunicación serie a una velocidad de 9600 baudios. Podemos encontrar el código completo en el Anexo [B.1.5](#).
- “PINS.hpp”: Aquí se definen los pines que deben coincidir con el esquemático mostrado en la Figura [3.22](#). Podemos encontrar el código completo en el Anexo [B.1.1.1](#).

- Controladores

- “MAIN_CONTROLLER.hpp”: Controlador principal que se encargará de llamar al resto de controladores en orden para que funcione con la lógica debida. Podemos encontrar el código completo en el Anexo [B.1.5](#).
- “PERCEPTION.hpp”: Capa controladora que capta la información para ser procesada posteriormente. La información que capta es el color del nodo en el que está posicionado el robot. Podemos encontrar el código completo en el Anexo [B.1.5](#).
- “LOGIC.hpp”: Capa que controla el procesamiento debido, como entrada recibe un color, el que haya leído en la percepción y como salida ofrece a la siguiente capa, que es la actuadora los grados que debe girar el robot que coincidirá con el siguiente nodo que ha calculado en esta misma capa utilizando la fórmula de elección de nodo vista en la Sección [2.1.3.2](#). Podemos encontrar el código completo en el Anexo [B.1.5](#).
- “ACTUATION.hpp”: Esta capa se encarga del movimiento del robot recibido después del procesamiento, de que el robot vaya de un nodo al siguiente. Podemos encontrar el código completo en el Anexo [B.1.5](#).

- Librerías de sensores y actuadores

- “CNY.hpp”: Administración de los sensores CNY70 de manera individual. Podemos encontrar el código completo en el Anexo [B.1.1.2](#).
- “2xCNY.hpp”: Administración de dos sensores CNY70 juntos el derecho y el izquierdo así como la identificación del lugar de cada uno. Podemos encontrar el código completo en el Anexo [B.1.1.3](#).
- “MOTOR.hpp”: Administración de cada motor, con operaciones para avanzar, retroceder, girar...
- “TSC3200.hpp”: Administración del sensor de color TCS3200. Podemos encontrar el código completo en el Anexo [B.1.1.4](#).

- Librerías software

- “MAP.hpp”: Contiene la matriz de feromonas, así como la fórmula de elección de nodo. Clase esencial en la capa de procesamiento. Podemos encontrar el código completo en el Anexo [B.1.1.9](#).

- “PATH.hpp”: Engloba un objeto que contiene el camino que va recorriendo la hormiga. Podemos encontrar el código completo en el Anexo [B.1.1.8](#).
- “RATIONAL.hpp”: Clase de números racionales con la función de sustituir el uso de números racionales. Podemos encontrar el código completo en el Anexo [B.1.1.7](#).
- “COMMON_TYPES.hpp”: Envoltura de un tipo enumerado que contiene los tres colores disponibles (Rojo, Verde, Azul). Se escribió por si hacía falta algún tipo común más. Podemos encontrar el código completo en el Anexo [B.1.1.6](#).

A continuación se muestra una pequeña ilustración de la interacción de controladores en el programa “ACO.ino”:

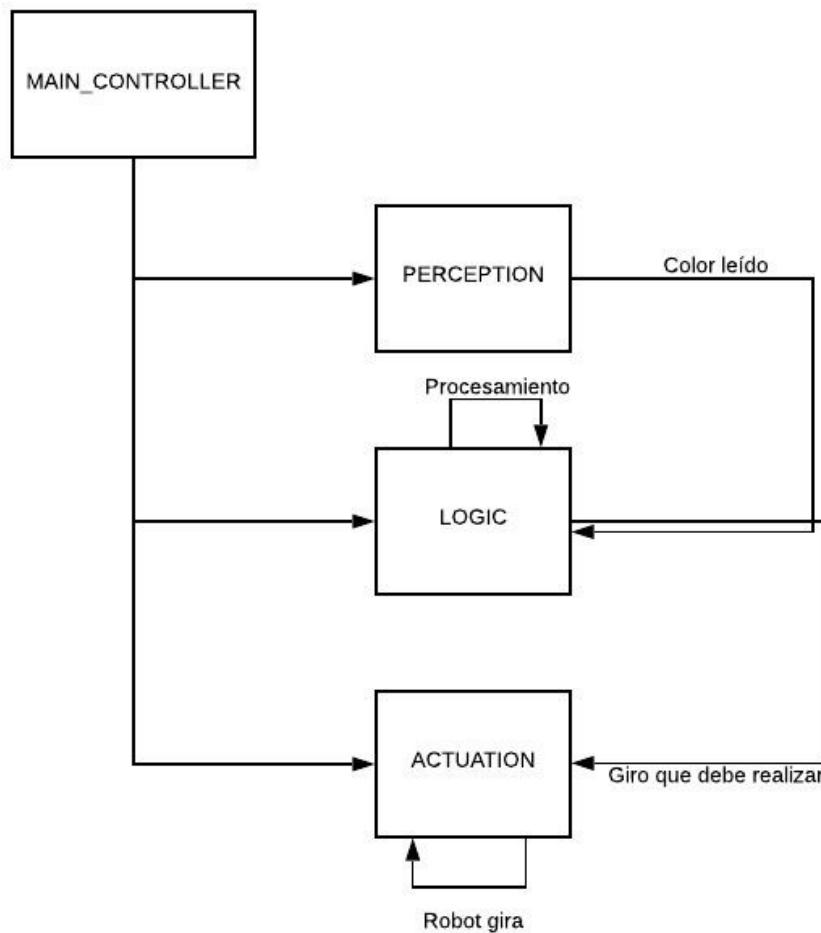


FIGURA 3.37: Diagrama de flujo de controladores en el programa “ACO.ino”.

3.5.3. Problemas durante el desarrollo en Arduino

A continuación se enumeran los problemas encontrados a la hora de crear el código en Arduino, concretamente en Arduino Nano, y la solución que se dio.

- Problema a la hora de realizar operaciones con números en coma flotante ya sean de precisión simple o precisión doble, el robot se interbloqueaba y no respondía. Para solventarlo se creó una clase de números racionales, aunque esta sea mas ineficiente en cuanto a memoria ya que contiene dos atributos de tipo entero, para numerador y denominador, el programa no se interbloquea. Como ventaja, ganamos en precisión a la hora de realizar cálculos.
- Problema a la hora de subir el programa por el uso de memoria dinámica, este problema se debe al tamaño de las variables globales donde se ocupó un 70 % de ésta dejando un 30 % libre para variables locales, el mismo IDE de Arduino lanza un aviso de que el programa podría mal funcionar como así fue debido a que las variables locales en un momento determinado del programan necesitaban más de ese 30 %. La solución que se dió fue pasar de 7 nodos que contenía el primer mapa como podemos apreciar en la Figura 3.31 a 5 nodos como podemos apreciar en la Figura 3.35.
- Al principio se pretendió utilizar una lista enlazada de nodos como estructura para albergar el camino que el robot hubiera recorrido, utilizando el TAD Lista, con la implementación de celdas enlazadas, pero al ver los problemas de memoria que Arduino Nano ofrecía se omitió dicho TAD y se implemento un TAD llamado “Path” de tamaño definido en tiempo de compilación.
- Una posible mejora en el robot que ahorraría bastante memoria, es la de incluir como electrónica adicional un giroscopio puesto que esta permite medir el ángulo de rotación en este caso de nuestro robot, ahorraríamos la matriz de giros así como la matriz de compensación pudiendo hacerse esto en procesamiento y no almacenando en memoria en una matriz de valores de giros como está hecho.

3.6. Comunicación Cliente Servidor

En esta Sección se detalla la interfaz de comunicación entre el robot y el servidor, como sabemos tomamos la decisión de usar Bluetooth luego se explica también qué tecnología implementada se utiliza para dicha comunicación.

3.6.1. Interfaz de comunicación

Para comunicar el cliente con el servidor mediante Bluetooth, se establece una interfaz de comunicación, es decir, un orden en el que se transmiten los datos de un tipo concreto y un formato concreto. En este apartado explicamos cómo se transmiten además de un ejemplo concreto de transmisión.

Comenzamos explicando el orden en que se transmiten los datos, cómo y por qué.

Inicialmente se envía la matriz de feromonas del servidor al robot para que éste quede sincronizado. Esto se hace únicamente al encender el robot.

Acto seguido, se intenta realizar una comunicación en la que el primero que salude sea el servidor, de manera manual, es decir pulsando el botón de transmisión explicado en la Sección 3.8.3.1. El servidor transmitirá la matriz de feromonas una vez el robot esté sobre un nodo Azul que representa el hormiguero, es decir, la llegada de haber realizado una iteración. La letra elegida para el saludo inicial es la “p”.

Una vez enviado el saludo, el cliente comienza con la transmisión del camino. El camino es un conjunto de nodos recogidos en una cadena de texto separados por comas. Por ejemplo: “0,2,4,3” sería un posible envío del cliente al servidor.

Una vez recibido, el servidor desmenuza el camino y realiza las operaciones pertinentes del ACO, actualizando así su matriz de feromonas la cual devolverá en formato cadena de caracteres de números racionales separados por coma con un carácter de final representado con la letra “E”. Por ejemplo: “1/100,13/38,1/10,[...],1/100E”.

Una vez Arduino reciba este dato lo desmenuzará uno a uno y actualizará la matriz de feromonas con dichos valores, con lo cual ya habría sido ésta actualizada y se dispone a continuar la ejecución de un nuevo camino.

Podemos encontrar las líneas de código correspondientes al servidor en el Anexo [B.1.7.2](#), mientras que el código correspondiente a la transmisión/recepción de datos del robot la podemos encontrar en el Anexo [B.1.7.1](#).

Sintetizamos lo comentado anteriormente en pocos pasos:

1. El cliente está sobre un nodo azul a la espera de una comunicación.
2. El servidor comienza la comunicación y envía un carácter “p”.
3. El cliente recibe dicho carácter y transmite al servidor el camino que ha tomado en una cadena de caracteres separados por comas, como por ejemplo: “nodo1,nodo2,nodo3,[...],nodoN” tal que $N \in \mathbb{Z}^+$.
4. El servidor procesará el camino para dar fin a una actualización en la matriz de feromonas y responderá con esta misma matriz al cliente enviándola en una cadena de caracteres separado por comas de números racionales como por ejemplo: “x/y,z/k,[...],i/jE” tal que $x,y,z,k,i,j \in \mathbb{Z}^+$.
5. Arduino recibirá la cadena de caracteres de la matriz y actualizará la suya propia en base a los valores recibidos.
6. Se puede cerrar manualmente la conexión una vez hayamos llegado a este paso de la comunicación.

Para facilitar aún más la visualización de la interfaz se ofrece la siguiente figura:

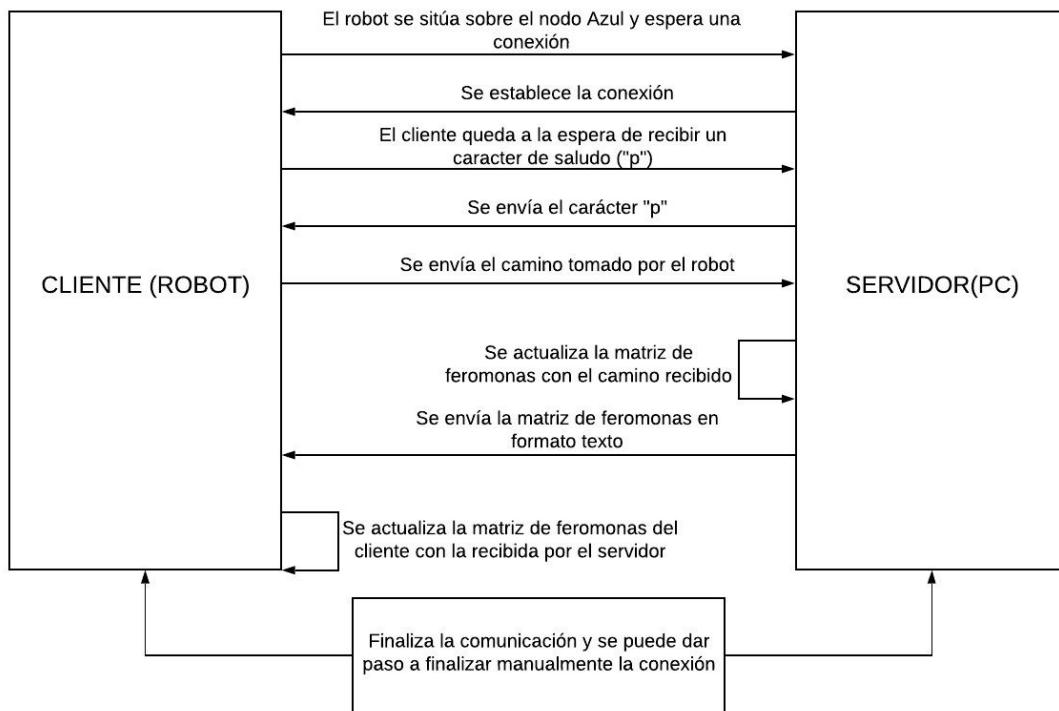


FIGURA 3.38: Interfaz de comunicación tras cada iteración. Podemos encontrar su implementación en el Anexo [B.1.7](#).

3.7. Implementación de la comunicación

Para implementar la interfaz descrita en la Sección 3.6.1 he utilizado la clase QBluetooth [50] por parte del servidor y los pines Rx y Tx a través de la clase Serial en Arduino.

El dispositivo físico en el cliente es el módulo HC-05 y como requisito previo debe estar enlazado con el equipo servidor a nivel de sistema operativo.

Se dispone adicionalmente de un programa de testeo en Arduino para el bluetooth bajo el nombre de “BLUETOOTH_PRUEBA.ino” en el que se transmite un camino inferido que se puede modificar en tiempo de compilación. Podemos encontrar el código del programa en el Anexo [B.1.4](#).

3.8. Aplicación servidor

En esta Sección se procede a describir la necesidad de una aplicación servidor, el rol que ocupa en el proyecto así como la descripción del desarrollo en sí.

3.8.1. Antecedentes de la aplicación servidor

Puesto que nuestra arquitectura es una arquitectura cliente-servidor, y como se ha comentado anteriormente el robot juega el rol de cliente, surge la necesidad de crear una aplicación servidor con la que los clientes o robots puedan interactuar.

La aplicación servidor realizada está diseñada en el IDE Qt versión 5.15.2, implementado con C++11 y Qt Creator versión 4.14.0.

En las siguientes Secciones se entra en detalle sobre esta aplicación.

3.8.2. Rol de la aplicación servidor

La aplicación servidor que se ha creado se encarga de una serie de puntos que se exponen a continuación, con ellos podemos identificar el rol que ocupa la aplicación servidor en dicho proyecto:

- Gestiona la comunicación inalámbrica entre el cliente y el servidor, comunicación que ha sido descrita en la Sección [3.6](#).
- Se encarga de actualizar la información de los clientes, una vez estos han terminado una iteración ofrecerán al servidor el camino que han tomado y recibirán como respuesta la matriz de feromonas actualizada.
- Surge la necesidad de monitorizar todo recorrido que realice cualquier hormiga con el fin de poder estudiar el algoritmo y en un futuro, poder mejorarlo.

- Al albergarse en un equipo más potente completa las carencias del cliente ya que ésta realiza el algoritmo de hormigas completo a excepción de la parte de elección de nodo la cual corresponde al cliente.

3.8.3. Desarrollo de la aplicación

El programa principal consta de varios módulos los cuales se explican a continuación:

- “ACO.pro”: Archivo de coordinación del proyecto en el que se incluyen las librerías externas necesarias, así como de los enlaces de los archivos versión de compilador que se quiere utilizar además de generar el script para la compilación.
- “main_controller.h”: Permite realizar operaciones sobre la matriz de feromonas y el camino que se esté procesando. Podemos encontrar el código completo en el Anexo [B.1.6](#).
- “mainwindow.h”: Es el archivo principal de donde se carga el entorno gráfico del programa, donde se recogen y programan todos los eventos disponibles del programa, donde se gestiona la comunicación Bluetooth entre el robot y la propia aplicación y se procede a la llamada del algoritmo en sí. Podemos encontrar el código completo en el Anexo [B.1.6](#).
- “matrix.h”: Recoge varias operaciones como la de expandir los adyacentes de un nodo, pasar la matriz de feromonas a cadena de caracteres para su posterior transmisión y contiene como atributo principal, el mapa del robot y la matriz de feromonas o el coste de evaporación, velocidad a la que se evaporan las feromonas en la operación de actualización de feromonas. Podemos encontrar el código completo en el Anexo [B.1.2.1](#).
- “path.h”: clase necesaria como receptora del camino emisor en el cliente para poder desgranarla y utilizarla en la actualización de feromonas. La operación de actualización de feromonas se realiza en esta clase. Podemos encontrar el código completo en el Anexo [B.1.2.2](#).
- “rational.h”: Clase receptora ya que la matriz de feromonas en el cliente es de tipo números racionales, en la aplicación servidora al tener más potencia podemos utilizar números en coma flotante los cuales son más cómodos para operar así que podemos hacer la conversión de racional a coma flotante operar pero hay que devolverlo al cliente como

cadena de caracteres de un número racional. Podemos encontrar el código completo en el Anexo B.1.2.3.

- “mainwindow.ui”: La recogida de eventos de dicho formulario se realiza en el archivo “mainwindow.cpp” cuyo código podemos encontrar en el Anexo B.1.6.
- “Resource.qrc”: Para almacenar el tema de multimedia e incluirlo en cualquier formulario, en este caso concreto se utiliza para almacenar una imagen del mapa que se mostrará en el programa.

Explicadas cada clase la operación que realiza, mostraremos a continuación la venta principal del programa y a partir de ahí explicaremos paso a paso como se realizaría una ejecución completa.

3.8.3.1. Ejecución de la aplicación servidor

A continuación se muestra la ventana del programa principal:

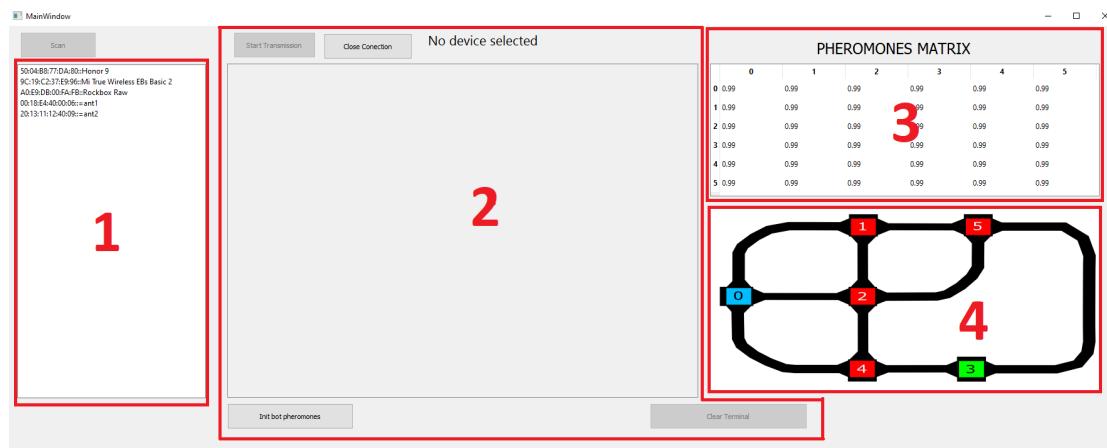


FIGURA 3.39: Ventana principal del programa servidor.

A continuación se describe cada ventana enumerada en la Figura 3.39:

1. En esta ventana podemos encontrar listados todos los Bluetooth disponibles, en mi caso solo nos interesarían los dispositivos con el nombre ant1 o ant2 los cuales corresponden a ambas hormigas.

2. Esta ventana es un registro de eventos en el que se incluye cualquier cosa que se transmita o reciba en el programa, datos tales como el camino que se recibe del cliente o la matriz de feromonas enviada, además de un mensaje al comienzo y al final de cada transmisión. Encontramos una errata en este apartado pendiente de reparar, el botón de transmisión ha de ser pulsado dos veces para que el envío sea realizado, he leído en [51] que colocando una resistencia en pull-up en el pin de transmisión de Arduino se solucionaría, pero queda pendiente de comprobarse. Hay un botón para limpiar los eventos en la terminal, y una etiqueta que en tiempo real indica a qué Bluetooth estamos conectados.
3. Se muestra la matriz de feromonas actualizada en cada cambio que se realice.
4. Se muestra una imagen del mapa que se está testeando como se indicó en la Sección 3.8.3 la imagen se toma del recurso “Resource.qrc”.

El procedimiento a realizar pasaría a ser casi en orden de izquierda a derecha indicado en la Figura 3.39 de 1 a 4.

1. Inicialmente se listarán los Bluetooth enlazados, pinchamos sobre el dispositivo al que queramos conectarnos y en cuanto la conexión se establezca, se habilitará la ventana número 2.
2. Si acabamos de encender el robot, debemos sincronizar la matriz de feromonas para que pueda iniciar, esto se hace pulsando sobre el botón “Init bot pheromones”. Si el robot ha realizado al menos una iteración este paso es innecesario y pasariamos al siguiente.
3. Una vez habilitada la ventana número 2, tenemos conexión con el robot, el diodo Led que contiene el módulo Bluetooth del robot habrá pasado de parpadear a estar fija, y podremos iniciar la comunicación, pulsamos 2 veces sobre el botón “Start Transmission” y el robot nos transmitirá sus datos, y nosotros responderemos con los nuestros, acto seguido termina la comunicación pero la conexión queda abierta hasta que no se corte explícitamente, para cortar la conexión podemos conectarnos directamente a otro dispositivo, o pulsar el botón “Scan” recogido en la ventana número 1 de la Figura 3.39.
4. Cuando se ha terminado la comunicación podemos observar como la ventana 3 de la Figura 3.39 que contiene la matriz de feromonas se actualiza automáticamente.

5. Para interpretar dicha matriz nos podemos basar en la ventana 4 de la Figura 3.39 la cual nos muestra el mapa con los nodos enumerados, dicha enumeración corresponde directamente con los índices de la matriz. Por ejemplo si queremos comprobar que valor tiene la feromona que va del nodo 1 al nodo 2, comprobamos en la matriz el valor que hay en la fila 1, columna 2. Como apunte puesto que se esta estudiando el ACO solo en la dirección de ida, y en la vuelta el robot toma el mismo camino de ida pero a la inversa la matriz nos quedará simétrica, si se hiciera el ACO a la vuelta, ésta sería asimétrica.

3.9. Integración del robot con el servidor

A continuación se describen los pasos seguidos para unir el proyecto completo que corresponden a las partes individuales:

- Cliente: Visto en la Sección 3.5.
- Comunicación: Visto en la Sección 3.6.
- Servidor: Visto en la Sección 3.8.

En primer lugar se realizó el cliente o robot, una vez completado este se realizó una pequeña aplicación en Qt utilizando la biblioteca QBluetooth [50] e interaccionando con el robot, enviando mensajes de “Hola mundo” y respondiendo desde el robot, luego se hizo un programa que simulara el envío de un camino y respondiera con una matriz inferida desde ambos programas, de esta manera se probó también que la matriz se actualizaba en el robot con los datos enviados desde el programa de Qt.

Posteriormente se creó la aplicación servidor final que incluía la parte de la comunicación y se probó con el programa final del cliente, dando como resultado el proyecto que podemos observar.

3.10. Ejecución del proyecto completo

En esta Sección se muestra una ejecución detallada de cómo funciona el proyecto realizado, es decir, como realizar una iteración con el robot Arduino con la interacción con el servidor.

Lo primero que debemos hacer es arrancar nuestro servidor, es decir, el programa en Qt. Una vez arrancado nos saldrá una ventana como esta:

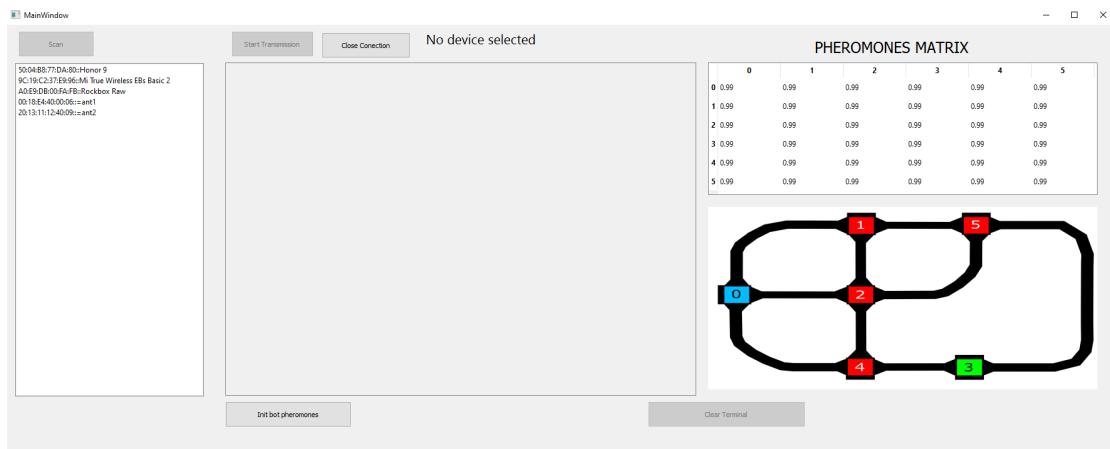


FIGURA 3.40: Servidor en su estado inicial.

Una vez tengamos esto colocamos el robot sobre la casilla de salida orientado a la derecha tal como se muestra en la siguiente figura y lo encendemos pulsando el interruptor de encendido.

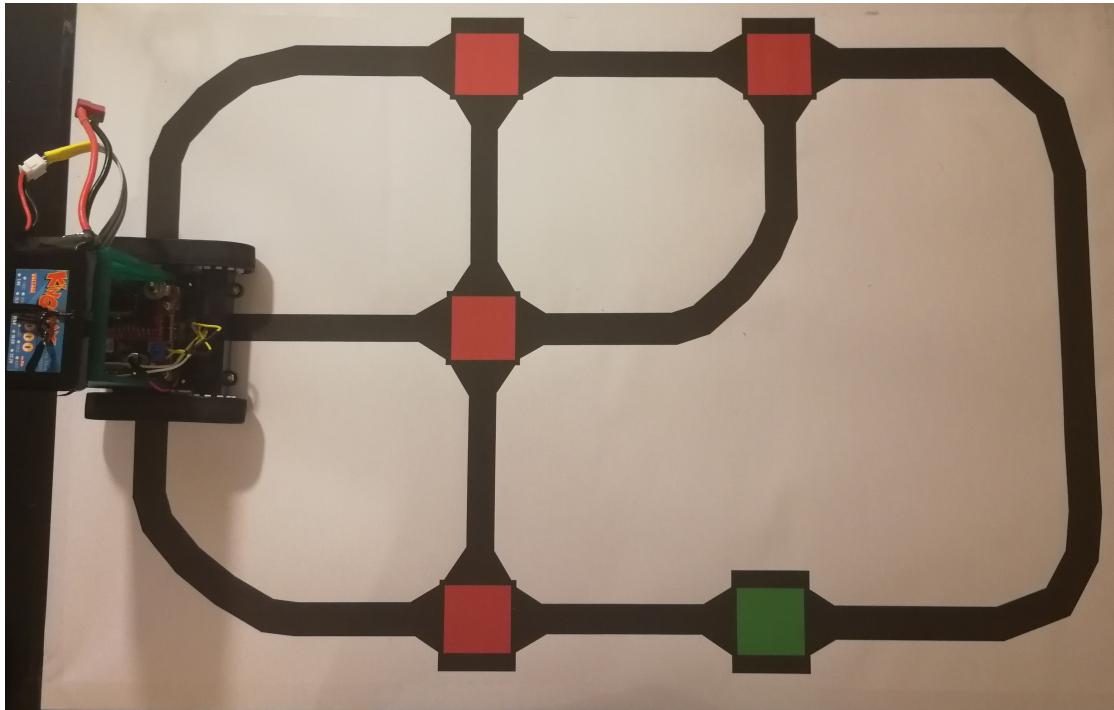


FIGURA 3.41: Estado inicial del robot en Arduino.

Cuando el robot esté encendido nos conectamos a éste mediante Bluetooth, seleccionandolo en la lista de dispositivos disponibles, entonces la venta se habilitará y pulsaremos sobre el botón “Init bot pheromones”, se sincronizarán las feromonas en el robot y éste comenzará con la ejecución. Acto seguido cerramos la conexión pinchando sobre el botón “Close Conection”.

Una vez complete la ejecución volverá al punto de inicio, pero esta vez orientado a la izquierda, como se muestra en la siguiente Figura:

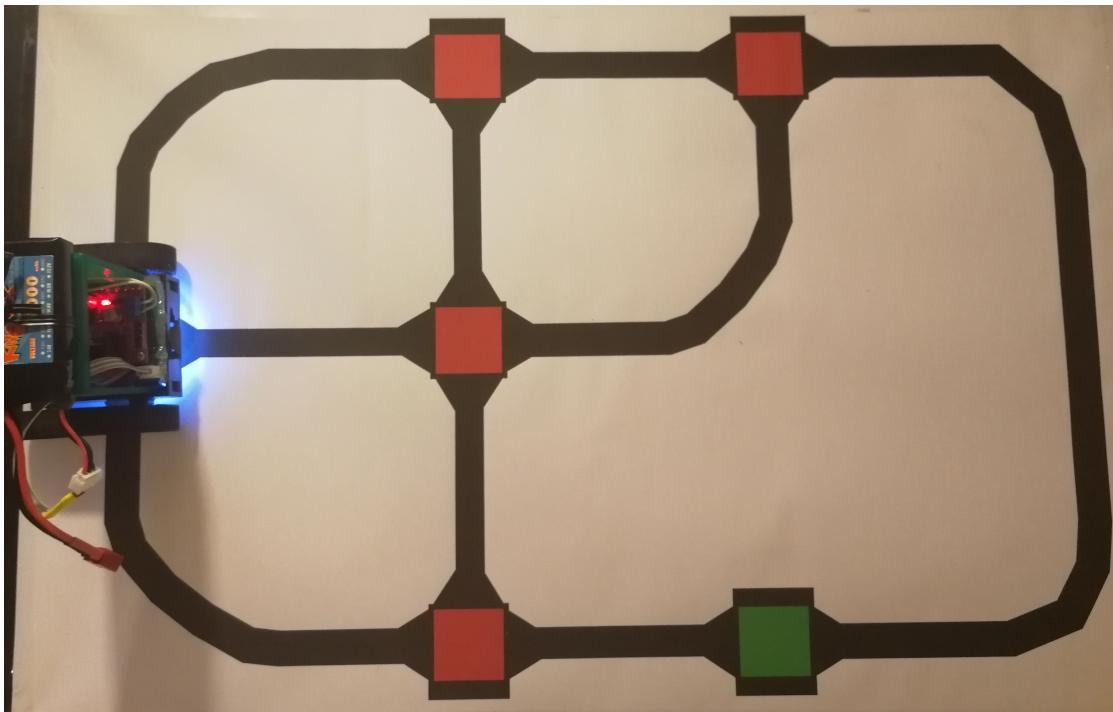


FIGURA 3.42: Estado final del robot en Arduino.

Ahora el robot, se quedará esperando a descargar la información captada en el servidor y hasta que no lo haga no continuará con su marcha, por ahora, este paso debe hacerse manualmente. Seguimos los pasos marcados a continuación:

- Conectamos mediante Bluetooth seleccionando el dispositivo (el robot).
- Pinchamos sobre el botón “Start Transmission” 2 veces, hasta que veamos que se ha descargado la información en la consola, tal como se muestra en la Figura 3.42.
- El robot, automáticamente continuará su marcha y tendremos que cerrar la conexión con el pinchando en el botón “Close Connection”.

Al final de cada iteración, el robot quedará en la misma posición que en la Figura 3.42

El resto de pasos son exactamente igual, solo habrá que descargar la información. Si el robot falla durante la ejecución se pulsará el interruptor de apagado, se volverá a encender y se resincronizará de nuevo.

Capítulo 4

Experimentación y resultados

En este Capítulo se exponen los distintos resultados obtenidos de tres maneras distintas:

- Teóricamente.
- Ejecución del [ACO](#) implementado en C++. (Código disponible en el Anexo [B.1.8.2](#)).
- Ejecución del [ACO](#) sobre el robot. (Código del servidor disponible en el Anexo [B.1.6](#) y código del robot disponible en el Anexo [B.1.5](#)).

