



**ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO**

**FACULTAD DE INFORMÁTICA Y ELECTRONICA**

**ESCUELA DE INGENIERÍA EN SISTEMAS**

**ANÁLISIS DE LOS ALGORITMOS HEURÍSTICOS APLICADO EN EL  
APRENDIZAJE Y GUÍA DE SISTEMAS AUTÓNOMOS.**

**TESIS DE GRADO**

**PREVIA A LA OBTENCIÓN DEL TÍTULO DE**

**INGENIERO EN SISTEMAS INFORMÁTICOS**

**Presentado por:**

**MARCO WILSON PINEDA GUARACA**

**RIOBAMBA-ECUADOR**

2014

## **AGRADECIMIENTO**

La vida estudiantil es una incesante lucha por alcanzar metas y cumplir objetivos, los mismos que luego de mucho tiempo los he logrado alcanzar poco a poco gracias al incondicional apoyo de mis padres, por eso mi agradecimiento va dirigido principalmente a mi mamá Roció del Carmen Guaraca Criollo y a mi papá Marco Wilson Pineda Terán, sin ellos nunca hubiera podido haber dado este primer paso hacia futuras metas, un agradecimiento muy especial a mi director de tesis Doctor Alonso Washington Alvares Olivo por el apoyo brindado durante la realización de la tesis también agradezco a todos mis amigos principalmente a Álvaro Roberto Santillán y Paul David Cumba quienes siempre estuvieron preocupados y pendientes de la realización de este trabajo final y por ultimo agradezco a mi hermana Martha Estefanía Pineda y su esposo Jorge Patricio Murquincho quienes siempre tuvieron sus manos abiertas para ayudarme en lo que necesite.

*Marco Wilson Pineda*

## **DEDICATORIA**

Dedico este trabajo a mi Mamita **Roció del Carmen Guaraca Criollo** que ha sido el pilar fundamental en todo el desarrollo de mi vida, que siempre supo extenderme su abrigo cuando más lo necesite, que gracias a su apoyo, comprensión y constancia me hicieron poder ser una persona de bien para la sociedad.

*Marco Wilson Pineda*

**FIRMAS RESPONSABILIDAD Y NOTA**

<b>NOMBRE</b>	<b>FIRMA</b>	<b>FECHA</b>
Ing. Iván Menes Camejo <b>DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRONICA</b>	.....	.....
Ing. Jorge Huilca Palacios. <b>DIRECTOR DE LA ESCUELA DE INGENIERIA EN SISTEMAS</b>	.....	.....
Dr. Alonso Alvarez. <b>DIRECTOR DE TESIS</b>	.....	.....
Ing. Paul Romero <b>MIEMBRO DEL TRIBUNAL</b>	.....	.....
Tlgo. Carlos Rodríguez <b>DIRECTOR DEL CENTRO DE DOCUMENTACIÓN</b>	.....	.....

## **RESPONSABILIDAD DEL AUTOR**

“Yo Marco Wilson Pineda, soy el responsable de las ideas, doctrinas y resultados expuestos en esta Tesis de Grado, y el patrimonio intelectual de la misma pertenece a la Escuela Superior Politécnica de Chimborazo.”

**FIRMA:**

---

Marco Wilson Pineda Guaraca

## ÍNDICE DE ABREVIATURAS

### ABREVIATURAS

#### A

**AI:** Artificial Intelligent (Inteligencia Artificial)

**AOP:** Aspect-Oriented Programming (Programación Orientada al Aspecto)

**ACID:** Atomicity (atomicidad), Consistency (coherencia), Isolation (aislamiento), Durability (permanencia).

**API:** Application Programming Interface (Interfaz de Programación de Aplicaciones).

#### B

**BD:** Base de Datos.

**BFS:** Breadth first search (Búsqueda por anchura).

#### C

**CNE:** Camino no encontrado.

**CGI:** *Common Gateway Interface (Interfaz de Pasarela Común)*

**CMP:** Container Managed Persistence (Persistencia Gestionada por el Contenedor)

#### D

**DAO:** Data Access Object (Objeto de Acceso a Datos)

**DDL:** Data Definition Language (Lenguaje de definición de Datos).

**DFS:** Depth first search (Búsqueda por profundidad)

**DML:** Data Manipulation Language (Lenguaje de Manipulación de Datos).

#### E

**EJB:** Enterprise Java Bean

#### G

**GNU:** GNU no es Unix.

#### H

**HQL:** Hibernate Query Lenguaje (Lenguaje de Consultas de Hibernate).

**HTML:** Hyper Text Markup Language (Lenguaje de Marcado Hipertextual)

**HTTP:** Hyper Text Transfer Protocol (Protocolo de Transferencia De Hipertexto)

**HR:** Hand Right (Mano derecha)

#### I

**IDE:** Integrated Development Environment (Entorno Integrado de Desarrollo)

**J**

**JMX:** Java Management Extensions (Administración de Extensiones Java)

**JNDI:** Java Naming and Directory Interface (Interfaz de Nombres y Directorios Java)

**JSF:** Java Server Faces

**O**

**OOP:** Object Oriented Programming (Programación Orientada a Objetos)

**ORM:** Object Relational Mapping.

**P**

**PAO:** Programación Orientada a Aspectos

**POO:** Programación Orientada a Objetos

**R**

**RUP:** Rational Unified Process (Proceso Unificado de Racional)

**S**

**SQL:** Structured Query Language (Lenguaje de Consulta Estructurados)

**T**

**TCP/IP:** *Transmission Control Protocol/Internet Protocol* (Protocolo de Control de Transmisión/Protocolo de Internet)

**U**

**URL:** Uniform Resource Locator (Localizador de Recursos Uniforme)

**W**

**WWW:** World Wide Web, Sistema de Documentos de Hipertexto

**X**

**XML:** Extensible Markup Language (Lenguaje de Marcas Extensibles)

## **INDICE**

PORTADA

AGRADECIMIENTO

DEDICATORIA

FIRMAS RESPONSABILIDAD Y NOTA

RESPONSABILIDAD DEL AUTOR

### **CAPÍTULO I**

MARCO REFERENCIAL .....	21
1.1 Antecedentes .....	21
1.2. Justificación del proyecto de tesis.....	23
1.3. Objetivos.....	24
1.3.1 Objetivo General: .....	24
1.3.1 Objetivos específicos: .....	24
1.5. Métodos y Técnicas.....	25
1.5.1. Métodos.....	25
1.5.2. Técnicas.....	25

### **CAPÍTULO II**

MARCO TEORICO .....	27
2.1. Definiciones.....	27
2.1.1. Inteligencia Artificial. ....	27
2.1.1.1. Categorías de inteligencia artificial.....	27
2.1.2. Sistemas inteligentes autónomos.....	28
2.1.2.1. Ontología.....	29
2.1.3. Heurística. ....	29
2.1.3.1. Procedimientos heurísticos.....	30
2.1.3.2. Estrategias heurísticas. ....	31
2.1.3.2.1. Trabajo hacia delante.....	31
2.1.3.2.2. Trabajo hacia atrás.....	32
2.1.3.3. Gnoseología de la heurística .....	33
2.1.4. Laberintos.....	34
2.1.4.1. Historia.....	35



2.1.4.2. Camino más corto en un laberinto usando el algoritmo de la mano derecha o izquierda.....	38
2.1.4.3. Algoritmo Backtracking.....	38
2.1.4.4. Aplicaciones del método Backtracking.....	39
2.1.4.5. Esquema General .....	39
2.1.4.6. Ventajas y desventajas .....	42
2.1.4.7. Árbol de búsqueda.....	43
2.1.5. Mecanismos autónomos .....	44
2.1.5.1. Características .....	45
2.1.5.2. Paradigmas en la robótica .....	45
2.1.5.2.1.Paradigma jerárquico.....	46
2.1.5.2.2.Paradigma reactivo .....	48
2.1.5.2.3.Paradigma híbrido deliberativo/reactivo .....	49
2.1.5.3. Importancia del aprendizaje en mecanismos autónomos .....	50
2.1.5.4. Problemas de la falta de adaptabilidad [10] .....	50
2.1.5.5. Aproximaciones simbólicas vs subsimbólicas. ....	51
2.1.5.6. Aprendizaje por refuerzo.....	53
2.1.5.6.1.Agente .....	54
2.1.5.6.2.Entorno .....	54
2.1.5.6.3.Recompensa.....	55
2.1.5.7. Interfaz Agente – Entorno.....	56
2.1.6. GINKO. Una Arquitectura de Aprendizaje y planificación en mecanismos autónomos .....	57
2.1.6.1. Aprendizaje .....	58
2.1.6.2. Planificación.....	58
2.1.6.3. Ejecución y monitoreo .....	59
2.1.6.4. Control.....	59
2.1.7. Mecanismo Lego Mindstorms 2.0.....	60
2.1.7.1. Historia.....	60
2.1.7.1.1.Colaboración con el grupo de epistemología y aprendizaje del MIT.....	61
2.1.7.1.2.Antecedentes del bloque programable .....	62

2.1.7.2. Microcontrolador.....	63
2.1.7.3. Entradas y salidas .....	63
2.1.7.4. Comunicaciones .....	63
2.1.7.5. Firmware .....	64
2.1.7.6. Motores .....	64
2.1.7.7. Sensores.....	66
2.1.7.7.1.Sensor ultrasónico .....	66
2.1.7.7.2.Sensor Detector de color .....	67
2.1.7.8. Programación .....	67
2.1.7.9. Lenguajes de programación .....	68
2.1.8. Lenguaje de programación C# (Visual Studio.net).....	68
2.1.8.1. Introducción .....	68
2.1.8.2. Historia.....	69
2.1.8.3. Tipos básico de datos .....	69
2.1.8.4. Metas del diseño del lenguaje [14].....	71
2.1.9. El Modelo de Gestión SCRUM.....	71
2.1.9.1. Origen.....	71
2.1.9.2. Introducción al modelo .....	72

### **CAPÍTULO III**

ANÁLISIS DE LOS ALGORITMOS HEURÍSTICOS APLICADO AL APRENDIZAJE Y GUÍA DE SISTEMAS AUTÓNOMOS.....	76
3.1.Análisis heurístico .....	76
3.2.Introducción.....	76
3.3Determinación de parámetros que conlleve al análisis heurístico para la guía de un mecanismo autónomo.....	77
3.3.1. Descripción inicial. ....	77
3.3.2. Eficiencia.....	77
3.3.2.1. Numero de nodos visitados. ....	78
3.3.2.2. Camino valido (Numero que conforma el camino valido).....	78
3.3.2.3. Nodos resultado(Relación nodos visitados con nodos respuesta).....	79
3.3.3. Tiempo. ....	79

3.3.3.1. Tiempo individual .....	79
3.3.3.2. Tiempo eficiente.....	79
3.3.4. Aprendizaje (Inteligencia).....	80
3.3.5. Administración de dispositivos (Uso del CPU). .....	80
3.3.5.1. Valores máximos CPU.....	80
3.3.5.2. Valores promedios. ....	81
3.4.Representación de parámetros para el análisis de algoritmos heurísticos.....	81
3.5.Apuntes del experimento.....	81
3.5.1. Inicialización del experimento. ....	82
3.5.2. Generación y resolución de laberintos. ....	83
8. Análisis de datos resultados y tabulación.....	90
3.6.4. Representación de Resultados.....	97
3.6.4.1. Análisis Heurístico .....	97
3.6.4.1.1.Calidad.....	98
3.6.4.1.2.Parámetro 1: Eficiencia. ....	98
3.6.4.1.3.Parámetro 2: Tiempo. ....	99
3.6.4.2. Comportamiento real.....	99
3.6.4.2.1.Parámetro 3: Aprendizaje.....	99
3.6.4.3. Administración de recursos.....	100
3.6.4.3.1.Parámetro 4: Uso de la CPU: .....	100
3.6.4.4. Resultados finales.....	100
3.6.5. Interpretación. ....	101
3.6.6. Calificación .....	102
3.6.7Interpretación.....	104
3.6.8Análisis de Resultados.....	105
<b>CAPITULO IV</b>	
4.1Algoritmos de Planificación.....	106
4.1.1. Planificación de movimiento.....	108
4.2.Determinación de un método adecuado de planificación.....	115
4.2.1. Valoración de un modelo adecuado de planificación. ....	115
4.2.2. Interpretación .....	118

4.2.3. Análisis de resultados.....	119
4.3.Comprobación de Hipótesis. ....	120
4.3.1. Resultados de mejor algoritmo en la búsqueda y aprendizaje. ....	120
4.3.2. Resultado del análisis heurístico aplicado al aprendizaje y guía de mecanismos autónomos. ....	121

## **CAPÍTULO V**

5.1DESARROLLO DEL SISTEMA DIDÁCTICO “LABERINTO VIRTUAL”.....	122
5.2.Ingeniería de la Información .....	123
5.3.Definición del ámbito.....	123
5.3.1. Estudio de Factibilidad.....	124
5.3.1.1. Factibilidad Técnica.....	124
5.4.Desarrollo del sistema “Laberinto virtual”.....	127
7.Análisis del Sistema .....	135
7.1. Definir Casos de Uso esenciales en formato extendido.....	135
8.Implementación y Pruebas .....	148
8.1. Definición de estándares de Programación.....	148
8.1.1. Pruebas Unitarias .....	148
8.1.2. Pruebas de Módulos y del Sistema.....	148

CONCLUSIONES

RECOMENDACIONES

RESUMEN

BIBLIOGRAFIA

ANEXOS

## ÍNDICES DE ILUSTRACIONES

ILUSTRACIÓN I 1: ARQUITECTURA GINKO.....	24
ILUSTRACIÓN II 1: ESQUEMA HEURÍSTICA.....	30
ILUSTRACIÓN II 2: LABERINTO FORMADO DE PIEDRAS EN EL PISO DEL TEMPLO DE SAN QUINTÍN EN FRANCIA. LA ENTRADA ES POR ABAJO POR LA LÍNEA VERTICAL.....	37
ILUSTRACIÓN II 3: LABERINTO EN LA CATEDRAL DE CHARTRES, FRANCIA.....	37
ILUSTRACIÓN II 4: ÁRBOL DE BÚSQUEDA.....	44
ILUSTRACIÓN II 5: INTERFAZ AGENTE-ENTORNO.....	56
ILUSTRACIÓN II 6: ARQUITECTURA GINKO.....	58
ILUSTRACIÓN II 7: ARQUITECTURA GINKO.....	60
ILUSTRACIÓN II 8: CARACTERÍSTICAS DE LOS MOTORES.....	65
ILUSTRACIÓN II 9: SERVOMOTOR NXT.....	65
ILUSTRACIÓN II 10: SENSOR ULTRASÓNICO.....	66
ILUSTRACIÓN II 11: SENSOR DE COLOR.....	67
ILUSTRACIÓN II 12: GAMA DE COLORES.....	67
ILUSTRACIÓN II 13: TIPO DE DATOS ENTEROS.....	70
ILUSTRACIÓN II 14: TIPO DE DATOS FLOTANTES.....	70
ILUSTRACIÓN II 15: ESTRUCTURA DEL DESARROLLO ÁGIL.....	72
ILUSTRACIÓN II 16: ESTRUCTURA CENTRAL DE SCRUM.....	73
ILUSTRACIÓN II 17: CICLOS SCRUM.....	75
ILUSTRACIÓN III 1: DE LABERINTO A NODOS.....	78
ILUSTRACIÓN III 2: CAMINO VALIDO.....	78
ILUSTRACIÓN III 3: EFICIENCIA RECORRIDO.....	79
ILUSTRACIÓN III 4: USO DE LA CPU.....	80
ILUSTRACIÓN III 5: CONFIGURACIÓN Y RESOLUCIÓN DE LABERINTOS (LABERINTOS VIRTUALES).....	82
ILUSTRACIÓN III 6: LABERINTO Y CAMINO ENCONTRADO EXP 1.....	83
ILUSTRACIÓN III 7: USO DE LA CPU EXP 1.....	83
ILUSTRACIÓN III 8: RESULTADOS PARCIALES EXP 1.....	83
ILUSTRACIÓN III 9: LABERINTO Y CAMINO ENCONTRADO EXP 2.....	84