Cada una de estas pruebas se realizarán bajo las mismas condiciones aunque en algunas variaran los parámetros de entrada. Por último, una vez obtenidos los resultados individualmente se contrastarán entre sí y se comentará la diferencia que hay entre estos, siendo el mismo algoritmo ejecutado en las mismas condiciones pero de distintas formas.

Se muestra a continuación el mapa sobre el que se realizarán todas las pruebas:

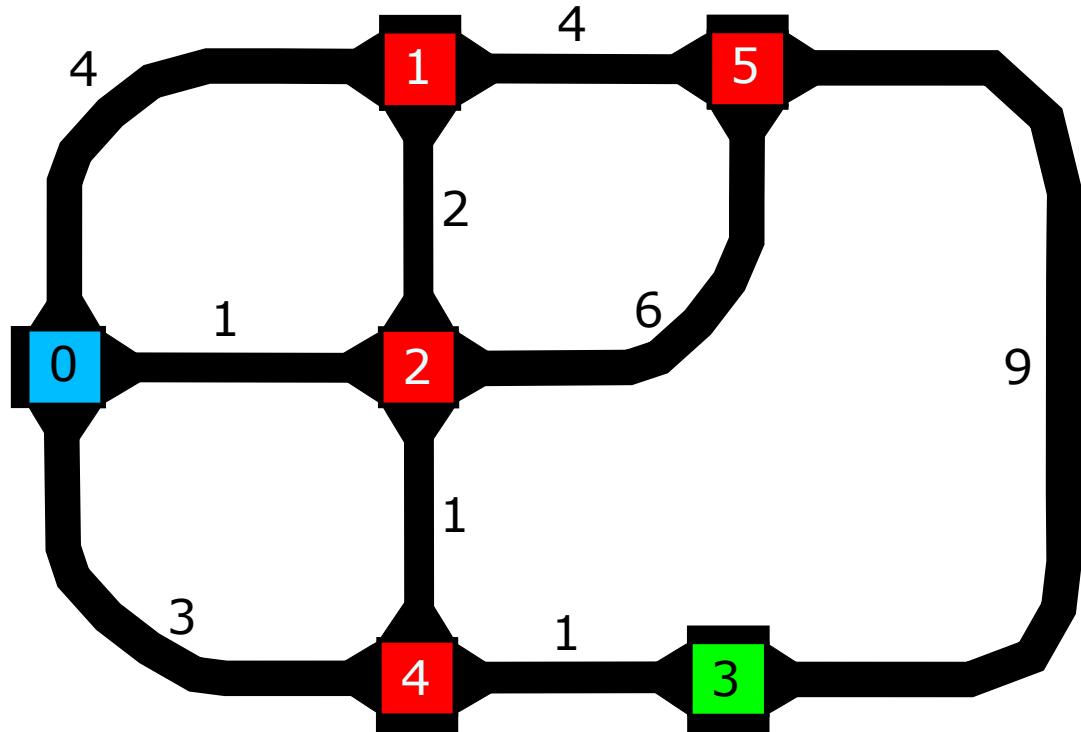


FIGURA 4.1: Mapa ponderado del experimento.

Puesto que en Arduino no podemos trabajar con potencias porque se nos irían unos números muy grandes, los pesos de Alpha y Beta que corresponden respectivamente a priorizar, o bien las feromonas, o bien el camino, no se utilizarán con la finalidad de que las 3 pruebas estén en las mismas condiciones, con lo cual se elevan a 1 ambas potencias.

La tasa de evaporación de las feromonas es igual a 0.001 para los 3 experimentos, excepto en Arduino que es de 0.01.

4.1. Experimento teórico del Algoritmo de Colonias de Hormigas

Basándonos en la Figura 4.1 simulamos manualmente las siguientes ejecuciones:

Nodo actual, Nodo 0.

$$P(0 \rightarrow 1) = \frac{(0.01) * (\frac{1}{4})}{(0.01) * (\frac{1}{4}) + (0.01) * (\frac{1}{1}) + (0.01) * (\frac{1}{3})} = 0.158 = 15.8\%$$

$$P(0-2) = \frac{(0,01)*(\frac{1}{4})}{(0,01)*(\frac{1}{4})+(0,01)*(\frac{1}{1})+(0,01)*(\frac{1}{3})} = 0.631 = 63.1\%$$

$$P(0-4) = \frac{(0,01)*(\frac{1}{3})}{(0,01)*(\frac{1}{4})+(0,01)*(\frac{1}{1})+(0,01)*(\frac{1}{3})} = 0.21 = 21\%$$

Seleccionamos aleatoriamente el nodo número 2 con un 63.1 % de probabilidad de tomarlo.

Nodo actual, Nodo 2.

$$P(2-1) = \frac{(0,01)*(\frac{1}{2})}{(0,01)*(\frac{1}{2})+(0,01)*(\frac{1}{6})+(0,01)*(\frac{1}{1})} = 0.3 = 30\%$$

$$P(2-5) = \frac{(0,01)*(\frac{1}{6})}{(0,01)*(\frac{1}{2})+(0,01)*(\frac{1}{6})+(0,01)*(\frac{1}{1})} = 0.1 = 10\%$$

$$P(2-4) = \frac{(0,01)*(\frac{1}{1})}{(0,01)*(\frac{1}{2})+(0,01)*(\frac{1}{6})+(0,01)*(\frac{1}{1})} = 0.6 = 60\%$$

Seleccionamos aleatoriamente el nodo número 4 con un 60 % de probabilidad de tomarlo.

Nodo actual, Nodo 4.

$$P(4-3) = \frac{(0,01)*(\frac{1}{1})}{(0,01)*(\frac{1}{1})} = 1 = 100\%$$

Llegamos a nuestra fuente de alimento que corresponde al Nodo 3.

Hemos tomado por tanto, el camino 0,2,4,3 con un coste igual a 3 unidades.

Con esto terminamos nuestra primera ejecución y damos pie a la segunda, ambas dentro de la misma iteración, es decir, no se ha producido aún la evaporación de feromonas.

Nodo actual, Nodo 0.

$$P(0-1) = 15.8\%$$

$$P(0-2) = 63.1\%$$

$$P(0-4) = 21\%$$

Seleccionamos aleatoriamente el nodo número 1 con un 15.8 % de probabilidad de tomarlo.

Nodo actual, Nodo 1.

$$P(1 - 2) = \frac{(0,01)*(\frac{1}{2})}{(0,01)*(\frac{1}{2}) + (0,01)*(\frac{1}{4})} = 0.66 = 66\%$$

$$P(1 - 5) = \frac{(0,01)*(\frac{1}{4})}{(0,01)*(\frac{1}{2}) + (0,01)*(\frac{1}{4})} = 0.33 = 33\%$$

Seleccionamos aleatoriamente el nodo número 5 con un 33 % de probabilidad de tomarlo.

Nodo actual, Nodo 5.

$$P(5 - 3) = \frac{(0,01)*(\frac{1}{9})}{(0,01)*(\frac{1}{9}) + (0,01)*(\frac{1}{6})} = 0.4 = 40\%$$

$$P(5 - 2) = \frac{(0,01)*(\frac{1}{6})}{(0,01)*(\frac{1}{9}) + (0,01)*(\frac{1}{6})} = 0.6 = 60\%$$

Seleccionamos aleatoriamente el nodo número 3 con un 40 % de probabilidad de tomarlo.

Llegamos a nuestro destino que corresponde al Nodo 3. Damos fin a la segunda ejecución.

El camino seleccionado por tanto, es el 0,1,5,3 con un coste total de 17 unidades.

Ambas ejecuciones descritas se han podido realizar concurrentemente. A continuación procedemos a realizar la actualización de feromonas.

Camino	$(1 - \rho)\sigma_{xy}$	Ejecución 1	Ejecución 2	Feromona
0-1	$(1-0.001) = 0.099$		1/17	0.1578
0-2	$(1-0.001) = 0.099$	1/3		0.4323
0-4	$(1-0.001) = 0.099$			0.099
1-5	$(1-0.001) = 0.099$		1/17	0.1578
1-2	$(1-0.001) = 0.099$			0.099
2-5	$(1-0.001) = 0.099$			0.099
2-4	$(1-0.001) = 0.099$	1/3		0.4323
4-3	$(1-0.001) = 0.099$	1/3		0.4323
5-3	$(1-0.001) = 0.099$		1/17	0.1578

TABLA 4.1: Actualización de feromonas, primera iteración.

Finalizada la primera iteración pasamos a la segunda, la cuál depende de los valores modificados en la primera iteración.

Nodo actual, Nodo 0.

$$P(0 - 1) = \frac{(0,1578)*(\frac{1}{4})}{(0,1578)*(\frac{1}{4}) + (0,4323)*(\frac{1}{1}) + (0,099)*(\frac{1}{3})} = 0.078 = 7.8\%$$

$$P(0 - 2) = \frac{(0,4323)*(\frac{1}{1})}{(0,1578)*(\frac{1}{4}) + (0,4323)*(\frac{1}{1}) + (0,099)*(\frac{1}{3})} = 0.856 = 85.6\%$$

$$P(0 - 4) = \frac{(0,099)*(\frac{1}{3})}{(0,1578)*(\frac{1}{4}) + (0,4323)*(\frac{1}{1}) + (0,099)*(\frac{1}{3})} = 0.065 = 6.5\%$$

Seleccionamos aleatoriamente el nodo número 2 con un 85.6% de probabilidad de tomarlo.

Nodo actual, Nodo 2.

$$P(2 - 1) = \frac{(0,099)*(\frac{1}{2})}{(0,099)*(\frac{1}{2}) + (0,099)*(\frac{1}{6}) + (0,4232)*(\frac{1}{1})} = 0.101 = 10.1\%$$

$$P(2 - 5) = \frac{(0,099)*(\frac{1}{6})}{(0,099)*(\frac{1}{2}) + (0,099)*(\frac{1}{6}) + (0,4232)*(\frac{1}{1})} = 0.033 = 3.3\%$$

$$P(2 - 4) = \frac{(0,4323)*(\frac{1}{1})}{(0,099)*(\frac{1}{2}) + (0,099)*(\frac{1}{6}) + (0,4323)*(\frac{1}{1})} = 0.865 = 86.5\%$$

Seleccionamos aleatoriamente el nodo número 5 con un 3.3% de probabilidad de tomarlo.

Nodo actual, Nodo 5.

$$P(5 - 1) = \frac{(0,1578)*(\frac{1}{4})}{(0,1578)*(\frac{1}{4}) + (0,1578)*(\frac{1}{9})} = 0.6923 = 69.23\%$$

$$P(5 - 3) = \frac{(0,1578)*(\frac{1}{9})}{(0,1578)*(\frac{1}{4}) + (0,1578)*(\frac{1}{9})} = 0.3076 = 30.76\%$$

Seleccionamos aleatoriamente el nodo número 3 con un 30.76% de probabilidad de tomarlo.

Hemos tomado el camino 0,2,5,3 con un coste total de 16 unidades

Nodo actual, Nodo 0.

$$P(0-1) = 7.8\%$$

$$P(0-2) = 85.6\%$$

$$P(0-4) = 6.5\%$$

Seleccionamos aleatoriamente el nodo número 2 con un 85.6 % de probabilidad de tomarlo.

Nodo actual, Nodo 2.

$$P(2-1) = 10.1\%$$

$$P(2-5) = 3.3\%$$

$$p(2-4) = 86.5\%$$

Seleccionamos aleatoriamente el nodo número 4 con un 86.5 % de probabilidad de tomarlo.

Nodo actual, Nodo 4.

$$P(4 - 2) = \frac{(0,4323) * (\frac{1}{1})}{(0,4323) * (\frac{1}{1}) + (0,4323) * (\frac{1}{1})} = 0.5 = 50\%$$

$$P(4 - 3) = \frac{(0,4323) * (\frac{1}{1})}{(0,4323) * (\frac{1}{1}) + (0,4323) * (\frac{1}{1})} = 0.5 = 50\%$$

Seleccionamos aleatoriamente el nodo número 3 con un 50 % de probabilidad de tomarlo.

Finalmente, llegamos al destino, habiendo tomado el camino 0,2,4,3 y completamos la última ejecución pasando así a la actualización de feromonas.

Camino	(1-ro)sigmaxy	Ejecución 1	Ejecución 2	Feromona
0-1	$(1-0.001) * 0.1578 = 0.01562$			0.0156
0-2	$(1-0.001) * 0.4323 = 0.04279$	1/16	1/3	0.4385
0-4	$(1-0.001) * 0.099 = 0.0098$			0.0098
1-5	$(1-0.001) * 0.1578 = 0.01562$			0.0156
1-2	$(1-0.001) * 0.099 = 0.0098$			0.0098
2-5	$(1-0.001) * 0.099 = 0.0098$	1/16		0.0723
2-4	$(1-0.001) * 0.4323 = 0.04279$		1/3	0.3761
4-3	$(1-0.001) * 0.4323 = 0.04279$		1/3	0.3760
5-3	$(1-0.001) * 0.1578 = 0.01562$	1/16		0.0781

TABLA 4.2: Actualización de feromonas, segunda iteración.

Procedemos ahora a contrastar la información de ambas iteraciones basándonos en los resultados obtenidos en la Tabla 4.1 y en la Tabla 4.2.

Camino	Feromonas de la iteración 1	Feromonas de la iteración 2
0-1	0.1578	0.0156
0-2	0.4323	0.4385
0-4	0.099	0.0098
1-5	0.1578	0.0156
1-2	0.099	0.0098
2-5	0.099	0.0723
2-4	0.4323	0.3761
4-3	0.4323	0.3760
5-3	0.1578	0.0781

TABLA 4.3: Resultado de feromonas obtenidas de las iteraciones 1, en la Tabla 4.1 y 2, en la Tabla 4.2.

Si entramos en detalle de los resultados obtenidos podemos observar pérdidas de probabilidad cuando pasamos de la iteración 1 a la iteración 2. Esto puede resultar engañoso puesto que mientras la diferencia de probabilidad (probabilidad máxima - probabilidad mínima) en una misma iteración es de 33.33 % en la primera iteración, en la segunda es de un 42.87 % con lo

cual nos está indicando que los caminos visitados se están fortificando más con respecto a los no visitados de manera relativa.

Adicionalmente, mostramos un histograma de ambas iteraciones:

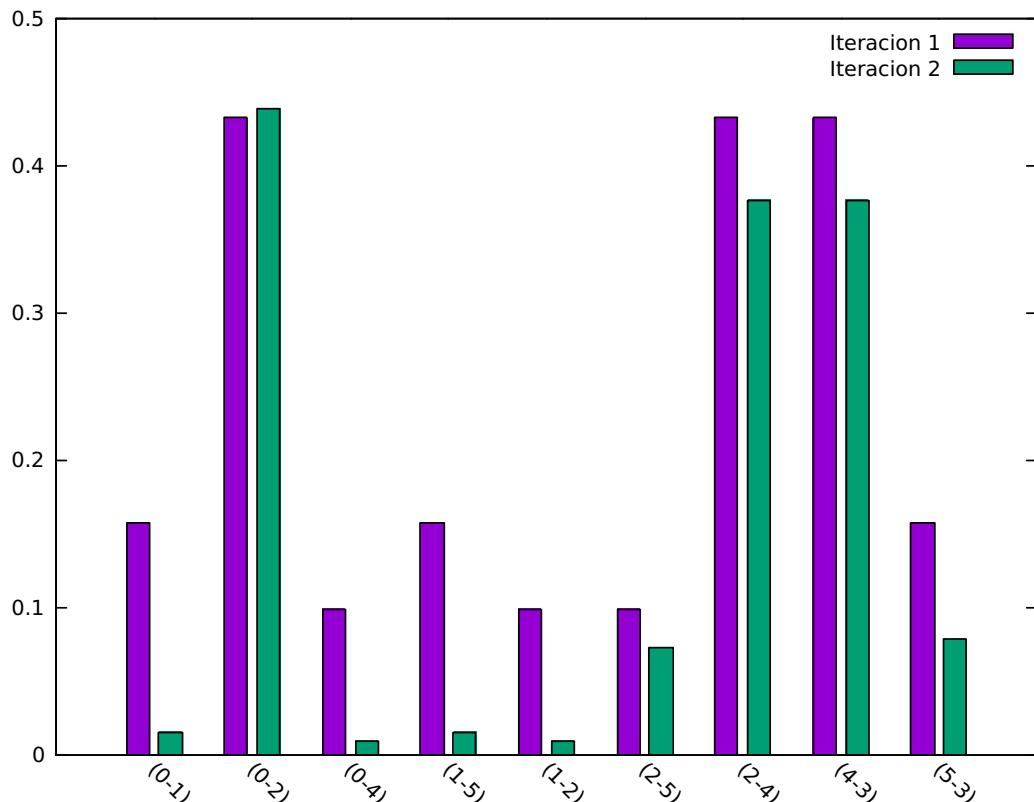


FIGURA 4.2: Gráfica referente a los datos obtenidos en la Tabla 4.3.

En la gráfica descrita en la Figura 4.2 se puede observar gráficamente el efecto de la evaporación de feromonas siendo menor la intensidad en la barra verde que corresponde a la segunda iteración. Además también se puede ver como el camino mejor camino y que más hormigas han tomado destaca del resto.

Puesto que son muy pocas iteraciones, y completar una iteración de manera teórica es laborioso, contrastaremos la información obtenida con una pequeña ejecución en C++ en la Sección 4.5 ya que no podemos sacar buenas conclusiones en tan solo dos iteraciones.

4.2. Experimento del Algoritmo de Colonias de Hormigas en C++

Basándonos en el mapa que ofrece la Figura 4.1 procedemos a realizar una ejecución de dos iteraciones en C++ con el algoritmo comentado en la Sección 3.2.

Después de dos iteraciones obtenemos el siguiente resultado:

	0	1	2	3	4	5
0	INFI	4.00	1.00	INFI	3.00	INFI
1	4.00	INFI	2.00	INFI	INFI	4.00
2	1.00	2.00	INFI	INFI	1.00	6.00
3	INFI	INFI	INFI	INFI	1.00	9.00
4	3.00	INFI	1.00	1.00	INFI	INFI
5	INFI	4.00	6.00	9.00	INFI	INFI

*****FEROMONAS*****						
	0	1	2	3	4	5
0	0.099	0.098	0.098	0.098	0.598	0.098
1	0.098	0.099	0.098	0.098	0.098	0.098
2	0.098	0.098	0.099	0.098	0.098	0.098
3	0.098	0.098	0.098	0.099	0.598	0.098
4	0.598	0.098	0.098	0.598	0.099	0.098
5	0.098	0.098	0.098	0.098	0.098	0.099

*****GRAFO*****						
	0	1	2	3	4	5
0	INFI	4.00	1.00	INFI	3.00	INFI
1	4.00	INFI	2.00	INFI	INFI	4.00
2	1.00	2.00	INFI	INFI	1.00	6.00
3	INFI	INFI	INFI	INFI	1.00	9.00
4	3.00	INFI	1.00	1.00	INFI	INFI
5	INFI	4.00	6.00	9.00	INFI	INFI

*****FEROMONAS*****						
	0	1	2	3	4	5
0	0.098	0.096	0.096	0.096	0.836	0.096
1	0.096	0.098	0.096	0.096	0.096	0.096
2	0.096	0.096	0.098	0.096	0.096	0.096
3	0.096	0.096	0.096	0.098	0.836	0.096
4	0.836	0.096	0.096	0.836	0.098	0.096
5	0.096	0.096	0.096	0.096	0.096	0.098

Camino recorrido: 0 -> 4 -> 3
Coste del camino: 4.000

FIGURA 4.3: Ejecución obtenida de lanzar 2 hormigas durante dos períodos de tiempo.

Como podemos observar, al ejecutar en un equipo el ACO, 2 iteraciones son muy pocas para que encuentre un buen camino, aún así no ha cogido por el peor camino ni de lejos, siendo este la segunda mejor opción a tomar.

Traduciremos a una tabla mas legible los resultados obtenidos en la Figura 4.3:

Camino	Feromona en iteración 1	Feromona en iteración 2
0-1	0.098	0.096
0-2	0.098	0.096
0-4	0.598	0.836
1-5	0.098	0.096
1-2	0.098	0.096
2-5	0.098	0.096
2-4	0.098	0.096
4-3	0.598	0.836
5-3	0.098	0.096

TABLA 4.4: Resultado obtenido de ejecutar 2 hormigas en 2 iteraciones en C++, datos recogidos de la Figura 4.3.

A continuación se muestra una gráfica de la ejecución de los resultados de la Tabla 4.4:

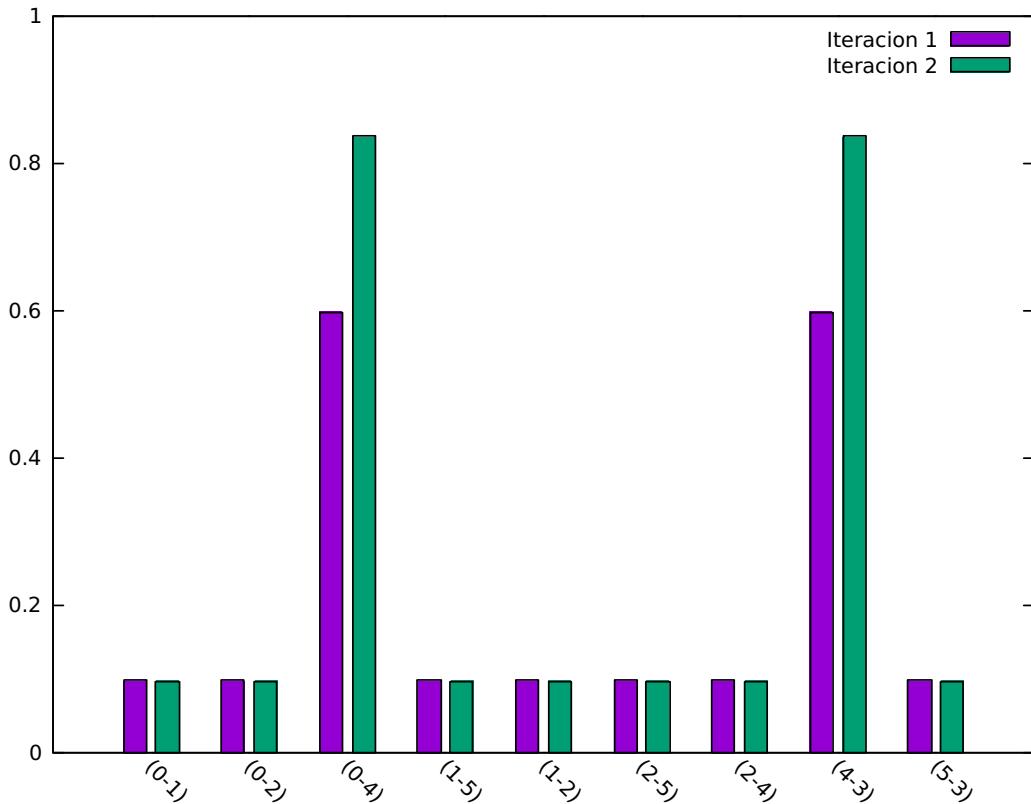


FIGURA 4.4: Gráfica obtenida a partir de los resultados de la Tabla 4.4.

Podemos observar que el camino preferido en este caso por las hormigas ha sido el 0-4-3, reforzando bastante las feromonas de la primera a la segunda iteración, mientras los caminos no visitados, se van evaporando muy lentamente.

Estos datos obtenidos se contrastarán en la Sección 4.5 junto con los datos de la ejecución teórica realizada en la Sección 4.1.

A continuación se hace un estudio más extenso de varias ejecuciones del ACO en C++.

4.2.1. 10, 100, 1000, 10000 iteraciones con 2 hormigas

A continuación se proceden a mostrar los datos obtenidos tras lanzar 2 hormigas al mapa que presenta la Figura 4.1 durante 10 iteraciones, 100 iteraciones, 1000 iteraciones y 10000 iteraciones y a realizar una comparativa entre los resultados obtenidos.

Lanzamos el ACO para dos hormigas y 10 iteraciones y obtenemos los siguientes resultados:

*****GRAFO*****

	0	1	2	3	4	5
0	INFI	4.00	1.00	INFI	3.00	INFI
1	4.00	INFI	2.00	INFI	INFI	4.00
2	1.00	2.00	INFI	INFI	1.00	6.00
3	INFI	INFI	INFI	INFI	1.00	9.00
4	3.00	INFI	1.00	1.00	INFI	INFI
5	INFI	4.00	6.00	9.00	INFI	INFI

*****FEROMONAS*****

	0	1	2	3	4	5
0	0.090	0.082	1.678	0.082	1.177	0.082
1	0.082	0.090	0.082	0.082	0.082	0.082
2	1.678	0.082	0.090	0.082	1.616	0.144
3	0.082	0.082	0.082	0.090	2.712	0.144
4	1.177	0.082	1.616	2.712	0.090	0.082
5	0.082	0.082	0.144	0.144	0.082	0.090

Camino recorrido: 0 -> 2 -> 4 -> 3

Coste del camino: 3.000

FIGURA 4.5: Ejecución del ACO de dos hormigas durante diez iteraciones.

Seguidamente ejecutaremos el ACO para dos hormigas durante 100 iteraciones:

*****GRAFO*****

	0	1	2	3	4	5
0	INFI	4.00	1.00	INFI	3.00	INFI
1	4.00	INFI	2.00	INFI	INFI	4.00
2	1.00	2.00	INFI	INFI	1.00	6.00
3	INFI	INFI	INFI	INFI	1.00	9.00
4	3.00	INFI	1.00	1.00	INFI	INFI
5	INFI	4.00	6.00	9.00	INFI	INFI

*****FEROMONAS*****

	0	1	2	3	4	5
0	0.037	0.013	0.013	0.013	10.927	0.013
1	0.013	0.037	0.013	0.013	0.013	0.013
2	0.013	0.013	0.037	0.013	0.013	0.013
3	0.013	0.013	0.013	0.037	10.927	0.013
4	10.927	0.013	0.013	10.927	0.037	0.013
5	0.013	0.013	0.013	0.013	0.013	0.037

Camino recorrido: 0 -> 4 -> 3

Coste del camino: 4.000

FIGURA 4.6: Ejecución del ACO de dos hormigas durante cien iteraciones.

Ahora haremos lo mismo pero durante 1000 iteraciones:

*****GRAFO*****

	0	1	2	3	4	5
0	INFI	4.00	1.00	INFI	3.00	INFI
1	4.00	INFI	2.00	INFI	INFI	4.00
2	1.00	2.00	INFI	INFI	1.00	6.00
3	INFI	INFI	INFI	INFI	1.00	9.00
4	3.00	INFI	1.00	1.00	INFI	INFI
5	INFI	4.00	6.00	9.00	INFI	INFI

*****FEROMONAS*****

	0	1	2	3	4	5
0	0.000	0.000	16.750	0.000	0.000	0.000
1	0.000	0.000	0.000	0.000	0.000	0.000
2	16.750	0.000	0.000	0.000	16.750	0.000
3	0.000	0.000	0.000	0.000	16.750	0.000
4	0.000	0.000	16.750	16.750	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000	0.000

Camino recorrido: 0 -> 2 -> 4 -> 3

Coste del camino: 3.000

FIGURA 4.7: Ejecución del ACO de dos hormigas durante mil iteraciones.

Y para terminar la misma ejecución para 10000 iteraciones:

```
*****
*****GRAFO*****
    0      1      2      3      4      5
0  INFI   4.00   1.00  INFI   3.00  INFI
1  4.00   INFI   2.00  INFI   INFI   4.00
2  1.00   2.00   INFI   INFI   1.00   6.00
3  INFI   INFI   INFI   INFI   1.00   9.00
4  3.00   INFI   1.00   1.00  INFI   INFI
5  INFI   4.00   6.00   9.00  INFI   INFI

*****
*****FEROMONAS*****
    0      1      2      3      4      5
0  0.000  0.000  16.750  0.000  0.000  0.000
1  0.000  0.000  0.000  0.000  0.000  0.000
2  16.750 0.000  0.000  0.000  16.750  0.000
3  0.000  0.000  0.000  0.000  16.750  0.000
4  0.000  0.000  16.750  16.750  0.000  0.000
5  0.000  0.000  0.000  0.000  0.000  0.000

Camino recorrido: 0 -> 2 -> 4 -> 3
Coste del camino: 3.000
```

FIGURA 4.8: Ejecución del ACO de dos hormigas durante diez mil iteraciones.

Una vez hecho esto, reduciremos los resultados obtenidos a la siguiente tabla:

Camino	2h10i	2h100i	2h1000i	2h10000i
0-1	0.082	0.013	0	0
0-2	1.678	0.013	16.750	16.750
0-4	1.177	10.927	0	0
1-5	0.082	0.013	0	0
1-2	0.082	0.013	0	0
2-5	0.144	0.013	0	0
2-4	1.616	0.013	16.750	16.750
4-3	2.712	10.927	16.750	16.750
5-3	0.144	0.013	0	0

TABLA 4.5: Recopilación de resultados de las ejecuciones anteriores.

Realizamos un histograma en la siguiente figura de los datos obtenidos en la Tabla 4.5:

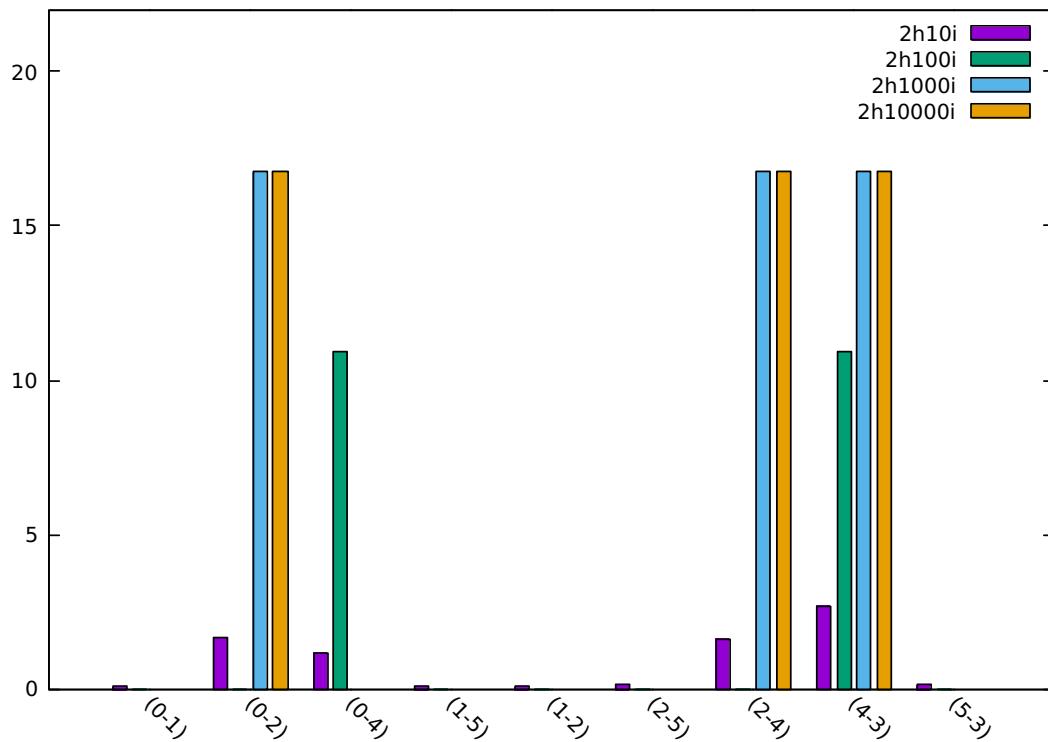


FIGURA 4.9: Gráfica referente a los datos obtenidos en la Tabla 4.5.

Como podemos observar, los caminos más transcurridos son:

- (0-2).
- (2-4).
- (4-3).

Correspondientes al mejor camino posible en este grafo. A continuación deducimos lo siguiente de la gráfica descrita en la Figura 4.9

Adicionalmente vemos algunos picos con 100 iteraciones que han tomado distinto camino (0-4-3) con un coste de 4 unidades. Esto no es incorrecto puesto que, como sabemos, el **ACO** es probabilístico y esta probabilidad se va haciendo más intensa a medida que las hormigas pasan por los nodos correspondientes. Se puede apreciar que al principio muchas hormigas tomaron este camino y la probabilidad de tomar dichos nodos fue aumentando por cada iteración.