ILUSTRACIÓN III 10: USO DE LA CPU EXP 2. ....	84
ILUSTRACIÓN III 11: RESULTADOS PARCIALES EXP 2. ....	84
ILUSTRACIÓN III 12: LABERINTO Y CAMINO ENCONTRADO EXP 3. ....	85
ILUSTRACIÓN III 13: USO DE LA CPU EXP 3. ....	85
ILUSTRACIÓN III 14: RESULTADOS PARCIALES EXP 3. ....	85
ILUSTRACIÓN III 15: LABERINTO Y CAMINO ENCONTRADO EXP 4. ....	86
ILUSTRACIÓN III 16: USO DE LA CPU EXP 4. ....	86
ILUSTRACIÓN III 17: RESULTADOS PARCIALES EXP 4. ....	86
ILUSTRACIÓN III 18: LABERINTO Y CAMINO ENCONTRADO EXP 5. ....	87
ILUSTRACIÓN III 19: USO DE LA CPU EXP 5. ....	87
ILUSTRACIÓN III 20: RESULTADOS PARCIALES EXP 5. ....	87
ILUSTRACIÓN III 21: LABERINTO Y CAMINO ENCONTRADO EXP 6. ....	88
ILUSTRACIÓN III 22: USO DE LA CPU EXP 6. ....	88
ILUSTRACIÓN III 23: RESULTADOS PARCIALES EXP 6. ....	88
ILUSTRACIÓN III 24: LABERINTO Y CAMINO ENCONTRADO EXP 7. ....	89
ILUSTRACIÓN III 25: USO DE LA CPU EXP 7. ....	89
ILUSTRACIÓN III 26: RESULTADOS PARCIALES EXP 7. ....	89
ILUSTRACIÓN III 27: BARRAS REPRESENTANDO EL RENDIMIENTO DE TIEMPOS INDIVIDUALES. ....	90
ILUSTRACIÓN III 28: LÍNEAS REPRESENTANDO EL RENDIMIENTO DE TIEMPOS INDIVIDUALES. ....	90
ILUSTRACIÓN III 29: RESULTADOS DE PORCENTAJES TOTALES SIN TIEMPO EFECTIVO. ....	92
ILUSTRACIÓN III 30: RESULTADOS DE PORCENTAJES TOTALES CON TIEMPO EFECTIVO. ....	93
ILUSTRACIÓN III 31: BARRAS REPRESENTANDO LA EFICIENCIA INDIVIDUAL. .....	93
ILUSTRACIÓN III 32: LÍNEAS REPRESENTANDO LA EFICIENCIA INDIVIDUAL. .....	94
ILUSTRACIÓN III 33: PORCENTAJES TOTALES DE LA EFICIENCIA AL ENCONTRAR LA SOLUCIÓN. ....	95

ILUSTRACIÓN III 34: BARRAS REPRESENTANDO EL USO DE LA CPU.....	95
ILUSTRACIÓN III 35: LÍNEAS REPRESENTANDO EL USO DE LA CPU.....	96
ILUSTRACIÓN III 36: PORCENTAJES TOTALES DEL USO DE LA CPU.....	97
ILUSTRACIÓN III 37: GRAFICA DE RESULTADOS .....	103
ILUSTRACIÓN III 38: GRAFICA GENERAL DE RESULTADOS.....	103
ILUSTRACIÓN IV 1: ESPACIO DE ESTADOS .....	111
ILUSTRACIÓN IV 2: RESULTADOS DE PORCENTAJES TOTALES OBTENIDOS DEL ANÁLISIS DE MÉTODOS DE PLANIFICACIÓN. ....	118
ILUSTRACIÓN IV 3: GRAFICA DE RESULTADOS FINAL DEL ANÁLISIS. ....	121
ILUSTRACIÓN V 1: PARÁMETROS PARA LA GENERACIÓN DE LABERINTOS.	129
ILUSTRACIÓN V 2: DESCRIPCIÓN DE LA CANTIDAD DE NODOS GENERADOS. .....	129
ILUSTRACIÓN V 3: EJEMPLO DE SALIDAS DE LABERINTOS GENERADOS. ....	130
ILUSTRACIÓN V 4: EJEMPLO DE ESTADÍSTICA ALMACENADA TEMPORALMENTE.....	132
ILUSTRACIÓN V 5: EJEMPLO EN QUE EL CAMINO ES ENCONTRADO NORMALMENTE.....	133
ILUSTRACIÓN V 6: EJEMPLO EN QUE EL CAMINO ES ENCONTRADO USANDO LA MEMORIA DEL SISTEMA. ....	134
ILUSTRACIÓN V 7: DIAGRAMA CASO DE USO .....	136
ILUSTRACIÓN V 8: DIAGRAMA DE SECUENCIA.....	136
ILUSTRACIÓN V 9: DIAGRAMA DE COMPONENTES. ....	137
ILUSTRACIÓN VI 1: SENSOR ULTRASÓNICO. ....	139
ILUSTRACIÓN VI 2: RANGOS EN LA MEDICIÓN DE VARIAS DISTANCIAS.....	139
ILUSTRACIÓN VI 3: FUNCIONAMIENTO DEL SENSOR.....	140
ILUSTRACIÓN VI 4: SENSOR DE COLOR.....	140
ILUSTRACIÓN VI 5: AGENTE O MECANISMO AUTÓNOMO. ....	143
ILUSTRACIÓN VI 6: AGENTE O MECANISMO AUTÓNOMO USADO PARA EL EXPERIMENTO.....	143
ILUSTRACIÓN VI 7: DIAGRAMA GENERAL DEL AGENTE.....	145

ILUSTRACIÓN VI 8: AUTÓMATA EN EL INICIO DEL ESCENARIO PLANTEADO. .....	146
ILUSTRACIÓN VI 9: PRIMER OBSTÁCULO A SORTEAR.....	146
ILUSTRACIÓN VI 10: AUTÓMATA ENCONTRANDO LA SALIDA.....	147



## ÍNDICE DE TABLAS

TABLA II 1: PASOS HEURÍSTICOS GENERALES. ....	33
TABLA II 2: VENTAJAS Y DESVENTAJAS DEL ALGORITMO. ....	43
TABLA III I: PARÁMETROS ALGORITMOS PARA LA RESOLUCIÓN DE LABERINTOS. ....	81
TABLA III II: DATOS EXPERIMENTALES. ....	82
TABLA III III: PORCENTAJES OBTENIDOS (TIEMPOS SIN REFERENCIA DE TIEMPO EFECTIVO). ....	91
TABLA III IV: PORCENTAJES OBTENIDOS (TIEMPOS CON REFERENCIA DE TIEMPO EFECTIVO). ....	92
TABLA III V: PORCENTAJES OBTENIDOS (EFICIENCIA). ....	94
TABLA III VI: PORCENTAJES OBTENIDOS (USO DE LA CPU). ....	96
TABLA III VII: ESCALA DE VALORACIÓN CUALITATIVA Y CUANTITATIVA PARA LOS INDICADORES. ....	98
TABLA III VIII: VALORACIÓN PARA INDICADORES (EFICIENCIA). ....	98
TABLA III IX: VALORACIÓN PARA INDICADORES (TIEMPOS). ....	99
TABLA III X: VALORACIÓN PARA INDICADORES (TIEMPOS). ....	100
TABLA III XI: VALORACIÓN PARA INDICADORES (USO DE LA CPU). ....	100
TABLA III XII: VALORACIÓN PARA INDICADORES ESTABLECIDOS. ....	101
TABLA III XIII: RESULTADOS GENERALES. ....	104
TABLA IV I: DESCRIPCIÓN DE VALORES. ....	115
TABLA IV II: VALORACIONES INDIVIDUALES (MÉTODO BÚSQUEDA HACIA DELANTE). ....	116
TABLA IV III: VALORACIONES INDIVIDUALES (MÉTODO BÚSQUEDA HACIA ATRÁS). ....	116
TABLA IV IV: PONDERACIÓN DE RESULTADOS (TOTAL). ....	117
TABLA IV V: RESULTADOS DE PORCENTAJES TOTALES (CUMPLIR HIPÓTESIS). ....	120
TABLA IV VI: RESULTADOS TOTALES (CUMPLIR HIPÓTESIS). ....	121
TABLA V I: RECURSO HUMANO REQUERIDO. ....	125
TABLA V II: PERSONAL A CAPACITAR. ....	125

TABLA V III: COSTO DE DESARROLLO.....	126
TABLA VI I: ACCIONES TOMADAS POR EL AUTÓMATA.....	141
TABLA VI II: DESCRIPCIONES DE POSIBLES CAMINOS. ....	141
TABLA VI III: ALGORITMO GENERAL.....	144
TABLA VI IV: ALGORITMO GENERAL CON MEMORIA ACTIVA.....	144

## **Introducción**

El presente trabajo está dirigido al estudio de la inteligencia artificial orientada a la automatización de los mecanismos autónomos tanto en la guía como en la capacidad de aprender.

El hombre se ha caracterizado siempre por su búsqueda constante de nuevas vías para mejorar sus condiciones de vida. Estos esfuerzos le han servido para reducir el trabajo en aquellas operaciones en las que la fuerza juega un papel primordial. Los progresos obtenidos han permitido dirigir estos esfuerzos a otros campos, como por ejemplo, a la construcción de máquinas que ayuden a resolver rápida determinadas operaciones que resultan tediosas cuando se realizan a mano.

La robótica siempre ha estado unida a la construcción de "artefactos" con la idea de asemejarse al ser humano y de ahorrarle trabajo, creando mecanismo cada día más autónomos.

La palabra "robot" proviene del escritor checo Karel Capek, el cual acuñó en 1921 dicho término en una de sus obras a partir de la palabra checa "robota", que significaba servidumbre o trabajo forzado. Posteriormente, sería Isaac Asimov quien utilizaría el término "robótica" la ciencia que estudia los robots.

Actualmente, la robótica se define como la ciencia y tecnología de los robots. Se ocupa del diseño, manufactura y aplicaciones de éstos y combina diversas disciplinas como la mecánica, Electrónica, información, inteligencia artificial e ingeniería de control.

Aquí es donde aparece el término de Inteligencia Artificial, el cual está adquiriendo mayor protagonismo con el tiempo en la robótica. Se trata de una ciencia perteneciente a la rama de la Cibernética, que estudia el mecanismo de la inteligencia humana con el fin de

crear máquinas inteligentes, capaces de realizar cálculos y de "pensar", elaborar juicios y tomar decisiones.

Mientras que la robótica, en principio, evoluciona la mecánica de los robots, la inteligencia artificial fundamentada en la teoría de la evolución se basa en los mecanismos de selección que utiliza la naturaleza, donde los individuos más fuertes de una población son los que sobreviven.

## **CAPÍTULO I:**

### **MARCO REFERENCIAL**

#### **1.1 Antecedentes**

El Aprendizaje Automático es una disciplina dentro de los Sistemas Inteligentes cuya importancia ha quedado establecida desde hace dos décadas.

De las variadas formas que se han explorado, la que mejor imita el aprendizaje humano es la que toma en los sistemas con aprendizaje por observación y descubrimiento.

El principio de que los programas de aprendizaje permiten comprobar teorías de aprendizaje y define los sistemas de producción adaptativos como sistemas que aprenden nuevas reglas basadas en lo que han observado, agregando estas nuevas reglas al sistema de producción existente.

Los sistemas autónomos operan el aprendizaje y planificación mediante la observación de las consecuencias de ejecutar acciones planeadas en un ambiente. Para acelerar este tipo de convergencia, se utilizan las heurísticas de generalización de las observaciones y estimadores de probabilidad para manejar las contradicciones que presentan los operadores de planificación generados. El mecanismo de aprendizaje no solo permite

adquirir descripciones de los operadores, sino también adaptar dichos conocimientos adquiridos a los cambios del ambiente.

El conflicto con los sistemas actuales es la falta de adaptabilidad al ambiente en donde se desenvuelven, estos sistemas resuelven los problemas de manera muy mecánica dando soluciones a problemas ya programados en sus memorias, pero los ambientes nunca son los mismos incluso en ambientes ya estudiados pueden ocurrir cambios inesperados los cuales los cuales no estuvieron contemplados en la mente del programador.

En síntesis el análisis de los algoritmos heurísticos consolida una base sólida en la elaboración de mecanismos de aprendizaje que permite compartir el conocimiento entre varios agentes que tomen decisiones en tiempo real usando la memoria como fuente.

En una forma general un algoritmo de muy buenas características es el algoritmo backtraining, la idea de este algoritmo se asemeja a un recorrido en profundidad dentro de un grafo dirigido. El grafo en cuestión suele ser un árbol, o por lo menos no contiene ciclos. Sea cual sea su estructura, existe sólo implícitamente. El objetivo del recorrido es encontrar soluciones para algún problema. Esto se consigue construyendo soluciones parciales a medida que progresa el recorrido; estas soluciones parciales limitan las regiones en las que se puede encontrar una solución completa. El recorrido tiene éxito si, procediendo de esta forma, se puede definir por completo una solución.

Otra forma de resolución es la aplicada en los laberintos en la antigüedad, que consistía en colocar la **mano derecha** (Hand Right) en el muro y seguir caminando con la mano presionada hasta encontrar la salida.

El estudio y resultado de la investigación está destinada a fortalecer las bases de la enseñanza acerca de las inferencias lógicas dentro de la inteligencia artificial en la materia de Bases del Conocimiento en la Escuela de Ingeniería en Sistemas de la ESPOCH.

Este análisis está dividido en el análisis de los algoritmos con la elaboración de módulos, implementación de sistema y ejecución en busca de resultados.