Podemos observar también que cuantas más iteraciones se realicen, más claro se tiene el camino, es decir, menos probabilidad de que se vaya por una variante tiene, ya que, la evaporación va causando efecto llegando a hacer que la probabilidad sea mínima, tanto que el tipo de dato que alberga la probabilidad de la feromona, es decir, un número en coma flotante acaba despreciando dichos decimales por falta de precisión, si usáramos un número en coma flotante de precisión doble, acabaría ocurriendo lo mismo, simplemente que más tarde.

Otra apreciación que podemos ver, es que en ningún caso, excepto en el de 10 iteraciones alguna vez, se ha tomado el camino más largo que recorrería por el Nodo 5 para llegar al 3, con un coste de 9 unidades por tomar dicho camino. Simétricamente en el inicio, vemos que los adyacentes del nodo de partida, es decir, el Nodo 0 apenas se toma el peor camino que corresponde a pasar por el Nodo 1, siendo este el más improbable de tomar de los 3 adyacentes posibles.

4.2.2. 10, 100, 1000, 10000 iteraciones con 6 hormigas

Seguidamente procedemos a mostrar los datos obtenidos tras lanzar 6 hormigas al mapa que se presenta la Figura 4.1 durante 10 iteraciones, 100 iteraciones, 1000 iteraciones y 10000 iteraciones y a realizar una comparativa entre los resultados obtenidos.

Lanzamos pues, el ACO para seis hormigas y 10 iteraciones y obtenemos los siguientes resultados:

```
*****GRAFO*****
    0      1      2      3      4      5
0  INFI   4.00   1.00   INFI   3.00   INFI
1  4.00   INFI   2.00   INFI   INFI   4.00
2  1.00   2.00   INFI   INFI   1.00   6.00
3  INFI   INFI   INFI   INFI   1.00   9.00
4  3.00   INFI   1.00   1.00   INFI   INFI
5  INFI   4.00   6.00   9.00   INFI   INFI

*****FEROMONAS*****
    0      1      2      3      4      5
0  0.090  0.082  0.082  0.082  11.728  0.082
1  0.082  0.090  0.082  0.082  0.082  0.082
2  0.082  0.082  0.090  0.082  0.082  0.082
3  0.082  0.082  0.082  0.090  11.728  0.082
4  11.728 0.082  0.082  11.728  0.090  0.082
5  0.082  0.082  0.082  0.082  0.082  0.090

Camino recorrido: 0 -> 4 -> 3
Coste del camino: 4.000
```

FIGURA 4.10: Ejecución del ACO de seis hormigas después de diez iteraciones.

Seguidamente ejecutaremos el ACO para seis hormigas durante 100 iteraciones.

```
*****GRAFO*****
    0      1      2      3      4      5
0  INFI   4.00  1.00  INFI   3.00  INFI
1  4.00  INFI   2.00  INFI   INFI   4.00
2  1.00  2.00  INFI   INFI   1.00  6.00
3  INFI   INFI   INFI   INFI   1.00  9.00
4  3.00  INFI   1.00  1.00  INFI   INFI
5  INFI   4.00   6.00  9.00  INFI   INFI

*****FEROMONAS*****
    0      1      2      3      4      5
0  0.037  0.013  0.060  0.013  54.411  0.013
1  0.013  0.037  0.013  0.013  0.013  0.013
2  0.060  0.013  0.037  0.013  0.060  0.013
3  0.013  0.013  0.013  0.037  54.457  0.013
4  54.411  0.013  0.060  54.457  0.037  0.013
5  0.013  0.013  0.013  0.013  0.013  0.037
```

Camino recorrido: 0 -> 4 -> 3

Coste del camino: 4.000

FIGURA 4.11: Ejecución del ACO de seis hormigas después de cien iteraciones.

Ahora lo haremos para 6 hormigas y 1000 iteraciones.

```
*****GRAFO*****
    0      1      2      3      4      5
0  INFI   4.00   1.00  INFI   3.00  INFI
1  4.00   INFI   2.00  INFI   INFI   4.00
2  1.00   2.00   INFI  INFI   1.00   6.00
3  INFI   INFI   INFI  INFI   1.00   9.00
4  3.00   INFI   1.00  INFI   INFI   INFI
5  INFI   4.00   6.00   9.00  INFI   INFI

*****FEROMONAS*****
    0      1      2      3      4      5
0  0.000  0.000  0.000  0.000  62.814 0.000
1  0.000  0.000  0.000  0.000  0.000  0.000
2  0.000  0.000  0.000  0.000  0.000  0.000
3  0.000  0.000  0.000  0.000  62.814 0.000
4  62.814 0.000  0.000  62.814  0.000  0.000
5  0.000  0.000  0.000  0.000  0.000  0.000
```

Camino recorrido: 0 -> 4 -> 3

Coste del camino: 4.000

FIGURA 4.12: Ejecución del ACO de seis hormigas después de mil iteraciones.

Y por último lo haremos para 6 hormigas y 10000 iteraciones.

```
*****GRAFO*****
    0      1      2      3      4      5
0  INFI   4.00   1.00  INFI   3.00  INFI
1  4.00   INFI   2.00  INFI   INFI   4.00
2  1.00   2.00  INFI   INFI   1.00   6.00
3  INFI   INFI   INFI   INFI   1.00   9.00
4  3.00   INFI   1.00   1.00  INFI   INFI
5  INFI   4.00   6.00   9.00  INFI   INFI

*****FEROMONAS*****
    0      1      2      3      4      5
0  0.000  0.000  83.753  0.000  0.000  0.000
1  0.000  0.000  0.000  0.000  0.000  0.000
2  83.753  0.000  0.000  0.000  83.753  0.000
3  0.000  0.000  0.000  0.000  83.753  0.000
4  0.000  0.000  83.753  83.753  0.000  0.000
5  0.000  0.000  0.000  0.000  0.000  0.000

Camino recorrido: 0 -> 2 -> 4 -> 3
Coste del camino: 3.000
```

FIGURA 4.13: Ejecución del ACO de seis hormigas después de diez mil iteraciones.

Procedemos a continuación a realizar una recopilación de las ejecuciones anteriores:

Camino	6h10i	6h100i	6h1000i	6h10000i
0-1	0.082	0.013	0	0
0-2	0.082	0.060	0	83.753
0-4	11.728	54.411	62.814	0
1-5	0.082	0.013	0	0
1-2	0.082	0.013	0	0
2-5	0.082	0.013	0	0
2-4	0.082	0.060	0	83.753
4-3	11.728	54.457	62.814	83.753
5-3	0.082	0.013	0	0

TABLA 4.6: Recopilación de resultados de las ejecuciones anteriores.

Mostramos los resultados obtenidos en la Tabla 4.6 en la siguiente gráfica:

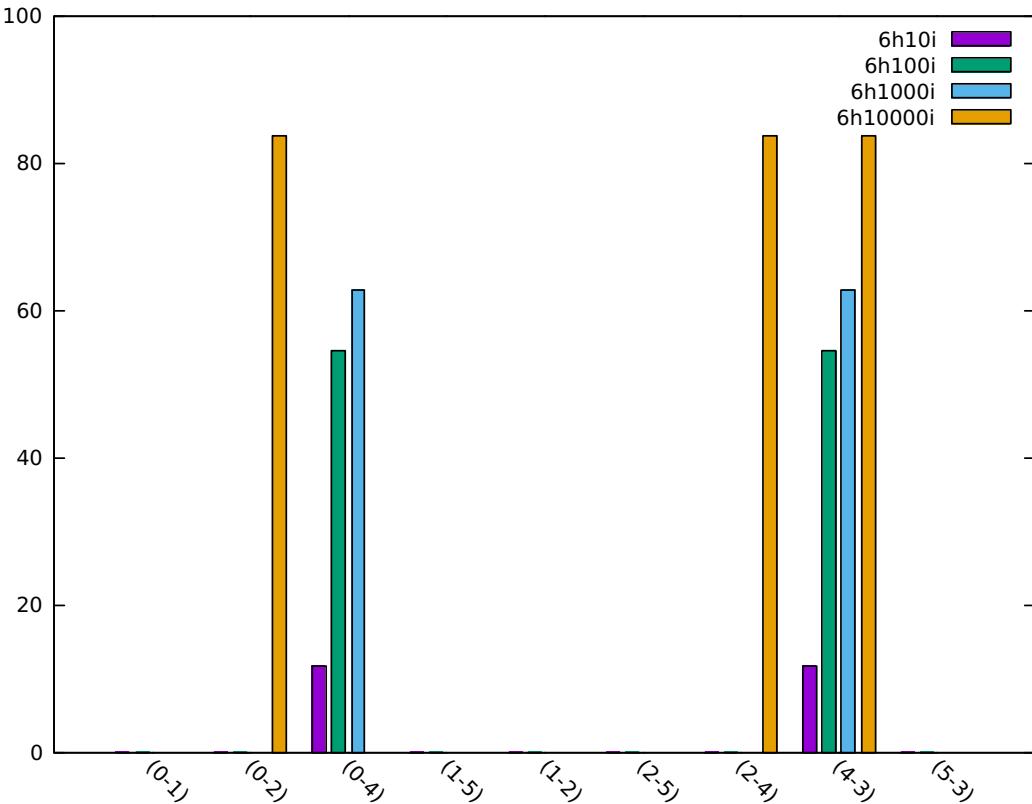


FIGURA 4.14: Gráfica de resultados de ejecutar seis hormigas tras varias iteraciones.

En la gráfica que muestra la Figura 4.14 podemos observar un comportamiento radical, más si cabe que en la gráfica mostrada en la Figura 4.9, las feromonas tienen un valor más alto puesto que pasan más hormigas por los nodos, aunque podemos observar también un dato curioso y es que el camino preferido por las hormigas no coincide con el óptimo a excepción de la ejecución con 10000 iteraciones, pero esto no significa que esté mal diseñado el algoritmo, simplemente en las ejecuciones arbitrarias lanzadas las hormigas han mostrado el resultado de dicho camino.

vemos que la evaporación de la feromona es mucho más dura que con dos hormigas, y deducimos también que al incrementar el número de hormigas, tanto la evaporación, como el aumento de probabilidad se produce antes.

También podemos apreciar que prácticamente no se ha tomado un camino malo, el algoritmo tiende a escoger en la mayoría de los casos los mejores, aunque los otros, en menos probabilidad, podrían tomarse.

Adicionalmente, podemos observar la intensidad que tiene la feromonía dependiendo incluso, del camino que vaya a tomar, se expone aquí una comparativa de dos ejecuciones, una la que se muestra en la Figura 4.13 que corresponde a 10000 iteraciones con 6 hormigas y la que se muestra a continuación con las mismas condiciones de lanzamiento:

```
*****GRAFO*****
    0      1      2      3      4      5
0  INFI   4.00   1.00  INFI   3.00  INFI
1  4.00   INFI   2.00  INFI   INFI   4.00
2  1.00   2.00  INFI   INFI   1.00   6.00
3  INFI   INFI   INFI   INFI   1.00   9.00
4  3.00   INFI   1.00   1.00  INFI   INFI
5  INFI   4.00   6.00   9.00  INFI   INFI

*****FEROMONAS*****
    0      1      2      3      4      5
0  0.000  0.000  0.000  0.000  62.814 0.000
1  0.000  0.000  0.000  0.000  0.000  0.000
2  0.000  0.000  0.000  0.000  0.000  0.000
3  0.000  0.000  0.000  0.000  62.814 0.000
4  62.814 0.000  0.000  62.814  0.000  0.000
5  0.000  0.000  0.000  0.000  0.000  0.000

Camino recorrido: 0 -> 4 -> 3
Coste del camino: 4.000
```

FIGURA 4.15: Resultados de ejecutar seis hormigas tras diez mil iteraciones.

Si recopilamos dicha información obtenemos la siguiente tabla:

Camino	Ejecución 1	Ejecución 2
0-1	0	0
0-2	83.753	0
0-4	0	62.814
1-5	0	0
1-2	0	0
2-5	0	0
2-4	83.753	0
4-3	83.753	62.814
5-3	0	0

TABLA 4.7: Recopilación de resultados tras dos ejecuciones del mismo programa, con seis hormigas y diez mil iteraciones.

Exponemos una gráfica de los datos recopilados en la Tabla 4.7:

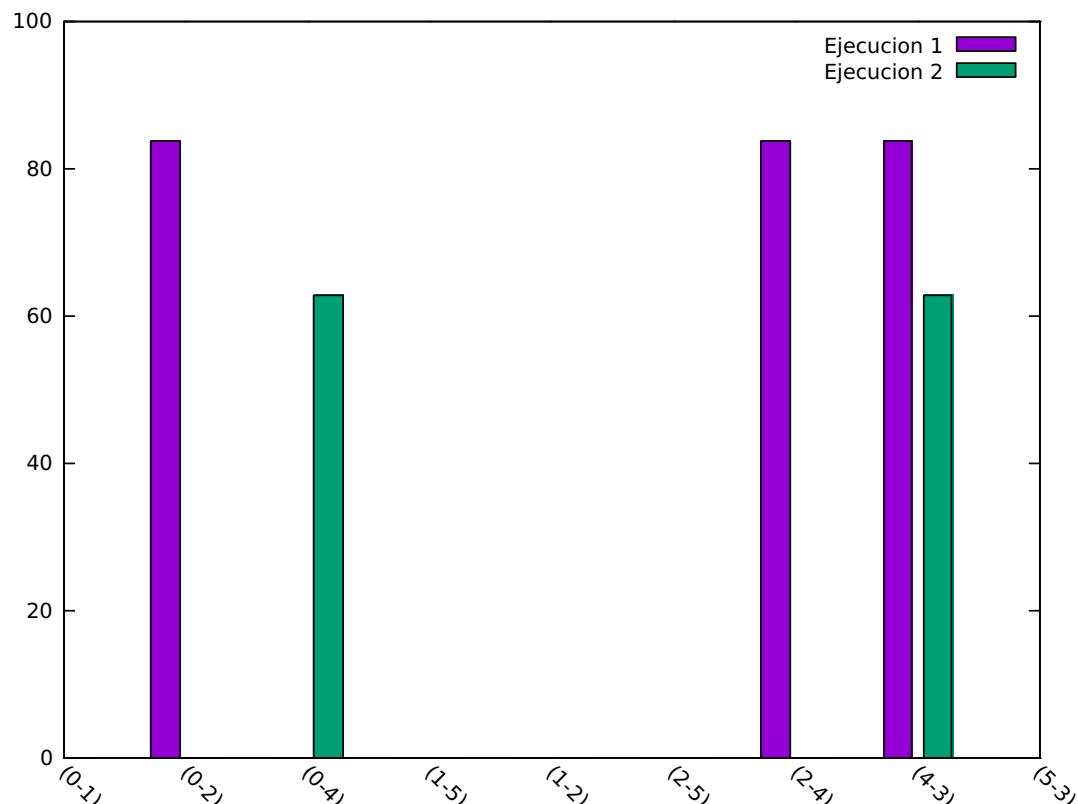
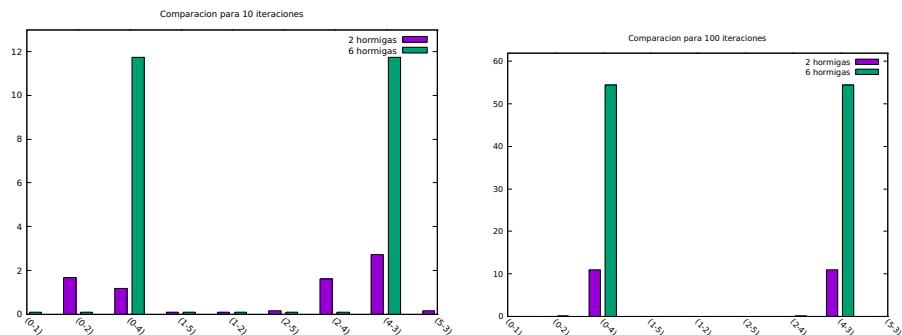


FIGURA 4.16: Gráfica de resultados recopilados en la Tabla 4.7.

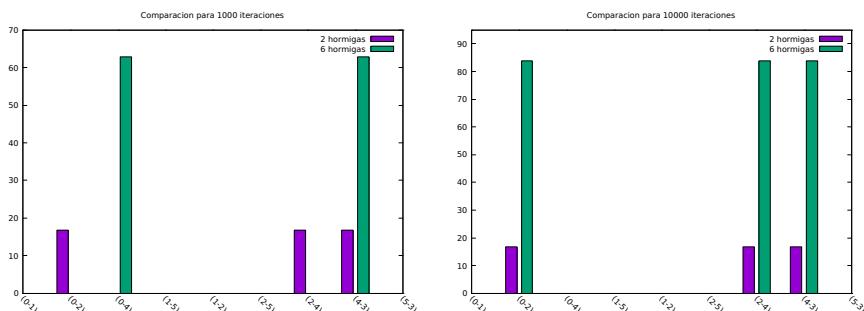
Podemos observar tras dos ejecuciones idénticas que las hormigas se han decantado por distintos caminos, en la primera ejecución se ha tomado el camino 0-2-4-3, mientras que en la segunda se tomó el camino 0-4-3 siendo este último más largo. Al ser más largo el segundo camino podemos observar que el valor de la feromona que se suelta en cada nodo visitado es más baja que en la primera ejecución, esto es también un punto fuerte de este algoritmo, el que tenga más peso el mejor camino que se toma.

4.3. Contraste de resultado al lanzar dos y seis hormigas

A continuación vamos a hacer una pequeña comparativa de los resultados obtenidos en la Sección 4.2.1 y en la Sección 4.2.2. Tomando datos de las secciones anteriormente mencionadas, vamos a generar varias gráficas y a comentarlas.



(a) Comparativa de 10 iteraciones con 2 y 6 hormigas. (b) Comparativa de 100 iteraciones con 2 y 6 hormigas.



(c) Comparativa de 1000 iteraciones con 2 y 6 hormigas. (d) Comparativa de 10000 iteraciones con 2 y 6 hormigas.

FIGURA 4.17: Comparativa entre varias ejecuciones con 2 hormigas y 6 hormigas

Contrastamos a continuación las gráficas descritas en la Figura 4.17:

- Podemos observar en la Figura 4.17(a) una gran diferencia de intensidad en las feromonas de ambas, vemos como en pequeña medida, pero perceptible, se pasa por todos los nodos del mapa, pero aunque esto sea así, prevalece la intensidad en los caminos mas cortos.
- Para 100 iteraciones, correspondiente a la Figura 4.17(b) vemos como se pierde el rastro de los caminos menos visitados, siendo prácticamente el camino elegido por ambas ejecuciones el 0-4-3.
- Con 1000 iteraciones, correspondiente a la Figura 4.17(d), da la casualidad de que ambas ejecuciones toman caminos distintos, pero vemos que con 2 hormigas, al ser mejor camino el que ha tomado, respecto a la ejecución de 6 hormigas relativamente la fuerza de la feromona es más intensa que en la ejecución con 6 hormigas, pero en cuanto a intensidad absoluta la ejecución de 6 hormigas es mayor puesto que están recorriendo el mapa el triple de hormigas.
- Por último podemos observar en la Figura 4.17(d) que las hormigas en ambos casos, han tomado el mismo camino, vemos en igualdad de condiciones la intensidad de las feromonas de una ejecución respecto a la otra, siendo el nivel de evaporación prácticamente equivalente para ambas ejecuciones.

Concluimos pues la Sección y procedemos al último experimento en nuestro robot real.

4.4. Experimento del proyecto en Arduino

Procedemos pues a recoger datos sobre el robot realizado en Arduino.

Primero tomaremos datos tras dos iteraciones, la primera, con una matriz inicial, y la segunda sobre los resultados de la primera matriz.

Inicialmente cada feromona vale 0.99, puesto que la tasa de evaporación es igual a 0.01 como observamos en la fórmula de evaporación de feromonas [2.1.3.2](#) $(1 - 0.01) = 0.99$. Tras la primera iteración nos queda la siguiente tabla de feromonas en nuestro programa.

		PHEROMONES MATRIX					
		0	1	2	3	4	5
		0 0.98	0.97	1.08	0.97	0.97	0.97
		1 0.97	0.98	0.97	0.97	0.97	0.97
		2 1.08	0.97	0.98	0.97	1.08	0.97
		3 0.97	0.97	0.97	0.98	1.08	0.97
		4 0.97	0.97	1.08	1.08	0.98	0.97
		5 0.97	0.97	0.97	0.97	0.97	0.98

FIGURA 4.18: Resultados obtenidos de los robot en Arduino tras 1 iteración.

Sobre la iteración mostrada en la Figura 4.18, volvemos a realizar otra iteración y nos queda la siguiente tabla de feromonas.

		PHEROMONES MATRIX					
		0	1	2	3	4	5
		0 0.97	0.95	1.15	0.95	0.95	0.95
		1 0.95	0.97	0.95	0.95	0.95	0.95
		2 1.15	0.95	0.97	0.95	1.15	0.95
		3 0.95	0.95	0.95	0.97	1.15	0.95
		4 0.95	0.95	1.15	1.15	0.97	0.95
		5 0.95	0.95	0.95	0.95	0.95	0.97

FIGURA 4.19: Resultados obtenidos de los robot en Arduino tras 2 iteraciones.

De éstos datos, obtenemos la siguiente tabla:

Camino	Iteración 1	Iteración 2
0-1	0.97	0.95
0-2	1.08	1.15
0-4	0.97	0.95
1-5	0.97	0.95
1-2	0.97	0.95
2-5	0.97	0.95
2-4	1.08	1.15
4-3	1.08	1.15
5-3	0.97	0.95

TABLA 4.8: Recopilación de resultados tras dos iteraciones en Arduino.

A continuación se muestra el histograma de los datos de la Tabla 4.8.

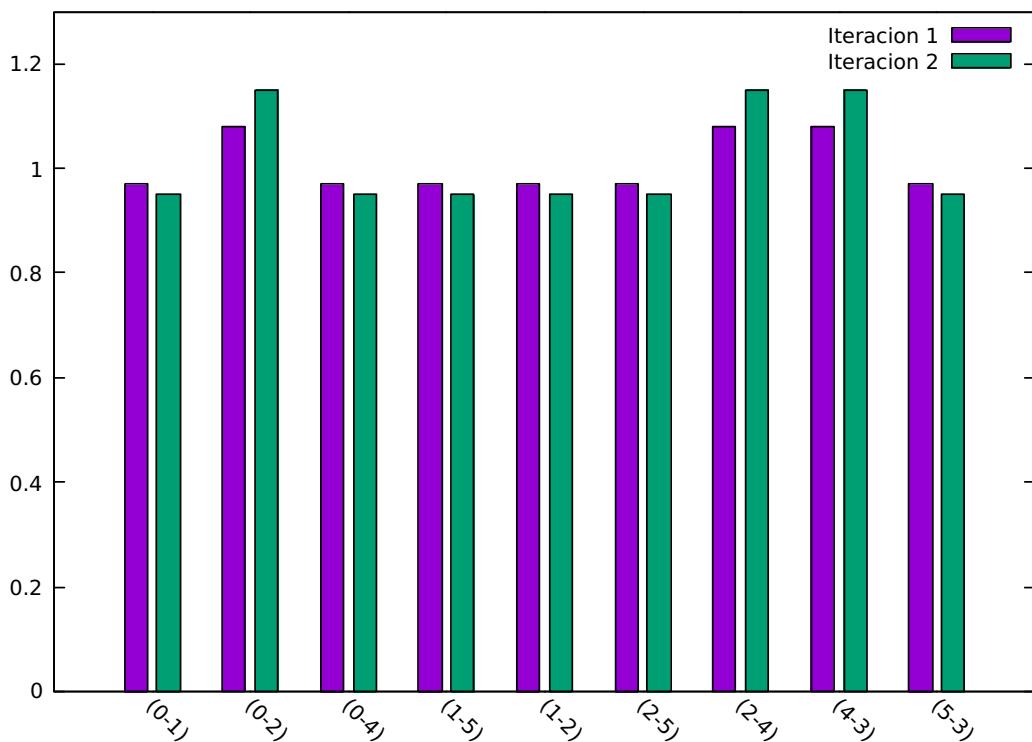


FIGURA 4.20: Gráfica recogida de la Tabla 4.8.

Como podemos apreciar, el camino tomado en ambas iteraciones fue el 0-2-4-3. La evaporación se sucede lentamente esto es debido a que hay solo una hormiga dentro de la iteración, contando con esto y los truncamientos realizados en Arduino, la evaporación será bastante lenta, en la Sección 4.5 contrastaremos estas ejecuciones.

Adicionalmente, realizaremos como prueba 10 iteraciones, inferidas, es decir, se le pasará el mismo camino desde Arduino, en este caso el 0-2-4-3 con un coste de 3 unidades y, acto seguido, inferiremos el camino 0-1-5-3 con un coste de 17 unidades otras 10 veces, y discutiremos el resultado final en forma de gráfica.

Mostramos primero el resultado tras las 10 primeras iteraciones correspondientes al camino 0-2-4-3.

PHEROMONES MATRIX						
	0	1	2	3	4	5
0	0.881	0.783	1.78	0.783	0.783	0.783
1	0.783	0.881	0.783	0.783	0.783	0.783
2	1.78	0.783	0.881	0.783	1.78	0.783
3	0.783	0.783	0.783	0.881	1.78	0.783
4	0.783	0.783	1.78	1.78	0.881	0.783
5	0.783	0.783	0.783	0.783	0.783	0.881

FIGURA 4.21: Tabla de feromonas tras 10 iteraciones

Seguidamente lanzamos 10 iteraciones más recorriendo el camino 0-1-5-3 y obtenemos la siguiente tabla de feromonas.

PHEROMONES MATRIX						
	0	1	2	3	4	5
0	0.783	0.939	1.67	0.618	0.618	0.618
1	0.939	0.783	0.618	0.618	0.618	0.939
2	1.67	0.618	0.783	0.618	1.67	0.618
3	0.618	0.618	0.618	0.783	1.67	0.939
4	0.618	0.618	1.67	1.67	0.783	0.618
5	0.618	0.939	0.618	0.939	0.618	0.783

FIGURA 4.22: Tabla de feromonas tras 20 iteraciones.

Pasamos ahora a recoger la información obtenida en la Figura 4.21 y en la Figura 4.22 en la siguiente tabla:

Camino	10 iteraciones	20 iteraciones
0-1	0.783	0.939
0-2	1.78	1.67
0-4	0.783	0.618
1-5	0.783	0.939
1-2	0.783	0.618
2-5	0.783	0.618
2-4	1.78	1.67
4-3	1.78	1.67
5-3	0.783	0.939

TABLA 4.9: Recopilación de resultados tras veinte iteraciones en Arduino.

A continuación sacaremos una gráfica de los resultados obtenidos en la Tabla 4.9.

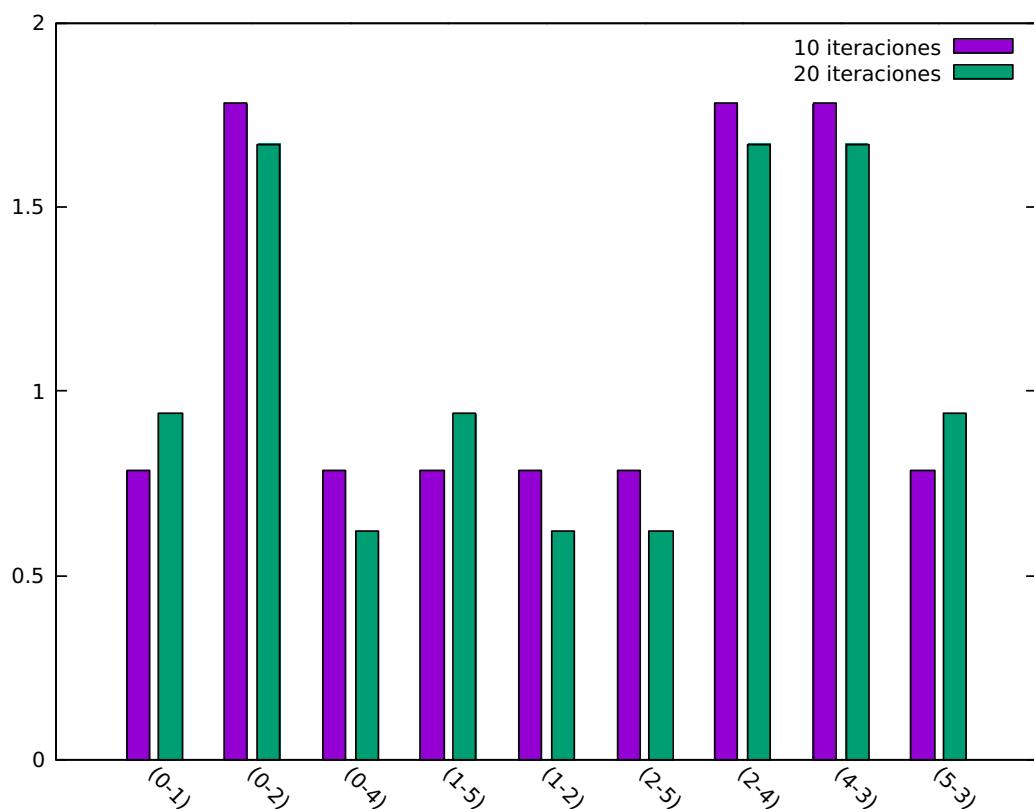


FIGURA 4.23: Gráfica obtenida sobre los resultados captados en la Tabla 4.9.

Como podemos observar en la Figura 4.23 en las primeras iteraciones sobresale bastante el

camino tomado respecto del que no se tomó. Al iterar otras 10 veces (20 iteraciones) vemos como el camino tomado empieza a destacar del que no se tomó pero sin llegar al punto del otro camino tomado, esto se debe a que las 10 primeras iteraciones se realizaron sobre un camino de coste 3 unidades, y las 10 segundas iteraciones con uno de coste 17, vemos la diferencia de ambos, vemos como correctamente, la feromonía más fuerte sigue siendo la del camino 3, a pesar de que ambos caminos se han ejecutado el mismo número de veces.

También podemos observar la evaporación con 10 iteraciones de diferencia entre las barras de 10 iteraciones, y las de 20.

4.5. Contraste de resultados obtenidos

En esta Sección compararemos las 3 iteraciones sobre las mismas condiciones, a excepción de la hormiga en Arduino que solo puede lanzarse 1 concurrentemente, puesto que la tecnología Bluetooth no admite difundir información y no funciona bien con varias conexiones concurrentes.

Procedemos pues a mostrar una gráfica con los resultados obtenidos de las tablas:

- Tabla 4.3: Datos de ejecución teórica.
- Tabla 4.4: Datos de ejecución en C++.
- Tabla 4.8: Datos de ejecución robot Arduino.

Procederemos a realizar los siguientes experimentos:

- Comparativa de los 3 métodos de ejecución de la primera iteración y argumentación.
- Comparativa de los 3 métodos de ejecución de la segunda iteración y argumentación.

Comenzamos recopilando la información de la primera iteración de las tablas de datos teórica, en C++ y en Arduino:

Camino	Teórico	C++	Arduino
0-1	0.1578	0.098	0.97
0-2	0.4323	0.098	1.08
0-4	0.099	0.598	0.97
1-5	0.1578	0.098	0.97
1-2	0.099	0.098	0.97
2-5	0.099	0.098	0.97
2-4	0.4323	0.098	1.08
4-3	0.4323	0.598	1.08
5-3	0.1578	0.098	0.97

TABLA 4.10: Resultados obtenidos tras una iteración.

A continuación mostramos la gráfica de los resultados obtenidos en la Tabla 4.10.