## **1.2. Justificación del proyecto de tesis.**

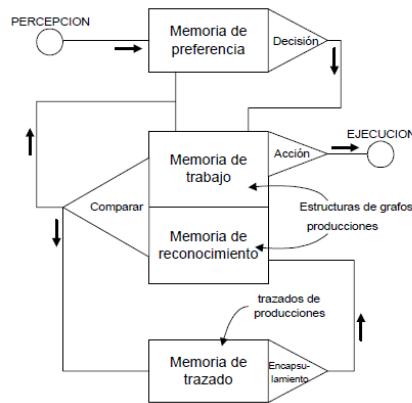
### **Justificación Teórica.**

La robótica es una de las principales herramientas que la humanidad tiene para cumplir con tareas que muchas veces son imposibles para una persona promedio, del cumplimiento de estas tareas dependen la economía de muchas empresas incluso en algunos escenarios las vidas de personas, de tal manera que dar la mayor independencia posible es el principal objetivo en la nueva generación de mecanismos auto desplazable.

En la actualidad el desarrollo de la robótica está muy desarrollado, nuevos sensores, nuevas unidades de memoria y nuevas formas de desplazamiento, pero los sistemas de control son pre programados para cumplir funciones fijas, simulando un ambiente fijo, los cuales no forman parte muchas veces de una predicción exacta de los manipuladores o administradores de estos sistemas, viendo la necesidad de crear parches, actualizaciones y limpiezas manuales de sus memorias en sistemas que pueden acoger una forma de conocer cuando actualizarse, que no le sirve en su memoria y desecharlo, sin la necesidad de parches software extras.

### **Justificación Práctica.**

Actualmente la Facultad de Informática y Electrónica de la ESPOCH, cuenta con una gran variedad de sistemas electrónico mecánicos desarrollados para el estudio de la robótica desarrollado por estudiantes y profesores, pese a contar con un buen sistema informático implementado para el control de dichos mecanismos, no podríamos decir que son sistemas totalmente autónomos y menos aún inteligentes.



**Ilustración I 1: Arquitectura GINKO.**

El análisis permitirá crear un sistema inteligente autónomo basado en la aplicación de un algoritmo lógico heurístico, el mismo que podrá ser implementado en diferentes tipos de mecanismos autónomos que cuenten con la configuración adecuada para la instalación del software resultante.

### **1.3. Objetivos.**

#### **1.3.1 Objetivo General:**

Analizar los algoritmos heurísticos para la aplicación en el aprendizaje y guía de sistemas autónomos.

#### **1.3.1 Objetivos específicos:**

- Analizar la característica de un de los algoritmos heurísticos más aplicados en la inteligencia artificial como es backtraining (vuelta atrás), frente a la técnica llamada la de la mano derecha.
- Determinar y definir los parámetros para realizar un análisis comparativo de los algoritmos heurísticos, el cual permitirá medir la eficiencia entre el algoritmo backtraining (vuelta atrás), y métodos clásicos como es la técnica de la mano derecha.
- Implementar el algoritmo heurístico seleccionado después del análisis, en un sistema inteligente autónomo con el fin de demostrar su aprendizaje y guía en un



escenario real, el sistema resultado será muy compatible con los mecanismos desarrollados en la Escuela de Ingeniería Electrónica de la ESPOCH y con el modelo comercial NXT 2.0, el sistema de “laberintos virtuales” será implementado con fines didácticos dirigidos a los estudiantes de Bases del Conocimiento en la Escuela de Ingeniería en Sistemas en la ESPOCH.

#### **1.4. Hipótesis.**

El análisis de los algoritmos heurísticos permitirá aplicar una adecuada planificación para el aprendizaje y guía de mecanismos autónomos.

#### **1.5. Métodos y Técnicas**

##### **1.5.1. Métodos**

Para la comprobación de la hipótesis será necesaria la aplicación de un método científico, que permitirá establecer una secuencia ordenada de acciones que nos llevarán a establecer las conclusiones sobre la aplicación de la lógica heurística en máquinas autónomas. Para el desarrollo de la aplicación de escritorio la cual será una herramienta didáctica, se empleará la metodología ágil SCRUM.

##### **1.5.2. Técnicas**

Para lo que tiene que ver en cuanto a fuentes de información se utilizarán principalmente lo que se refieran al tema de investigación como la observación, libros, revistas, páginas web, etc. además, se contactará a especialistas en los temas mediante video-llamadas, foros en línea, etc. También se empleará la observación y análisis por parte de los investigadores para hacer deducciones e inducciones sobre el tema de tesis.

Para la aplicación se realizarán entrevistas y encuestas a las personas que utilizarán el sistema y que actualmente realizan los procesos de forma manual, para determinar las necesidades que se deben cubrir.

## **CAPÍTULO II: MARCO TEORICO**

### **2.1. Definiciones**

#### **2.1.1. Inteligencia Artificial.**

La Inteligencia Artificial (IA) es la rama de las ciencias de la computación que se ocupa de construir sistemas que permitan exhibir un comportamiento cada vez más inteligente. En general la definición de inteligencia es recursiva, etimológicamente deriva de la voz latina "legere" que significa recolectar y por lo tanto elegir, "intellegere" significa elegir entre varias cosas. Inteligencia sería entonces la capacidad de discernir, discriminar, evaluar pero a medida que el conocimiento humano se fue ampliando, el concepto de inteligencia fue abarcando cada vez mayor cantidad de facetas del comportamiento no automático o repetitivo, cada vez más asociado a la resolución de problemas y al proceso creativo. [1]

##### **2.1.1.1. Categorías de inteligencia artificial.**

- **Sistemas que piensan como humanos.-** Estos sistemas tratan de emular el pensamiento humano; por ejemplo las redes neuronales artificiales. La automatización de

actividades que vinculamos con procesos de pensamiento humano, actividades como la Toma de decisiones, resolución de problemas, aprendizaje. [2]

- **Sistemas que actúan como humanos.**- Estos sistemas tratan de actuar como humanos; es decir, imitan el comportamiento humano; por ejemplo la robótica. El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor. [2]

- **Sistemas que piensan racionalmente.**- Es decir, con lógica (idealmente), tratan de imitar o emular el pensamiento lógico racional del ser humano; por ejemplo los sistemas expertos. El estudio de los cálculos que hacen posible percibir, razonar y actuar. [2]

- **Sistemas que actúan racionalmente (idealmente).** Tratan de emular de forma racional el comportamiento humano; por ejemplo los agentes inteligentes. Está relacionado con conductas inteligentes en artefactos. [2]

### 2.1.2. Sistemas inteligentes autónomos

Un rasgo comúnmente asociado con la inteligencia es la capacidad de adquirir nuevos conocimientos. Esto se manifiesta en los procesos de aprendizaje, que aceptan ser descritos en términos de asimilación e incorporación de información extraída del contexto. De esta forma, un sistema inteligente autónomo puede definirse como aquél capaz de descubrir y registrar si una acción efectuada sobre una situación dada fue adecuada para lograr su objetivo. Para aprender en un mundo real, un sistema necesita formular una teoría acerca de los efectos de las acciones sobre su entorno. Necesita construir planes, monitorizar la ejecución de esos planes para detectar expectativas violadas y diagnosticar y rectificar errores que los datos inconsistentes revelen. [3]

El aprendizaje es necesario porque en un nuevo entorno, el sistema no puede saber “a priori” las consecuencias de sus acciones ni las relaciones existentes entre las acciones y las percepciones. [3]

Recientemente se aborda el problema de compartir el conocimiento entre distintos agentes (sistemas inteligentes autónomos). Los agentes actúan y comparten conocimiento en un entorno distribuido, el empleo de Agentes Basados en Sistemas Distribuidos es compartir entre ellos el entendimiento común de la estructura de información como sucede entre personas, para lo cual los agentes de software que dan la inteligencia lógica a agente, requieren desarrollar ontologías. [3]

#### **2.1.2.1. Ontología.**

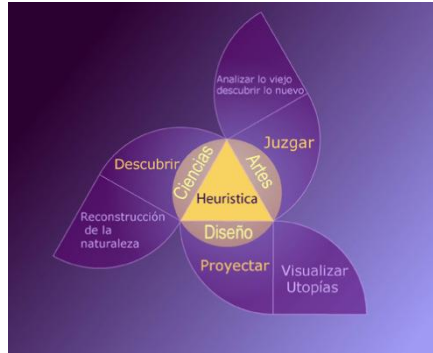
Una ontología contiene definiciones, conceptos básicos y relaciones entre estos que pueden ser interpretados por un sistema. El desarrollo de ontologías permite compartir el entendimiento común de las estructuras de información entre personas o agentes de software, la reutilización de conocimientos del dominio, explicitar suposiciones del dominio, separar el conocimiento del dominio del conocimiento operacional, analizar el conocimiento de un dominio. En términos prácticos desarrollar una ontología incluye: definir las clases, organizar las clases en una jerarquía taxonómica (superclase / sub-clase), definir slots y describir valores permitidos para esos slots. Se pueden distinguir tres tipos fundamentales de ontologías: de un dominio, genéricas, representacionales o también denominadas meta-ontologías. [3]

#### **2.1.3. Heurística.**

Se denomina heurística a la capacidad de un sistema para realizar de forma inmediata innovaciones positivas para sus fines. La capacidad heurística es un rasgo característico de los humanos, desde cuyo punto de vista puede describirse como **el arte y la ciencia del descubrimiento** y de la invención o de **resolver problemas mediante la creatividad y el pensamiento lateral** o pensamiento divergente.

La etimología de heurística es la misma que la de la palabra eureka, cuya exclamación se atribuye a Arquímedes en un episodio tan famoso como apócrifo. La palabra heurística aparece en más de una categoría gramatical. Cuando se usa como sustantivo, identifica el arte o la ciencia del descubrimiento, una disciplina susceptible de ser investigada

formalmente. Cuando aparece como adjetivo, se refiere a cosas más concretas, como estrategias heurísticas, reglas heurísticas o silogismos y conclusiones heurísticas. Claro está que estos dos usos están íntimamente relacionados ya que la heurística usualmente propone estrategias heurísticas que guían el descubrimiento. [4]



**Ilustración II 1: Esquema Heurística.**

### **2.1.3.1. Procedimientos heurísticos**

La heurística como disciplina científica, es relativamente joven, y en épocas muy recientes es que aparecen sistematizados los procedimientos heurísticos en la literatura pedagógica. El método heurístico se caracteriza como un método de enseñanza mediante el cual se le plantean en las personas impulsos que facilitan la búsqueda independiente de problemas y de soluciones, donde no se extienden problemas terminado, sino que los lleva al redescubrimiento de las suposiciones y reglas correspondientes, de forma independiente. La instrucción heurística es la enseñanza consciente y planificada de reglas generales y especiales de la heurística para la solución de problemas. [5]

La actividad heurística se debe analizar como variedad del pensamiento humano, la que crea un nuevo sistema de acciones o abre regularidades desconocidas hasta entonces, de los objetos que rodean al hombre (u objetos de la ciencia a estudiar).

El objetivo principal de la Heurística es investigar las reglas y métodos que conducen a los descubrimientos y a las invenciones e incluye la elaboración de principios, reglas,

estrategias y programas que facilitan la búsqueda de vías de solución a tareas de carácter no algorítmico de cualquier tipo y de cualquier dominio científico o práctico. [5]

### **2.1.3.2. Estrategias heurísticas.**

Son los procedimientos principales para buscar los medios matemáticos concretos que se necesitan para resolver un problema en sentido amplio y para buscar la idea fundamental de solución, por lo que, se les llama también estrategias de búsqueda.

Existen dos estrategias heurísticas que pueden ser aplicadas a cualquier tipo de ejercicio (estrategias generales o universales), ellas son: el trabajo hacia adelante o método sintético y el trabajo hacia atrás o método analítico. [5]

#### **2.1.3.2.1. Trabajo hacia adelante**

Se caracteriza por partir de los datos, deducir de ellos lo que se busca, pasando por una serie de casos intermedios, apoyándose en los conocimientos que tienen, de manera que se obtenga la cadena de ideas que permite elaborar el plan de solución. La estrategia consiste en buscar cuáles objetivos parciales o resultados intermedios se pueden alcanzar partiendo de los elementos dados.

#### *Ejemplo*

Utilizando la desigualdad triangular, demuestre que en todo triángulo, la mitad del perímetro es mayor que la longitud de cada lado.

*P: ABC triángulo cualquiera*

*T:  $a < a + b + c / 2$*

Trabajando hacia adelante se llega fácilmente a encontrar los medios y la idea de la demostración, pues sólo habría que plantearse la pregunta:

¿Qué datos ofrece la premisa, o que se puede plantear a partir de ella, que tenga relación con lo que se quiere probar? Y cómo en el texto se sugiere utilizar la desigualdad triangular, no será difícil responderla.

$A < b + c$  La comparación con la tesis sugiere sumar a

$$a = a$$

$a+a < a+b+c$  Efectuando se observa que al dividir

$a < a+b+c / 2$  por 2 se obtiene lo que se busca. [5]

### **2.1.3.2.2. Trabajo hacia atrás**

Se caracteriza por partir de lo que se busca, apoyándose en los conocimientos que se tienen, analizar posibles resultados intermedios de los que se puede deducir lo buscado (y cada resultado intermedio anterior) hasta llegar a los datos. De modo que recorriendo el camino a la inversa se tiene la idea de la solución. [5]

#### *Ejemplo*

Aplicando el trabajo hacia atrás (transformando la tesis) se logra establecer relaciones con la premisa (ABC triángulo cualquiera) que pueden sugerir los medios a utilizar y con ello la idea de la demostración.

*Se busca:  $a < a + b + c / 2$  lo que equivale a*

$$2a < a + b + c$$

$$a+a < a+b+c$$

$$a < b + c$$

Como se conoce que para todo triángulo es siempre posible plantear esta relación (desigualdad triangular) y que también, aplicando teoremas conocidos (propiedades de las operaciones con desigualdades) se pueden lograr las transformaciones necesarias (en sentido inverso a lo realizado) se tiene ya resuelto el problema. Para la planificación y dirección de los procesos de resolución de problemas se utilizan los llamados programas heurísticos. [5]

De interés especial resulta el conocido como programa heurístico general, el cual constituye para el profesor el instrumento universal de dirección, y para el alumno un a base de orientación para el trabajo con problemas, según el eminente matemático húngaro



George Polya se distinguen en el proceso de resolución de todo problema las cuatro etapas siguientes:

**Tabla II 1: Pasos heurísticos Generales.**

Fases Fundamentales	Tareas Principales
<ul style="list-style-type: none"><li>• <b>Orientación hacia el problema.</b></li></ul>	➤ Comprensión del problema.
<ul style="list-style-type: none"><li>• <b>Trabajo en el problema.</b></li></ul>	➤ Búsqueda de la idea de solución. ➤ Reflexión sobre los medios.
<ul style="list-style-type: none"><li>• <b>Solución del problema.</b></li></ul>	➤ Reflexión sobre la vía.
<ul style="list-style-type: none"><li>• <b>Evaluación de la solución del problema.</b></li></ul>	➤ Ejecución del plan de soluciones. ➤ Comprobación de la solución. ➤ Reflexión sobre los métodos aplicados.