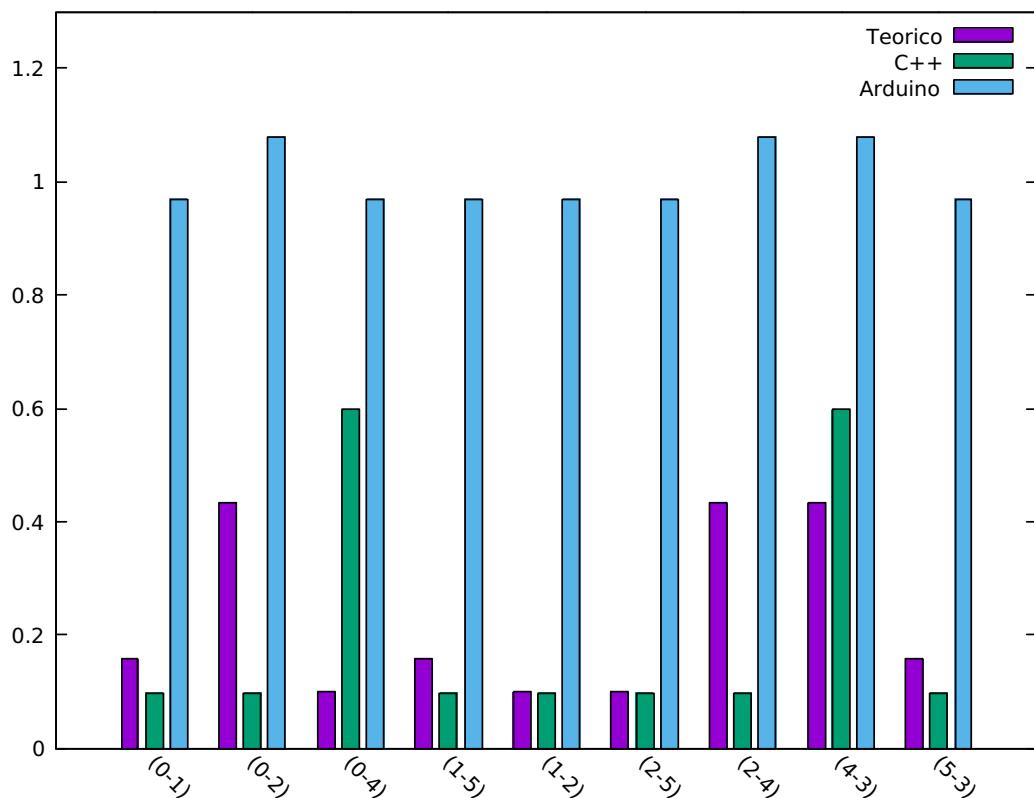


FIGURA 4.24: Gráfica recogida de la Tabla 4.10.

Como podemos observar en la gráfica hay un salto de escala entre la ejecución en Arduino y

el resto, esto se debe a la poca precisión de números decimales que ofrece Arduino, luego hemos tenido que reducir la precisión de la feromonas en 0.01 y no en 0.001 como está en C++ y en la ejecución teórica.

Si comprobamos relativamente, la pérdida de precisión anteriormente comentada deja huella en la evaporación de feromonas de la ejecución en Arduino, siendo esta menor aún teniendo la tasa de evaporación más alta que en los otros dos casos, esto es debido al truncamiento que se produce por la falta de precisión en Arduino. Otro factor influyente en la evaporación es que Arduino solo tiene una hormiga por cada iteración mientras los otros dos sistemas dos.

Contrastando los 3 podemos observar que la ejecución teórica no es tan radical respecto a las otras dos iteraciones, esto se debe a que la aleatoriedad fue generada por un humano, y se quiso que las hormigas exploraran distintos caminos, con lo cuál las hormigas no se decantaban solo por uno como es el caso de C++ o Arduino.

Comparando la ejecución en C++ y Arduino podemos observar que son equivalentes, pero C++ goza de más precisión en sus decimales además de dos hormigas por iteración, aun así el GAP que dejan los nodos visitados y los que no en ambas ejecuciones, podríamos decir que es casi equivalente.

Pasamos ahora a la segunda iteración, recopilamos los datos tal como hemos hecho con la Tabla 4.10.

Camino	Teórico	C++	Arduino
0-1	0.0156	0.096	0.95
0-2	0.4385	0.096	1.15
0-4	0.0098	0.836	0.95
1-5	0.0156	0.096	0.95
1-2	0.0098	0.096	0.95
2-5	0.0723	0.096	0.95
2-4	0.3761	0.096	1.15
4-3	0.3760	0.836	1.15
5-3	0.0781	0.096	0.95

TABLA 4.11: Resultados obtenidos tras dos iteraciones.

Mostramos la gráfica de los datos recopilados en la Tabla 4.11.

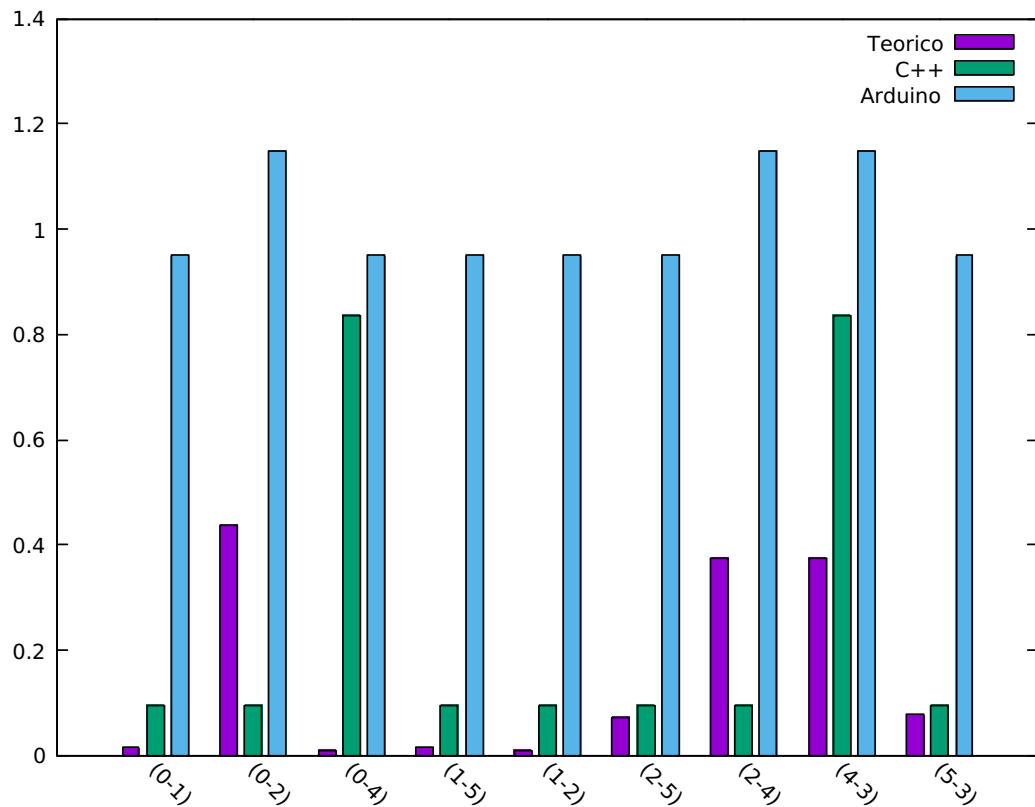


FIGURA 4.25: Gráfica recogida de la Tabla 4.11.

Respecto a la anterior iteración mostrada en la Figura 4.24 vemos como los caminos se van radicalizando mucho, excepto en la ejecución de Arduino por los motivos comentados en el análisis de la Figura 4.24.

Por lo demás vemos más de lo mismo, el funcionamiento del ACO tal como debe funcionar.

Capítulo 5

Presupuesto

A continuación se presenta el presupuesto de fabricación de una hormiga, adicionalmente, puesto que es parte del presupuesto, se especifican las horas dedicadas al desarrollado en un previo *diagrama de Gantt* incluido en este capítulo.

5.1. Diagrama de Gantt

Como la normativa establece en estos casos, un proyecto de final de carrera tiene que tener una duración máxima de 450 horas, agotadas en este proyecto. A continuación se desglosa la actividad realizada durante dichas horas distribuidas en el *diagrama de Gantt*.

El proyecto comienza a realizarse el día *15 de Enero de 2021* y tiene finalización el día *5 de Marzo de 2021*, cumpliendo un total de 450 horas, distribuidas en 11 semanas y 2 días, invirtiendo 8 horas cada día de Lunes a Viernes.

Seguidamente se muestra el *diagrama de Gantt*:

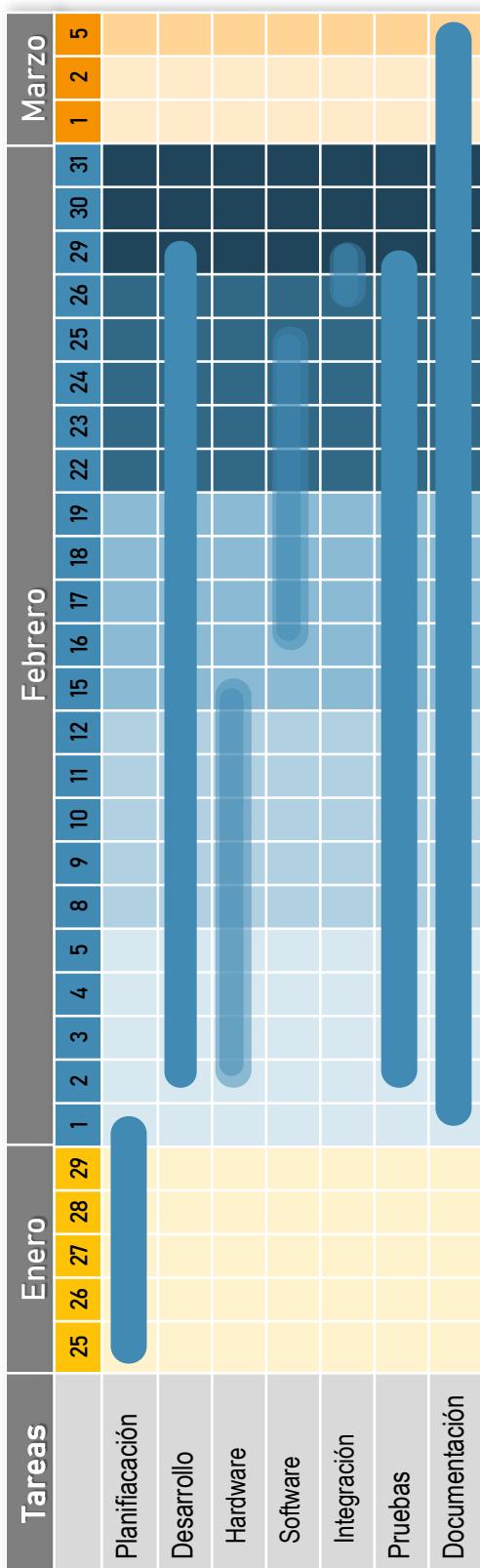


FIGURA 5.1: Diagrama de Gantt.

A continuación desglosaremos cada una de las operaciones realizadas en el *diagrama de Gantt* que se muestra en la figura 5.1:

- **Planificación:** Las tareas de planificación abarcan desde que surge la idea del proyecto a realizar hasta que se crea el esqueleto del documento que están leyendo, escribiendo también en borrador qué se va a realizar, a saber, los dos robot, el software que implementan, la documentación, el tablero, el medio de comunicación así como la arquitectura, etc. Para esto se han invertido 6 días, que equivalen a 48 horas del proyecto desde el día *25 de Enero de 2021* hasta el día *1 de Febrero de 2021*.
- **Desarrollo:** La fase de desarrollo incluye desde que no hay absolutamente nada hasta que tenemos un robot con el software implementado. Se ha subdividido esta fase en tres subfases: Hardware, Software e Integración. Estas tres fases están intercaladas temporalmente con la fase de Pruebas que veremos posteriormente. Esta fase ocupa gran parte del proyecto abarcando 20 días y 160 horas, desde el día *2 de Febrero de 2021* hasta el día *29 de Febrero de 2021*.
- **Hardware:** La subfase de hardware abarca desde que no tenemos nada hasta que tenemos los dos robots montados y con todos sus sensores funcionando, pasando por la fase de diseño y pedidos de componentes que se ha intercalado con la documentación mientras llegaban los envíos. Esta subfase abarca 10 días del proyecto, equivalente a 80 horas, distribuidas entre los días *2 de Febrero de 2021* y *15 de Febrero de 2021*.
- **Software:** La subfase de software va desde que no tenemos nada hasta que tenemos el software del robot en perfecto funcionamiento, pasando por un software simple del algoritmo en C++, y a partir de éste, se ha distribuido parte de dicho software en el robot, y la otra parte en la aplicación realizada en Qt que corresponde al servidor. Esta operación ha durado 8 días, es decir, 64 horas, que van desde los días *16 de Febrero de 2021* hasta el *25 de Febrero de 2021*.
- **Integración:** Esta pequeña subfase que abarca 2 días, es decir 16 horas, que van desde el día *26 de Febrero de 2021* al *27 de Febrero de 2021*. En esta subfase se integra el software con el hardware probando completamente la lógica y su funcionamiento, dejándolo todo preparado para la toma de resultados.
- **Pruebas:** La fase de pruebas incluye todas y cada una de las pruebas que se hayan realizado en la fase de desarrollo tanto software como hardware, así como las de integración

y al finalizar dicha fase, tendríamos realizada la toma de datos a falta de contrastarla en la documentación. Dicha fase abarca 20 días, unas 160 horas, distribuidas desde el día *2 de Febrero de 2021* al *29 de Febrero de 2021*.

- **Documentación:** La fase de documentación, la cuál es la más importante de todas, abarca desde que tenemos el esqueleto del proyecto indicando qué puntos vamos a abarcar, hasta que se finaliza dicha documentación. Esta fase esta constantemente intercalando con el resto puesto que todo lo que se haga debe quedar debidamente documentado. Esta fase abarca 26 días que corresponden a 208 horas, distribuidas entre los días *1 de Febrero de 2021* y *5 de Marzo de 2021*.

Finalizada la justificación de horas del operario, procedemos a documentar el presupuesto.

5.2. Presupuesto

En esta Sección se muestra el presupuesto que supone el proyecto completo, se desglosará en un presupuesto y, posteriormente se describirá cada elemento del presupuesto.

PRESUPUESTO PROYECTO FINAL DE CARRERA

Concepto	Número de unidades o cantidad	Precio unitario (€)	Importe (€)
Motor DC	4	6,798	27,192
Controladora de motor LN298N	2	2,553	5,106
Sensor óptico infrarrojo CNY70	4	1,015	4,06
Resistencia SMD 180 Ohmios	4	0,018	0,072
Resistencia SMD 10 KOhmios	4	0,023	0,092
PCB para sensores ópticos CNY70	2	0,92	1,84
PCB para interconectar hardware	2	0,92	1,84
Interruptor	2	0,735	1,47
Sensor de color TCS3200	2	7,35	14,7
Arduino Nano	2	3,659	7,318
Módulo Bluetooth HC-05	2	6,89	13,78
Batería LiPo	2	23,91	47,82
Chasis robot zumo Pololu	2	16,376	32,752
Bobina de cable 0.08mm	160 cm	18,39 (4200 cm)	0,7
Goma termorretráctil	12	0,01	0,12
Tornillos M3	16	0,033	0,539
Tornillos de separación hexagonales	8	0,076	0,612
Filamento para impresora 3D	21.8 m	21,15 (335 m)	1,376
Horas de impresión	7h 30m	10 €/ h	7,5
Pines macho	2 (x4 pins)	8,27 (x200 pins)	0,33
Pines hembra	8 (x4 pins)	8,27 (x200 pins)	1,32
Conector JST-XH hembra	4	9,089 (x170 conectores)	0,213
Conector JST-XH macho	2	9,089 (x170 conectores)	0,106
Tablero impreso	2	1,84	3,68
Mano de obra	450 h	20 €/ h	9000
Amortización de ordenador portátil	1	825,62/19	43,42
Horas de electricidad	450 h	0,084	37,8
Total sin IVA (€)			9255.76
IVA (€)			1943.70
Total + IVA (€)			11199.46

TABLA 5.1: Presupuesto del proyecto de final de carrera.

Procedemos a continuación a describir cada elemento del presupuesto mostrado en la Tabla 5.1:

- Motor DC: Los motores de corriente continua se utilizan para mover el robot, cada robot constará de dos motores de corriente continua.
- Controladora de motor LN298N: Controladora de motor, se utiliza para dar alimentación a los motores además de controlarlos ya que está comunicada con los pines de Arduino. Se utiliza una por cada robot.
- Sensor óptico infrarrojo CNY70: Los sensores CNY70 detectan los colores blanco y negro, en el robot son utilizados para hacer el siguelínea. Se utilizan 2 por cada robot.
- Resistencia SMD 180 Ohmios: Resistencia de 180 Ohmios utilizada para los sensores CNY70. Se utilizan 2 por cada robot, es decir, uno por cada sensor CNY70.
- Resistencia SMD 10 KOhmios: Resistencia de 10 Kilo Ohmios utilizada para los sensores CNY0. Se utilizan 2 por cada robot, es decir, uno por cada sensor CNY70.
- PCB para sensores ópticos CNY70: Placa de circuito impreso donde se soldará el circuito de los CNY70. Se utiliza uno por cada robot.
- PCB para interconectar hardware: Placa de circuito impreso donde se soldará todo el hardware electrónico que contenga el robot. Se utiliza uno por cada robot.
- Interruptor: Interruptor de encendido o apagado del robot. Se utiliza uno por cada robot.
- Sensor de color TCS3200: El sensor de color detecta los colores rojo, verde y azul, en el robot se utiliza para detectar sobre qué tipo de nodo estamos ubicados. Se utiliza uno por cada robot.
- Arduino Nano: Microcontrolador que actuará de cerebro de nuestro robot. Uno por cada robot.
- Módulo Bluetooth HC-05: Módulo Bluetooth a través del cual, el robot se comunicará con el servidor. Uno por cada robot.
- Batería LiPo: Batería que alimentará al robot. Una por cada robot.
- Chasis robot zumo Pololu: Estructura sobre la que se albergará el robot. Uno por cada robot.

- Bobina de cable 0.08mm: Cable que se utilizará para conectar los motores a sus controladoras. 80cm por cada robot.
- Goma termorretráctil: Se utiliza para fijar bien los cables en las soldaduras. 6 unidades por cada robot.
- Tornillos M3: Tornillos para fijar el robot a la estructura y la controladora de motor a la PCB principal. 8 por cada robot.
- Tornillos de separación hexagonales: Se utilizan como separador de la controladora de motor con la PCB principal. 4 por cada robot.
- Filamento para impresora 3D: Se utiliza para imprimir el soporte de la batería del robot. 10.9m por cada robot.
- Horas de impresión: Horas que se mantiene la impresora ocupada. 3 horas 45 minutos por cada robot.
- Pines macho: Pines para conectar componentes electrónicos. 4 por cada robot.
- Pines hembra: Pines para conectar componentes electrónicos. 16 por cada robot.
- Conector JST-XH hembra: Conector hembra donde se conectará la batería. 2 por cada robot.
- Conector JST-XH macho: Conector macho donde se conectará la batería. 1 por cada robot.
- Tablero impreso: Tablero sobre el cuál se moverán los robot. 1 por cada robot.
- Mano de obra: Horas de trabajo. 450 horas estipuladas.
- Amortización de ordenador portátil: Se divide el precio del portátil con el número de días que se tarda en realizar el proyecto, aproximadamente 19.
- Horas de electricidad: Horas de electricidad utilizadas para hacer el proyecto.

Capítulo 6

Conclusiones

A continuación se exponen las principales conclusiones obtenidas.

- Se han alcanzado los objetivos propuestos, tanto entender el funcionamiento del ACO, como implementarlo en un robot real, previamente diseñado y fabricado para ello.
- Se ha realizado un proyecto novedoso en su campo, con vistas a una futura publicación.
- Se comprueba que el ACO es un algoritmo que permite obtener muy buenos resultados en un tiempo de ejecución óptimo.
- Se muestra la fase de diseño paso a paso, discriminando por qué se utiliza cada tecnología y por qué no otras. Además del diseño de circuitos impresos específicos para hacer el robot más compacto.
- Se ofrece un software específico para la ejecución del ACO, útil a la hora de obtener resultados en el Capítulo 4. Gracias al cual somos conscientes de la importancia de lanzar una hormiga más o incrementar el número de iteraciones que deban realizar las hormigas.
- Se ofrece un software que ejerce de servidor entre los robots, y un cliente en forma de robot creado en Arduino con el cuál se construye la matriz de feromonas y se puede comparar el rendimiento respecto a otras formas de implementar el algoritmo en cuestión.
- Conseguimos conciencia de la importancia de cada elemento en el ACO.

- Se ofrece un presupuesto el cual nos indica el ridículo coste de construir una hormiga haciendo este proyecto factible económicamente para que cualquiera pueda construirlo.
- Abrimos un camino lleno de posibilidades si seguimos trabajando por esta vía, ya que se podría llegar incluso a ejecutar el algoritmo completo en procesadores poco potentes lo cuál sería un gran avance en la ciencia.

Anexo A

Anexo: Manual de usuario

A.1. Introducción

El manual de usuario se dividirá en 3 apartados:

- Qué incluye.
- Metodología de uso.
- Qué hacer en caso de error.

A.2. Elementos incluidos

El proyecto incluye los siguientes elementos:

- Robot basado en Arduino.
- Software que actuará como servidor.

A.3. Metodología de uso

A continuación se describe como utilizar el proyecto.

1. Lo primero que se debe hacer es ejecutar el software servidor.
2. Una vez hecho esto colocamos la hormiga en la casilla de salida, la de color azul, orientado a la derecha.
3. Encendemos el robot pulsando el interruptor y podremos observar que la luz del módulo Bluetooth se queda parpadeando.
4. Pasamos al programa servidor de nuevo y buscamos nuestro dispositivo Bluetooth del cual debemos saber el nombre, una vez haya sido enlazado con el sistema operativo, nos aparecerá en la lista de dispositivos disponibles, dicha ventana se encuentra a la izquierda en el programa. Damos doble click sobre el dispositivo y podremos observar al mismo tiempo como la ventana central se habilita y la luz del módulo Bluetooth se queda fija.
5. La primera vez, debemos sincronizar la matriz de feromonas, esto se realiza pulsando sobre el botón “Init bot pheromones”, y podemos observar como el robot comienza a ejecutarse. Una vez hecho esto cerramos la conexión Bluetooth pulsando sobre el botón “Close Conection”.
6. Dejamos que el robot encuentre la fuente de alimento y regrese de nuevo al punto de origen.
7. Se quedará esperando a poder volcar la información capturada en el servidor así que conectamos de nuevo con éste exactamente igual que se hizo en el paso 4.
8. Pulsamos ahora sobre el botón “Start Transmission” dos veces, y observaremos en la venta principal que se ha volcado el camino tomado. Cerramos la conexión con el botón “Close Conection”.
9. Volvemos al paso 6.

A.4. Plan de contingencia en caso de error

En caso de error en la ejecución del robot realizaremos los siguientes pasos:

1. Mantendremos en todo momento el servidor encendido.
2. Apagaremos el robot y lo volveremos a colocar en la casilla de salida orientado a la derecha.
3. Volvemos a encenderlo y seguimos los pasos descritos en el apartado [A.3](#).

Es aconsejable limpiar la ventana central periódicamente pulsando sobre el botón “Clear Terminal” ya que si no se hace al cabo de ciertas iteraciones el servidor podría cerrarse de manera incorrecta.

Anexo B

Anexo: Código empleado en el proyecto

B.1. Introducción

Mostramos en este apéndice el código relevante utilizado en el proyecto dividiéndolo en los siguientes apartados:

- Bibliotecas propias del robot Arduino.
- Bibliotecas propias del servidor.
- Código de testeo del robot.
- Código de testeo de la comunicación Bluetooth
- Código del robot Arduino.
- Código del servidor.
- Código específico de comunicación.
- Implementación del Código en C++.

Las bibliotecas tanto del robot como del servidor serán utilizadas cuando el controlador en cuestión lo requiera.

B.1.1. Bibliotecas propias del robot Arduino

A continuación se muestran las bibliotecas utilizadas en el robot del proyecto. Son las que se listan a continuación:

- “PINS.hpp”
- “CNY.hpp” / “CNY.cpp”.
- “2xCNY.hpp” / “2xCNY.cpp”.
- “TSC3200.hpp” / “TSC3200.cpp”.
- “MOTOR.hpp” / “MOTOR.cpp”.
- “COMMON_TYPES.hpp”.
- “RATIONAL.hpp” / “RATIONAL.cpp”.
- “PATH.hpp” / “PATH.cpp”.
- “MAP.hpp” / “MAP.cpp”.

B.1.1.1. PINS

En esta Sección se muestra el contenido del archivo “PINS.hpp”.

```
#ifndef PINS_HPP_
#define PINS_HPP_
//DEFINITION OF PINS
/*********************CNYx2********************/
* ARDUINO PIN    NAME          *
*   A1           CNY_LEFT      *
*   A2           CNY_RIGHT     *
/*********************BLUETOOTH*****************/
* ARDUINO PIN    NAME          *
```

```

*      D1/TX          BLUETOOTH_TRANSMIT      *
*      D2/RX          BLUETOOTH_RECEIVE      *
******/*****
/******COLOR_SENSOR******/
*  ARDUINO PIN    NAME          *
*  D8           S1           *
*  D9           S0           *
*  D10          S3           *
*  D11          S2           *
*  D12          OUT          *
******/*****
/******LN298******/
*  ARDUINO PIN    NAME          *
*  D5           IN1          *
*  D4           IN2          *
*  D3           IN3          *
*  D2           IN4          *
******/*****
//CNYx2
#define CNY_LEFT A2
#define CNY_RIGHT A1
//BLUETOOTH
#define BLUETOOTH_TRANSMIT 1
#define BLUETOOTH_RECEIVE 2
//COLOR SENSOR
#define S1 8
#define S0 9
#define S3 10
#define S2 11
#define OUT 12
//LN298
#define LN298_IN1 5
#define LN298_IN2 4
#define LN298_IN3 3
#define LN298_IN4 2
#endif

```

B.1.1.2. CNY

En esta Sección se muestra el contenido de los archivos “CNY.hpp” y “CNY.cpp”.

```

#ifndef CNY_HPP_
#define CNY_HPP_
#include "Arduino.h"
#include "PINS.hpp"

```

```

class CNY{
public:
    //CONSTRUCTOR
    /* CNY(uint8_t CNY_PIN, uint8_t SILL)
     * Precondition: -
     * Postcondition: Creates a new CNY object, sill parameter
     * is for calibration of the CNY, by default 512. CNY_PIN is
     * the Arduino pin that this CNY uses.
     */
    CNY(uint8_t, float = 512);
    //GETTER
    /* bool isWhite() const;
     * Precondition: -
     * Postcondition: Returns TRUE if the CNY is over white, else returns FALSE
     */
    bool isWhite() const;
    /* bool isBlack() const;
     * Precondition: -
     * Postcondition: Returns TRUE if the CNY is over black, else returns FALSE
     */
    bool isBlack() const;
private:
    uint8_t CNY_;
    float SILL_;
};

#endif

```

```

#include "CNY.hpp"
CNY::CNY(uint8_t CNY_PIN, float SILL):
    CNY_(CNY_PIN), SILL_(SILL) {}

bool CNY::isWhite() const {
    uint32_t cny;
    cny = analogRead(CNY_);
    return SILL_ >= cny;
}

bool CNY::isBlack() const {
    uint32_t cny;
    cny = analogRead(CNY_);
    return SILL_ < cny;
}

```

B.1.1.3. 2xCNY

En esta Sección se muestra el contenido de los archivos “2xCNY.hpp” y “2xCNY.cpp”.

```
#ifndef x2CNY_HPP_
#define x2CNY_HPP_
#include "Arduino.h"
#include "CNY.hpp"

class x2CNY {
public:
    //CONSTRUCTOR
    /* 2xCNY()
     * Precondition: -
     * Postcondition: Creates a new CNY object which includes 2 CNYs,
     *      one each to the other. (left, right)
     */
    x2CNY(const CNY&, const CNY&);

    //GETTER
    /* bool bothWhite() const
     * Precondition: -
     * Postcondition: returns TRUE if left CNY and right CNY are focusing
     *      a white background, any other case returns FALSE
     */
    bool bothWhite() const;
    /* bool bothBlack() const
     * Precondition: -
     * Postcondition: returns TRUE if left CNY and right CNY are focusing a black background,
     *      any other case returns FALSE
     */
    bool bothBlack() const;
    /* bool leftWhiteRightBlack() const
     * Precondition: -
     * Postcondition: returns TRUE if left CNY is focusing a white background
     *      and right CNY is focusing a black background, any other case returns FALSE
     */
    bool leftWhiteRightBlack() const;
    /* bool leftBlackRightWhite() const
     * Precondition: -
     * Postcondition: returns TRUE if left CNY is focusing a black background
     *      and right CNY is focusing a white background, any other case returns FALSE
     */
    bool leftBlackRightWhite() const;
    /* bool leftWhite() const
     * Precondition: -
     * Postcondition: returns TRUE if left CNY focuses a white background,
     *      any other case returns FALSE and does not matter about right CNY value
     */
    bool leftWhite() const;
    /* bool leftBlack() const
     * Precondition: -
     */
```

```

    * Postcondition: returns TRUE if left CNY is focusing a black background,
    *      any other case returns FALSE and doesn't matter about right CNY value
    */
    bool leftBlack() const;
    /* bool rightWhiteLeftBlack() const
     * Precondition: -
     * Postcondition: returns TRUE if right CNY is focusing a white background
     *      and left CNY is focusing a black background, any other case returns FALSE
     */
    bool rightWhiteLeftBlack() const;
    /* bool rightBlackLeftWhite() const
     * Precondition: -
     * Postcondition: returns TRUE if right CNY is focusing a black background
     *      and left CNY is focusing a white background, any other case returns FALSE
     */
    bool rightBlackLeftWhite() const;
    /* bool rightWhite() const
     * Precondition: -
     * Postcondition: returns TRUE if right CNY is focusing a white background,
     *      any other case returns FALSE and does not matter about left CNY value
     */
    bool rightWhite() const;
    /*bool rightBlack() const
     * Precondition: -
     * Postcondition: returns TRUE if right CNY is focusing a black background,
     *      any other case returns FALSE and does not matter about left CNY value
     */
    bool rightBlack() const;
private:
    CNY CNY_LEFT_, CNY_RIGHT_;
};

#endif

```

```

#include "2xCNY.hpp"
x2CNY::x2CNY(const CNY& CNY_LEFT, const CNY& CNY_RIGHT):
    CNY_LEFT_(CNY_LEFT),
    CNY_RIGHT_(CNY_RIGHT)
{}

bool x2CNY::bothWhite() const {
    return (CNY_LEFT_.isWhite() && CNY_RIGHT_.isWhite());
}

bool x2CNY::bothBlack() const {
    return (CNY_LEFT_.isBlack() && CNY_RIGHT_.isBlack());
}

bool x2CNY::leftWhiteRightBlack() const {
    return (CNY_LEFT_.isWhite() && CNY_RIGHT_.isBlack());
}

bool x2CNY::leftBlackRightWhite() const {

```

```

    return (CNY_LEFT_.isBlack() && CNY_RIGHT_.isWhite());
}

bool x2CNY::leftWhite() const {
    return (CNY_LEFT_.isWhite());
}

bool x2CNY::leftBlack() const {
    return (CNY_LEFT_.isBlack());
}

bool x2CNY::rightWhiteLeftBlack() const {
    return (CNY_RIGHT_.isWhite() && CNY_LEFT_.isBlack());
}

bool x2CNY::rightBlackLeftWhite() const {
    return (CNY_RIGHT_.isBlack() && CNY_LEFT_.isWhite());
}

bool x2CNY::rightWhite() const {
    return (CNY_RIGHT_.isWhite());
}

bool x2CNY::rightBlack() const {
    return (CNY_RIGHT_.isBlack());
}

```

B.1.1.4. TSC3200

En esta Sección se muestra el contenido de los archivos “TSC3200.hpp” y “TSC3200.cpp”.

```

#ifndef TSC3200_HPP_
#define TSC3200_HPP_
#include "Arduino.h"
#include "PINS.hpp"
#include "COMMON_TYPES.hpp"

class TSC3200 {
public:
    //CONSTRUCTOR
    /*
     * TSC3200()
     * Precondition: -
     * Postcondition: Creates a new TSC3200 object
     */
    TSC3200();
    /*
     * Color getColour();
     * Precondition: -
     * Postcondition: returns the readen color
     */
    Color getColor();

```

```

/*
 * Color getLastColour() const;
 * Precondition: -
 * Postcondition: Returns the las color
 * that the sensor TSC3200 read
 */
Color getLastColour() const;

private:
    Color color_;
    uint8_t S0_, S1_, S2_, S3_, OUT_;

};

#endif


---


#include "TSC3200.hpp"

TSC3200::TSC3200():
    color_ (Color::NONE),
    S0_ (S0),
    S1_ (S1),
    S2_ (S2),
    S3_ (S3),
    OUT_ (OUT) {
    pinMode (S0_, OUTPUT);
    pinMode (S1_, OUTPUT);
    pinMode (S2_, OUTPUT);
    pinMode (S3_, OUTPUT);
    pinMode (OUT_, INPUT);
    digitalWrite (S0_, HIGH);
    digitalWrite (S1_, HIGH);
}

Color TSC3200::getColor() {
    byte countRed, countBlue, countGreen;
    for(int i = 0; i < 50; ++i){
        digitalWrite(S2_, LOW);
        digitalWrite(S3_, LOW);
        countRed = pulseIn(OUT_, digitalRead(OUT_) == HIGH ? LOW : HIGH);
        digitalWrite(S3_, HIGH);
        countBlue = pulseIn(OUT_, digitalRead(OUT_) == HIGH ? LOW : HIGH);
        digitalWrite(S2_, HIGH);
        countGreen = pulseIn(OUT_, digitalRead(OUT_) == HIGH ? LOW : HIGH);
    }
    if (countRed < countBlue && countRed < countGreen && countRed < 80)
        color_ = Color::RED;
    else if (countGreen < countRed && (countGreen - countBlue) < 4 )
        color_ = Color::GREEN;
    else if (countBlue < countRed && countBlue < countGreen)
        color_ = Color::BLUE;
    return color_;
}

```

```
Color TSC3200::getLastColour() const {
    return color_;
}
```

B.1.1.5. MOTOR

En esta Sección se muestra el contenido de los archivos “MOTOR.hpp” y “MOTOR.cpp”.

```
#ifndef MOTOR_HPP_
#define MOTOR_HPP_
#include "Arduino.h"
#include "PINS.hpp"

class MOTOR {
public:
    //CONSTRUCTOR
    /* MOTOR()
     * Precondition: -
     * Postcondition: Creates a new MOTOR object
     */
    MOTOR();
    //END CONSTRUCTOR
    //SETTERS
    /* void straight()
     * Precondition: -
     * Postcondition: Makes the bot move straight
     */
    void straight();
    /* void left()
     * Precondition: -
     * Postcondition: Makes the bot turn left
     */
    void left();
    /* void right()
     * Precondition: -
     * Postcondition: Makes the bot turn right
     */
    void right();
    /* void back()
     * Precondition: -
     * Postcondition: Makes the bot move backwards
     */
    void back();
    /* void crossLeft()
     * Precondition: -
     * Postcondition: Turns the bot left and straight
     */
}
```

```

        */
void crossLeft();
/*void crossRight()
 * Precondition: -
 * Postcondition: Turns the bot right and straight
*/
void crossRight();
/*void backRight()
 * Precondition: -
 * Postcondition: Turns the bot right foward and
 * left backwards at the same time
*/
void backRight();
/*void backLeft()
 * Precondition: -
 * Postcondition: Turns the bot left foward and
 * right backwards at the same time
*/
void backLeft();
/* void turnBack()
 * Precondition: -
 * Postcondition: Makes the bot turn the way back
 * from where it came
*/
void turnBack();
/* void botStop()
 * Precondition: -
 * Postcondition: if bot is running, it stops it,
 * else it does nothing.
*/
void botStop();
/* void roll_degrees (unsigned grades)
 * Precondition: -
 * Postcondition: makes the bot turn on x grades
*/
void roll_grades(int grades);
//END SETTERS
private:
    uint8_t IN1_, IN2_, IN3_, IN4_;
};

#endif

```

```

#include "MOTOR.hpp"
MOTOR::MOTOR():
    IN1_(LN298_IN1),
    IN2_(LN298_IN2),
    IN3_(LN298_IN3),
    IN4_(LN298_IN4)

```

```

{
    pinMode(IN1_, INPUT);
    pinMode(IN2_, INPUT);
    pinMode(IN3_, INPUT);
    pinMode(IN4_, INPUT);
    digitalWrite(IN1_, LOW);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, LOW);
    digitalWrite(IN4_, LOW);
}
void MOTOR::straight() {
    digitalWrite(IN1_, HIGH);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, HIGH);
    digitalWrite(IN4_, LOW);
}
void MOTOR::left() {
    digitalWrite(IN1_, LOW);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, HIGH);
    digitalWrite(IN4_, LOW);
}
void MOTOR::right() {
    digitalWrite(IN1_, HIGH);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, LOW);
    digitalWrite(IN4_, LOW);
}
void MOTOR::back() {
    digitalWrite(IN1_, LOW);
    digitalWrite(IN2_, HIGH);
    digitalWrite(IN3_, LOW);
    digitalWrite(IN4_, HIGH);
}
void MOTOR::crossLeft() {
    digitalWrite(IN1_, LOW);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, HIGH);
    digitalWrite(IN4_, LOW);
    digitalWrite(IN1_, LOW);
    digitalWrite(IN2_, HIGH);
    digitalWrite(IN3_, HIGH);
    digitalWrite(IN4_, LOW);
}
void MOTOR::crossRight() {
    digitalWrite(IN1_, HIGH);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, LOW);
}

```

```

digitalWrite(IN4_, LOW);
digitalWrite(IN1_, HIGH);
digitalWrite(IN2_, LOW);
digitalWrite(IN3_, HIGH);
digitalWrite(IN4_, LOW);
}

void MOTOR::backLeft() {
    digitalWrite(IN1_, LOW);
    digitalWrite(IN2_, HIGH);
    digitalWrite(IN3_, HIGH);
    digitalWrite(IN4_, LOW);
}

void MOTOR::backRight() {
    digitalWrite(IN1_, HIGH);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, LOW);
    digitalWrite(IN4_, HIGH);
}

void MOTOR::turnBack() {
    digitalWrite(IN1_, HIGH);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, LOW);
    digitalWrite(IN4_, HIGH);
}

void MOTOR::botStop() {
    digitalWrite(IN1_, LOW);
    digitalWrite(IN2_, LOW);
    digitalWrite(IN3_, LOW);
    digitalWrite(IN4_, LOW);
}

void MOTOR::roll_grades(int grades) {
    switch(grades) {
        case 90:
            left();
            backLeft();
            delay(1150);
            //delay(360); //ATOMICA
            //delay(1400); //BATERIA GRANDE
            botStop();
            break;
        case 180:
            right();
            backRight();
            delay(2300);
            //delay(730); //ATOMICA
            //delay(2688); //BATERIA GRANDE
            botStop();
            break;
    }
}

```

```

    case 270:
        right();
        backRight();
        delay(1200);
        //delay(360); //ATOMICA
        //delay(1400); //BATERIA GRANDE
        botStop();
    break;
}
}

```

B.1.1.6. COMMON_TYPES

En esta Sección se muestra el contenido deL archivos “COMMON_TYPES.hpp”.

```

#ifndef COMMON_TYPES_HPP_
#define COMMON_TYPES_HPP_
enum Color {RED, GREEN, BLUE, NONE};
#endif

```

B.1.1.7. RATIONAL

En esta Sección se muestra el contenido de los archivos “RATIONAL.hpp” y “RATIONAL.cpp”.

```

#ifndef RATIONAL_NUMBER_HPP_
#define RATIONAL_NUMBER_HPP_
#include "Arduino.h"
struct Rational_number {
    Rational_number();
    explicit Rational_number(float, int = 1);
    Rational_number(const Rational_number&);
    void modify_rational_number (int, int = 1);
    void modify_rational_number_WITHOUT_SIMPLIFY (int, int = 1);
    int numerator() const;
    int denominator() const;
    float resolv() const;
    void from_str(String);
    String toString() const;
    void printR() const;
    void simplify();
    Rational_number operator = (const Rational_number&);

```

```

private:
    int numerator_, denominator_;
};

int mcd(int, int);
int mcm(int, int);

Rational_number operator + (const Rational_number&, const Rational_number&);
Rational_number operator - (const Rational_number&, const Rational_number&);
Rational_number operator * (const Rational_number&, const Rational_number&);
Rational_number operator / (const Rational_number&, const Rational_number&);
bool operator > (const Rational_number&, const Rational_number&);
bool operator >= (const Rational_number&, const Rational_number&);
bool operator < (const Rational_number&, const Rational_number&);
bool operator <= (const Rational_number&, const Rational_number&);
bool operator == (const Rational_number&, const Rational_number&);

#endif


---


#include "RATIONAL.hpp"

Rational_number::Rational_number():numerator_(1), denominator_(100) {}

Rational_number::Rational_number(float r, int d = 1):
    numerator_ (r),
    denominator_(d) {
    simplify();
}

Rational_number::Rational_number(const Rational_number& r):
    numerator_(r.numerator_),
    denominator_(r.denominator_)
{
    simplify();
}

//explicit Rational_number::Rational_number(int n): numerator_(n), denominator_(1) {}

void Rational_number::modify_rational_number (int numerator, int denominator = 1) {
    if(denominator_ != 0) {
        numerator_ = numerator;
        denominator_ = denominator;
        simplify();
    }
}

void Rational_number::modify_rational_number_WITHOUT_SIMPLIFY
    (int numerator, int denominator = 1) {
    if(denominator_ != 0) {
        numerator_ = numerator;
        denominator_ = denominator;
    }
}

int Rational_number::numerator() const { return numerator_; }
int Rational_number::denominator() const { return denominator_; }
float Rational_number::resolv() const { return (numerator_ / denominator_); }

void Rational_number::from_str(String str) {

```

```

String number = "";
bool numerator = true;
for(size_t i = 0; i < str.length(); i++) {
    if(str.charAt(i) != '/' & str.charAt(i) != 'E') {
        number += str.charAt(i);
    } else {
        if (numerator) {
            numerator_ = number.toInt();
            numerator = false;
            number = "";
        } else {
            denominator_ = number.toInt();
            number = "";
        }
    }
}
String Rational_number::toString() const {
    return (((String)numerator_) + "/" + ((String)denominator_));
}
void Rational_number::printR() const {
    Serial.print(numerator_);
    Serial.print(" / ");
    Serial.println(denominator_);
}
void Rational_number::simplify() {
    int prime_numbers[] = {2, 3, 5, 7, 11, 13, 17, 19};

    for (size_t i = 0; i < 8; i++)
        if((((numerator_ % prime_numbers[i]) == 0) &&
            ((denominator_ % prime_numbers[i]) == 0)) {
            numerator_ /= prime_numbers[i];
            denominator_ /= prime_numbers[i];
            i = -1;
        }
}
Rational_number Rational_number::operator = (const Rational_number& r) {
    if(denominator_ != 0) {
        numerator_ = r.numerator_;
        denominator_ = r.denominator_;
        simplify();
    }
}

return *this;
}
int mcd(int num1, int num2) {
    int mcd = 0;
    int a = (num1 > num2)? num1 : num2;
}

```

```

    int b = (num1 < num2) ? num1 : num2;
    do {
        mcd = b;
        b = a%b;
        a = mcd;
    } while(b != 0);

    return mcd;
}

int mcm(int num1, int num2) {
    int mcm = 0;
    int a = (num1 > num2) ? num1 : num2;
    int b = (num1 < num2) ? num1 : num2;
    mcm = (a/mcd(a,b))*b;
    return mcm;
}

Rational_number operator + (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    int numerator =
        (mcm_/r1.denominator())*r1.numerator() + (mcm_/r2.denominator())*r2.numerator();
    r.modify_rational_number (
        (numerator),
        (mcm_)
    );
    if(r.denominator() == 0) {
        Serial.println("Math Error, denominator = 0");
        r.modify_rational_number(0, 1);
    }
    return r;
}

Rational_number operator - (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    int numerator =
        (mcm_/r1.denominator())*r1.numerator() - (mcm_/r2.denominator())*r2.numerator();
    r.modify_rational_number (
        (numerator),
        (mcm_)
    );
    if(r.denominator() == 0) {
        Serial.println("Math Error, denominator = 0");
        r.modify_rational_number(0, 1);
    }
    return r;
}

Rational_number operator * (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;

```

```

r.modify_rational_number (
    (r1.numerator() * r2.numerator()),
    (r1.denominator() * r2.denominator())
);
if(r.denominator() == 0) {
    Serial.println("Math Error, denominator = 0");
    r.modify_rational_number(0, 1);
}
return r;
}

Rational_number operator / (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;

    r.modify_rational_number (
        (r1.numerator() * r2.denominator()),
        (r1.denominator() * r2.numerator())
);
if(r.denominator() == 0) {
    Serial.println("Math Error, denominator = 0");
    r.modify_rational_number(0, 1);
}
return r;
}

bool operator > (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return
        (mcm_/r1.denominator()*r1.numerator()) > (mcm_/r2.denominator()*r2.numerator());
}

bool operator >= (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return
        ((mcm_/r1.denominator()*r1.numerator()) > (mcm_/r2.denominator()*r2.numerator()))
        ||
        r1 == r2
};

bool operator < (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return
        (mcm_/r1.denominator()*r1.numerator()) < (mcm_/r2.denominator()*r2.numerator());
}

bool operator <= (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return
        ((mcm_/r1.denominator()*r1.numerator()) < (mcm_/r2.denominator()*r2.numerator()))
        ||
        r1 == r2
}

```

```

;
}

bool operator == (const Rational_number& r1, const Rational_number& r2) {
    r1.simplify();
    r2.simplify();

    return ((r1.numerator() == r2.numerator()) && (r1.denominator() == r2.denominator()));
}

```

B.1.1.8. PATH

En esta Sección se muestra el contenido de los archivos “PATH.hpp” y “PATH.cpp”.

```

#ifndef PATH_HPP_
#define PATH_HPP_
#include "Arduino.h"

class Path {
public:
    Path();
    Path(const Path&);

    Path operator = (Path);
    void add_path(uint8_t);
    void clear_path();
    uint8_t remove_last();
    uint8_t last_element() const;
    bool is_empty() const;
    String path() const;
    ~Path();

private:
    uint8_t *path_;
    unsigned int path_length_;
    unsigned int n_;
};

#endif

```

```

#include "PATH.hpp"
Path::Path() :
    path_length_ (0),
    n_ (51)
{
    path_ = new uint8_t[n_];

    for(int i = 0; i < n_; i++)
        path_[i] = -1;

```

```

        ++path_length_;
        path_[path_length_ - 1] = 0;
    }

Path::Path(const Path& p) :
    path_length_(p.path_length_),
    n_(p.n_)
{
    path_ = new uint8_t[n_];
    for(int i = 0; i < n_; i++)
        path_[i] = p.path_[i];
}

Path Path::operator = (Path p) {
    path_length_ = p.path_length_;
    n_ = p.n_;

    path_ = new uint8_t[n_];
    for(int i = 0; i < path_length_; i++)
        path_[i] = p.path_[i];
}

void Path::add_path(uint8_t element) {
    if(path_length_ < n_) {
        ++path_length_;
        path_[path_length_ - 1] = element;
        path_[path_length_] = -1;
    }
}

void Path::clear_path() {
    path_length_ = 0;
    for(int i = 0; i < n_; i++)
        path_[i] = -1;

    ++path_length_;
    path_[path_length_ - 1] = 0;
}

uint8_t Path::remove_last() {
    uint8_t element_to_return = path_[path_length_ - 1];

    path_[path_length_ - 1] = -1;
    --path_length_;

    return element_to_return;
}

uint8_t Path::last_element() const {
    return path_[path_length_ - 1];
}

bool Path::is_empty() const {
    return path_length_ == 0;
}

```

```

String Path::path() const {
    String str = "";

    for(int i = 0; i < path_length_; i++)
        str += ((i < (path_length_ - 1)))?
            String(path_[i]) + "," : String(path_[i]);

    return str;
}

Path::~Path() {
    delete [] path_;
}

```

B.1.1.9. MAP

En esta Sección se muestra el contenido de los archivos “MAP.hpp” y “MAP.cpp”.

```

#ifndef MAP_HPP_
#define MAP_HPP_
#include "Arduino.h"
#include "COMMON_TYPES.hpp"
#include "RATIONAL.hpp"

class Map {
public:
    /**************************************************************************
     *                                         ATRIBUTOS DE PUBLICOS
     */
    /*const size_t alpha = 1;
    const size_t beta = 1;
    const float LEARNING = 1.;
    const float EVAPORATION = 0.01;*/
    static const unsigned int n_ = 6;
    Rational_number PHEROMONES[n_][n_];
    bool VISITED[n_];
    uint8_t map_[n_][n_] =
    {
        {0,4,1,INF,3,INF},
        {4,0,2,INF,INF,4},
        {1,2,0,INF,1,6},
        {INF,INF,INF,0,1,9},
        {3,INF,1,1,0,INF},
        {INF,4,6,9,INF,0}
    };
    uint8_t grades[n_][n_] =
    {

```

```

        {0,1,0,INF,3,INF},
        {2,0,3,INF,INF,0},
        {2,1,0,INF,3,0},
        {INF,INF,INF,0,2,0},
        {2,INF,1,0,0,INF},
        {INF,2,3,0,INF,0}
    };
    uint8_t stragihten_grades[n_] [n_] =
{
    {0,0,0,INF,0,INF},
    {1,0,1,INF,INF,0},
    {2,3,0,INF,1,3},
    {INF,INF,INF,0,2,2},
    {3,INF,3,0,0,INF},
    {INF,2,2,2,INF,0}
};
//     static const unsigned int n_ = 6;
//
//     uint8_t map_[n_] [n_] =
// {
//     {0,4,1,INF,3,INF},
//     {4,0,2,INF,INF,4},
//     {1,2,0,INF,1,6},
//     {INF,INF,INF,0,7,5},
//     {3,INF,1,7,0,INF},
//     {INF,4,6,5,INF,0}
// };
//
//     uint8_t grades[n_] [n_] =
// {
//     {0,1,0,INF,3,INF},
//     {2,0,3,INF,INF,0},
//     {2,1,0,INF,3,0},
//     {INF,INF,INF,0,3,1},
//     {2,INF,1,0,0,INF},
//     {INF,2,3,0,INF,0}
// };
//
//     uint8_t stragihten_grades[n_] [n_] =
// {
//     {INF,0,0,INF,0,INF},
//     {1,0,1,INF,INF,0},
//     {2,3,0,INF,1,3},
//     {INF,INF,INF,0,2,2},
//     {3,INF,3,3,0,INF},
//     {INF,2,2,1,INF,0}
// };
static const unsigned int n_ = 7;

```

```

//      uint8_t map_[n_] [n_] =
//      {
//          {0, 4, 2, INF, INF, 5, INF},
//          {4, 0, 2, INF, INF, INF, 3},
//          {2, 2, 0, INF, INF, 2, 7},
//          {INF, INF, INF, 0, 6, INF, 5},
//          {INF, INF, INF, 6, 0, 3, INF},
//          {5, INF, 2, INF, 3, 0, INF},
//          {INF, 3, 7, 5, INF, INF, 0}
//      };
//      uint8_t grades[n_] [n_] =
//      {
//          {0, 1, 0, INF, INF, 3, INF},
//          {2, 0, 3, INF, INF, INF, 0},
//          {2, 1, 0, INF, INF, 3, 0},
//          {INF, INF, INF, 0, 3, INF, 1},
//          {INF, INF, INF, 0, 0, 2, INF},
//          {2, INF, 1, INF, 0, 0, INF},
//          {INF, 2, 3, 0, INF, INF, 0}
//      };
//      uint8_t stragihten_grades[n_] [n_] =
//      {
//          {0, 0, 0, INF, INF, 0, INF},
//          {1, 0, 1, INF, INF, INF, 0},
//          {2, 3, 0, INF, INF, 1, 3},
//          {INF, INF, INF, 0, 2, INF, 2},
//          {INF, INF, INF, 3, 0, 2, INF},
//          {3, INF, 3, INF, 0, 0, INF},
//          {INF, 2, 2, 1, INF, INF, 0}
//      };
//      /*Actualizaci n de feromonas
//      PRECONDICIN: las feromonas han sido inicializadas
//      POSTCONDICIN: actualiza las feromonas utilizando
//      la f rmula de actualizaci n de feromonas
//      */
//      void grades_toogle_conversion();
//      void modify_pheromones_from_str(String str);
// **** DECLARACION DE SUBTIPOS ****
// **** OPERACIONES DEL TAD TABLERO ****
//      typedef uint8_t summit;
//      //V rtices del grafo(nodos)
//      static const uint8_t INF;
//      //Coste infinito entre dos nodos(no hay camino directo entre ellos)
// **** OPERACIONES DEL TAD TABLERO ****
// **** CONSTRUCTOR sobrecargado para crear el tablero

```

```

PRECONDICION: el parametro formal N debe ser > 0
POSTCONDICIN: crea un tablero nuevo, asigna el tamaño a las feromonas
e inicializa la lista de hormigas.

*/
explicit Map(summit initial);
/*M todo para hacer avanzar una hormiga de un nodo al siguiente utilizando
* la formula de la probabilidad sobre la cual se elige uno arbitrariamente
* y condicionado segun su probabilidad.
PRECONDICIN: el nodo pasado por parametro tiene al menos un adyacente.
El nodo no es el nodo objetivo. La hormiga ha estado recorriendo el tablero.
POSTCONDICIN: devuelve el siguiente nodo por el que pasar la hormiga
*/
summit next_node(summit x);
/*M todo para saber la distancia de un punto x a otro y,
* se utiliza la formula de la distancia
PRECONDICIN: ambos parametros son adyacentes entre s ,
puesto que el grafo es no dirigido, con el que el primer
parametro sea adyacente al segundo nos vale
POSTCONDICION: devuelve la distancia entre dos puntos
*/
uint8_t distance(uint8_t x, uint8_t y);
/*M todo observador del nodo inicial
PRECONDICIN: -
POSTCONDICIN: devuelve el vertice inicial
*/
summit initial(){
    return initial_;
}
/*M todo observador del nodo actual
PRECONDICIN: -
POSTCONDICIN: devuelve el vertice actual
*/
summit actual(){
    return actual_node_;
}

/*M todo modificador del nodo inicial
PRECONDICIN: -
POSTCONDICIN: devuelve el vertice actual
*/
void actual(summit actual){
    actual_node_ = actual;
}
/*M todo observador del nodo actual
PRECONDICIN: -
POSTCONDICIN: devuelve el vertice actual
*/
summit previous()

```

```

        return previous_node_;
    }

    //String print_adjacent_list() const;
    void restart_visited();
    int number_of_adjacents() const;
private:
    /***** DECLARACION DE TIPOS PRIVADOS *****/
    *
    * DECLARACION DE METODOS PRIVADOS
    */
    int adjacent_[4];
    summit actual_node_, previous_node_;
    summit initial_;
    /***** DECLARACION DE METODOS PRIVADOS *****/
    *
    */
    void adjacent_list(summit x);
};

#endif


---


#include "MAP.hpp"

const uint8_t Map::INF = 255;

void Map::modify_pheromones_from_str(String str) {
    String rational_number_string = "";
    Rational_number r;
    size_t i = 0, j = 0;
    for(size_t it = 0; it < str.length(); it++) {
        if(str.charAt(it) != ',' && str.charAt(it) != 'E') {
            rational_number_string += str.charAt(it);
        } else {
            rational_number_string += 'E';
            r.from_str(rational_number_string);
            PHEROMONES[i][j] = r;
            PHEROMONES[j][i] = PHEROMONES[i][j];
            rational_number_string = "";
            j++;
        }
        if(j == n_) {
            j = 0;
            i++;
        }
    }
}

void Map::grades_toogle_conversion() {
    int conversion = 0;
    uint8_t equivalence[4][2] =
    {
        {0, 2},
        {1, 3},

```

```

    {2, 0},
    {3, 1}
};

for (int i = 0; i < n_; i++)
for (int j = 0; j < n_; j++)
if (grades[i][j] != INF && i != j) {
    grades[i][j] = equivalence[
        grades[i][j]
    ][1];
    stragihтен_grades[i][j] = equivalence[
        stragihтен_grades[i][j]
    ][1];
}
}

Map::Map (Map::summit initial):
    initial_ (initial),
    actual_node_ (initial_),
    previous_node_(initial_)
{
    /*for(size_t i = 0; i < n_; i++)
    for(size_t j = 0; j < n_; j++)
        PHEROMONES[i][j] = 1;*/
    for(size_t i = 0; i < n_; i++)
        VISITED[i] = false;
    VISITED[initial_] = true;
}

typename Map::summit Map::next_node(Map::summit x) {
    randomSeed(analogRead(A0));
    size_t vector_adjacent_position;
    adjacent_list(x);
    size_t n_adjacents = number_of_adjacents();
    Rational_number p[n_];
    Rational_number aux (0);
    Rational_number p_random(0);
    if(n_adjacents == 0) {
        actual_node_ = previous_node_;
        previous_node_ = x;
    } else {
        /*Serial.println("GRADES MATRIX");
        for(int i = 0; i < n_; i++) {
            for(int j = 0; j < n_; j++) {
                Serial.print(String(grades[i][j]));
                Serial.print(" ");
            }
            Serial.println();
        }
        Serial.println("CORRECTION GRADES MATRIX");
        for(int i = 0; i < n_; i++) {
            */
    }
}

```

```

        for(int j = 0; j < n_; j++) {
            Serial.print(String(stragihten_grades[i][j]));
            Serial.print(" ");
        }
        Serial.println();
    } */

//NEXT NODE ACO
for(size_t i = 0; i < n_adjacents; i++) {
    aux =
        aux + (PHEROMONES[x][adjacent_[i]] * Rational_number(1, distance(x, adjacent_[i])));
    for(size_t i = 0; i < n_adjacents; i++) {
        p[ adjacent_[i] ] =
            (
                (
                    (PHEROMONES[x][adjacent_[i]]) * Rational_number(1, distance(x, adjacent_[i]))
                    /
                    aux
                )
            );
    };

    if(i > 0) {
        p[ adjacent_[i] ] = p[ adjacent_[i] ] + p[adjacent_ [i-1]];
    }
    /*Serial.print("PROBABILITY TO GET NODE ");
    Serial.print(adjacent_[i]);
    Serial.print(": ");
    p[ adjacent_[i] ].printR();*/
}
p_random.modify_rational_number(
    random(0,100),
    100
);
//p_random.printR();
previous_node_ = x;
for(size_t i = 0; i < n_adjacents; i++) {
    if(i == 0) {
        if(p_random >= Rational_number(0,1) && p_random <= p[ adjacent_[i] ])
            actual_node_ = adjacent_[i];
    } else {
        if(p_random > p[ adjacent_[i - 1] ] && p_random <= p[ adjacent_[i] ])
            actual_node_ = adjacent_[i];
    }
}
/*Serial.print("SELECTED NODE: ");
Serial.println(actual_node_);*/
VISITED[actual_node_] = true;
}

return actual_node_;

```

```

}

uint8_t Map::distance(Map::summit x, Map::summit y) {
    return map_[x][y];
}

int Map::number_of_adjacents() const {
    int i = 0;
    for(; i < 4 && adjacent_[i] != -1; i++);
    return i;
}

void Map::adjacent_list(Map::summit x) {
    uint8_t it = 0;
    for(uint8_t i = 0; i < n_; i++)
        if(map_[x][i] != INF && i != x && !VISITED[i]){
            adjacent_[it] = i;
            ++it;
        }
    if(it < 4) adjacent_[it] = -1;
    return adjacent_;
}

/*String Map::print_adjacent_list() const {
    String adjacent_str = "";
    for(int i = 0; i < 4 && adjacent_[i] != -1; i++)
        adjacent_str += (i == 4 - 1 || adjacent_[i] != -1)?
            String(adjacent_[i]) :
            String(adjacent_[i] + ", ");
    return adjacent_str;
} */

void Map::restart_visited() {
    for(size_t i = 0; i < n_; i++)
        VISITED[i] = false;
    VISITED[initial_] = true;
}

```

B.1.2. Bibliotecas propias del servidor

A continuación se exponen las bibliotecas correspondientes al servidor:

- “matrix.h” / “matrix.cpp”.
- “path.h” / “path.cpp”.
- “rational.h” / “rational.cpp”.