### 2.1.3.3. Gnoseología de la heurística

Al estudiar cómo se generan los conocimientos innovadores bien podríamos estar hablando de una nueva disciplina que tiene como misión analizar los factores determinantes, del conocimiento heurístico, su naturaleza y su génesis, así como los procesos de definición y decisión referentes a un campo de conocimiento y transformación concreto, con el afán de dilucidar el carácter de las estrategias cognitivas para la innovación. [6]

La heurística refiere a estrategias, métodos, criterios o astucias utilizados para hacer posible la solución de problemas complejos y difíciles. [6]

El conocimiento heurístico es un tipo especial de conocimiento empleado a través del tiempo y en diversas latitudes por los seres humanos para resolver problemas de alta complejidad.

Al conocimiento en la actualidad se le demanda el adjetivo heurístico que significa comprender, esclarecer, descubrir, transformar, innovar, desarrollar, evolucionar, solucionar.

Un método heurístico es un conjunto de procesos cognitivos, propositivos y reflexivos que son necesarios realizar para identificar en el menor tiempo posible alternativas de solución de alta calidad y flexibilidad para un determinado problema. [6]

Al principio esta forma de resolver problemas no fue bien vista en los círculos académicos, debido aparentemente a su escaso rigor lógico y matemático. Sin embargo, gracias a su potencial práctico para solucionar problemas reales se fueron abriendo poco a poco las puertas a los métodos heurísticos, sobre todo a partir de los años 60 del siglo XX. Actualmente las versiones matemáticas y diseñadores de métodos heurísticos continúan desarrollándose y están incrementando el rango de sus aplicaciones, así como su variedad de enfoques. [6]

Nuevos métodos y técnicas heurísticas son utilizadas a diario por científicos de diversos campos, por empresarios, por diseñadores, por desarrolladores de informática y cibernética, para visualizar y hacer prospectivas con las cuales resolver problemas que antes eran demasiado complejos e impensables en las anteriores generaciones. [6]

#### **2.1.4. Laberintos**

El origen de los problemas sobre laberintos se refiere a tiempos antiquísimos y se pierde en las tinieblas de leyendas legendarias. Los antiguos y quizás algunos en nuestros tiempos, consideraban que los problemas sobre laberintos, en general, eran irresolubles. La persona que entraba en un laberinto no podía ya salir de él, de no ser que sucediese un milagro o que una casualidad lo ayudase.

Veremos, por el contrario, que laberintos sin salida no existen, que orientarse y encontrar salida del laberinto no supone gran esfuerzo. [7]

#### **2.1.4.1. Historia**

La palabra laberinto es de procedencia griega y significa pasos subterráneos. Efectivamente, existen multitudes de cuevas naturales subterráneas con una cantidad tan enorme de corredores, rincones y callejones sin salida, cruzados en todas las direcciones, que no es difícil perderse en ellos, extraviarse y al no encontrar la salida, morir de hambre y sed. [7]

Ejemplos de laberintos de esta clase, pero ya artificiales, pueden ser muchas ruinas de ciertos yacimientos, o las llamadas "catacumbas".

Lo más probable es que estas cuevas subterráneas excitaron, ya en los arquitectos antiguos, el deseo de edificar algo semejante a ellas. Por eso, en algunas obras de escritores antiguos (por ejemplo, egipcios) encontramos referencias a la existencia de laberintos artificiales.

Por último, la palabra "laberinto", precisamente con mayor frecuencia se refería a edificios artificiales sumamente complicados, con multitud de paseos o galerías, infinidad de ramificaciones, cruces y pasos sin salida, que obligaban al que entrase a errar inútilmente en busca de una salida. Sobre la construcción de estos laberintos se componían leyendas.

La más conocida es la leyenda sobre el laberinto, construido el mítico Dédalo en la isla de Creta para el mítico rey Minos. En el centro del laberinto vivía el monstruo Minotauro y nadie que entrase en ese laberinto podía salir de él; al fin y al cabo era víctima del monstruo. Siete mozos y siete mozas daban de tributo cada año los atenienses al monstruo, que los devoraba sin piedad. Por fin Teseo, no sólo mató al Minotauro, sino que consiguió salir del laberinto, sin extraviarse en él, orientándose por el hilo de un ovillo que le dio la princesa Ariadna. Desde entonces la expresión "el hilo de Ariadna" posee un significado simbólico como medio que permite salir de las situaciones más difíciles. [7]

Los laberintos tienen diversidad de formas y composición. Hasta nuestros días se han conservado galerías intrincadas y complejas, caminos por cuevas, laberintos

arquitectónicos sobre sepulturas, planes sinuosos en las paredes o pisos, marcados con mármol de color o con tejas, senderos tortuosos en el terreno y sinuosidades en el relieve de las rocas. [7]

Con dibujos de laberintos se adornaban las vestiduras de los emperadores cristianos hasta el siglo IX y restos de esta clase de adornos se conservan hasta hoy día en las iglesias y catedrales de aquellos tiempos. Es posible que estos adornos simbolizasen la complejidad del camino de la vida y de los extravíos del hombre. Sobre todo se practicaban mucho los laberintos en la primera mitad del siglo XII. En la Francia de aquellos tiempos, los laberintos se construían de piedra o se representaban en el piso de iglesias y catedrales. Con mayor frecuencia eran llamados "camino a Jerusalén" y simbolizaban el difícil camino terrenal hacia los "lugares santos", recompensado con la felicidad celestial, por eso, el centro de los laberintos con frecuencia se denominaba "cielo". [7]

En Inglaterra no se encuentran laberintos en los suelos de las iglesias, pero sí había muchos en praderas hechos con césped. Estos laberintos llevaban distintos nombres: "Ciudad de Troya", "Huella del pastor", etc. Laberintos como estos menciona Shakespeare en sus obras "Sueño de una noche de verano" y "La Tormenta". [7]

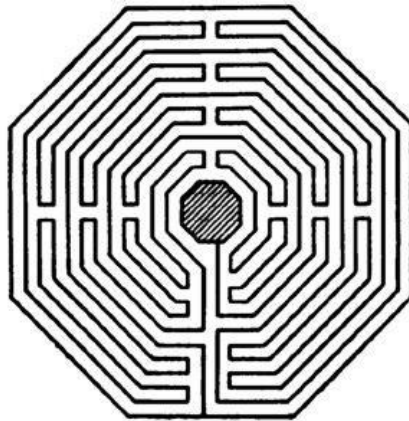
Todos estos laberintos poseen más bien interés histórico que matemático. Desenredarlos no es difícil. Con el transcurso del tiempo estas figuras perdieron su significación simbólica y poco a poco se convirtieron en objeto de distracción. Los laberintos pasan a jardines y por donde, mediante sendas sinuosas que se cruzan, separan, ramifican o terminan inesperadamente, se obtienen figuras muy enredadas, en las que efectivamente, no es fácil encontrar el camino del borde al centro y en las que no es difícil perderse. [7]

La nota histórica dada nos demuestra cuán viejo es el tema sobre los laberintos y a cuántas personas interesó él en sus tiempos. Los hombres se esmeraban en inventar los laberintos más complicados, "sin salida". ¿Pero acaso es posible construir realmente e incluso solamente dibujar, un laberinto sin salida, es decir, en el hecho de encontrar el camino hacia el centro y el camino de regreso hacia la salida fuera exclusivamente cuestión de

suerte, casualidad o fortuna y no producto de un cálculo matemático determinado y correcto?

La resolución de estos problemas pertenece a tiempos relativamente posteriores y su comienzo fue puesto por el famoso Euler. Los resultados de las investigaciones condujeron a la deducción de que no hay laberintos sin salida. [7]

La solución para cada laberinto puede ser hallada y además mediante procedimientos relativamente fáciles.



**Ilustración II 2: Laberinto formado de piedras en el piso del templo de San Quintín en Francia. La entrada es por abajo por la línea vertical.**



**Ilustración II 3: Laberinto en la catedral de Chartres, Francia**

#### **2.1.4.2. Camino más corto en un laberinto usando el algoritmo de la mano derecha o izquierda**

Teniendo un laberinto en el cual empezamos en el interior y buscamos la salida, uno de los algoritmos más conocidos para resolverlo es el conocido como de la “mano derecha” (o izquierda), y el cual consiste en simplemente mantener cualquiera de las dos manos pegada a la pared y avanzar hasta llegar a la salida. Este algoritmo funciona siempre que el laberinto tenga una conexión simple, lo que significa que todas las paredes están conectadas, y que exista una salida. [8]

#### **2.1.4.3. Algoritmo Backtracking**

El backtracking es una estrategia usada para encontrar soluciones a problemas que tienen una solución completa, en los que el orden de los elementos no importa y en los que existen una serie de variables a cada una de las cuales debemos asignarle un valor teniendo en cuenta unas restricciones dadas. El término fue acuñado por el matemático D.H. Lehmer e la década de los cincuenta y formalizado por Walker, Golom y Baumert en el siguiente decenio. El NIST (U.S. National Institute of Standards and Technology) lo define en su Diccionario de Algoritmos y Estructuras de Datos de la siguiente manera:

**Find a solution by trying one of several choices. If the choice proves incorrect, computation backtracks or restarts at the point of choice and tries another choice.**

Traducción:

**Encontrar una solución intentándolo con una de varias opciones. Si la elección es incorrecta, el cómputo retrocede o vuelve a comenzar desde el punto de decisión anterior y lo intenta con otra opción.**

Este esquema algorítmico es utilizado para resolver problemas en los que la solución consta de una serie de decisiones adecuadas hacia nuestro objetivo. El backtracking está muy relacionado con la búsqueda combinatoria. De manera más exacta podemos indicar que los problemas a considerar son los siguientes:

**1. Problemas de Decisión:** Búsqueda de las soluciones que satisfacen ciertas restricciones.

**2. Problemas de Optimización:** Búsqueda de la mejor solución en base a una función objetivo.

De forma general, el método del backtracking, concebido como tal, genera todas las secuencias de forma sistemática y organizada, de manera que prueba todas las posibles combinaciones de un problema hasta que encuentra la correcta. En general, la forma de actuar consiste en elegir una alternativa del conjunto de opciones en cada etapa del proceso de resolución, y si esta elección no funciona (no nos lleva a ninguna solución), la búsqueda vuelve al punto donde se realizó esa elección, e intenta con otro valor. Cuando se han agotado todos los posibles valores en ese punto, la búsqueda vuelve a la anterior fase en la que se hizo otra elección entre valores. Si no hay más puntos de elección, la búsqueda finaliza. [9]

Para implementar algoritmos con Backtracking, se usan llamadas a funciones recursivas, en la que en cada llamada la función asigna valores a un determinado punto de la posible solución, probando con todos los valores posibles, y manteniendo aquella que ha tenido éxito en las posteriores llamadas recursivas a las siguientes partes de la solución. [9]

#### **2.1.4.4. Aplicaciones del método Backtracking**

La técnica de Backtracking es usada en muchos ámbitos de la programación, por ejemplo, para el cálculo de expresiones regulares o para tareas de reconocimiento de texto y de sintaxis de lenguajes regulares. También es usado incluso en la implementación de algunos lenguajes de programación, tales como Planner o Prolog y da soporte a muchos algoritmos en inteligencia artificial. [9]

#### **2.1.4.5. Esquema General**

El esquema general para este tipo de algoritmos, es una función recursiva, como se describe a continuación. Considerando que el algoritmo está diseñado para resolver

problemas en los que estamos buscando una solución, es decir, cuando encontramos la primera que cumpla todas las restricciones, finalizaremos, sin importarnos si en realidad existen algunas más o si se trata de la mejor de las posibles. [9]

*Funcion Backtracking (Etapai) devuelve: boolean*

*Inicio*

*Éxito = falso;*

*IniciarOpciones(i, GrupoOpciones o);*

*Repetir*

*SeleccionarnuevaOpcion(o, Opcion n);*

*Si (Aceptable(n)) entonces*

*AnotarOpcion(i, n);*

*SiSolucionCompleta(i) entonces*

*Éxito = verdadero;*

*Sino*

*Éxito = Backtracking(i+1);*

*Si Éxito = false*

*entonces*

*cancelamosAnotacion(i, n);*

*Fin - si;*

*Fin - si;*

*Fin - si;*

*Hasta (éxito = verdadero) o (NoQuedanOpciones(o));*

*Retorna Éxito;*

*Fin;*

## **Descripción.**

### **1. IniciarOpciones(Etapa i, GrupoOpciones o);**

En base a las alternativas elegidas en la solución parcial i que nos llega como parámetro, introducimos en “O” el grupo de opciones posibles que podemos probar en la etapa actual.



Si ninguno de estos valores condujese a una solución válida, la llamada a la función finalizaría y volveríamos a una etapa posterior.

## **2. SeleccionarNuevaOpcion(GrupoOpciones o, Opcion n);**

De entre todas las alternativas posibles que tenemos en O, elegimos una. Aquí podemos realizar funciones de Poda, que pueden ser muy beneficiosas en ciertos algoritmos, proporcionando opciones que tienen una alta probabilidad de convertirse en solución y evitando muchas pruebas innecesarias con valores que según nuestra estimación no nos conducirá al éxito. Hay que señalar que, de alguna forma esta opción elegida tiene que ser desechada o marcada como usada en el grupo de opciones O, para que no volvamos a probar con el mismo valor repetidas veces, es decir, el conjunto vaya reduciéndose hasta que no queden más alternativas por probar.

## **3. Aceptable (Opción n);**

Los problemas que resuelve este tipo de esquemas algorítmicos tienen como característica que buscan un grupo de valores que cumplen entre si una serie de restricciones. Pues bien, el cumplimiento de estas restricciones se comprueba en esta función. Si la opción no es aceptable no será necesario expandir por esa rama las posibilidades porque en ningún caso nos llevarán a una solución. De esa manera evitamos tener que explorar zonas del espacio de alternativas que sólo nos harán perder tiempo de búsqueda sin proporcionar ningún resultado. Esta función es por tanto de vital importancia, porque tiene funciones de poda que pueden convertir el algoritmo en mucho más eficiente. Para construirla, se suele llevar a cabo un análisis exhaustivo del problema a tratar, para identificar esas situaciones de la solución parcial en la que no debemos expandir más porque nunca llegaremos a una solución. Hay también que considerar que si el coste computacional es elevado, esta función puede incluso empeorar el tiempo de ejecución pues esta función que se ejecutará muchas veces, y si el coste computacional de la misma es elevado y los casos que evita son pocos debemos optar por otra más sencilla y menos difícil de calcular.

#### **4. AnotarOpcion(Etapa i, Opción n);**

Esta función anota la opción n elegida en las funciones anteriores dentro de la solución parcial que llevamos construida hasta la etapa i. Esta opción fue elegida en SeleccionarNuevaAcción y comprobada como válida en Aceptable, de manera que tenemos asegurado que es una alternativa que aún no hemos probado, y que no incumple ninguna de las reglas de poda que hayamos introducido en nuestro programa.

#### **5. cancelamosAnotacion(Etapa i, Opcion n);**

Realiza la acción contraria. Debido a que esa opción n que habíamos anotado no ha llevado a una solución, o estamos buscando soluciones alternativas, la eliminamos de la solución parcial en la etapa i, para que podamos probar con otra si es que existe.