B.1.2.1. matrix

En esta Sección se muestra el contenido de los archivos “matrix.hpp” y “matrix.cpp”.

```
#ifndef MATRIX_H
#define MATRIX_H
#include <QString>
#include <QtDebug>
#include <QList>
#include "rational.h"

class Matrix {
public:
    Matrix();
    QString toString() const;
    QList<int> adjacents_from_node(size_t node);
    //void evaporate (size_t);
    static const int n_ = 6;
    static const int INF = 255;
    const Rational_number EVAPORATION_COST;
    Rational_number pheromones_[n_][n_];
    static const int map_[n_][n_];
    Matrix operator = (const Matrix&);

private:
    QList<int> adjacents_;
};

#endif // MATRIX_H



---


#include "matrix.h"
const int Matrix::map_[n_][n_] =
{
    {0,4,1,INF,3,INF},
    {4,0,2,INF,INF,4},
    {1,2,0,INF,1,6},
    {INF,INF,INF,0,1,9},
    {3,INF,1,1,0,INF},
    {INF,4,6,9,INF,0}
};

Matrix::Matrix() {
    for(int i = 0; i < n_; i++)
        for(int j = 0; j < n_; j++)
            pheromones_[i][j] = (Rational_number(1,1) - EVAPORATION_COST);
}

QString Matrix::toString() const {
    QString str = "";
    for(int i = 0; i < n_; i++)
        for(int j = 0; j < n_; j++) {
            str += (i == (n_ - 1) && j == (n_ - 1)) ? pheromones_[i][j].toString() : (pheromones_[i][j]
```

```

        }
        str += "E";
        return str;
    }

QList<int> Matrix::adjacents_from_node(size_t node) {
    adjacents_.clear();
    for(size_t i = 0; i < n_; i++)
        if(map_[node][i] != INF && i != node)
            adjacents_.append(i);
    return adjacents_;
}

Matrix Matrix::operator = (const Matrix& m) {
    for(size_t i = 0; i < n_; i++)
        for(size_t j = 0; j < n_; j++)
            pheromones_[i][j] = m.pheromones_[i][j];
    return *this;
}

```

B.1.2.2. path

En esta Sección se muestra el contenido de los archivos “path.hpp” y “path.cpp”.

```

#ifndef PATH_HPP_
#define PATH_HPP_
#include <cstdint>
#include <QString>
#include <QStringList>
#include "matrix.h"

class Path {
public:
    Path();
    Path(const Path&);
    Path(QString);
    void add_path(int);
    void clear_path();
    int remove_last();
    int last_element() const;
    bool is_empty() const;
    QString path() const;
    int cost();
    const Path& fromString(QString);
    Matrix update_pheromones(Matrix&);

    ~Path();

private:
    int *path_;

```

```

    unsigned int path_length_;
    unsigned int n_;
}

#endif


---


#include "path.h"
Path::Path() :
    path_length_ (0),
    n_ (51)
{
    path_ = new int[n_];
    for(size_t i = 0; i < n_; i++)
        path_[i] = -1;
    ++path_length_;
    path_[path_length_ - 1] = 0;
}
Path::Path(const Path& p) :
    path_length_ (p.path_length_),
    n_ (p.n_)
{
    path_ = new int[n_];
    for(size_t i = 0; i < n_; i++)
        path_[i] = p.path_[i];
}
Path::Path(QString str) {
    QStringList pieces = str.split(",");
    QStringList numbers;
    n_ = pieces.size();
    int i = 0;
    for(const auto& p : pieces) {
        path_[i] = p.toInt();
        path_length_++;
    }
}
void Path::add_path(int element) {
    if(path_length_ < n_) {
        ++ path_length_;
        path_[path_length_ - 1] = element;
        path_[path_length_] = -1;
    }
}
void Path::clear_path() {
    path_length_ = 0;
    for(size_t i = 0; i < n_; i++)
        path_[i] = -1;
    ++path_length_;
    path_[path_length_ - 1] = 0;
}

```

```

int Path::remove_last() {
    int element_to_return = path_[path_length_ - 1];
    path_[path_length_ - 1] = -1;
    --path_length_;
    return element_to_return;
}

int Path::cost() {
    int previous, next;
    int cost = 0;
    for(unsigned int i = 0; i < path_length_; i++) {
        previous = path_[i];
        next = path_[((i + 1))];
        cost += Matrix::map_[previous][next];
    }
    return cost;
}

int Path::last_element() const {
    return path_[path_length_ - 1];
}

bool Path::is_empty() const {
    return path_length_ == 0;
}

QString Path::path() const {
    QString str = "";
    int aux;
    for(size_t i = 0; i < path_length_; i++) {
        aux = path_[i];
        str += (i == (path_length_ - 1)) ?
            QString(aux) : QString(aux) + ">";
    }
    qDebug() << "lets see: " << str;
    return str;
}

const Path& Path::fromString(QString str) {
    QStringList str_list = str.split(",");
    path_length_ = 0;
    for(int i = 0; i < (str_list.size()); i++)
        add_path(str_list[i].toInt());
    return *this;
}

Matrix Path::update_pheromones(Matrix& matrix) {
    int fnode, snode;
    float aux;
    for(size_t i = 0; i < matrix.n_; i++)
        for(size_t j = 0; j < matrix.n_; j++) {
            matrix.pheromones_[i][j].from_float_to_rational(matrix.pheromones_[i][j].resolv() * (Ratio
            matrix.pheromones_[j][i] = matrix.pheromones_[i][j];
            qDebug() << i << ", " << j << ":" << matrix.pheromones_[i][j].resolv();
        }
}

```

```

        }
    for(size_t i = 0; i < (path_length_ - 1); i++) {
        fnode = path_[i];
        snode = path_[i + 1];
        aux = matrix.pheromones_[fnode][snode].resolv();
        aux += (float(1) / float(cost()));
        matrix.pheromones_[fnode][snode] = Rational_number(aux);
        matrix.pheromones_[snode][fnode] = matrix.pheromones_[fnode][snode];
    }
    return matrix;
}
Path::~Path() {
    delete [] path_;
}

```

B.1.2.3. rational

En esta Sección se muestra el contenido de los archivos “rational.hpp” y “rational.cpp”.

```

#ifndef RATIONAL_NUMBER_HPP_
#define RATIONAL_NUMBER_HPP_
#include <cstdint>
#include <QString>
#include <QDebug>
struct Rational_number {
    Rational_number();
    Rational_number(int, int = 1);
    explicit Rational_number(float);
    Rational_number(const Rational_number&);
    void modify_rational_number (int, int = 1);
    void modify_rational_number_WITHOUT_SIMPLIFY (int, int = 1);
    int numerator() const;
    int denominator() const;
    float resolv() const;
    Rational_number from_float_to_rational(float);
    //String toString() const;
    void printR() const;
    void simplify();
    Rational_number simplify() const;
    Rational_number operator = (const Rational_number&);
    QString toString() const;
    private:
        int numerator_, denominator_;
};
int mcd(int, int);

```

```

int mcm(int, int);

Rational_number operator + (const Rational_number&, const Rational_number&);
Rational_number operator - (const Rational_number&, const Rational_number&);
Rational_number operator * (const Rational_number&, const Rational_number&);
Rational_number operator / (const Rational_number&, const Rational_number&);

bool operator > (const Rational_number&, const Rational_number&);
bool operator >= (const Rational_number&, const Rational_number&);
bool operator < (const Rational_number&, const Rational_number&);
bool operator <= (const Rational_number&, const Rational_number&);
bool operator == (const Rational_number&, const Rational_number&);

#endif


---


#include "rational.h"

Rational_number::Rational_number(): numerator_(1), denominator_(100) {}

Rational_number::Rational_number(int n, int d): numerator_(n), denominator_(d) {}

Rational_number::Rational_number(float f) {
    int pot = 10;
    numerator_ = int (f);
    denominator_ = 1;
    while(numerator_ < 100) {
        numerator_ = f * pot;
        denominator_ = pot;
        pot *= 10;
    }
    simplify();
}

Rational_number::Rational_number(const Rational_number& r):
    numerator_(r.numerator_),
    denominator_(r.denominator_)

{
    simplify();
}

void Rational_number::modify_rational_number (int numerator, int denominator) {
    if(denominator_ != 0) {
        numerator_ = numerator;
        denominator_ = denominator;
        simplify();
    }
}

void Rational_number::modify_rational_number_WITHOUT_SIMPLIFY (int numerator, int denominator) {
    if(denominator_ != 0) {
        numerator_ = numerator;
        denominator_ = denominator;
    }
}

int Rational_number::numerator() const { return numerator_; }

int Rational_number::denominator() const { return denominator_; }

float Rational_number::resolv() const { return (float(numerator_) / float(denominator_)); }

```

```

Rational_number Rational_number::from_float_to_rational(float f) {
    qDebug() << "float val: " << f;
    int pot = 10;
    numerator_ = int(f);
    denominator_ = 1;
    while(numerator_ < 100) {
        numerator_ = f * pot;
        denominator_ = pot;
        pot *= 10;
    }
    simplify();
    return *this;
}

QString Rational_number::toString() const { return (QString::number(numerator_) + "/" + QString::number(denominator_)); }

void Rational_number::simplify() {
    int prime_numbers[] = {2, 3, 5, 7, 11, 13, 17, 19};
    for (size_t i = 0; i < 8; i++)
        if((numerator_ % prime_numbers[i]) == 0 && ((denominator_ % prime_numbers[i]) == 0)) {
            numerator_ /= prime_numbers[i];
            denominator_ /= prime_numbers[i];
            i = -1;
        }
    numerator_ %= 32000;
    denominator_ %= 32000;
}

Rational_number Rational_number::simplify() const {
    int prime_numbers[] = {2, 3, 5, 7, 11, 13, 17, 19};
    int n = numerator_;
    int d = denominator_;
    for (size_t i = 0; i < 8; i++)
        if((n % prime_numbers[i]) == 0 && ((d % prime_numbers[i]) == 0)) {
            n /= prime_numbers[i];
            d /= prime_numbers[i];
            i = -1;
        }
    n %= 32000;
    d %= 32000;
    return Rational_number(n, d);
}

Rational_number Rational_number::operator = (const Rational_number& r) {
    if(denominator_ != 0) {
        numerator_ = r.numerator_;
        denominator_ = r.denominator_;
        simplify();
    }
    return *this;
}

int mcd(int num1, int num2) {

```

```

int mcd = 0;
int a = (num1 > num2)? num1 : num2;
int b = (num1 < num2)? num1 : num2;
do {
    mcd = b;
    b = a%b;
    a = mcd;
} while(b != 0);
return mcd;
}

int mcm(int num1, int num2) {
    int mcm = 0;
    int a = (num1 > num2)? num1 : num2;
    int b = (num1 < num2)? num1 : num2;
    mcm = (a/mcd(a,b))*b;
    return mcm;
}

Rational_number operator + (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    int numerator = (mcm_/r1.denominator()*r1.numerator()) + (mcm_/r2.denominator()*r2.numerator());
    r.modify_rational_number (
        (numerator),
        (mcm_)
    );
    if(r.denominator() == 0)
        r.modify_rational_number(0, 1);
    return r;
}

Rational_number operator - (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    int numerator = (mcm_/r1.denominator()*r1.numerator()) - (mcm_/r2.denominator()*r2.numerator());
    r.modify_rational_number (
        (numerator),
        (mcm_)
    );
    if(r.denominator() == 0)
        r.modify_rational_number(0, 1);
    return r;
}

Rational_number operator * (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;
    r.modify_rational_number (
        (r1.numerator() * r2.numerator()),
        (r1.denominator() * r2.denominator())
    );
    if(r.denominator() == 0)

```

```

    r.modify_rational_number(0, 1);
    return r;
}

Rational_number operator / (const Rational_number& r1, const Rational_number& r2) {
    Rational_number r;
    r.modify_rational_number (
        (r1.numerator() * r2.denominator()),
        (r1.denominator() * r2.numerator())
    );
    if(r.denominator() == 0)
        r.modify_rational_number(0, 1);
    return r;
}

bool operator > (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return (mcm_/r1.denominator()*r1.numerator()) > (mcm_/r2.denominator()*r2.numerator());
}

bool operator >= (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return ((mcm_/r1.denominator()*r1.numerator()) > (mcm_/r2.denominator()*r2.numerator()))
        ||
        r1 == r2
    ;
}

bool operator < (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return (mcm_/r1.denominator()*r1.numerator()) < (mcm_/r2.denominator()*r2.numerator());
}

bool operator <= (const Rational_number& r1, const Rational_number& r2) {
    int mcm_ = mcm(r1.denominator(), r2.denominator());
    return ((mcm_/r1.denominator()*r1.numerator()) < (mcm_/r2.denominator()*r2.numerator()))
        ||
        r1 == r2
    ;
}

bool operator == (const Rational_number& r1, const Rational_number& r2) {
    Rational_number rc1, rc2;
    rc1 = r1.simplify();
    rc2 = r2.simplify();
    return ((rc1.numerator() == rc2.numerator()) && (rc1.denominator() == rc2.denominator()));
}

```

B.1.3. Código de testeo del robot

Código utilizado para probar individualmente los sensores y actuadores del robot, así como para calibrar éste.

```
#ifndef TEST_HORMIGAS_INO_
#define TEST_HORMIGAS_INO_
#include "PINS.hpp"
#include "MOTOR.hpp"
#include "2xCNY.hpp"
#include "CNY.hpp"
#include <SoftwareSerial.h>
/**ONLY FOR DEBUG**/
//#define SERIAL_BLUETOOTH_ONLY_
//uncomment this line if you are using bluetooth to comunicate the ant with the PC
#define SERIAL_WIRED_ONLY_
//uncomment this line if you are using wired connection between arduino and the PC
//#define STRAIGHT_FOWARD_
//uncomment this line if you want the robot to go straight and foward
//#define RUN_ROBOT_
//uncomment if you want the robot to run
//#define RUN_WHITE_PATH_ROBOT_
//uncomment if you want the robot to run through a white path
//#define PRINT_CNY_READ_
//uncomment if you want to display the cny analog read
//#define PROBAR_COLORES_
//uncomment if you want to activate and display wich colour is the sensor colour reading
//#define NODE_OPERATIONS_
//uncomment to enable actions to do when the robot finds a node
//#define NEIGHBORS_
//uncomment this line to search for the neighbors and continue
//#define TURN_AROUND_
//uncomment this line to calibrate the robots turnning around
//#define FOLLOW_BLACK_LINES_
//uncomment this line to follow black lines
#define GRADES_
//uncomment this line to calibrate de grades of the robot
//SOME INITIALIZATIONS
#endif SERIAL_BLUETOOTH_ONLY_
SoftwareSerial bluetoothSerial(BLUETOOTH_RECEIVE, BLUETOOTH_TRANSMIT);
#endif
String cadena_iteracion;
MOTOR motor;
CNY cny_left(CNY_LEFT, 800), cny_right(CNY_RIGHT, 800);
x2CNY cnys(cny_left, cny_right);
byte countRed = 0;
byte countGreen = 0;
```

```

byte countBlue = 0;

void setup() {
    // put your setup code here, to run once:
    #ifdef SERIAL_BLUETOOTH_ONLY_
    #ifndef SERIAL_WIRED_ONLY_
        bluetoothSerial.begin(9600);
    #endif
    #endif
    #ifdef SERIAL_WIRED_ONLY_
    #ifndef SERIAL_BLUETOOTH_ONLY_
        Serial.begin(9600);
    #endif
    #endif
    pinMode(A1, INPUT);
    pinMode(S0, OUTPUT);
    pinMode(S1, OUTPUT);
    pinMode(S2, OUTPUT);
    pinMode(S3, OUTPUT);
    pinMode(OUT, INPUT);
    digitalWrite(S0, HIGH);
    digitalWrite(S1, HIGH);
}

//PROBAR CROSS RIGHT AND LEFT

void loop() {
    #ifdef STRAIGHT_FOWARD_
        motor.straight();
        delay(1000);
        motor.back();
        delay(1000);
    #endif
    #ifdef PROBAR_COLORES_
        Serial.println("prueba colores");
        for(int i = 0; i < 50; ++i){
            digitalWrite(S2, LOW);
            digitalWrite(S3, LOW);
            countRed = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
            digitalWrite(S3, HIGH);
            countBlue = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
            digitalWrite(S2, HIGH);
            countGreen = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
            cadena_iteracion = "Contadores\n";
            cadena_iteracion += "Rojo: ";
            cadena_iteracion += String(countRed);
            cadena_iteracion += "\nAzul: ";
            cadena_iteracion += String(countBlue);
            cadena_iteracion += "\nVerde: ";
            cadena_iteracion += String(countGreen);
            cadena_iteracion += "\n";
        }
    #endif
}

```

```

        cadena_iteracion += "Color: ";
        if (countRed < countBlue && countRed < countGreen && countRed < 80)
            cadena_iteracion += "ROJO\n";
        else if (countGreen < countRed && (countGreen - countBlue) < 4)
            cadena_iteracion += "VERDE\n";
        else if (countBlue < countRed && countBlue < countGreen)
            cadena_iteracion += "AZUL\n";
        else cadena_iteracion += "-\n";
    }

    Serial.println(cadena_iteracion);
    delay(1000);
    #endif

#ifdef RUN_ROBOT_
/*while(cnys.bothBlack())
motor.straight();
if(cnys.rightWhite())
while(cnys.rightWhite() && !cnys.bothWhite())
motor.left();
if(cnys.leftWhite())
while(cnys.leftWhite() && !cnys.bothWhite())
motor.right();*/
if(cnys.leftWhite())
while (!cnys.bothBlack() && !cnys.bothWhite())
motor.right();
if(cnys.rightWhite())
while (!cnys.bothBlack() && !cnys.bothWhite())
motor.left();
while(cnys.bothBlack()) {
    motor.straight();
    #ifdef PRINT_CNY_READ_
    cadena_iteracion = "CNY IZQDA.: ";
    cadena_iteracion += analogRead(CNY_LEFT);
    cadena_iteracion += ((cny_left.isWhite() == true) ? " White" : " Black");
    cadena_iteracion += "\nCNY DRCHA.: ";
    cadena_iteracion += analogRead(CNY_RIGHT);
    cadena_iteracion += ((cny_right.isWhite()) ? " White" : " Black");
    cadena_iteracion += "\n";
    Serial.println(cadena_iteracion);
    //delay(2000);
    #endif
}
while(cnys.bothWhite()) {
    motor.botStop();
    #ifdef PRINT_CNY_READ_
    cadena_iteracion = "CNY IZQDA.: ";
    cadena_iteracion += analogRead(CNY_LEFT);
    cadena_iteracion += ((cny_left.isWhite() == true) ? " White" : " Black");
    cadena_iteracion += "\nCNY DRCHA.: ";
}

```

```

cadena_iteracion += analogRead(CNY_RIGHT);
cadena_iteracion += ((cny_right.isWhite()) ? " White" : " Black");
cadena_iteracion += "\n";
Serial.println(cadena_iteracion);
//delay(2000);
#endif
}

#endif
#ifndef RUN_WHITE_PATH_ROBOT_
if(cnys.leftBlack())
while(!cnys.bothBlack() && !cnys.bothWhite())
    motor.right();
if(cnys.rightBlack())
while(!cnys.bothBlack() && !cnys.bothWhite())
    motor.left();
while(cnys.bothWhite())
    motor.straight();
while(cnys.bothBlack())
    motor.botStop();
#endif
#endif
#ifdef PRINT_CNY_READ_
cadena_iteracion = "CNY IZQDA.: ";
cadena_iteracion += analogRead(CNY_LEFT);
cadena_iteracion += ((cny_left.isWhite() == true) ? " White" : " Black");
cadena_iteracion += "\nCNY DRCHA.: ";
cadena_iteracion += analogRead(CNY_RIGHT);
cadena_iteracion += ((cny_right.isWhite()) ? " White" : " Black");
cadena_iteracion += "\n";
Serial.println(cadena_iteracion);
delay(100);
/*Serial.print(analogRead(CNY_LEFT));
Serial.print(" - ");
Serial.println(analogRead(CNY_RIGHT));*/
#endif
//GET COLOUR
/*#ifdef PROBAR_COLORES_
digitalWrite(S2, LOW);
digitalWrite(S3, LOW);
countRed = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
digitalWrite(S3, HIGH);
countBlue = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
digitalWrite(S2, HIGH);
countGreen = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
cadena_iteracion = "Contadores\n";
cadena_iteracion += "Rojo: ";
cadena_iteracion += String(countRed);
cadena_iteracion += "\nAzul: ";
cadena_iteracion += String(countBlue);
```

```

cadena_iteracion += "\nVerde: ";
cadena_iteracion += String(countGreen);
cadena_iteracion += "\n";
cadena_iteracion += "Color: ";
if (countRed < countBlue && countRed < countGreen && countRed < 80)
    cadena_iteracion += "ROJO\n";
else if (countBlue < countRed && countBlue < countGreen)
    cadena_iteracion += "AZUL\n";
else if (countGreen < countRed && countGreen < countBlue)
    cadena_iteracion += "VERDE\n";
else cadena_iteracion += "-\n";
delay(3000);
#endif*/
#endif SERIAL_BLUETOOTH_ONLY_
bluetoothSerial.write(cadena_iteracion.c_str());
#endif
#endif SERIAL_WIRED_ONLY_
//Serial.println(cadena_iteracion.c_str());
#endif
////////////////// NODE OPERATIONS //////////////////
////////////////// *****PSEUDOCODE*****////////////////
* 1. ANT-ROBOTS CNY DETECTS WHITE IN BOTH CNYs
* 2. MOVES STRAIGHT UNTIL BOTH CNYs DETECT BLACK
* 3. DELAY 2 SECONDS
* 4. READS COLOUR
* 5. ANT-ROBOT CONTINUES FOLLOWING LINES
*/
#endif NODE_OPERATIONS_
if(cnys.bothWhite()){
    //Serial.println("both white");
    while(!cnys.bothBlack()){
        if(cnys.leftBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.left();
        if(cnys.rightBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.right();
        while(cnys.bothWhite())
            motor.straight();
    }
    //Serial.println("1. ANT-ROBOTS CNY DETECTS WHITE IN BOTH CNYs");
    motor.botStop();
}
//Serial.println("both black");
if(cnys.leftWhite())
    if(cnys.leftWhite())

```

```

while(!cnys.bothBlack() && !cnys.bothWhite())
    motor.right();
if(cnys.rightWhite())
    while(!cnys.bothBlack() && !cnys.bothWhite())
        motor.left();
while(cnys.bothBlack())
    motor.straight();
for(int i = 0; i < 50; ++i){
    #ifdef PROBAR_COLORES_
    digitalWrite(S2, LOW);
    digitalWrite(S3, LOW);
    countRed = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
    digitalWrite(S3, HIGH);
    countBlue = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
    digitalWrite(S2, HIGH);
    countGreen = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
    cadena_iteracion = "Contadores\n";
    cadena_iteracion += "Rojo: ";
    cadena_iteracion += String(countRed);
    cadena_iteracion += "\nAzul: ";
    cadena_iteracion += String(countBlue);
    cadena_iteracion += "\nVerde: ";
    cadena_iteracion += String(countGreen);
    cadena_iteracion += "\n";
    cadena_iteracion += "Color: ";
    if (countRed < countBlue && countRed < countGreen && countRed < 80)
        cadena_iteracion += "ROJO\n";
    else if (countBlue < countRed && countBlue < countGreen)
        cadena_iteracion += "AZUL\n";
    else if (countGreen < countRed && countGreen < countBlue)
        cadena_iteracion += "VERDE\n";
    else
        cadena_iteracion += "-\n";
    #endif
}
Serial.println(cadena_iteracion);
#endif
#ifndef NEIGHBORS_
/*********************************************
*           SEARCH FOR THE NEIGBORS          *
*********************************************
* 1. STOPS AND READS THE COLOUR
* 2. THE ROBOT MAKES A SWEEP SEARCHING FOR ITS NEIGBORS
* 3. CONTINUES THROUGH THE APPROPRIATE PATH
*****/
bool continueFollowing = false;
while(!cnys.bothWhite()){
    if(cnys.leftWhite())
        while(!cnys.bothBlack() && !cnys.bothWhite())

```

```

        motor.right();
    if(cnys.rightWhite())
        while(!cnys.bothBlack() && !cnys.bothWhite())
            motor.left();
        while(cnys.bothBlack())
            motor.straight();
        continueFollowing = true;
    }

//Serial.println("both white");
if(continueFollowing) {
    continueFollowing = false;
    while(!cnys.bothBlack()) {
        if(cnys.leftBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.right();
        if(cnys.rightBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.left();
        while(cnys.bothWhite())
            motor.straight();
    }
    motor.botStop();
    for(int i = 0; i < 1000; ++i) {
        digitalWrite(S2, LOW);
        digitalWrite(S3, LOW);
        countRed = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
        digitalWrite(S3, HIGH);
        countBlue = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
        digitalWrite(S2, HIGH);
        countGreen = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
        /*cadena_iteracion = "Contadores\n";
        cadena_iteracion += "Rojo: ";
        cadena_iteracion += String(countRed);
        cadena_iteracion += "\nAzul: ";
        cadena_iteracion += String(countBlue);
        cadena_iteracion += "\nVerde: ";
        cadena_iteracion += String(countGreen);
        cadena_iteracion += "\n";*/
        /*do
            motor.straight();
            while(!cnys.bothBlack());*/
    }
    cadena_iteracion += "Color: ";
    if (countRed < countBlue && countRed < countGreen && countRed < 80)
        cadena_iteracion += "ROJO\n";
    else if (countBlue < countRed && countBlue < countGreen)
        cadena_iteracion += "AZUL\n";
    else if (countGreen < countRed && countGreen < countBlue)

```

```

        cadena_iteracion += "VERDE\n";
    else cadena_iteracion += "-\n";
    Serial.println(cadena_iteracion);
    continueFollowing = true;
}

//Serial.println("1. ANT-ROBOTS CNY DETECTS WHITE IN BOTH CNYs");
delay(5000);
if(cnys.bothBlack() && continueFollowing) {
    continueFollowing = false;
    motor.right();
    delay(2000);
    motor.straight();
    delay(1000);
/*while(!cnys.bothBlack()) {
    motor.straight();
}
motor.botStop();*/
}

//NEW aqui se decide si izquierda o dcha
/*do
    motor.crossLeft();
while(!cnys.bothWhite());
delay(1000);
do
    motor.straight();
while(!cnys.bothBlack());
delay(1000);*/
#endif
#ifndef TURN_AROUND_
bool continueFollowing = true;
if(continueFollowing) {
    Serial.println("1 - Avanzo hasta encontrar negro, PRELECTURA");
    continueFollowing = false;
    while(!cnys.bothBlack()) {
        if(cnys.leftBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.right();
        if(cnys.rightBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.left();
        while(cnys.bothWhite())
            motor.straight();
    }
    motor.botStop();
    for(int i = 0; i < 1000; ++i) {
        digitalWrite(S2, LOW);
        digitalWrite(S3, LOW);
        countRed = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
    }
}

```

```

        digitalWrite(S3, HIGH);
        countBlue = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
        digitalWrite(S2, HIGH);
        countGreen = pulseIn(OUT, digitalRead(OUT) == HIGH ? LOW : HIGH);
    }

    cadena_iteracion += "Color: ";
    if (countRed < countBlue && countRed < countGreen && countRed < 80)
        cadena_iteracion += "ROJO\n";
    else if (countBlue < countRed && countBlue < countGreen)
        cadena_iteracion += "AZUL\n";
    else if (countGreen < countRed && countGreen < countBlue)
        cadena_iteracion += "VERDE\n";
    else cadena_iteracion += "-\n";
    Serial.println("2 - Lectura de color");
    Serial.println(cadena_iteracion);
    continueFollowing = true;
}

motor.right();
motor.backRight();
delay(3650);
motor.botStop();

if(continueFollowing){
    Serial.println("3 - Avanzo hasta encontrar negro, POSTLECTURA");
    continueFollowing = false;
    while(!cnys.bothBlack()) {
        if(cnys.leftBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.right();
        if(cnys.rightBlack())
            while(!cnys.bothBlack() && !cnys.bothWhite())
                motor.left();
        while(cnys.bothWhite())
            motor.straight();
    }
    motor.botStop();
}

while(!cnys.bothWhite()){
    Serial.println("4 - Siguelinea");
    if(cnys.leftWhite())
        while(!cnys.bothBlack() && !cnys.bothWhite())
            motor.right();
    if(cnys.rightWhite())
        while(!cnys.bothBlack() && !cnys.bothWhite())
            motor.left();
    while(cnys.bothBlack())
        motor.straight();
    continueFollowing = true;
}

```

```

#endif
#ifndef FOLLOW_BLACK_LINES_
while(!cnys.bothWhite()){
    Serial.println("4 - Sigue linea");
    if(cnys.leftWhite())
        while(!cnys.bothBlack() && !cnys.bothWhite())
            motor.right();
    if(cnys.rightWhite())
        while(!cnys.bothBlack() && !cnys.bothWhite())
            motor.left();
    while(cnys.bothBlack())
        motor.straight();
    Serial.print("RIGHT: ");
    Serial.println(analogRead(CNY_RIGHT));
    Serial.print("LEFT: ");
    Serial.println(analogRead(CNY_LEFT));
}
#endif
#ifndef GRADES_
//90    RIGHT
motor.right();
motor.backRight();
delay(1150);
motor.botStop();
delay(5000);
//90    LEFT
motor.left();
motor.backLeft();
delay(1200);
motor.botStop();
delay(5000);
//180
motor.right();
motor.backRight();
delay(2300);
motor.botStop();
delay(5000);
#endif
}
#endif

```

B.1.4. Código de testeo de la comunicación Bluetooth

Código utilizado para probar la comunicación Bluetooth.

```
#ifndef ACO_INO_
#define ACO_INO_
#include "PINS.hpp"
#include "MAIN_CONTROLLER.hpp"
Main_controller main_controller;
void setup() {
    Serial.begin(9600);
}
void loop() {
    main_controller.run_main_controller();
}
#endif

#ifndef MAIN_CONTROLLER_HPP_
#define MAIN_CONTROLLER_HPP_
#include "PERCEPTION.hpp"
#include "LOGIC.hpp"
#include "ACTUATION.hpp"
#include "COMMON_TYPES.hpp"
class Main_controller {
public:
    //DEFAULT CONSTRUCTOR
    void run_main_controller();
    Perception this_perception() const;
    Logic this_logic() const;
    Actuation this_actuation() const;
private:
    Perception perception_;
    Logic logic_;
    Actuation actuation_;
    String inputString_;
    char inChar_;
    char continue_ = 'Y';
};
#endif

#include "MAIN_CONTROLLER.hpp"
void Main_controller::run_main_controller() {
    Color readen;
    uint8_t grades;
    String on_transmit = "0";
    char junk;
    //readen = perception_.Perceive();
    //if(logic_.isHome(readen)) {
    Path p;
    p.add_path(1);
    p.add_path(5);
```

```

p.add_path(3);
/*p.add_path(2);
p.add_path(4);
p.add_path(3);*/
/*p.add_path(2);
p.add_path(5);
p.add_path(3);*/
/*while(on_transmit == "0") {
    //Serial.println("Vamos a ver...");
    if(Serial.available() && continue_ == 'y') {
        on_transmit = "1";
        //Serial.println("Dentro");
    }
} */
on_transmit = "1";
while(Serial.available() && on_transmit == "1") {
    //Serial.println("HOLA");
    while(inputString_ != "p") {
        inChar_ = (char)Serial.read();
        inputString_ = inChar_;
    }
    Serial.write(p.path().c_str());
    inputString_ = "";
    do {
        if(Serial.available()) {
            inChar_ = (char)Serial.read();
            inputString_ += inChar_;
        }
    } while(inChar_ != 'E');
    on_transmit = "0";
    continue_ = 'n';
    while(Serial.available() > 0) junk = Serial.read();
}
logic_.modify_pheromones(inputString_);
inChar_ = 'X';
//logic_.print_pheromones();
p.clear_path();
//}
//grades = logic_.Proceed(readen);
//actuation_.Response(grades, logic_.correction(), logic_.toogle());
}

Perception Main_controller::this_perception() const {
    return perception_;
}

Logic Main_controller::this_logic() const {
    return logic_;
}

Actuation Main_controller::this_actuation() const {

```

```
    return actuation_;
}
```

B.1.5. Código del robot Arduino

Código para ejecutar el robot en Arduino. Bibliotecas restantes en el Apartado [B.1.1](#).

```
#ifndef ACO_INO_
#define ACO_INO_
#include "PINS.hpp"
#include "MAIN_CONTROLLER.hpp"
Main_controller main_controller;
void setup() {
    Serial.begin(9600);
}
void loop() {
    main_controller.run_main_controller();
}
#endif

#ifndef MAIN_CONTROLLER_HPP_
#define MAIN_CONTROLLER_HPP_
#include "PERCEPTION.hpp"
#include "LOGIC.hpp"
#include "ACTUATION.hpp"
#include "COMMON_TYPES.hpp"
class Main_controller {
public:
    //DEFAULT CONSTRUCTOR
    void run_main_controller();
    Perception this_perception() const;
    Logic this_logic() const;
    Actuation this_actuation() const;
private:
    Perception perception_;
    Logic logic_;
    Actuation actuation_;
};
#endif

#include "MAIN_CONTROLLER.hpp"
void Main_controller::run_main_controller() {
    Color readen;
    uint8_t grades;
    readen = perception_.Perceive();
```

```

//perception_.color_str();
grades = logic_.Proceed(readen);
actuation_.Response(grades, logic_.correction(), logic_.toogle());
}

Perception Main_controller::this_perception() const {
    return perception_;
}

Logic Main_controller::this_logic() const {
    return logic_;
}

Actuation Main_controller::this_actuation() const {
    return actuation_;
}

```

```

#ifndef PERCEPTION_HPP_
#define PERCEPTION_HPP_
#include "Arduino.h"
#include "TSC3200.hpp"
#include "COMMON_TYPES.hpp"

class Perception {
public:
    //DEFAULT CONSTRUCTOR
    //PERCEPTION
    /*
     * TSC3200::Color Perceive();
     * Precondition: -
     * Postcondition: Returns the color that
     * it read
     */
    Color Perceive();
    String color_str() const;
private:
    TSC3200 tsc3200_;
};
#endif

```

```

#include "PERCEPTION.hpp"

String Perception::color_str() const {
    String color_str;
    switch(tsc3200_.getLastColour()){
        case Color::RED:
            color_str = "Red";
            break;
        case Color::GREEN:
            color_str = "Green";
            break;
        case Color::BLUE:
            color_str = "Blue";
    }
}
```

```

break;
case Color::NONE:
    color_str = "None";
    break;
}
return color_str;
}

Color Perception::Perceive() {
    return tsc3200_.getColor();
}

#ifndef LOGIC_HPP_
#define LOGIC_HPP_
#include "Arduino.h"
#include "MAP.hpp"
#include "PATH.hpp"
#include "COMMON_TYPES.hpp"

class Logic {
public:
    // CONSTRUCTOR
    Logic();
    class Way {
public:
    const bool WAY_HOME = false;
    const bool WAY_TO_FOOD = true;
    Way(): way_(WAY_TO_FOOD) {}
    bool WAY() const { return way_; }
    void WAY(bool way) { way_ = way; }
    String toString() const {
        String str = "";
        if(way_ == WAY_HOME) str = "Way Home";
        else str = "Way to Food";
        return str;
    }
private:
    bool way_;
    };
/*
 * uint8_t Proceed()
 * Precondition: -
 * Postcondition: Returns de grades from the next node
 */
uint8_t Proceed(Color);
uint8_t correction() const;
bool isGoal(Color) const;
bool isHome(Color) const;
bool isNone(Color) const;
bool toggle() const;

```

```

        bool& toogle();
        String actual_node_str() const;
        String path_str() const;
        String print_pheromones() const;
        void modify_pheromones(String);
        //String actual_adjacent_list() const;
        String way_str() const;

private:
    Map map_;
    Path path_, final_path_;
    bool first_;
    uint8_t correction_;
    Way way_;
    bool toogle_;
    String inputString_;
    char inChar_;
    char continue_;

};

#endif


---


#include "LOGIC.hpp"
Logic::Logic():
    map_(0),
    first_(true),
    continue_('y')
{
}

uint8_t Logic::Proceed(Color color) {
    Map::summit next_node;
    uint8_t selected_grade, actual_node_path;
    String on_transmit = "0";
    char junk;

    if(isHome(color)) way_.WAY( way_.WAY_TO_FOOD );
    if(isGoal(color)) way_.WAY( way_.WAY_HOME );
    if(way_.WAY() == way_.WAY_HOME || isGoal(color)) {
        if(isGoal(color)) {
            final_path_ = path_;
            map_.grades_toogle_conversion();
            toogle_ = true;
            //Serial.println("IS GOAL");
            /*Serial.print("Common toogle: ");
            Serial.println(toogle_);*/
        }
        /*Serial.print("PATH POP ");
        if(way_.WAY() == way_.WAY_HOME)
            Serial.println("WAY home");
        else
            Serial.println("WAY food");*/
    }
}