#### **6. NoQuedanOpciones(o);**

Indica si todavía existe en el grupo inicial que calculamos en cada función recursiva, alguna alternativa más que probar. Si no queda ninguna opción, inevitablemente hemos terminado de buscar en la etapa actual, y debemos ir hacia atrás a la anterior llamada recursiva para buscar nuevas alternativas por donde expandir el árbol de posibilidades.

#### **2.1.4.6. Ventajas y desventajas**

Los algoritmos de backtracking no son eficientes, ya que se basan en el método de prueba y error y en muchas ocasiones para conseguir solución se tiene que recorrer todo el árbol de búsqueda o gran parte de él. También tendremos que achacar que la recursividad contribuye a su ineficiencia, por la memoria que gasta durante las diferentes llamadas recursivas. [9]

Ahora bien hay que decir que la eficiencia depende de:

- El número de nodos del árbol de búsqueda que se visitan para conseguir la solución  $\rightarrow v(n)$ .

- El trabajo realizado en cada nodo, esto es, el coste de la función de solución completa o ver si la solución es aceptable hasta el momento. Este coste lo podemos expresar como  $\rightarrow p(n)$ , ya que generalmente será polinómico.

- El coste en general será:  $O(p(n) v(n))$ , este coste será exponencial en el peor caso.

Para conseguir mejoras en los costes se suele recurrir a la poda del árbol de búsqueda, lo cual se hace marcando los caminos que ya se han estudiado y los no prometedores como cerrados con lo cual el algoritmo no perderá tiempo con los nodos que estén dentro de estos caminos. También se podrían crear predicados acotadores reduzcan mucho el número de nodos generados, si el predicado acotador sólo dejara un nodo el algoritmo se convertiría en voraz y su coste bajaría a  $O(p(n)n)$ . Aunque normalmente los costes más bajos que se pueden conseguir son el orden de  $O(p(n) 2n)$ .

En conclusión debido al coste creado en tiempo y memoria (por la pila recursiva) los algoritmos de vuelta atrás no son todo lo eficientes que deberían, y debemos dejarlos para resolver parte de otros problemas o problemas reducidos. Aun así la gran ventaja que tienen es que si hay solución la encontrarán. [9]

**Tabla II 2: Ventajas y desventajas del algoritmo.**

Ventajas	Desventajas
<b>Si existe una solución, la calcula.</b> <b>Es un esquema sencillo de implementar.</b> <b>Adaptable a las características específicas de cada problema.</b>	Coste exponencial $O(ji)$ en la mayoría de los casos. Si el Espacio de Búsqueda es infinito, la solución, aunque exista, no se encontrará nunca. Por término medio consume mucha memoria al tener que almacenar las llamadas recursivas.

#### 2.1.4.7. Árbol de búsqueda

Cómo ya hemos comentado anteriormente el algoritmo de vuelta atrás proporciona una manera sistemática de generar las todas las posibles soluciones siempre que se puedan resolver por etapas, lo que se asemeja mucho a una búsqueda combinatoria (probar todas

la posibles combinaciones). Para conseguir este estudio tan exhaustivo del problema, se considera que se trabaja con un árbol que cuya existencia es sólo implícita, para nosotros cada nodo del nivel k representa una parte de la solución y nuestro árbol estará formado por las k etapas que se considerarán ya realizadas.

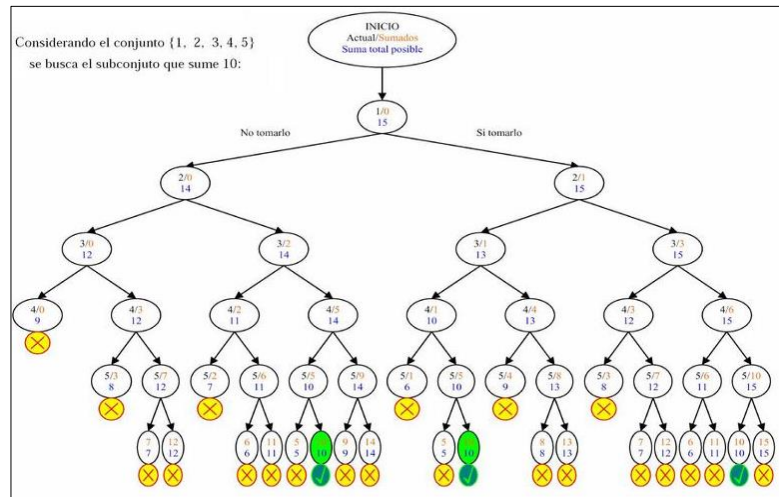


Ilustración II 4: Árbol de Búsqueda.

La búsqueda que realizada sobre el árbol es una búsqueda en profundidad. En el transcurso de la búsqueda si se encuentra un estado incorrecto, se ha de retroceder hasta la decisión anterior y si existe uno o más caminos aún no explorados que puedan conducir a la solución, el recorrido del árbol continúa por uno de ellos (hijos si nos referimos a un árbol). Si no quedasen más alternativas la búsqueda fallaría y el problema no tendría solución. En este caso en el que consideramos el problema en forma de árbol, la solución sería un camino que llevara desde el nodo raíz hasta uno nodo hoja, y las soluciones parciales llevarían desde el nodo raíz a los nodos interiores del árbol. [9]

### 2.1.5. Mecanismos autónomos

Los mecanismos autónomos son entidades físicas con capacidad de percepción sobre un entorno y que actúan sobre el mismo en base a dichas percepciones, sin supervisión directa de otros agentes.

En muchas ocasiones se asimila el término "mecanismo autónomo" con el de "robot móvil", pero en realidad son términos diferentes. Un mecanismo autónomo suele ser móvil, entendiendo por móvil que no se encuentra fijado a una posición y puede desplazarse por su entorno, pero no hay nada que en principio obligue a ello. A la inversa, un robot móvil no es necesariamente autónomo: existen multitud de robots móviles que son teleoperados en mayor o menor medida, con lo que contravienen nuestra definición.

#### **2.1.5.1. Características**

**Están situados:** Cuando decimos que un mecanismo autónomo percibe un entorno y actúa sobre él, no lo hacemos a la ligera. El mecanismo está literalmente inmerso en el entorno, esto es, el mecanismo no actúa sobre abstracciones o modelos, sino directamente sobre la realidad material. [10]

**Son entidades corpóreas:** Como hemos comentado anteriormente, los mecanismos operan sobre el mundo físico; su experiencia del mundo y sus acciones sobre el mismo se producen de forma directa haciendo uso de sus propias capacidades físicas. [10]

#### **2.1.5.2. Paradigmas en la robótica**

En las ciencias de la computación y en la ingeniería frecuentemente se hace uso del término "paradigma". Resulta interesante examinar la etimología de esta palabra, así como el significado con el que se usa en la actualidad. Paradigma proviene del griego *paradeigma*, y su significado era originalmente "ejemplo ilustrativo". En particular se usaba para denotar un enunciado modelo que mostraba todas las inflexiones de una palabra. Sin embargo, en su famoso libro "La estructura de las revoluciones científicas" [Kuhn70], el historiador Thomas Kuhn amplió dicha definición, dándole el significado de un conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento, esto es, una forma de ver el mundo. [10]

En la robótica esta definición se adapta mejor que en cualquier otra disciplina, ya que en este campo un paradigma es literalmente una manera de conseguir que los robots tengan una forma de ver el mundo. [10]

Los principales paradigmas en la robótica son el jerárquico, el reactivo y el híbrido.

#### **2.1.5.2.1. Paradigma jerárquico.**

El paradigma jerárquico es el más antiguo. Históricamente tuvo su mayor influencia en las décadas de los 60 y 70. Este se basa en gran medida en una aproximación antropomórfica, bajo la cual los robots se desarrollan tomando como inspiración la manera en la que los seres humanos resuelven los problemas. Esto supone que este paradigma tiene mucho de simbólico, y se da mucha importancia a las tareas de planificación, más si cabe teniendo en cuenta que en la época en la que surgió, la aproximación simbólica a la inteligencia artificial se encontraba en su momento de mayor auge. [10]

Este paradigma se denomina también paradigma Percepción, Planificación, Acción o PPA, por los nombres de las tres operaciones básicas que lo componen.

La denominación de jerárquico se debe a que las tres operaciones que acabamos de mencionar se llevan a cabo siempre secuencialmente, en el orden en que las hemos nombrado, y porque cada operación toma como entrada las salidas de la operación anterior.

La percepción es clave en todo mecanismo autónomo, ya que para poder influir sobre el entorno, es necesario que exista una manera de percibir el estado actual del mismo. [10]

En la fase de planificación el robot decide qué acciones llevar a cabo tomando como base las percepciones recién adquiridas y sus propios objetivos. Para poder efectuar la planificación, las percepciones deberán haber sido preprocesadas y transformadas en una abstracción del entorno. [10]

En la fase de acción se llevan a cabo las acciones de alto nivel que se seleccionaron en la fase de planificación, traduciéndolas en movimientos de los actuadores del robot. [10]

En la época en la que se desarrolló este paradigma se consideraba que los grandes problemas que planteaban las técnicas de inteligencia artificial eran los relativos a la demostración de teoremas matemáticos, la comprensión del lenguaje natural y otras tareas formales. Estos fueron los temas fundamentales que se trataron en la conferencia de Dartmouth en el año 56. Los primeros trabajos surgidos de aquí fueron los que dieron origen a la Inteligencia Artificial tal y como la conocemos, y centraron las tareas a las que se dedicarían los investigadores en los años siguientes. [10]

El paradigma PPA es heredero de esta filosofía, ya que en este paradigma se considera que el problema fundamental es el de la planificación de tareas en base a un razonamiento simbólico. [10]

Posteriormente los investigadores comenzaron a darse cuenta de que las tareas realmente complejas son las relacionadas con la percepción. A principios de los años 60 se consideraba que problemas como el de la visión artificial se solucionarían definitivamente en un plazo relativamente corto. Sin embargo la práctica demostró que este tipo de tareas presentan muchas más dificultades que las que tienen que ver con el razonamiento formal. [10]

Las tareas relacionadas con la percepción han resultado ser computacionalmente intratables, y su abstracción y transformación a símbolos no ha sido resuelta todavía. [10] Los problemas relativos a la planificación hicieron fracasar al paradigma PPA. Los investigadores se dieron cuenta de que para poder efectuar la planificación de la manera que se había planteado, era necesario que en la capa de percepción pudiera llevarse a cabo una complicadísima tarea de abstracción para proveer a la capa de planificación con los símbolos de alto nivel con los que ésta trabaja. Esto estaba fuera del alcance de las tecnologías de los años 60-70 y sigue siendo inviable en la actualidad. Por ello fue necesario buscar otras aproximaciones en las que no fuera imprescindible un preprocesamiento intensivo de las percepciones del robot. Así fue cómo surgió el paradigma reactivo. [10]

### **2.1.5.2.2. Paradigma reactivo**

El paradigma reactivo fue la respuesta a los problemas que introducía el paradigma jerárquico. Este paradigma, que surgió en los años 80 y estuvo vigente hasta principios de los noventa, está inspirado en las ciencias cognitivas y en la biología, y como su nombre indica su manera de operar obvia la planificación y se basa en correspondencias directas entre percepción y acción. Por ello, este paradigma se denomina también paradigma Percepción Acción o PA. [10]

El paradigma PPA resultaba demasiado ambicioso y ponía demasiado énfasis en el razonamiento. La nueva generación de robots comenzó a tomar como inspiración una inteligencia más práctica, ligada a la acción directa a partir de las percepciones, sin un preprocesamiento intensivo de las mismas. Los científicos comenzaron a darse cuenta de que la inteligencia no está limitada ni mucho menos al razonamiento abstracto. Todo esto queda resumido en el título de un famoso artículo de Rodney Brooks, apóstol de este paradigma: "Los elefantes no juegan al ajedrez" [Brooks90]. [10]

Con esto, Brooks quiere decir que un animal, como el elefante de su ejemplo, no necesita llevar a cabo razonamientos simbólicos de alto nivel para sobrevivir en su entorno. Lo realmente imprescindible es que el animal disponga de una manera de percibir su entorno y actuar sobre el mismo reactivamente, sin llevar a cabo un razonamiento intensivo. Brooks propone la arquitectura de subsunción para implantar este tipo de comportamientos en los robots autónomos. [10]

El paradigma PA presenta unos requerimientos mucho menores para las tareas de percepción, ya que al no haber planificación no resulta necesario abstraer lo percibido transformándolo en símbolos, con lo que se evita la explosión combinatoria. En lugar de ello, las percepciones son pasadas al módulo de acción con un mínimo preprocesamiento. El módulo de acción trabaja en base a estas entradas subsimbólicas y a partir de ellas genera un comportamiento. Este módulo suele dividirse en varios niveles que se encargan de tareas de distinta prioridad. La acción de "evitar obstáculos" suele ser la de mayor prioridad. Por encima de esta acción se sitúan las acciones más abstractas, relacionadas



con los objetivos finales del robot, pero que tienen menor prioridad comparadas con las acciones primarias o "actos reflejos", como el de evitar los obstáculos. [10]

A pesar de su mayor sencillez, o precisamente debido a ella, los robots construidos empleando este paradigma sólo pueden emplearse para llevar a cabo tareas moderadamente sencillas, y que no implican razonamientos complejos. [10]

Esto no es óbice para considerar este paradigma como todo un éxito. De hecho, algunos de los mayores éxitos de la robótica hasta el momento fueron desarrollados siguiendo el paradigma PA. Por ejemplo, el Mars Explorer, que aunque en parte era teleoperado también contaba con cierta dosis de autonomía desarrollada siguiendo el paradigma PA.

Los partidarios de este paradigma están en lo correcto cuando afirman que el razonamiento abstracto no es la prioridad en un robot o mecanismo autónomo, y que lo primero con lo que hay que dotarle es con cierta inteligencia práctica que le permita sobrevivir y manejarse en su entorno. Sin embargo si queremos ser más ambiciosos no podemos limitarnos a los modelos reactivos. Una vez que el robot consigue manejarse y es capaz de llevar a cabo tareas rudimentarias, una progresión natural nos lleva a intentar conseguir comportamientos más complejos. El paradigma reactivo no permite conseguir esto. Por ello resulta necesario dar un paso más y buscar una manera de conseguir ese aumento de capacidades en los robots.