```

```

actual_node_path = path_.remove_last();
next_node = path_.last_element();
map_.actual(next_node);
selected_grade = map_.grades[actual_node_path][next_node];
correction_ = map_.stragihten_grades[actual_node_path][next_node];
/*Serial.print("ACTUAL NODE: ");
Serial.println(actual_node_path);
Serial.print("NEXT_NODE: ");
Serial.println(next_node);
Serial.println(path_.path());*/
} else {
    /*Serial.print("First value: ");
    Serial.println(first_);*/
    if(isHome(color) && first_) {
        while(on_transmit == "0") {
            if(Serial.available() && continue_ == 'y') {
                on_transmit = "1";
            }
        }
        while(Serial.available() && on_transmit == "1") {
            inputString_ = "";
            do {
                if(Serial.available()) {
                    inChar_ = (char)Serial.read();
                    inputString_ += inChar_;
                }
            } while(inChar_ != 'E');
            modify_pheromones(inputString_);
            //print_pheromones();
            on_transmit = "0";
            continue_ = 'n';
        }
    } else if(isHome(color) && !first_) {
        /*Serial.println("PATH...");*/
        Serial.write(final_path_.path().c_str());
        Serial.println();*/
        while(on_transmit == "0") {
            if(Serial.available() && continue_ == 'y') {
                on_transmit = "1";
            }
        }
        while(Serial.available() && on_transmit == "1") {
            while(inputString_ != "p") {
                inChar_ = (char)Serial.read();
                inputString_ = inChar_;
            }
            inputString_ = "";
            Serial.write(final_path_.path().c_str());

```

```

    do {
        if(Serial.available()) {
            inChar_ = (char)Serial.read();
            inputString_ += inChar_;
        }
    } while(inChar_ != 'E');

    modify_pheromones(inputString_);
    //print_pheromones();
    on_transmit = "0";
    continue_ = '\n';
    //while(Serial.available() > 0) junk = Serial.read();
}

inChar_ = 'X';
map_.grades_toogle_conversion();
final_path_.clear_path();
path_.clear_path();
map_.restart_visited();
toogle_ = true;
/*Serial.print("Common toogle: ");
Serial.println(toogle_);*/
} else
first_ = false;
/*Serial.print("PATH POP ");
if(way_.WAY() == way_.WAY_HOME)
Serial.println("WAY home");
else
Serial.println("WAY food");*/
next_node = map_.next_node(map_.actual());
path_.add_path(next_node);
/*Serial.println("INDEX OF MAP_GRADES:");
Serial.print("MAP_.PREVIOUS(): ");
Serial.println(map_.previous());
Serial.print("NEXT_NODE: ");
Serial.println(next_node);*/
selected_grade = map_.grades[map_.previous()][next_node];
correction_ = map_.stragighten_grades[map_.previous()][next_node];
}
//p.clear_path();
continue_ = 'Y';
return selected_grade;
}

uint8_t Logic::correction() const {
    return correction_;
}

bool Logic::isGoal(Color color) const {
    return (color == Color::GREEN);
}

bool Logic::isHome(Color color) const {

```

```

    return (color == Color::BLUE);
}

bool Logic::isNone(Color color) const {
    return (color == Color::NONE);
}

bool Logic::toogle() const {
    return toogle_;
}

bool& Logic::toogle() {
    return toogle_;
}

String Logic::actual_node_str() const {
    return String(map_.actual());
}

String Logic::path_str() const {
    return path_.path();
}

void Logic::modify_pheromones(String inputString) {
    map_.modify_pheromones_from_str(inputString);
}

String Logic::print_pheromones() const {
    for(size_t i = 0; i < map_.n_; i++) {
        for(size_t j = 0; j < map_.n_; j++) {
            Serial.print(map_.PHEROMONES[i][j].toString() + " ");
        }
        //Serial.println();
    }
}

/*String Logic::actual_adjacent_list() const {
    return map_.print_adjacent_list();
} */

String Logic::way_str() const {
    return way_.toString();
}



---


#ifndef ACTUATION_HPP_
#define ACTUATION_HPP_
#include "Arduino.h"
#include "MOTOR.hpp"
#include "CNY.hpp"
#include "2xCNY.hpp"
#include "COMMON_TYPES.hpp"

class Actuation {
public:
    //CONSTRUCTOR
    Actuation();
    //ACTUATION
    void Response(uint8_t grades, uint8_t correction, bool& toogle);
}

```

```

void straighten();
String previous_grade() const;
int to_grades(uint8_t) const;
private:
    MOTOR motor_;
    CNY cny_left_, cny_right_;
    x2CNY cnys_;
    int previous_grade_;
};

#endif


---


#include "ACTUATION.hpp"
Actuation::Actuation():
    cny_left_(CNY_LEFT, 900),
    cny_right_(CNY_RIGHT, 900),
    cnys_(cny_left_, cny_right_),
    previous_grade_(0)
{}

void Actuation::Response(uint8_t grades, uint8_t correction, bool& toogle) {
    if(toogle) {
        //Serial.println("TOOGLE ROBOT");
        motor_.roll_grades(180);
        toogle = false;
    }
    /*Serial.print("INPUT GRADES: ");
    Serial.println(grades);*/
    previous_grade_ = to_grades(grades);
    /*Serial.print("GRADES CONVERTED: ");
    Serial.println(previous_grade_);*/
    motor_.roll_grades(previous_grade_);
    if(cnys_.bothWhite()) {
        while(!cnys_.bothBlack()) {
            if(cnys_.leftBlack())
                while(!cnys_.bothBlack() && !cnys_.bothWhite())
                    motor_.right();
            if(cnys_.rightBlack())
                while(!cnys_.bothBlack() && !cnys_.bothWhite())
                    motor_.left();
            while(cnys_.bothWhite())
                motor_.straight();
        }
    }
    while(!cnys_.bothWhite()){
        if(cnys_.leftWhite())
            while(!cnys_.bothBlack() && !cnys_.bothWhite())
                motor_.right();
        if(cnys_.rightWhite())
            while(!cnys_.bothBlack() && !cnys_.bothWhite())

```

```

        motor_.left();
        while(cnys_.bothBlack())
            motor_.straight();
    }

motor_.botStop();
delay(2000);
while (!cnys_.bothBlack()) {
    if(cnys_.leftBlack())
        while (!cnys_.bothBlack() && !cnys_.bothWhite())
            motor_.right();
    if(cnys_.rightBlack())
        while (!cnys_.bothBlack() && !cnys_.bothWhite())
            motor_.left();
    while(cnys_.bothWhite())
        motor_.straight();
}

motor_.botStop();
delay(2000);
/*Serial.print("INPUT CORRECTION: ");
Serial.println(correction);*/
motor_.roll_grades(to_grades(correction));
}

int Actuation::to_grades(uint8_t matrix_grades) const {
    int grades_to_roll = 0;
    switch(matrix_grades) {
        case 0:
            grades_to_roll = 0;
            break;
        case 1:
            grades_to_roll = 90;
            break;
        case 2:
            grades_to_roll = 180;
            break;
        case 3:
            grades_to_roll = 270;
            break;
    }
    return grades_to_roll;
}

void Actuation::straighten() {
    /*int grades_to_roll;

switch(previous_grade_) {
    case 90:
        grades_to_roll = 270;
        break;
    case 270:

```

```

        grades_to_roll = 90;
    break;
default:
    grades_to_roll = previous_grade_;
break;
}
Serial.print("GRADES TO ROLL TO STRAIGHT BOT: ");
Serial.println(grades_to_roll);
motor_.roll_grades(grades_to_roll);
*/
}
String Actuation::previous_grade() const {
    return String(previous_grade_);
}

```

B.1.6. Código del servidor

Código referente al servidor. Bibliotecas externas en [B.1.2](#).

```

#include "mainwindow.h"
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QListWidgetItem>
#include <QStandardItemModel>
#include <QBluetoothSocket>
#include <QBluetoothDeviceDiscoveryAgent>
#include "main_controller.h"
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);

```

```

~MainWindow();
void update_pheromones_view();

signals:
void messageReceived(const QString &sender, const QString &message);

private slots:
void device_discovered(const QBluetoothDeviceInfo&);
void when_connected();
void when_disconnected();
void on_scan_button_clicked();
void on_bt_name_list_itemClicked(QListWidgetItem *item);
void on_trasnmit_button_clicked();
void on_clear_terminal_button_clicked();
void on_close_button_clicked();
void on_init_bot_button_clicked();

private:
Ui::MainWindow *ui;
QListWidgetItem *selected_;
QStandardItemModel* model_ = new QStandardItemModel(6,6,this);
Main_controller controller_;
QBluetoothDeviceDiscoveryAgent *agent_ = new QBluetoothDeviceDiscoveryAgent;
QBluetoothSocket *socket_;
QString string_address_;
QString string_name_;

};

#endif // MAINWINDOW_H

```

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow)
{
    QFont font;
    font.setWeight(QFont::Bold);
    ui->setupUi(this);
    for(int i = 0; i < 6; i++) {
        model_->setHorizontalHeaderItem(i, new QStandardItem(QString::number(i)));
        model_->setVerticalHeaderItem(i, new QStandardItem(QString::number(i)));
    }
    ui->pheromones_table->horizontalHeader()->setFont(
        font
    );
    ui->pheromones_table->verticalHeader()->setFont(
        font
    );
    ui->pheromones_table->setModel(model_);
    ui->pheromones_table->setShowGrid(false);
    ui->Lpicture->setVisible(true);
}

```

```

connect (    agent_ ,
             SIGNAL(deviceDiscovered(QBluetoothDeviceInfo)),
             this,
             SLOT(device_discovered(QBluetoothDeviceInfo))
         );
agent_->start ();
socket_ = new QBluetoothSocket (
             QBluetoothServiceInfo::RfcommProtocol
         );
connect (socket_ ,
             SIGNAL (disconnected()),
             this,
             SLOT (when_disconnected())
         );
connect (socket_ ,
             SIGNAL (connected()),
             this,
             SLOT (when_connected())
         );
QList<QStandardItem*> item;
for(int i = 0; i < 6; i++)
    for(int j = 0; j < 6; j++) {
        item.append(
            new QStandardItem(
                QString::number(controller_.matrix().pheromones_[i][j].resolv())
            )
        );
    }

    model_->setItem(i, j, item[(i*6)+j]);
}
ui->pheromones_table->setModel (model_);
}

void MainWindow::device_discovered(const QBluetoothDeviceInfo& device) {
    ui->bt_name_list->addItem(
        device.address().toString() + ":" + device.name()
    );
}

void MainWindow::when_connected() {
    ui->event_list->setEnabled(true);
    ui->clear_terminal_button->setEnabled(true);
    ui->connected_name_label->setText(string_name_);
    ui->trasmit_button->setEnabled(true);
}

void MainWindow::when_disconnected() {
    ui->event_list->setEnabled(false);
    ui->connected_name_label->setText("No device connected");
    selected_->setSelected(false);
}

```

```

void MainWindow::update_pheromones_view() {
    QList<QStandardItem*> item;
    for(int i = 0; i < 6; i++)
        for(int j = 0; j < 6; j++) {
            item.append(
                new QStandardItem(
                    QString::number(controller_.matrix().pheromones_[i][j].resolv())
                )
            );
            model_->setItem(i, j, item[(i*6)+j]);
        }
    ui->pheromones_table->setModel(model_);
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::on_scan_button_clicked()
{
    selected_->setSelected(false);
    ui->bt_name_list->clear();
    socket_->disconnectFromService();
    agent_->stop();
    agent_->start();
}
void MainWindow::on_bt_name_list_itemClicked(QListWidgetItem *item)
{
    socket_->disconnectFromService();
    selected_ = item;
    selected_->setSelected(true);
    QString string = item->text();
    QStringList slist = string.split(":");
    string_address_ = slist[0];
    string_name_ = slist[1];
    static const QLatin1String serviceUuid("00001101-0000-1000-8000-00805F9B34FB");
    socket_->connectToService(QBluetoothAddress(string_address_),
                               QBluetoothUuid(serviceUuid),
                               QIODevice::ReadWrite
                           );
}
void MainWindow::on_trasnmit_button_clicked()
{
    QByteArray line[10];
    QString path = "";
    QString aux = "";
    int i = 0;
    socket_->write("p");
    ui->event_list->addItem(string_name_ + " transmitting...");
}

```

```

while(socket_->bytesAvailable() > 0) {
    qDebug() << socket_->isOpen();
    qDebug() << socket_->canReadLine();
    qDebug() << socket_->bytesToWrite();
    line[i] = socket_->readLine().trimmed();
    emit messageReceived(socket_->peerName(),
                         QString::fromUtf8(line[i].constData(), line[i].length()));
    ui->event_list->addItem(line[i]);
    path = line[0];
    i++;
}
if(!path.isEmpty()) {
    controller_.path_from_str(
        QString::fromUtf8(line[0].constData(), line[0].length()
));
    controller_.path().update_pheromones(
        controller_.matrix()
);
    socket_->write(controller_.matrix().toString().toStdString().c_str());
    ui->event_list->addItem(controller_.matrix().toString().toStdString().c_str());
    path = "";
    for(int i = 0; i < 6; i++) {
        for(int j = 0; j < 6; j++) {
            aux += controller_.matrix().pheromones_[i][j].resolv();
            aux += " ";
        }
        aux += "\n";
    }
}
qDebug() << aux;
while(socket_->bytesAvailable() > 0) {
    i = 0;
    line[i] = socket_->readLine().trimmed();
    emit messageReceived(socket_->peerName(),
                         QString::fromUtf8(line[i].constData(), line[i].length()));
}
update_pheromones_view();
ui->event_list->addItem(string_name_ + " end of transmission.");
}

void MainWindow::on_clear_terminal_button_clicked()
{
    ui->event_list->clear();
}

void MainWindow::on_close_button_clicked()
{
    selected_->setSelected(false);
    ui->bt_name_list->clear();
    socket_->disconnectFromService();
}

```

```

    agent_->stop();
    agent_->start();
}

void MainWindow::on_init_bot_button_clicked()
{
    ui->event_list->addItem(string_name_ + " transmitting...");
    socket_->write(controller_.matrix().toString().toStdString().c_str());
    ui->event_list->addItem(controller_.matrix().toString().toStdString().c_str());
    ui->event_list->addItem(string_name_ + " end of transmission.");
}

#ifndef MAIN_CONTROLLER_H
#define MAIN_CONTROLLER_H
#include <QString>
#include "matrix.h"
#include "path.h"
class Main_controller {
public:
    Matrix& matrix();
    Path& path();
    const Path& path_from_str(QString);
private:
    Matrix matrix_;
    Path path_;
};

#endif // MAIN_CONTROLLER_H

#include "main_controller.h"
Matrix& Main_controller::matrix() {
    return matrix_;
}
Path& Main_controller::path() {
    return path_;
}
const Path& Main_controller::path_from_str(QString str) {
    return path_.fromString(str);
}

```

B.1.7. Código específico de comunicación

Este Apartado se divide en dos partes:

- Comunicación por parte del robot.

- Comunicación por parte del servidor.

B.1.7.1. Comunicación del robot

Código referente a la comunicación por parte del robot en Arduino.

```

if(isHome(color) && first_) {
    while(on_transmit == "0") {
        if(Serial.available() && continue_ == 'y') {
            on_transmit = "1";
        }
    }
    while(Serial.available() && on_transmit == "1") {
        inputString_ = "";
        do {
            if(Serial.available()) {
                inChar_ = (char)Serial.read();
                inputString_ += inChar_;
            }
        } while(inChar_ != 'E');
        modify_pheromones(inputString_);
        //print_pheromones();
        on_transmit = "0";
        continue_ = 'n';
    }
} else if(isHome(color) && !first_) {
    /*Serial.println("PATH...");*/
    Serial.write(final_path_.path().c_str());
    Serial.println();*/
    while(on_transmit == "0") {
        if(Serial.available() && continue_ == 'y') {
            on_transmit = "1";
        }
    }
    while(Serial.available() && on_transmit == "1") {
        while(inputString_ != "p") {
            inChar_ = (char)Serial.read();
            inputString_ = inChar_;
        }
        inputString_ = "";
        Serial.write(final_path_.path().c_str());
        do {
            if(Serial.available()) {
                inChar_ = (char)Serial.read();
                inputString_ += inChar_;
            }
        }
    }
}

```

```

    } while(inChar_ != 'E');

    modify_pheromones(inputString_);

    //print_pheromones();

    on_transmit = "0";
    continue_ = 'n';

    //while(Serial.available() > 0) junk = Serial.read();

}

inChar_ = 'X';

map_.grades_toogle_conversion();

final_path_.clear_path();

path_.clear_path();

map_.restart_visited();

toogle_ = true;

/*Serial.print("Common toogle: ");

Serial.println(toogle_); */

} else

first_ = false;

```

B.1.7.2. Comunicación del servidor

Código referente a la comunicación por parte del servidor.

```

MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent)
, ui(new Ui::MainWindow)
{
    QFont font;
    font.setWeight(QFont::Bold);
    ui->setupUi(this);
    for(int i = 0; i < 6; i++) {
        model_->setHorizontalHeaderItem(i, new QStandardItem(QString::number(i)));
        model_->setVerticalHeaderItem(i, new QStandardItem(QString::number(i)));
    }
    ui->pheromones_table->horizontalHeader()->setFont(
        font
    );
    ui->pheromones_table->verticalHeader()->setFont(
        font
    );
    ui->pheromones_table->setModel(model_);
    ui->pheromones_table->setShowGrid(false);
    ui->Lpicture->setVisible(true);
    connect(    agent_,

        SIGNAL(deviceDiscovered(QBluetoothDeviceInfo)),
        this,

```

```

        SLOT(device_discovered(QBluetoothDeviceInfo))
    );
agent_->start();
socket_ = new QBluetoothSocket(
    QBluetoothServiceInfo::RfcommProtocol
);
connect(socket_,
    SIGNAL (disconnected()),
    this,
    SLOT (when_disconnected())
);
connect(socket_,
    SIGNAL (connected()),
    this,
    SLOT (when_connected())
);
QList<QStandardItem*> item;
for(int i = 0; i < 6; i++)
    for(int j = 0; j < 6; j++) {
        item.append(
            new QStandardItem(
                QString::number(controller_.matrix().pheromones_[i][j].resolv())
            )
        );
    }
    model_->setItem(i, j, item[(i*6)+j]);
}
ui->pheromones_table->setModel (model_);
}

void MainWindow::device_discovered(const QBluetoothDeviceInfo& device) {
    ui->bt_name_list->addItem(
        device.address().toString() + ":" + device.name()
    );
}

void MainWindow::when_connected() {
    ui->event_list->setEnabled(true);
    ui->clear_terminal_button->setEnabled(true);
    ui->connected_name_label->setText(string_name_);
    ui->trasmit_button->setEnabled(true);
}

void MainWindow::when_disconnected() {
    ui->event_list->setEnabled(false);
    ui->connected_name_label->setText("No device connected");
    selected_->setSelected(false);
}

void MainWindow::on_scan_button_clicked()
{
    selected_->setSelected(false);
}

```

```

ui->bt_name_list->clear();
socket_->disconnectFromService();
agent_->stop();
agent_->start();
}

void MainWindow::on_bt_name_list_itemClicked(QListWidgetItem *item)
{
    socket_->disconnectFromService();
    selected_ = item;
    selected_->setSelected(true);
    QString string = item->text();
    QStringList slist = string.split(":");
    string_address_ = slist[0];
    string_name_ = slist[1];
    static const QLatin1String serviceUuid("00001101-0000-1000-8000-00805F9B34FB");
    socket_->connectToService(QBluetoothAddress(string_address_),
                               QBluetoothUuid(serviceUuid),
                               QIODevice::ReadWrite
                             );
}

void MainWindow::on_trasnmit_button_clicked()
{
    QByteArray line[10];
    QString path = "";
    QString aux = "";
    int i = 0;
    socket_->write("p");
    ui->event_list->addItem(string_name_ + " transmitting...");
    while(socket_->bytesAvailable() > 0) {
        qDebug() << socket_->isOpen();
        qDebug() << socket_->canReadLine();
        qDebug() << socket_->bytesToWrite();
        line[i] = socket_->readLine().trimmed();
        emit messageReceived(socket_->peerName(),
                           QString::fromUtf8(line[i].constData(), line[i].length()));
        ui->event_list->addItem(line[i]);
        path = line[0];
        i++;
    }
    if(!path.isEmpty()) {
        controller_.path_from_str(
            QString::fromUtf8(line[0].constData(), line[0].length()
        ));
        controller_.path().update_pheromones(
            controller_.matrix()
        );
        socket_->write(controller_.matrix().toString().toStdString().c_str());
        ui->event_list->addItem(controller_.matrix().toString().toStdString().c_str());
    }
}

```

```

    path = "";
    for(int i = 0; i < 6; i++) {
        for(int j = 0; j < 6; j++) {
            aux += controller_.matrix().pheromones_[i][j].resolv();
            aux += " ";
        }
        aux += "\n";
    }
}

qDebug() << aux;
while(socket_->bytesAvailable() > 0) {
    i = 0;
    line[i] = socket_->readLine().trimmed();
    emit messageReceived(socket_->peerName(),
                         QString::fromUtf8(line[i].constData(), line[i].length()));
}
update_pheromones_view();
ui->event_list->addItem(string_name_ + " end of transmission.");
}

void MainWindow::on_clear_terminal_button_clicked()
{
    ui->event_list->clear();
}

void MainWindow::on_close_button_clicked()
{
    selected_->setSelected(false);
    ui->bt_name_list->clear();
    socket_->disconnectFromService();
    agent_->stop();
    agent_->start();
}

void MainWindow::on_init_bot_button_clicked()
{
    ui->event_list->addItem(string_name_ + " transmitting...");
    socket_->write(controller_.matrix().toString().toStdString().c_str());
    ui->event_list->addItem(controller_.matrix().toString().toStdString().c_str());
    ui->event_list->addItem(string_name_ + " end of transmission.");
}

```

B.1.8. Implementación del Código en C++

Código desarrollado en C++ que ejecuta el ACO. Lo desglosaremos en los siguientes apartados:

- Bibliotecas que usa el programa en C++.
- Código principal en C++.
- Makefile

B.1.8.1. Bibliotecas que usa el programa en C++

En el siguiente Apartado mostraremos las bibliotecas utilizadas por el programa en C++:

- “camino.hpp”.
- “hormigas.hpp”.
- “listaenla.hpp”.
- “tablero.hpp”.

Código del archivo camino.hpp.

```
#ifndef CAMINO_HPP_
#define CAMINO_HPP_
#include "listaenla.hpp"
#include "hormigas.hpp"
template <typename T>
class Camino{
public:
    typedef typename std::size_t vertice;
    typedef T tCoste;
    /*Metodo para agregar un nodo a la lista de vertices
     *recorridos junto al coste de ir del anterior al actual
     *PRECONDICION: el nodo n no ha sido visitado aun
     *POSTCONDICION: agrega un nuevo nodo a la lista de visitados
     *y a ade el coste
    */
    void nuevoNodo(vertice n, tCoste c);
    /*Metodo para volver al inicio
     *PRECONDICION: -
     *POSTCONDICION: deja el camino vacio(solo vertice inicial)
     *y el coste a 0
    */
    void volver();
    /*Metodo para saber en que nodo se encuentra nuestra hormiga
```

```

PRECONDICIoN: hay una hormiga asignada a un tablero (lo
sabemos si hemos inicializado el vertice inicial)
POSTCONDICIoN: devuelve el vertice en el que se encuentra
la hormiga
*/
const vertice& actual();
/* Metodo para saber si un nodo ya ha sido visitado
PRECONDICIoN: -
POSTCONDICIoN: devuelve verdadero si el camino ha sido
visitado, falso si no lo ha sido
*/
bool haVisitado(vertice i) const{
    return camino.buscar(i) != camino.fin();
}
/* Metodo para obtener el coste del camino recorrido
PRECONDICIoN: -
POSTCONDICIoN: devuelve el coste del camino recorrido
*/
const float& costeCamino() const{
    return coste;
}
/* Metodo para saber los nodos recorridos de un camino
PRECONDICIoN: -
POSTCONDICIoN: devuelve una lista de vertices recorridos
*/
const Lista<vertice>& caminoRecorrido() const{
    return camino;
}
*****OPERADOR DE INSERCION EN FLUJO*****
template <class U>
friend std::ostream& operator <<
    (std::ostream& os, const Camino<U>& cam);
private:
/*
***** DECLARACION DE TIPOS PRIVADOS *****
*****
Lista<vertice> camino;
tCoste coste;
};

template<typename U>
std::ostream& operator <<(std::ostream& os, const Camino<U>& cam) {
    os << "Camino recorrido: ";
    bool primera = false;
    for(typename Lista<typename Camino<U>::vertice>::posicion it =
cam.camino.primer(); it != cam.camino.fin();
        it = cam.camino.siguiente(it)){
        os << ((primera)? " -> ":"") << cam.camino[it];
        primera = true;
}

```

```

        }

        os << "\nCoste del camino: " << cam.coste << "\n";
        return os;
    }

    /*****IMPLEMENTACION DE METODOS*****/

    template <typename T>
    void Camino<T>::nuevoNodo
    {
        (Camino<T>::vertice n, Camino<T>::tCoste c) {
            coste += c;
            camino.insertar(n, camino.fin());
        }
    }

    template <typename T>
    void Camino<T>::volver()
    {
        for(Lista<vertice>::posicion it = camino.anterior(camino.fin()));
            it != camino.primera(); it = camino.anterior(it))
            camino.eliminar(it);
        coste = 0.0;
    }

    template <typename T>
    const typename Camino<T>::vertice& Camino<T>::actual()
    {
        return camino[
            camino.anterior(
                camino.fin()
            )
        ];
    }
}

#endif

```

Código del archivo hormigas.hpp.

```

#ifndef HORMIGA_HPP_
#define HORMIGA_HPP_
#include <iostream>
#include "listaenla.hpp"
#include "camino.hpp"

template <typename T>
class Hormiga{
    public:
    /***** DECLARACION DE SUBTIPOS *****/
    *                                     DECLARACION DE SUBTIPOS
    ****
    typedef typename std::size_t vertice;
    typedef T tCoste;
    /***** OPERACIONES DEL TAD HORMIGA *****/
    *                                     OPERACIONES DEL TAD HORMIGA
    ****
    /*CONSTRUCTOR sobrecargado pasandole un identificador

```

```

unico en un tablero

PRECONDICIoN: -
POSTCONDICIoN: se crea una nueva hormiga
*/
explicit Hormiga(size_t i):id(i), visitados(){}
/*OBSERVADOR del identificador de la hormiga
PRECONDICIoN: -
POSTCONDICIoN: devuelve el identificador de esta hormiga
*/
size_t identificador()const{return id;}
/*OBSERVADOR Y MODIFICADOR de la lista de nodos visitados*/
const Camino<tCoste>& verticesVisitados()const{
    return visitados;
}
Camino<tCoste>& verticesVisitados() {return visitados;}
/*********************************************
*
*               MetODOS WRAPPED DEL TAD CAMINO
*
********************************************/
/*Una hormiga visita un vertice nuevo
PRECONDICIoN: -
POSTCONDICIoN: agrega un nuevo nodo visitado con
el coste de ir del anterior a este
*/
void nuevoNodoVisitado(vertice n, tCoste c){
    visitados.nuevoNodo(n, c);
}
/*La hormiga vuelve al hormiguero
PRECONDICIoN: -
POSTCONDICIoN: vacia la lista de nodos visitados dejando
solo el primero que es el hormiguero, y deja
el coste a 0
*/
void volver(){visitados.volver();}
/*Metodo para saber si un nodo ya ha sido visitado
PRECONDICIoN: -
POSTCONDICIoN: devuelve verdadero si el camino
ha sido visitado, falso si no lo ha sido
*/
bool haVisitado(vertice i) const{
    return visitados.haVisitado(i);
}
/*Metodo para saber en que nodo se encuentra nuestra hormiga
PRECONDICIoN: hay una hormiga asignada a un tablero(lo
sabemos si hemos inicializado el vertice inicial)
POSTCONDICIoN: devuelve el vertice en el que se encuentra
la hormiga
*/
const vertice& actual(){

```

```

        return visitados.actual();
    }

*****OPERADOR DE INSERCION EN FLUJO*****

    template <class U>
    friend std::ostream& operator <<
        (std::ostream& os, const Hormiga<U>& h);

private:
    size_t id;
    Camino<tCoste> visitados;
};

*****OPERADORES == y !=*****

template<typename T>
bool operator ==(const Hormiga<T>& h1, const Hormiga<T>& h2) {
    return h1.identificador() == h2.identificador();
}

template<typename T>
bool operator !=(const Hormiga<T>& h1, const Hormiga<T>& h2) {
    return !(h1 == h2);
}

*****OPERADOR DE INSERCION EN FLUJO*****

template<typename T>
std::ostream& operator <<(std::ostream& os, const Hormiga<T>& h) {
    os << "*****HORMIGA " << h.id << "*****\n";
    os << h.visitados ;
    return os;
}

#endif

```

Código del archivo listaenla.hpp.

```

-----*/
/* listaenla.h */ 
/*
/* clase Lista genrica mediante celdas enlazadas. */
/* Despus de una insercin o eliminacin en una */
/* posicion p, las variables externas de tipo posicion */
/* posteriores a p continan representando las */
/* posiciones de los mismos elementos. */
-----*/

#ifndef LISTA_ENLA_H
#define LISTA_ENLA_H
#include <cassert>
#include <iostream>

template <typename T> class Lista {
    struct nodo; // declaracion adelantada privada
public:
    typedef nodo* posicion;

```

```

// posicion de un elemento
Lista();
// constructor, requiere ctor. T()
Lista(std::size_t n, const T& e);
Lista(const Lista<T>& l);
// ctor. de copia, requiere ctor. T()
Lista<T>& operator =(const Lista<T>& l);
// asignacion de listas
void insertar(const T& x, posicion p);
void eliminar(posicion p);
const T& elemento(posicion p) const;
// acceso a elto, lectura
T& elemento(posicion p);
// acceso a elto, lectura/escritura
posicion buscar(const T& x) const;
// T requiere operador ==
posicion siguiente(posicion p) const;
posicion anterior(posicion p) const;
posicion primera() const;
posicion fin() const;
// posicion despues del ltimo
bool vacia() const;
~Lista();
// destructor

// Usado para algoritmos de grafos
Lista<T>& operator +=(const Lista<T>& l);
// concatenacin
const T& operator [] (posicion) const;
T& operator [] (posicion);
size_t size() const;

private:
struct nodo {
    T elto;
    nodo* sig;
    nodo(const T& e, nodo* p = 0): elto(e), sig(p) {}
};

nodo* L; // lista enlazada de nodos
void copiar(const Lista<T>& l);
};

template <typename T>
std::ostream& operator << (std::ostream& os, Lista<T> l){
    for(typename Lista<T>::posicion p =
        l.primer(), p != l.fin(); p = l.siguiente(p))
        os << l.elemento(p) << " ";
    os << "\n";
    return os;
}

```

```

/*-----*/
/* clase Lista genrica mediante celdas enlazadas con      */
/* cabecera.                                              */
/* La posicion de un elemento se representa mediante      */
/* un puntero al nodo anterior.                           */
/* La primera posicion es un puntero al nodo cabecera. */
/* La posicion fin es un puntero al ltimo nodo.          */
/* Implementacin de operaciones                         */
/*-----*/
// Mtodo privado
template <typename T>
void Lista<T>::copiar(const Lista<T> &l)
{
    L = new nodo(T()); // crear el nodo cabecera
    nodo* q = L;
    for (nodo* r = l.L->sig; r; r = r->sig) {
        q->sig = new nodo(r->elto);
        q = q->sig;
    }
}
template <typename T>
inline Lista<T>::Lista() : L(new nodo(T())) // crear cabecera
{}
template <typename T>
inline Lista<T>::Lista(std::size_t n, const T& e) {
    L = new nodo(T());
    nodo* q = L;
    for(std::size_t r = 0; r < n; r++) {
        q->sig = new nodo(e);
        q = q->sig;
    }
}
template <typename T>
inline Lista<T>::Lista(const Lista<T>& l)
{
    copiar(l);
}
template <typename T>
Lista<T>& Lista<T>::operator =(const Lista<T>& l)
{
    if (this != &l) { // evitar autoasignacin
        this->~Lista(); // vaciar la lista actual
        copiar(l);
    }
    return *this;
}
template <typename T> inline

```

```

void Lista<T>::insertar(const T& x, Lista<T>::posicion p)
{
    p->sig = new nodo(x, p->sig);
    // el nuevo nodo con x queda en la posicion p
}

template <typename T>
inline void Lista<T>::eliminar(Lista<T>::posicion p)
{
    assert(p->sig); // p no es fin
    nodo* q = p->sig;
    p->sig = q->sig;
    delete q;
    // el nodo siguiente queda en la posicion p
}

template <typename T> inline
const T& Lista<T>::elemento(Lista<T>::posicion p) const
{
    assert(p->sig); // p no es fin
    return p->sig->elto;
}

template <typename T>
inline T& Lista<T>::elemento(Lista<T>::posicion p)
{
    assert(p->sig); // p no es fin
    return p->sig->elto;
}

template <typename T>
typename Lista<T>::posicion
Lista<T>::buscar(const T& x) const
{
    nodo* q = L;
    bool encontrado = false;
    while (q->sig && !encontrado)
        if (q->sig->elto == x)
            encontrado = true;
        else q = q->sig;
    return q;
}

template <typename T> inline
typename Lista<T>::posicion
Lista<T>::siguiente(Lista<T>::posicion p) const
{
    assert(p->sig); // p no es fin
    return p->sig;
}

template <typename T>
typename Lista<T>::posicion
Lista<T>::anterior(Lista<T>::posicion p) const

```

```

{
    nodo* q;
    assert(p != L); // p no es la primera posicion
    for (q = L; q->sig != p; q = q->sig);
    return q;
}

template <typename T>
inline typename Lista<T>::posicion Lista<T>::primera() const
{
    return L;
}

template <typename T>
typename Lista<T>::posicion Lista<T>::fin() const
{
    nodo* p;
    for (p = L; p->sig; p = p->sig);
    return p;
}

template <typename T>
bool Lista<T>::vacia() const {
    return primera() == fin();
}

template <typename T>
const T& Lista<T>::operator [] (Lista<T>::posicion p) const{
    return elemento(p);
}

template <typename T>
T& Lista<T>::operator [] (Lista<T>::posicion p){
    return elemento(p);
}

template <typename T>
size_t Lista<T>::size() const{
    int i = 0;
    for(Lista<T>::posicion p = L; p != fin(); p = p->sig) i++;

    return i;
}

// Destructor: destruye el nodo cabecera y vaca la lista
template <typename T> Lista<T>::~Lista()
{
    nodo* q;
    while (L) {
        q = L->sig;
        delete L;
        L = q;
    }
}
/*-----*/

```

```

/* Para algoritmos de grafos */  

/* -----*/  

// Concatenacion de listas (para recorridos)  

template<typename T>  

Lista<T>& Lista<T>::operator +=(const Lista<T>& l)  

{  

    for (Lista<T>::posicion i =  

        l.primer(); i != l.fin(); i = l.siguiente(i))  

        insertar(l.elemento(i), fin());  

    return *this;  

}  

template<typename T>  

Lista<T> operator +(const Lista<T>& l1, const Lista<T>& l2)  

{  

    return Lista<T>(l1) += l2;  

}  

#endif // LISTA_ENLA_H

```

Código del archivo tablero.hpp

```

#ifndef TABLERO_HPP_
#define TABLERO_HPP_
#include "listaenla.hpp"
#include "hormigas.hpp"
#include "camino.hpp"
#include <vector>
#include <ctime>
#include <algorithm>
#include <limits>
#include <iostream>
#include <iomanip>
template <typename T>
class Tablero
{
public:
    **** ATRIBUTOS DE PUBLICOS ****
    *                                     ATRIBUTOS DE PUBLICOS
    ****
    float FEROMONAS[100][100];//Representacion de las feromonas.
    //Puesto que no vamos a tener un mapa con mas de 100 vertices
    //asigno 100 para que cuando reasigne memoria la matriz este alineada.
    const float APRENDIZAJE = 1.;
    const float COSTE_EVAPORACION = 0.01;
    /*Actualizacion de feromonas
    PRECONDICION: las feromonas han sido inicializadas
    POSTCONDICION: actualiza las feromonas utilizando
    la formula de actualizacion de feromonas

```

```

*/
void actualizar_feromonas();
void evaporar_feromonas();
*****
*                                     DECLARACION DE SUBTIPOS
*****
typedef T tCoste;
//Tipo de coste de las aristas
typedef typename std::size_t vertice;
//Vertices del grafo(nodos)
static const tCoste INFINITO;
//Coste infinito entre dos nodos(no hay camino directo entre ellos)
/*ESTRUCTURA DE CONEXION ENTRE DOS NODOS*/
struct conexion{
    vertice i, j; //Vertices que forman la arista
    tCoste coste; //El coste entre los dos vertices
    /*OPERADOR == para comprobar si dos aristas son iguales.
     PRECONDICION: -
     POSTCONDICION: devuelve true si el parametro formal es
     la misma arista que la de la clase miembro.*/
    bool operator == (const conexion& c) const{return i == c.i && j == c.j; }
};

*****
*                                     OPERACIONES DEL TAD TABLERO
*****
/*CONSTRUCTOR sobrecargado para crear el tablero
 PRECONDICION: el parametro formal N debe ser > 0
 POSTCONDICION: crea un tablero nuevo, asigna el tamaño a las
 feromonas e inicializa la lista de hormigas.
*/
explicit Tablero(std::size_t N, vertice inicial, vertice objetivo);
/*OBSERVADOR para saber el numero de vertices del grafo o tablero
 PRECONDICION: -
 POSTCONDICION: devuelve el numero de nodos
*/
size_t numeroNodos() const{ return adyacentes.size(); }
/*OPERADOR [] OBSERVADOR y MODIFICADOR
 PRECONDICION: -
 POSTCONDICION: devuelve el elemento en dicha posicion
*/
const std::vector<tCoste>& operator[](vertice v)const{
    return adyacentes[v];
}
std::vector<tCoste>& operator[](vertice v){return adyacentes[v]; }
/*OBSERVADOR para saber cuantas hormigas hay en el tablero
 PRECONDICION: -
 POSTCONDICION: devuelve el numero de hormigas que recorren el tablero
*/

```

```

size_t numeroHormigas() const {return hormigas.size();}

/*Metodo para hacer avanzar una hormiga de un nodo al siguiente utilizando la formula de la probabilidad sobre la cual se elige uno arbitrariamente y condicionado segun su probabilidad.

PRECONDICIoN: el nodo pasado por parametro tiene al menos un adyacente.

El nodo no es el nodo objetivo. La hormiga h esta recorriendo el tablero.

POSTCONDICIoN: devuelve el siguiente nodo por el que pasara la hormiga

*/

vertice siguienteNodo(vertice x, Hormiga<tCoste>* h);

/*Metodo para saber la distancia de un punto x a otro y, se utiliza la formula de la distancia

PRECONDICIoN: ambos parametros son adyacentes entre si, puesto que el grafo es no dirigido, con el que el primer parametro sea adyacente al segundo nos vale

POSTCONDICION: devuelve la distancia entre dos puntos

*/

float distancia(vertice x, vertice y);

/*Metodo para saber si una hormiga especifica ha salido del hormiguero en busca de comida

PRECONDICIoN: -

POSTCONDICIoN: devuelve verdadero si la hormiga existe en el tablero o falso en caso contrario

bool hormigaBuscaComida(const Hormiga<tCoste>* h) const{

    return hormigas.buscar(h) != hormigas.fin();

}

/*Metodo para mandar a una nueva hormiga a buscar comida

PRECONDICIoN: -

POSTCONDICIoN: aade una hormiga nueva al Tablero y marca como visitado el nodo de inicio

*/

void nuevaHormiga(Hormiga<tCoste>* h) {

    hormigas.insertar(h, hormigas.fin());
    h->verticesVisitados().nuevoNodo(inicial_, 0.0);

}

/*Metodo para saber si hemos alcanzado el objetivo

PRECONDICIoN: -

POSTCONDICIoN: devuelve true si hemos alcanzado el objetivo, false en caso contrario

*/

bool esObjetivo(vertice i) const{

    return i == objetivo_;

}

/*Metodo observador del nodo inicial

PRECONDICIoN: -

POSTCONDICIoN: devuelve el vertice inicial

*/

vertice inicial(){

}

```

```

        return inicial_;
    }

/*Metodo para saber cual es el mejor camino en un instante
de tiempo t

PRECONDICiON: -
POSTCONDICiON: devuelve un camino recorrido hasta el instante
de tiempo en que se llama a la funcion y su coste

*/

Camino<tCoste> mejorCaminoActual();
*****OPERADOR DE INSERCIoN EN FLUJO*****
template <class U> friend std::ostream& operator <<
    (std::ostream& os, const Tablero<U>& tab);

private:
***** DECLARACION DE TIPOS PRIVADOS *****
*****
*****
std::vector< std::vector<tCoste> >adyacentes;
Lista<const Hormiga<tCoste> *> hormigas;
vertice inicial_;
vertice objetivo_;

};

#endif
*****INFINITO*****
template <typename T> const T Tablero<T>::INFINITO =
    std::numeric_limits<T>::max();

*****CONSTRUCTOR*****
template <typename T>
Tablero<T>::Tablero(std::size_t N, vertice inicial, vertice objetivo)
    :adyacentes(N, std::vector<tCoste>(N,INFINITO)),
     hormigas(),
     inicial_(inicial),
     objetivo_(objetivo)
{
    // FEROMONAS[N][N];
    //Inicializamos las feromonas con una Tasa de Feromonas del 1% (aleatoria)
    for(size_t i = 0; i < N; i++)
        for(size_t j = 0; j < N; j++)
            FEROMONAS[i][j] = 0.1;
}

*****IMPLEMENTACION DEL RESTO DE METODOS DECLARADOS*****
template <typename T>
typename Tablero<T>::vertice Tablero<T>::siguienteNodo(vertice x, Hormiga<tCoste>* h)
{
    size_t n = adyacentes[x].size();
    //Adyacentes de x
    assert(n != 0 && x != objetivo_ && hormigas.buscar(h) != hormigas.fin());
    vertice siguiente = x;
    //El siguiente nodo por el que pasara, valor de retorno de la funcion
}

```

```

float Stn = 0;
//Sumatorio de feromonas y distancias
float caminosIndividuales[n];
//Numerador de la formula 1, producto de la feromona y la distancia desde
//el vertice x a uno de sus adyacentes, almacenados secuencialmente en un
//vector para calcular sobre ellos la probabilidad
std::srand(unsigned(time(0))); //semilla aleatoria para generar un numero
for(vertice i = adyacentes[x].front(); i < adyacentes[x].size(); i++){
    if(adyacentes[x][i] != INFINITO && !(h->haVisitado(i))){
        caminosIndividuales[i] = (1/adyacentes[x][i])*FEROMONAS[x][i];
        Stn += caminosIndividuales[i];
    }
    std::vector<vertice> anterior;
    //Conjunto de vertices adyacentes que no fueron INFINITO
    int ultimoValorAlmacenado = 0;
    //Para posteriormente se termine este bloque saber hasta que numero
    //aleatorio generar a la hora de elegir camino
/*Definir rangos con su porcentaje,
por ejemplo: 0----16.987%----56.654%----100% correspondientes a sus vertices*/
for(vertice i = adyacentes[x].front(); i < adyacentes[x].size(); i++){
    if(adyacentes[x][i] != INFINITO && !(h->haVisitado(i))){
        caminosIndividuales[i] = (caminosIndividuales[i] / Stn) * 100;
        if(!anterior.empty()){
            for(size_t j = 0; j < anterior.size(); j++)
                caminosIndividuales[i] += caminosIndividuales[anterior[j]];
            anterior.push_back(i);
            ultimoValorAlmacenado = caminosIndividuales[i];
        }
    }
    //CALLEJO SIN SALIDA: el nodo en el que estamos tiene todos
    //sus adyacentes ya marcados y no es la solucion
    if(anterior.size() == 0){
        //Cogemos el vertice anterior por el que hemos pasado
        vertice antesDelCallejonSinSalida =
            h->verticesVisitados().
                caminoRecorrido().elemento(
                    h->verticesVisitados()
                        .caminoRecorrido()
                        .anterior(
                            h->verticesVisitados()
                                .caminoRecorrido()
                                .buscar(h->actual())
                        )
                )
    };
    //Se agrega el nodo anterior de nuevo ya que hay que saber el
    //camino real que ha tomado la hormiga
    h->nuevoNodoVisitado(
        antesDelCallejonSinSalida,

```

```

        adyacentes[x] [antesDelCallejonSinSalida]
    );
    siguiente = siguienteNodo(antesDelCallejonSinSalida, h);
}
//Si hay solo un elemento no hay rangos y si utilizaramos el caso
//general saltaria una excepcion al intentar asignar el
//valor a la variable iBefore
if (anterior.size() == 1){
    siguiente = anterior.front();
    h->nuevoNodoVisitado(
        anterior.front(),
        adyacentes[x] [siguiente]
    );
} else if (anterior.size() > 1){
    /*Generamos numero aleatorio y sacamos, segun el rango,
    el vertice al que corresponde el numero aleatorio generado*/
    float numeroAzar = rand() % (0 + ultimoValorAlmacenado);
    //Para coger el ultimo elemento y su elemento anterior(para elegir un rango)
    vertice iAux, iBeforeAux;
    for(vertice i = anterior.back(); i != anterior.front(); i--) {
        iAux = i;
        anterior.pop_back();
        iBeforeAux = anterior.back();
        if(adyacentes[x] [iAux] != INFINITO) {
            if(numeroAzar >=
                caminosIndividuales[iBeforeAux] &&
                numeroAzar <= caminosIndividuales[iAux]) {
                siguiente = iAux;
                h->nuevoNodoVisitado(
                    iAux,
                    adyacentes[x] [siguiente]
                );
            }
        }
    }
    return siguiente;
}
template <typename T>
float Tablero<T>::distancia(vertice x, vertice y) {
    assert(adyacentes[x] [y] == INFINITO);
    return 0;
}
template <typename T>
void Tablero<T>::actualizar_feromonas() {
    //float sigmaXY = 0.0;
    Lista<
        typename Hormiga

```

```

        <typename Tablero<T>::tCoste>
        ::vertice> camino;
        //Lista de caminos recorridos para cada hormiga
        vertice x, y;
        //Variables auxiliares para saber que feromonas actualizar
        evaporar_feromonas();
        for(typename Lista
            <const Hormiga<T> *>::posicion itH = hormigas.primera();
            itH != hormigas.fin(); itH = hormigas.siguiente(itH)
        ) {
            camino = hormigas[itH]-
                verticesVisitados()
                .caminoRecorrido();
            for(Lista<vertice>::posicion itV = camino.primera();
                camino.siguiente(itV) != camino.fin();
                itV = camino.siguiente(itV)){
                x = camino[itV];
                y = camino[camino.siguiente(itV)];
                //valores iniciales de las feromonas del camino(la suma de
                //la inversa del coste del camino de cada hormiga en un tiempo t)
                FEROMONAS[x][y] += (1/hormigas[itH]-
                    verticesVisitados()
                    .costeCamino())
                );
                FEROMONAS[y][x] = FEROMONAS[x][y];
            }
        }
    }

    template <typename T>
    void Tablero<T>::evaporar_feromonas () {
        for(size_t i = 0; i < adyacentes.size(); i++)
            for(size_t j = 0; j < adyacentes.size(); j++) {
                FEROMONAS[i][j] *= (1 - COSTE_EVAPORACION);
                FEROMONAS[j][i] = FEROMONAS[i][j];
            }
    }

    template <typename T>
    Camino<T> Tablero<T>::mejorCaminoActual() {
        Camino<T> mejorCamino;
        vertice maxVerticeEnTiempot;
        //El vertice adyacente candidato a ser el siguiente mejor
        vertice VerticeEnTiempot;
        //El vertice actual que a adiremos al camino
        T costeEnTiempot;
        //coste actual de la feromona que estamos viendo
        T costeMax;
        //para coger el mayor coste y posteriormente guardar el
        //vertice anteriormente declarado
    }
}

```

```

T c;
//coste a asignar cuando se elija un vertice
//A adimos el vertice inicial a la lista del mejor camino
mejorCamino.nuevoNodo(inicial_, 0);
bool visitado[adyacentes.size()];
for(size_t i = 0; i < adyacentes.size(); i++)
    visitado[i] = false;
//Puesto que el mejor vertice inicialmente es el inicial, es
//al que inicializamos la variable maxVerticeEnTiempot
VerticeEnTiempot = inicial_;
visitado[inicial_] = true;
while(VerticeEnTiempot != objetivo_){
    costeMax = -1;
    for(Tablero<T>::vertice v = 0; v < adyacentes.size(); v++){
        if(adyacentes[VerticeEnTiempot][v] != INFINITO &&
           !visitado[v]){
            costeEnTiempot = FEROMONAS[VerticeEnTiempot][v];
            if(costeMax < costeEnTiempot){
                costeMax = costeEnTiempot;
                maxVerticeEnTiempot = v;
            }
        }
    }
    c = adyacentes[VerticeEnTiempot][maxVerticeEnTiempot];
    VerticeEnTiempot = maxVerticeEnTiempot;
    mejorCamino.nuevoNodo(
        VerticeEnTiempot,
        c
    );
    visitado[VerticeEnTiempot] = true;
}
return mejorCamino;
}

/*********************OPERADOR DE INSERCION EN FLUJO*****************/
template <typename T>
std::ostream& operator << (std::ostream& os, const Tablero<T>& tab)
{
    size_t n = tab.adyacentes.size(); //numero de vertices
    /*MOSTRAR EL GRAFO*/
    os << "*****GRAFO*****\n";
    for(size_t i = 0; i < n; i++) {
        if (i == 0) os << std::setfill(' ') << std::setw(15);
        os << std::setfill(' ') << std::setw(8) << i;
    }
    os << std::endl;
    for(size_t i = 0; i < n; i++) {
        for(size_t j = 0; j < n; j++) {
            if(j == 0) os << i;

```

```

    if (tab[i][j] == Tablero<T>::INFINITO)
        os << std::setfill(' ') << std::setw(8) << "INFI";
    else
        os << std::setfill(' ') << std::setw(8) <<
            std::fixed << std::setprecision(2) << tab[i][j];
    }
    os << "\n";
}
/*MOSTRAR FEROMONAS*/
os << "\n*****FEROMONAS*****\n";
for(size_t i = 0; i < n; i++) {
    if (i == 0) os << std::setfill(' ') << std::setw(15);
    os << std::setfill(' ') << std::setw(9) << i;
}
os << std::endl;
for(size_t i = 0; i < n; i++) {
    for(size_t j = 0; j < n; j++) {
        if(j == 0) os << i;
        os << std::setfill(' ') << std::setw(9) << std::fixed <<
            std::setprecision(3) << tab.FEROMONAS[i][j];
    }
    os << "\n";
}
/*MOSTRAR HORMIGAS*/
//os << "\n*****HORMIGAS*****\n";
/*for(typename Lista
    <const Hormiga<T> *>::posicion it = tab.hormigas.primera();
    it != tab.hormigas.fin(); it = tab.hormigas.siguiente(it))
    os << *(tab.hormigas[it]);*/
return os;
}

```

B.1.8.2. Código principal en C++

En el siguiente Apartado mostraremos el código principal que se utiliza en C++.

```

#include "tablero.hpp"
#include "hormigas.hpp"
#include <iostream>
using namespace std;
int main(){
    size_t n = 6;
    typedef typename Tablero<float>::tCoste tCoste;
    typedef typename Tablero<tCoste>::vertice vertice;
    Hormiga<tCoste> h1(1),h2(2),h3(3),h4(4)

```

```

        ,h5(5),h6(6),h7(7),h8(8);
vertice inicial = 0;
vertice objetivo = 3;
Tablero<tCoste> tab(n, inicial, objetivo);
vertice actual = tab.inicial();
tab[0][1] = tab[1][0] = 4;
tab[0][2] = tab[2][0] = 1;
tab[0][4] = tab[4][0] = 3;
tab[1][5] = tab[5][1] = 4;
tab[1][2] = tab[2][1] = 2;
tab[2][5] = tab[5][2] = 6;
tab[2][4] = tab[4][2] = 1;
tab[3][4] = tab[4][3] = 1;
tab[3][5] = tab[5][3] = 9;
tab.nuevaHormiga(&h1);
tab.nuevaHormiga(&h2);
tab.nuevaHormiga(&h3);
tab.nuevaHormiga(&h4);
tab.nuevaHormiga(&h5);
tab.nuevaHormiga(&h6);
/*tab.nuevaHormiga(&h7);
tab.nuevaHormiga(&h8);*/
tab.hormigaBuscaComida(&h3);
tab.hormigaBuscaComida(&h1);
for(size_t i = 0; i < 10000; i++){
    while(!tab.esObjetivo(actual))
        actual = tab.siguienteNodo(actual, &h1);
    actual = tab.inicial();
    while(!tab.esObjetivo(actual))
        actual = tab.siguienteNodo(actual, &h2);
    actual = tab.inicial();
    while(!tab.esObjetivo(actual))
        actual = tab.siguienteNodo(actual, &h3);
    actual = tab.inicial();
    while(!tab.esObjetivo(actual))
        actual = tab.siguienteNodo(actual, &h4);
    actual = tab.inicial();
    while(!tab.esObjetivo(actual))
        actual = tab.siguienteNodo(actual, &h5);
    actual = tab.inicial();
    while(!tab.esObjetivo(actual))
        actual = tab.siguienteNodo(actual, &h6);
/*actual = tab.inicial();
while(!tab.esObjetivo(actual))
    actual = tab.siguienteNodo(actual, &h7);
actual = tab.inicial();
while(!tab.esObjetivo(actual))
    actual = tab.siguienteNodo(actual, &h8);*/

```

```

        tab.actualizar_feromonas();
        h1.volver();
        h2.volver();
        h3.volver();
        h4.volver();
        h5.volver();
        h6.volver();
        /*h7.volver();
        h8.volver();*/
        /*h1.volver();
        tab.actualizar_feromonas();*/
        //cout << tab << endl;
    }
    cout << tab << endl;
    cout << tab.mejorCaminoActual() << endl;
}

```

B.1.8.3. Makefile

En el siguiente Apartado mostraremos el código del archivo de compilación del algoritmo en C++.

```

CXX = clang++
CXXFLAGS = -g -Wall -std=c++11 -pedantic
BASURA = test-tablero *.o

all: clean test-tablero

test-tablero: pt.o
    $(CXX) $(LDFLAGS) -o $@ $^
test-tablero.o: pt.cpp

clean:
    $(RM) $(BASURA)

```

Bibliografía

- [1] Algoritmo de la colonia de hormigas - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Algoritmo_de_la_colonia_de_hormigas.
Accedido: 15-01-2021.
- [2] Inteligencia de enjambre - Wikipedia, la enciclopedia libre. URL
https://es.wikipedia.org/wiki/Inteligencia_de_enjambre#Optimizacion_de_colonia_de_hormigas.
Accedido: 15-01-2021.
- [3] Pierre-Paul Grassé - Wikipedia, la enciclopedia libre. URL
https://es.wikipedia.org/wiki/Pierre-Paul_Grassé.
Accedido: 15-01-2021.
- [4] Hormigas: los genios de la comunicación química. URL
<https://www.xatakaciencia.com/biologia/hormigas-los-genios-de-la-comunicacion-quimica>.
Accedido: 18-01-2021.
- [5] Computación evolutiva - Wikipedia, la enciclopedia libre, . URL
https://es.wikipedia.org/wiki/Computacion_evolutiva.
Accedido: 18-01-2021.
- [6] Algoritmo colonia de abejas artificiales - Wikipedia, la enciclopedia libre. URL
https://es.wikipedia.org/wiki/Algoritmo_{_}colonia{_}de{_}abejas{_}artificiales.
Accedido: 18-01-2021.

- [7] Jose B. Escario, Juan F. Jimenez, and Jose M. Giron-Sierra. Optimisation of autonomous ship manoeuvres applying ant colony optimisation metaheuristic. *Expert Systems with Applications*, 39:10120–10139, 2012. ISSN 09574174.
- [8] Adel T. Abbas, Mohamed F. Aly, and Karim Hamza. Optimum drilling path planning for a rectangular matrix of holes using ant colony optimisation. *International Journal of Production Research*, 49:5877–5891, 2011. ISSN 00207543.
- [9] Stigmergy: El algoritmo de colonia de hormigas (aco) - stigmergy. URL
<http://www.stigmergy.es/stigmergy-el-algoritmo-de-colonia-de-hormigas-aco/>.
Accedido: 18-01-2021.
- [10] Diagrama de operaciones. — documentación empleada en programación de la producción. URL
https://ikastaroak.ulhi.net/edu/es/PPFM/PP/PP05/es_PPFM_PP05_Contenidos/website_211_diagrama_de_operaciones.html.
Accedido: 20-01-2021.
- [11] github juan manuel gómez hutchison. URL
<https://github.com/juanmaGHutchison/AlgoritmoHormigaTFG/tree/master/C%2B%2BALgorithm>.
Accedido: 20-01-2021.
- [12] Github - google/googletest: Googletest - google testing and mocking framework. URL
<https://github.com/google/googletest>.
Accedido: 21-01-2021.
- [13] Tdd como metodología de diseño de software - paradigma. URL
<https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/>.
Accedido: 21-01-2021.
- [14] Batería de polímero de litio - wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Bater%C3%ADa_de_pol%C3%A9mico_de_litio.
Accedido: 22-01-2021.
- [15] Puente h (electrónica) - wikipedia, la enciclopedia libre. URL
[https://es.wikipedia.org/wiki/Puente_H_\(electr%C3%B3nica\)](https://es.wikipedia.org/wiki/Puente_H_(electr%C3%B3nica)).

Accedido: 22-01-2021.

- [16] Robot seguidor de línea - wikipedia, la enciclopedia libre. URL
https://es.wikipedia.org/wiki/Robot_seguidor_de_l%C3%A1nea.
Accedido: 25-01-2021.
- [17] Teach, learn, and make with raspberry pi raspberry pi, . URL
<https://www.raspberrypi.org/>.
Accedido: 25-01-2021.
- [18] Arduino - home, . URL <https://www.arduino.cc/>.
Accedido: 25-01-2021.
- [19] Gpio - wikipedia, la enciclopedia libre. URL
<https://es.wikipedia.org/wiki/GPIO>.
Accedido: 25-01-2021.
- [20] (raspberry pi zero). URL <https://cdn-learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-zero.pdf>.
Accedido: 25-01-2021.
- [21] Arduino - home. URL <https://www.arduino.cc/>.
Accedido: 25-01-2021.
- [22] (arduino nano datasheet), . URL
<http://www.farnell.com/datasheets/1682238.pdf>.
Accedido: 25-01-2021.
- [23] Esta es la energía que consume cada modelo de raspberry pi. URL <https://www.redeszone.net/2017/03/02/energia-consumida-raspberry-pi/>.
Accedido: 25-01-2021.
- [24] Limites de voltaje, corriente y alimentación del arduino electrónica especializada, . URL
<https://www.alarduino.com.mx/blog/limites-de-voltaje-corriente-y-alimentacion-del-arduino/>.
Accedido: 25-01-2021.
- [25] Arduino nano pin diagram, features, pin uses programming. URL
<https://components101.com/microcontrollers/arduino-nano>.
Accedido: 25-01-2021.

- [26] Raspberry pi placa zero wh 512 mb: Amazon.es: Electrónica, .
Accedido: 26-01-2021.
- [27] Sandisk ultra tarjeta de memoria microsdhc con adaptador sd, hasta 98 mb/s, rendimiento de apps a1, clase 10, u1, 32 gb: Amazon.es: Informática.
Accedido: 26-01-2021.
- [28] Longrunner para arduinoide atmega328p 5v 16m micro controlador junta módulo ky64-5: Amazon.es: Electrónica, .
Accedido: 26-01-2021.
- [29] Wifi - wikipedia, la enciclopedia libre. URL
<https://es.wikipedia.org/wiki/Wifi>.
Accedido: 26-01-2021.
- [30] Bluetooth - wikipedia, la enciclopedia libre. URL
<https://es.wikipedia.org/wiki/Bluetooth>.
Accedido: 26-01-2021.
- [31] Zigbee - wikipedia, la enciclopedia libre. URL
<https://es.wikipedia.org/wiki/Zigbee>.
Accedido: 26-01-2021.
- [32] Esp8266-01 + arduino y thingspeak.com, . URL <https://arduproject.es/estacion-meteorologica-wifi-arduino-esp8266-01-thingspeak-esp8266-01/>.
Accedido: 26-01-2021.
- [33] Lista-de comandos at esp8266 esp8266ex módulo wifi — hetpro, . URL
<https://hetpro-store.com/comandos-at-esp8266-esp8266ex/>.
Accedido: 26-01-2021.
- [34] Azdelivery esp8266 esp01 esp-01s wlan wifi modulo sensor transceptor microcontrolador compatible con arduino y raspberry pi con e-book incluido!: Amazon.es: Industria, empresas y ciencia, .
Accedido: 26-01-2021.
- [35] Dsd tech hc-06 mÓdulo de soporte de transceptor serial inalámbrico bluetooth modo esclavo y maestro para arduino + 4pin dupont cable: Amazon.es: Informática.
Accedido: 26-01-2021.

- [36] Amazon.es: Opciones de compra: Cloud home xbee 2mw pcb antena serie 2 (zigbee mesh).
Accedido: 26-01-2021.
- [37] Modulo bluetooth hc-05 4 pin wireless bluetooth uno r3: Amazon.es: Electrónica.
Accedido: 27-01-2021.
- [38] Hrb 7.4v 2200mah 30c ec2 lipo batería para juguetes de avión helicóptero rc car model heli plano carro del barco fpv coches (7.4v 2200mah ec2): Amazon.es: Juguetes y juegos.
Accedido: 27-01-2021.
- [39] Motordc pdf, motordc description, motordc datasheets, motordc view :::: Alldatasheet :::: . URL <https://pdf1.alldatasheet.com/datasheet-pdf/view/96347/ETC/MOTORDC.html>.
Accedido: 29-01-2021.
- [40] L298n pdf, l298n description, l298n datasheets, l298n view :::: Alldatasheet :::: . URL <https://pdf1.alldatasheet.com/datasheet-pdf/view/22440/STMICROELECTRONICS/L298N.html>.
Accedido: 29-01-2021.
- [41] Cny70 pdf, cny70 description, cny70 datasheets, cny70 view :::: Alldatasheet :::: . URL <https://pdf1.alldatasheet.com/datasheet-pdf/view/26332/VISHAY/CNY70.html>.
Accedido: 29-01-2021.
- [42] Tcs3200 pdf, tcs3200 description, tcs3200 datasheets, tcs3200 view :::: Alldatasheet :::: . URL <https://pdf1.alldatasheet.com/datasheet-pdf/view/560507/AMSCO/TCS3200.html>.
Accedido: 29-01-2021.
- [43] Arduino nano — arduino official store, . URL <https://store.arduino.cc/arduino-nano>.
Accedido: 29-01-2021.
- [44] Hc-05 bluetooth module pinout, specifications, default settings, replacements datasheet, . URL <https://components101.com/wireless/hc-05-bluetooth-module>.
Accedido: 29-01-2021.

- [45] Circuito impreso - wikipedia, la enciclopedia libre. URL
https://es.wikipedia.org/wiki/Circuito_impresso.
Accedido: 01-02-2021.
- [46] John C. Shedd, Mayo D. Hershey, and John C. Shedd. The history of ohm's law. *Popular Science Monthly*, 83:599–614, 1913. URL
https://en.wikisource.org/wiki/Popular_Science_Monthly/Volume_83/December_1913/The_History_of_Ohm%27s_Law.
- [47] Pololu - zumo chassis kit (no motors). URL
<https://www.pololu.com/product/1418>.
Accedido: 01-02-2021.
- [48] Pololu Corporation. Pololu Zumo Chassis User's Guide. Technical report, Pololu Corporation, 2001.
- [49] Inkscape. Draw Freely — Inkscape. URL <https://inkscape.org/es/>.
Accedido: 22-02-2021.
- [50] Qbluetooth namespace — qt bluetooth 5.15.3. URL
<https://doc.qt.io/qt-5/qbluetooth.html>.
Accedido: 25-02-2021.
- [51] Conectar un módulo Bluetooth HC-05 al Arduino. Parte I — El blog del Tecñófilo. URL
<https://www.tecnofilo.es/blog/conectar-un-modulo-bluetooth-hc-05-al-arduino-parte-i/>.
Accedido: 23-01-2021.