#### **2.1.5.2.3. Paradigma híbrido deliberativo/reactivo**

El paradigma híbrido se desarrolló para tratar de conseguir este aumento de capacidades operativas de los robots autónomos y se ha venido empleando desde mediados de los años noventa hasta la actualidad. [10]

La idea de este paradigma es llegar a un compromiso entre las tareas deliberativas y las reactivas. En este caso el robot está compuesto, por un lado, de una cadena deliberativa típica (PPA) y por otro lado de un modelo reactivo (PA). [10]

La cadena deliberativa se encarga de llevar a cabo la planificación de alto nivel, como en el caso del modelo deliberativo tradicional. La diferencia es que se reduce el énfasis en la abstracción de las percepciones. La cadena reactiva entra en funcionamiento cuando resulta necesaria una acción inmediata para garantizar la supervivencia del robot (evitación de colisiones, por ejemplo). [10]

Otro tipo de modelo híbrido es el llamado Planificación, Percepción - Acción. En este modelo el robot planifica cómo descomponer una tarea en subtareas y elige cuáles son los comportamientos más adecuados para cumplir con esas tareas. En ese momento dichos comportamientos empiezan a ejecutarse a través del paradigma reactivo tradicional. En este caso las percepciones también se ponen a disposición del módulo de planificación. [10]

### **2.1.5.3. Importancia del aprendizaje en mecanismos autónomos**

En cualquier caso, los paradigmas y arquitecturas descritas, por sí solos adolecen de la capacidad de dotar al robot con herramientas para sobrevivir en un entorno abierto y de condiciones cambiantes. Es por ello por lo que se considera de tanta importancia el poder proporcionar a los robots mecanismos de aprendizaje. [10]

### **2.1.5.4. Problemas de la falta de adaptabilidad [10]**

Las capacidades de aprendizaje de los robots primitivos eran nulas. El principal interés de los investigadores era imbuir a las entidades físicas que diseñaban con ciertos comportamientos que ellos mismos elegían en base a su propia experiencia. Esto presenta tres problemas fundamentales:

- En primer lugar, sucede muy a menudo que los comportamientos que los desarrolladores de robots creen adecuados para una de sus creaciones se demuestran poco útiles o incluso contraproducentes al llevarlos a la práctica. Esto se debe a que en algunos casos la única manera de elaborar estos comportamientos es en base a simulaciones y a posteriores refinamientos de dichas simulaciones. En otros casos los comportamientos del

robot se obtienen a partir de modelos matemáticos, típicamente métodos de ingeniería de control, con los que ocurre algo parecido al llevarlos a la práctica: el modelo matemático no se ajusta totalmente a lo esperado debido a factores no tenidos en cuenta. En cualquiera de los dos casos se termina dotando al robot con una serie de reglas y parámetros ad-hoc, que no surgen realmente de un conocimiento profundo del propio robot y su entorno, sino que se obtienen en base a múltiples pruebas de ensayo y error.

- El segundo problema es la falta de capacidad de adaptación de un robot diseñado de esta forma. Un robot sin capacidades de aprendizaje puede funcionar adecuadamente en un entorno controlado, pero al enfrentarse a un entorno cambiante, las acciones que antes eran adecuadas pueden convertirse en inútiles.

- El tercer problema consiste en la dificultad de optimizar el consumo de energía y los movimientos del robot. La optimización del consumo de energía y la economización de movimientos pueden calcularse de manera teórica, empleando los mismos métodos que ya hemos comentado. Sin embargo, por las mismas razones esgrimidas antes, es más que improbable que los cálculos teóricos se ajusten a la perfección a las características particulares del robot. Si el robot tuviese la capacidad de calcular su propio consumo y aprender cuáles son las acciones que consiguen reducir dicho consumo al mínimo para cumplir un cierto objetivo, esto podría suponer grandes ahorros en tiempo y energía en la vida útil del robot.

Estos problemas que acabamos de describir podrían solucionarse satisfactoriamente proporcionando a los robots capacidades de aprendizaje. En concreto, las técnicas de aprendizaje por refuerzo nos pueden ayudar a solucionar cada uno de estos tres problemas.

#### **2.1.5.5. Aproximaciones simbólicas vs subsimbólicas.**

Analizando lo descrito anteriormente la simbología aplicada exige que existen varios problemas en las arquitecturas subsimbólicas, como la de subsunción. Sin el afán de defender una aproximación simbólica pura a la robótica o a la IA, sobre todo teniendo en cuenta

los fracasos por los que ha pasado y los éxitos que están cosechando en la actualidad las diferentes aproximaciones subsimbólicas, como por ejemplo las redes neuronales. En todo caso, si buscamos conseguir en los robots una inteligencia remotamente comparable a la humana, algo de simbolismo siempre va a ser necesario. Quizás dicho simbolismo pueda surgir de manera emergente de un sustrato subsimbólico, o por medio de mecanismos de operacionalización, pero posiblemente hagan falta otros elementos. [10]

Después de todo, resulta un hecho probado que a ciertos niveles los seres humanos operamos con símbolos, y que para poder llegar a un nivel de inteligencia y de razonamiento elevados es necesario contar con la aproximación simbólica, sobre todo si en algún momento se plantea la necesidad de la comunicación entre varios agentes. Una comunicación de este tipo necesita de manera imprescindible que exista un lenguaje simbólico compartido entre los diferentes agentes. [10]

La principal objeción que podemos presentar a las arquitecturas tradicionales es la falta total de mecanismos de aprendizaje y debido a ello la necesidad de incluir reglas y mecanismos muy específicos. Además, únicamente con esto no podemos generar comportamiento inteligente ni adaptación, más allá de lo programado en el robot, como ya hemos comentado.

Esto es independiente del paradigma empleado. Brooks critica la IA tradicional y en concreto el paradigma PPA, por el hecho de que los desarrolladores de robots tienen que imbuir a sus creaciones con grandes cantidades de conocimiento simbólico que no está "asentado" en la realidad física y porque esto supone que deben contemplarse muchos casos particulares y reglas ad-hoc. Sin embargo en el caso de su arquitectura de subsunción, basada en el paradigma PA, también existen estos casos particulares y reglas ad-hoc. La diferencia es que en la aproximación de Brooks dichas reglas no están basadas en manipulación de símbolos, sino en una cierta manera de interactuar entre sí de las distintas capas de su arquitectura. El diseño que hace Brooks no es a nivel simbólico, pero sigue siendo un diseño para casos particulares, a menos que se añadan capacidades de aprendizaje. [10]

Si quisiéramos construir un robot con inteligencia generalista, empleando la arquitectura de subsunción tendríamos que hacer algo similar a lo que habría que hacer en las aplicaciones de IA simbólica tradicional: Elaborar un grandísimo conjunto de reglas e incorporarlas a nuestro robot, lo cual no es viable. [10]

Probablemente ninguna de las aproximaciones actuales es correcta, y más aún, ni siquiera podemos decir que "una mezcla de las dos" sea suficiente para solucionar este tipo de problemas. [10]

Consideramos que se hace necesario elaborar un paradigma distinto, centrado sobre todo en el aprendizaje, y más concretamente, en el aprendizaje y la creación de símbolos a partir de entradas sub-simbólicas. [10]

Existen nuevas aproximaciones en diferentes campos que podrían servir de inspiración para buscar este nuevo paradigma. Por ejemplo, Marvin Minsky lleva años abogando por una nueva teoría de la mente, descrita en su libro "Society of mind" [Minsky88], título también de uno de sus cursos en el MIT. Dicha teoría tiene como objetivo explicar el funcionamiento de la mente humana y del sentido común, y para ello subdivide la mente misma en un conjunto de agentes que cooperan entre sí. Esta teoría llevada a la práctica podría conducirnos algo más cerca del objetivo de lograr mecanismos verdaderamente autónomos. [10]

Actualmente se está generando un gran interés en el campo de la robótica adaptativa, basada fundamentalmente en aproximaciones biológicas y en el aprendizaje. Probablemente esta aproximación nos permita avanzar algo más y acercarnos a nuestra meta. [10]

#### **2.1.5.6. Aprendizaje por refuerzo**

Hay un método concreto que permite a los mecanismos autónomos aprender de su propio entorno, y además, aprender mientras se encuentran inmersos en dicho entorno. Este

método se denomina aprendizaje por refuerzo y ha tenido un importante auge en los últimos años.

Este método de aprendizaje surge por un lado y conceptualmente, de una rama de estudios de psicología experimental, que pueden remontarse a las experiencias de Pavlov con el refuerzo condicionado, y por otro lado es heredero de los métodos de control óptimo que se originan a partir de los trabajos de Bellman. [10]

Dicho de forma breve, el aprendizaje por refuerzo es el problema de conseguir que un agente actúe en un entorno de manera que maximice la recompensa que obtiene por sus acciones.

Para entender esta definición, hay que definir a su vez los términos agente, entorno y recompensa.

#### **2.1.5.6.1. Agente**

El término agente es uno de los más empleados en la actualidad en el mundo de la informática. Incluso a lo largo de este documento lo hemos empleado en varias ocasiones. Existen múltiples definiciones para este término. Para nuestros intereses vamos a escoger por su claridad y concisión la de Pattie Maes [Maes95]. Según esta definición, un agente es un sistema computacional que habita en un entorno complejo y dinámico, con la capacidad de percibir y actuar autónomamente sobre dicho entorno, y de esta manera es capaz de cumplir un conjunto de objetivos o llevar a cabo ciertas tareas para las cuales fue diseñado.

#### **2.1.5.6.2. Entorno**

Por correspondencia con la anterior definición, se puede definir el entorno como todo aquello que no es el agente y que es de interés para llevar a cabo la tarea que se le ha asignado a dicho agente. Los agentes siempre se encuentran "situados" en un entorno, del que reciben sus percepciones y sobre el que ejecutan sus acciones.

El que los entornos sean complejos y dinámicos, como se decía en la definición anterior puede suponer que al efectuar una misma acción en el mismo estado en dos ocasiones distintas obtengamos consecuencias diferentes cada vez. [10]

De la manera que hemos definido el entorno, como "todo lo que no es el agente", debemos tener en cuenta que en el entorno podría incluir a otros agentes. [10]

Una particularidad de los entornos complejos es que en muchas ocasiones los agentes no tienen la capacidad de percibir completamente dicho entorno, y esto puede añadir muchas dificultades a la hora de llevar a cabo ciertas tareas, como veremos más adelante. [10]

### **2.1.5.6.3. Recompensa**

La recompensa es un valor escalar que indica lo deseable que es una situación para un agente. La recompensa puede tomar valores tanto positivos como negativos. Fisiológicamente podría compararse un valor de recompensa negativo con el dolor y un valor positivo con el placer.

Cada vez que el agente ejecuta una acción, recibe un valor de recompensa. Estas recompensas no tienen por qué estar asociadas directamente con la última acción ejecutada, sino que pueden ser consecuencia de acciones anteriores llevadas a cabo por el agente. [10]

No es sencillo efectuar una correspondencia directa entre acciones y recompensa obtenida. Es posible que la misma acción llevada a cabo en dos momentos diferentes devuelva una recompensa muy distinta debido a la "historia" previa. Por ello, no podremos saber de manera directa a qué acción concreta se deben los buenos o malos resultados obtenidos por un robot. Esto es una versión del clásico problema de la asignación de crédito. [10]

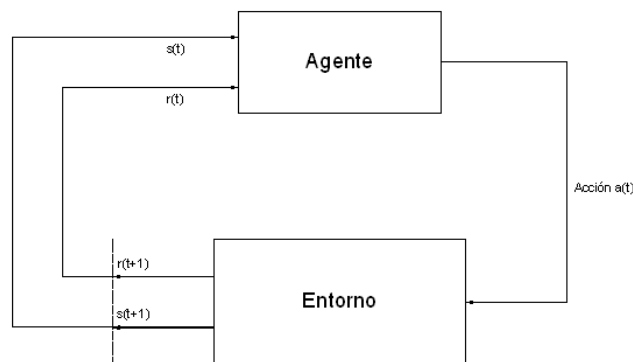
La suma de la recompensa se denomina "retorno". El objetivo final del aprendizaje por refuerzo es conseguir un agente que maximice el retorno a largo plazo. [10]

### 2.1.5.7. Interfaz Agente – Entorno

Analizado lo anterior, se puede comprender mucho mejor la definición anterior de aprendizaje por refuerzo. Para profundizar en esta definición y en el problema al que nos enfrentamos podemos pasar a estudiar el interfaz agente-entorno. [10]

El agente y el entorno interactúan en una secuencia de instantes de tiempo  $t=0,1,2,3,4,\dots$ . En cada instante de tiempo  $t$ , el agente recibe una representación del estado del entorno  $S_t \in S$ , donde  $S$  es el conjunto de posibles estados. En base a esto, el agente selecciona una acción  $a_t \in A(S_t)$ , donde  $A(S_t)$  es el conjunto de acciones disponibles en el estado  $S_t$ . En el instante de tiempo posterior, y en parte como consecuencia de la acción llevada a cabo, el agente recibe una recompensa numérica  $r_{t+1}$  y pasa a estar en un nuevo estado  $S_{t+1}$ .

En cada momento de tiempo el agente lleva a cabo un mapeo entre las representaciones de los estados y las probabilidades de seleccionar cada una de las acciones posibles. Llamamos a este mapeo la política del agente, y la denotamos por  $\pi_t$ , donde  $\pi_t(s, a)$  la probabilidad de que  $a_t = a$  si  $S_t = S$ . Los distintos métodos de aprendizaje por refuerzo especifican de qué manera cambia el agente su política como resultado de la experiencia que va adquiriendo enfrentándose al entorno. El objetivo del agente, expresado sucintamente, es maximizar a largo plazo la suma de las recompensas que obtiene. [10]



**Ilustración II 5: Interfaz Agente-Entorno.**

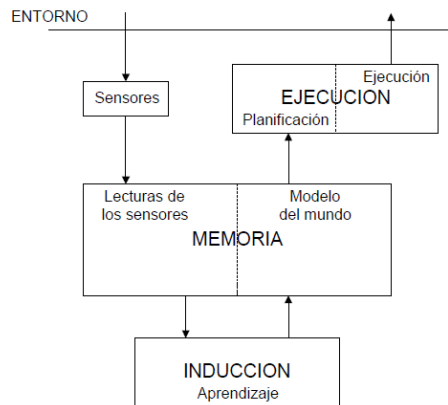
Sutton y Barto [Sutton98] nos menciona, lo mejor de este marco de trabajo es que es extremadamente flexible, lo que permite que sea aplicado a muchos problemas de



maneras muy distintas. En lo que compete a problemas de la abstracción y las diferencias entre las aproximaciones simbólicas y subsimbólicas, tenemos como resultado que con este marco las acciones pueden ser tanto controles directos, por ejemplo los voltajes aplicados a los motores del brazo de un robot, como decisiones de mayor nivel, por ejemplo escoger entre la ruta 1 y la ruta 2. De manera similar, los estados pueden obtenerse de una gran variedad de formas. Pueden estar completamente determinados por percepciones de bajo nivel, como las lecturas directas sobre los sensores, o pueden ser más abstractas y de mayor nivel, como las descripciones simbólicas de los objetos de una habitación, tal y como se pretendían obtener en los inicios de la robótica autónoma. [10] También existe una gran flexibilidad en la manera de diseñar los estados: Un estado puede consistir en las percepciones directas recibidas en el momento actual, o puede estar compuesto en parte por datos almacenados de percepciones anteriores. Incluso podrían existir estados completamente "mentales". [10]

#### **2.1.6. GINKO. Una Arquitectura de Aprendizaje y planificación en mecanismos autónomos**

El sistema GINKO consiste de cuatro componentes principales. Consta de un conjunto de sensores que actualizan continuamente al receptor. Posee una memoria que es dividida conceptualmente en dos partes: un modelo del mundo y las lecturas de los sensores que le dan importancia y soporte a ese modelo. Tiene un componente de inducción que forma un modelo del mundo consistente con los datos. Y finalmente, posee un elemento de ejecución que consiste de un planificador y monitores de planes en ejecución. El elemento de ejecución es responsable de llevar a cabo tareas en el ambiente y es completamente dependiente de los sensores y del modelo del mundo. [11]



**Ilustración II 6: Arquitectura GINKO.**

### **2.1.6.1. Aprendizaje**

Para cada par operador/sensor se particiona el espacio de configuración en regiones de comportamiento cualitativo uniforme. Sobre cada región, el efecto de ejecutar ese operador es incrementar, decrementar, o dejar sin cambios, el valor del sensor. Luego se combinan las particiones asociadas a cada operador para formar una representación compuesta de los efectos de ese operador. Los operadores se dividen en regiones y cada región de comportamiento cualitativo uniforme de cada partición de un operador es una región característica de ese operador. [11]

Cada región característica predice el cambio cualitativo en los valores del sensor que resultarán de ejecutar el operador asociado a esa región. [11]

### **2.1.6.2. Planificación**

El problema de la planificación consiste en encontrar una secuencia de operadores que, cuando se ejecuten, lleven al robot de una configuración inicial (generalmente la configuración actual) a alguna configuración que satisfaga el objetivo propuesto. El planificador del GINKO usa directamente las particiones de los operadores para construir sus planes. El planificador utiliza un algoritmo convencional de búsqueda encadenada hacia atrás desde la región objetivo hacia la configuración inicial. [11]

### **2.1.6.3. Ejecución y monitoreo**

Los planes generados por el planificador de GINKO son abstractos en el sentido de que son cualitativos e imprecisos. Un plan concreto es generado por asociación de monitores de ejecución con los operadores programados. El efecto principal de estos monitores es usar los sensores para detectar sucesos inesperados o fallas, y volver a planificar localmente cuando sea necesario. [11]

Un plan no tiene garantía de tener éxito ya que los efectos de los operadores son capturados sólo cualitativamente y, en general, pueden ser caracterizados incorrectamente. Podría parecer que un solo operador es efectivo para alcanzar un objetivo cuando en realidad se necesita una interpolación de varios operadores para dirigirse al objetivo, para corregir desviaciones cuantitativas de la trayectoria deseada.

Es importante saber que las particiones de los operadores son el resultado de la generalización de la experiencia y son sólo una aproximación. Las inconsistencias pueden introducir errores en el plan. En dichos casos, los bordes de las regiones deben redefinirse y se debe generar un plan más preciso. Se pueden hacer refinamientos de las regiones características inmediatamente o postergarlas por ejemplo hasta la falla del plan en ejecución. El refinamiento puede ser incremental o un reparticionamiento total del espacio de configuración. [11]

### **2.1.6.4. Control**

El elemento de ejecución es responsable de la selección y ejecución de los operadores. En general, el componente de aprendizaje es activado por el elemento de ejecución ante una falla en la ejecución del plan si la estrategia de refinamiento es pasiva, o puede ser activado por el proceso de integración de datos asociado con el almacenamiento de las lecturas de los sensores si la estrategia de refinamiento es activa. [11]

### 2.1.7. Mecanismo Lego Mindstorms 2.0

Lego Mindstorms es un juego de robótica fabricado por la empresa Lego, el cual posee elementos básicos de las teorías robóticas, como la unión de piezas y la programación de acciones, en forma interactiva. Este robot fue comercializado por primera vez en septiembre de 1998. [12]



**Ilustración II 7: Arquitectura GINKO.**

Comercialmente se publicita como “Robotic Invention System”, en español Sistema de Invención Robotizado (RIS). También se vende como herramienta educacional, lo que originalmente se pensó en una sociedad entre Lego y el MIT. La versión educativa se llama «Lego Mindstorms for Schools», en español Lego Mindstorms para la escuela y viene con un software de programación basado en la GUI de Robolab. [12]

Lego Mindstorms puede ser usado para construir un modelo de sistema integrado con partes electromecánicas controladas por computador. Prácticamente todo puede ser representado con las piezas tal como en la vida real, como un elevador o robots industriales. [12]

#### 2.1.7.1. Historia

Lego Mindstorms fue uno de los resultados de la fructífera colaboración entre Lego y el MIT. Esta asociación se emplea como ejemplo de relación entre la industria y la investigación académica que resulta muy beneficiosa para ambos socios [12].

### **2.1.7.1.1. Colaboración con el grupo de epistemología y aprendizaje del MIT**

La línea Lego Mindstorms nació en una época difícil para Lego, a partir de un acuerdo entre Lego y el MIT. Según este trato, Lego financiaría investigaciones del grupo de epistemología y aprendizaje del MIT sobre cómo aprenden los niños y a cambio obtendría nuevas ideas para sus productos, que podría lanzar al mercado sin tener que pagar regalías al MIT. Un fruto de esta colaboración fue el desarrollo del MIT Programmable Brick (Ladrillo programable). [12]

El mentor del grupo, Seymour Papert, era un matemático interesado desde la década de 1960 por la relación entre la ciencia, la adquisición del conocimiento y el desarrollo de la mente infantil. De hecho, el nombre del producto, Mindstorms, proviene del título de un libro suyo, llamado MindStorms: Children, Computers, and Powerful Ideas, en el que describe sus ideas respecto al empleo de las computadoras como impulsoras del aprendizaje. Papert, uno de los creadores de lenguaje de programación Logo, ampliamente empleado como herramienta para enseñar programación, toma de Jean Piaget la concepción de niño como “constructor de sus propias estructuras mentales”. Es partidario del construccionismo, tesis que sostiene que el niño crea su conocimiento de forma activa y que la educación debe de facilitarle herramientas para realizar actividades que impulsen esta actividad.<sup>4</sup> La lectura de su libro fue lo que impulsó al presidente de Lego a contactar en 1985 con el MIT, pues le hizo pensar que ambos grupos tenían ideas similares sobre el aprendizaje infantil. [12]

*“El aprender mejor no vendrá de ofrecer las mejores herramientas para que el profesor instruya, sino de dar las mejores oportunidades a los estudiantes para construir.”*

*Seymour Papert*

El grupo de epistemología y aprendizaje del MIT, dirigido por Mitchel Resnick, que a su vez había sido pupilo de Papert, estaba profundamente influido por el constructivismo de Piaget, extendido por el propio Papert bajo la denominación de construccionismo. Según esta perspectiva, en lugar de instruir al estudiante proporcionándole fórmulas y técnicas

(instruccionismo), es mejor potenciar el aprendizaje creando un entorno en el que los estudiantes puedan desempeñar actividades propias de ingenieros o inventores como vía para acceder a los principios fundamentales de la ciencia y la técnica; pues de esta forma es como se desarrolla la forma de pensar propia de los científicos, los estudiantes se interesan realmente en su trabajo y motu proprio tratan de informarse para resolver los problemas que van encontrando. Así que se concentraron, en palabras de Resnick, en “diseñar cosas que permitan a los estudiantes diseñar cosas”. [12]

### **2.1.7.1.2. Antecedentes del bloque programable**

La línea Mindstorms no fue el primer fruto de la relación entre Lego y el MIT, aunque sí el más exitoso. Con anterioridad, Lego se había interesado por el Lenguaje de programación Logo, Fruto de este interés nació en 1986 Lego TC Logo, creado por Resnick y Steve Ocko. Lego TC Logo era un sistema en el que se programaba en una computadora que estaba conectada por un cable a una construcción Lego que contaba con motores, luces y sensores. Aunque alcanzó un relativo éxito comercial, según Resnick el sistema “imponía restricciones tanto físicas como imaginativas”.

Esta línea de desarrollo continuaría en 1993 con el lanzamiento de Control Lab, de software mejorado. [12]

El paso de programar una computadora que se conectaba a una construcción Lego a programar un bloque de esa construcción era una idea natural que se estudió durante largo tiempo. Desde principios de los años 90 se empezó a investigar esta posibilidad. Sin embargo el proyecto tuvo que esperar a que el mercado fuera propicio. Por una parte, el coste de la tecnología era demasiado alto en un principio. Por otra, el bloque se programaría desde un computador, y por esas fechas los computadores no estaban tan extendidos como lo estaban ocho años más tarde, lo que afectaría negativamente a la demanda. Hubo que esperar un lustro hasta que las condiciones eran las apropiadas y decidieran empezar seriamente el desarrollo de lo que acabaría siendo el bloque RCX, un bloque de Lego que contaba con un microcontrolador, y que constituye el corazón del producto Mindstorms. De esta forma las construcciones Lego pasaban de ser estructuras estáticas a máquinas dinámicas que interactúan con el mundo. Por otra parte, mientras

que en muchos casos los productos Lego proporcionaban las piezas necesarias para construir algo con un objetivo fijo, como un tren o un puente, lo que permite “aprender haciendo”, en el desarrollo del nuevo bloque se siguió en cambio la filosofía de Papert y Resnick de fomentar el “aprender diseñando”, y tratar de dejar más abiertas las posibilidades. [12]

#### **2.1.7.2. Microcontrolador**

El microcontrolador que posee es un ARM7 de 32 bits, que incluye 256 Kb de memoria Flash y 64 Kb de RAM externa, la cual a diferencia del bloque RCX, posee mayores capacidades de ejecución de programas, evitando que los procesos inherentes de varios paquetes de datos colisionen y produzcan errores y un posible error en la ejecución del software. Su presentación es similar al Hitachi H8 ya que se encuentra en el circuito impreso del bloque, junto a la memoria FLASH. [12]

#### **2.1.7.3. Entradas y salidas**

En el bloque de NXT existen cuatro entradas para los sensores, pero los conectores son distintos de los del RCX, lo que impide la conexión de sus motores o sensores, sin embargo, el kit de NXT incluye el adaptador para que los sensores de RCX sean compatibles con NXT. [12]

Las salidas de energía aún son tres localizadas en la parte posterior del bloque, haciendo que la conexión para los motores y partes móviles sean de más fácil acceso. [12]

#### **2.1.7.4. Comunicaciones**

El bloque de NXT puede comunicarse con el computador mediante la interfaz de USB que posee, la cual ya viene en la versión 2.0. Además, para comunicarse con otros robots en las cercanías posee una interfaz Bluetooth que es compatible con la Clase II v 2.0. Esta conectividad con Bluetooth no sólo permite conectarse con otros bloques, sino también con computadores, palms, teléfonos móviles, y otros aparatos con esta interfaz de comunicación. [12]

Dentro de las posibilidades de conexión se encuentran.

- Conectar hasta tres dispositivos distintos.
- Buscar y conectarse a otros dispositivos que posean Bluetooth.
- Recordar dispositivos con los cuales se ha conectado anteriormente para conectarse más rápidamente,
- Establecer el bloque NXT como visible lo invisible para el resto de los dispositivos.

#### **2.1.7.5. Firmware**

El firmware del Lego Mindstorms consta de las instrucciones básicas que posee el bloque para hacer las distintas tareas que se le pueden programar en el bloque RCX. El firmware viene en el CD-ROM que se adjunta en el empaque original y debe ser cargado todas las veces que el robot se inicialice o se cambien las baterías y la memoria se borra. [12]

Si no se carga el firmware, el robot queda en modo de arranque, lo cual hace que se pueda jugar con un programa que viene en forma nativa dentro del robot. Para cargar el firmware debe ejecutarse el programa adjunto y luego esperar cerca de 3 minutos para que se cargue completamente el firmware básico. [12]

Las versiones más actuales de Lego Mindstorms RCX, como la versión 2.0, es compatible con las versiones anteriores del bloque, haciendo que los programas escritos en versiones más nuevas también puedan ser ejecutadas en las generaciones previas.

#### **2.1.7.6. Motores**

Los motores de la serie Lego Robotics han sido de tres tipos, los cuales son independientes al bloque, lo que entrega movilidad al sistema dinámico según las necesidades de construcción. [12]

En la tabla de medición, el motor estándar es más veloz que el de 9 volts, pero este último posee más fuerza para mover el robot, ya que pueden levantar cerca de 240 piezas de 8x8,



pero es más lento y a la vez más preciso. El motor Micro es sólo para funciones menores debido a su escaso torque y la mínima velocidad de rotación.

Los motores desmontables son alimentados mediante cables que poseen conductores eléctricos que transmiten la energía a los inductores. Como son motores paso a paso, el sentido de conexión no entrega la misma dirección de movimiento. [12]

Los motores integrados al bloque son menos versátiles, pero no dependen de conexiones externas, lo cual ayuda visualmente al robot en su presentación. [12]

El modelo NXT usa servo motores, los cuales permiten la detección de giros de la rueda, indicando los giros completos o medios giros, que es controlado por el software.

Motor	Velocidad normal (RPM)	Torque (kg/cm)	Velocidad estándar (RPM)
Estándar	3240	1,760	40
9 voltios	370	3.840	15
Micro	36	0,128	36

**Ilustración II 8: Características de los motores.**



**Ilustración II 9: Servomotor NXT.**

### **2.1.7.7. Sensores**

Un sensor es un dispositivo que está capacitado para detectar acciones o estímulos externos y responder en consecuencia. Estos aparatos pueden transformar las magnitudes físicas o químicas en magnitudes eléctricas. [12]

#### **2.1.7.7.1. Sensor ultrasónico**

El sensor Ultrasónico sólo se incluye en el empaque de Lego Mindstorms NXT, y su principal función detectar las distancias y el movimiento de un objeto que se interponga en el camino del robot, mediante el principio de la detección ultrasónica. Este sensor es capaz de detectar objetos que se encuentren desde 0 a 255 cm.

Mediante el principio del eco, el sensor es capaz de recibir la información de los distintos objetos que se encuentren en el campo de detección. El sensor funciona mejor cuando las señales ultrasónicas que recibe, provienen de objetos que sean grandes, planos o de superficies duras. Los objetos pequeños, curvos o suaves, como pelotas, pueden ser muy difíciles de detectar. Si en el cuarto se encuentra más de un sensor ultrasónico, los dispositivos pueden interferir entre ellos, resultando en detecciones pobres. [12]



**Ilustración II 10: Sensor Ultrasónico.**

### 2.1.7.7.2. Sensor Detector de color

Este tipo de sensor incluido en el conjunto de NXT, tiene la capacidad de detectar una amplia gama de colores y transformarle en un valor numérico en cual en programación puede ser usado y calibrado de acuerdo a la necesidad. [13]



**Ilustración II 11: Sensor de color.**



**Ilustración II 12: Gama de colores.**

### 2.1.7.8. Programación

La programación del Lego Mindstorms se realiza mediante el software que se adjunta en el empaque original, el cual trae el firmware del robot y un programa que emula un árbol de decisiones, para los cuales, el usuario debe programar las acciones a seguir por el robot. El software se encuentra dividido por cada tipo de robot que se puede construir, y que viene recomendado en el empaque. [12]

Una de las principales características de este software de programación, es su entorno visual, el cual emula la construcción por bloques, dando la posibilidad a cualquier usuario aprendiz acostumbrarse rápidamente a la programación de bloque. [12]

Este lenguaje permite las instrucciones secuenciales, instrucciones de ciclos e instrucciones de decisiones, éstas últimas, basadas en los datos reportados por los sensores que se puede añadir al robot. [12]

### **2.1.7.9. Lenguajes de programación**

El bloque del Lego Mindstorms como un producto de hardware y software integrado, puede ser programado con varias interfaces, pero todos logrando el mismo fin. Esto se puede realizar mediante la torre de comunicación y utilizando las herramientas correctas para poder acceder al firmware básico de Lego. [12]

Algunos de estos tipos de lenguajes son licenciados como las herramientas de visual studio .net o robot C y también los lenguajes de código libre pueden ser usados para programar como Java de netbeans, eclipse y C. [12]

### **2.1.8. Lenguaje de programación C# (Visual Studio.net)**

#### **2.1.8.1. Introducción**

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común. [14]

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre C Sharp fue inspirado por la notación musical, donde '#' (sostenido, en inglés sharp) indica que la nota (C es la nota do en inglés) es un semitono más alta, sugiriendo que C# es superior a C/C++. Además, el signo '#' se compone de cuatro signos '+' pegados.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux. [14]

### **2.1.8.2. Historia**

Durante el desarrollo de la plataforma .NET, las bibliotecas de clases fueron escritas originalmente usando un sistema de código gestionado llamado Simple Managed C (SMC). En enero de 1999, Anders Hejlsberg formó un equipo con la misión de desarrollar un nuevo lenguaje de programación llamado Cool (Lenguaje C orientado a objetos). Este nombre tuvo que ser cambiado debido a problemas de marca, pasando a llamarse C#. La biblioteca de clases de la plataforma .NET fue migrada entonces al nuevo lenguaje. [14]

Hejlsberg lideró el proyecto de desarrollo de C#. Anteriormente, ya había participado en el desarrollo de otros lenguajes como Turbo Pascal, J++ y Embarcadero Delphi.

### **2.1.8.3. Tipos básico de datos**

Los tipos de datos básicos son ciertos tipos de datos tan comúnmente utilizados en la escritura de aplicaciones que en C# se ha incluido una sintaxis especial para tratarlos.

Por ejemplo, para representar números enteros de 32 bits con signo se utiliza el tipo de dato System.Int32 definido en la BCL, aunque a la hora de crear un objeto a de este tipo que represente el valor 2 se usa la siguiente sintaxis:

```
System.Int32 a = 2;
```

Como se ve, no se utiliza el operador new para crear objeto System.Int32, sino que directamente se indica el literal que representa el valor a crear, con lo que la sintaxis necesaria para crear entero de este tipo se reduce considerablemente. Es más, dado lo frecuente que es el uso de este tipo también se ha predefinido en C# el alias int para el mismo, por lo que la definición de variable anterior queda así de compacta:

```
int a = 2;
```

System.Int32 no es el único tipo de dato básico incluido en C#. En el espacio de nombres System se han incluido todos estos:

Tipo de datos de enteros				
Tipo	Equivalente BCL	Tamaño	Rango	Significado
byte	System.Byte	8-bit (1-byte)	0 a 255	Entero sin signo
sbyte	System.SByte	8-bit (1-byte)	-128 a 127	Entero con signo
short	System.Int16	16-bit (2-byte)	-32.768 a 32.767	Entero corto con signo
ushort	System.UInt16	16-bit (2-byte)	0 a 65.535	Entero corto sin signo
int	System.Int32	32-bit (4-byte)	-2.147.483.648 a 2.147.483.647	Entero medio con signo
uint	System.UInt32	32-bit (4-byte)	0 a 4.294.967.295	Entero medio sin signo
long	System.Int64	64-bit (8-byte)	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Entero largo con signo
ulong	System.UInt64	64-bit (8-byte)	0 a 18.446.744.073.709.551.615	Entero largo sin signo

**Ilustración II 13: Tipo de datos enteros.**

Los tipos de coma flotante pueden representar números con componentes fraccionales. Existen dos clases de tipos de coma flotante: float y double. El tipo double es el más utilizado porque muchas funciones matemáticas de la biblioteca de clases de C# usan valores double. Quizá, el tipo de coma flotante más interesante de C# es decimal, dirigido al uso de cálculos monetarios. La aritmética de coma flotante normal está sujeta a una variedad de errores de redondeo cuando se aplica a valores decimales. El tipo decimal elimina estos errores y puede representar hasta 28 lugares decimales [14].

Tipo de datos de coma flotante				
Tipo	Equivalente BCL	Tamaño	Rango	Significado
float	System.Single	32-bit (4-byte)	±1.401298E-45 a ±3.402823E+38	Coma flotante corto
double	System.Double	64-bit (8-byte)	±4.94065645841246E-324 a ±1.79769313486232E+308	Coma flotante largo
decimal	System.Decimal	128-bit (16-byte)	-7.9228162514264337593543950335 a +7.9228162514264337593543950335	Coma flotante monetario

**Ilustración II 14: Tipo de datos flotantes.**

#### **2.1.8.4. Metas del diseño del lenguaje [14]**

El estándar ECMA-334 lista las siguientes metas en el diseño para C#:

- Lenguaje de programación orientado a objetos simple, moderno y de propósito general.
- Inclusión de principios de ingeniería de software tales como revisión estricta de los tipos de datos, revisión de límites de vectores, detección de intentos de usar variables no inicializadas, y recolección de basura automática.
- Capacidad para desarrollar componentes de software que se puedan usar en ambientes distribuidos.
- Portabilidad del código fuente.
- Fácil migración del programador al nuevo lenguaje, especialmente para programadores familiarizados con C, C++ y Java.
- Soporte para internacionalización.
- Adecuación para escribir aplicaciones de cualquier tamaño: desde las más grandes y sofisticadas como sistemas operativos hasta las más pequeñas funciones.
- Aplicaciones económicas en cuanto a memoria y procesado.

#### **2.1.9. El Modelo de Gestión SCRUM**

##### **2.1.9.1. Origen.**

Scrum es una metodología ágil de desarrollo de proyectos que toma su nombre y principios de los estudios realizados sobre nuevas prácticas de producción por Hirotaka Takeuchi e Ikujiro Nonaka a mediados de los 80.

Aunque surgió como modelo para el desarrollo de productos tecnológicos, también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas de software. En el desarrollo de software scrum está considerado como modelo ágil por la Agile Alliance.

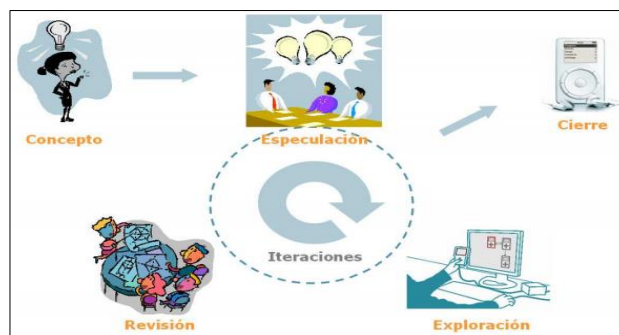
### 2.1.9.2. Introducción al modelo

Scrum es una metodología de desarrollo muy simple, que requiere trabajo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto.

Scrum es una metodología ágil, y como tal:

Es un modo de desarrollo de carácter adaptable más que predictivo.

- Orientado a las personas más que a los procesos.
- Emplea la estructura de desarrollo ágil: incremental basada en iteraciones y revisiones.



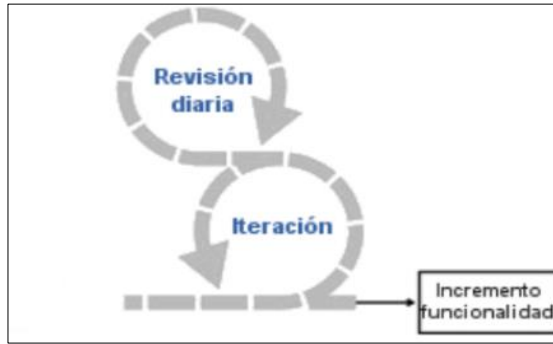
**Ilustración II 15: Estructura del desarrollo ágil.**

Se comienza con la visión general del producto, especificando y dando detalle a las funcionalidades o partes que tienen mayor prioridad de desarrollo y que pueden llevarse a cabo en un periodo de tiempo breve (normalmente de 30 días).

Cada uno de estos periodos de desarrollo es una iteración que finaliza con la producción de un incremento operativo del producto.

Estas iteraciones son la base del desarrollo ágil, y Scrum gestiona su evolución a través de reuniones breves diarias en las que todo el equipo revisa el trabajo realizado el día anterior y el previsto para el día siguiente.





**Ilustración II 16: Estructura central de Scrum.**

- **Control de la evolución del proyecto**

Scrum controla de forma empírica y adaptable la evolución del proyecto, empleando las siguientes prácticas de la gestión ágil:

- **Revisión de las Iteraciones**

Al finalizar cada iteración (normalmente 30 días) se lleva a cabo una revisión con todas las personas implicadas en el proyecto. Este es el periodo máximo que se tarda en reconducir una desviación en el proyecto o en las circunstancias del producto.

- **Desarrollo incremental**

Durante el proyecto, las personas implicadas no trabajan con diseños o abstracciones. El desarrollo incremental implica que al final de cada iteración se dispone de una parte del producto operativa que se puede inspeccionar y evaluar.

- **Desarrollo evolutivo**

Los modelos de gestión ágil se emplean para trabajar en entornos de incertidumbre e inestabilidad de requisitos.

Intentar predecir en las fases iniciales cómo será el producto final, y sobre dicha predicción desarrollar el diseño y la arquitectura del producto no es realista, porque las circunstancias obligarán a remodelarlo muchas veces.

Para qué predecir los estados finales de la arquitectura o del diseño si van a estar cambiando. En Scrum se toma a la inestabilidad como una premisa, y se adoptan técnicas de trabajo para permitir esa evolución sin degradar la calidad de la arquitectura que se irá generando durante el desarrollo.

El desarrollo Scrum va generando el diseño y la arquitectura final de forma evolutiva durante todo el proyecto. No los considera como productos que deban realizarse en la primera “fase” del proyecto.

(El desarrollo ágil no es un desarrollo en fases)

- **Auto-organización**

Durante el desarrollo de un proyecto son muchos los factores impredecibles que surgen en todas las áreas y niveles. La gestión predictiva confía la responsabilidad de su resolución al gestor de proyectos.

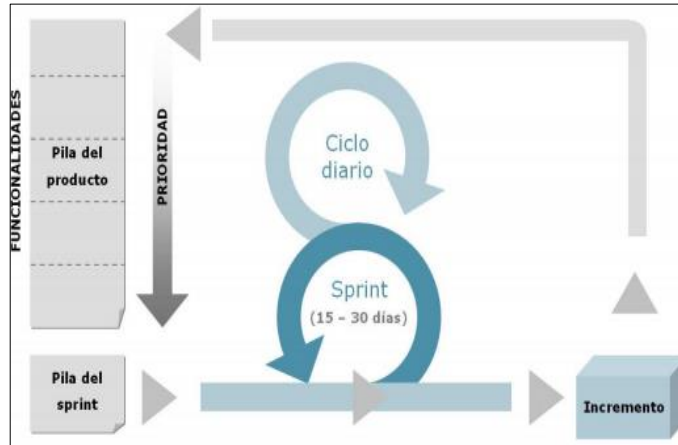
En Scrum los equipos son auto-organizados (no auto-dirigidos), con margen de decisión suficiente para tomar las decisiones que consideren oportunas.

- **Colaboración**

Las prácticas y el entorno de trabajo ágiles facilitan la colaboración del equipo. Ésta es necesaria, porque para que funcione la auto-organización como un control eficaz cada miembro del equipo debe colaborar de forma abierta con los demás, según sus capacidades y no según su rol o su puesto.

- **Visión general del proceso**

Scrum denomina “sprint” a cada iteración de desarrollo y recomienda realizarlas con duraciones de 30 días. El sprint es por tanto el núcleo central que proporciona la base de desarrollo iterativo e incremental.



**Ilustración II 17: Ciclos SCRUM.**

Los elementos que conforman el desarrollo Scrum son:

➤ **Las reuniones**

- Planificación de sprint: Jornada de trabajo previa al inicio de cada sprint en la que se determina cuál va a ser el trabajo y los objetivos que se deben cumplir en esa iteración.
- Reunión diaria: Breve revisión del equipo del trabajo realizado hasta la fecha y la previsión para el día siguiente.
- Revisión de sprint: Análisis y revisión del incremento generado.

➤ **Los elementos**

- Pila del producto: lista de requisitos de usuario que se origina con la visión inicial del producto y va creciendo y evolucionando durante el desarrollo.
- Pila del sprint: Lista de los trabajos que debe realizar el equipo durante el sprint para generar el incremento previsto.
- Incremento: Resultado de cada sprint.

## **CAPÍTULO III:**

### **ANÁLISIS DE LOS ALGORITMOS HEURÍSTICOS APLICADO AL APRENDIZAJE Y GUÍA DE SISTEMAS AUTÓNOMOS.**

#### **3.1. Análisis heurístico**

A continuación se detalla el análisis de los algoritmos heurísticos ya descritos anteriormente.

#### **3.2. Introducción.**

La constante evolución de la tecnología autónoma en dispositivos mecánicos ha llevado a que la inteligencia artificial constituya uno de los aspectos más importantes; es fundamental que los agentes (mecanismos autónomos) controlados por la computadora se comporten en forma inteligente. Un problema característico es la navegación, que consiste en determinar el camino más conveniente entre una posición inicial y una posición de destino. Si bien el planteo del problema es sencillo, el mismo está lejos de ser trivial debido a la creciente complejidad de los entornos de trabajo y los requerimientos a los que están sometidos dichos dispositivos.

La búsqueda es una de las técnicas más utilizadas para resolver los problemas de encontrar caminos o planificación que se presentan en la inteligencia artificial en los juegos de vídeo, laberintos, seguidores de líneas etc. En particular, la búsqueda es utilizada para resolver el problema de la navegación. De los distintos tipos de algoritmos de búsqueda,

los algoritmos de búsqueda heurística completa se encuentran ampliamente difundidos. Sin dudas, el algoritmo backtraning, es el algoritmo de búsqueda más adecuado como alternativa heurística en la aplicación para el desarrollo de software de guía y aprendizaje automático frente a los métodos clásicos e impredecibles.

Los algoritmos heurísticos tradicionales muestran limitaciones importantes cuando el espacio de búsqueda es demasiado grande o existen factores dinámicos. En la navegación, de los diferentes entornos en tiempo real, este problema aparece cuando las rutas a determinar son muy largas, el terreno es modificable o existen muchos objetos móviles. Bajo esas condiciones, los algoritmos de búsqueda básicos no pueden responder en el tiempo requerido y resultan inadecuados. De esta forma, surge la necesidad de desarrollar nuevas estrategias de búsqueda, que se adapten a los requerimientos de tiempo real en la automatización del aprendizaje en mecanismos y que resuelvan adecuadamente caminos en condiciones de incertidumbre sobre terrenos de gran extensión.

### **3.3. Determinación de parámetros que conlleve al análisis heurístico para la guía de un mecanismo autónomo.**

#### **3.3.1. Descripción inicial.**

Un mecanismo autónomo tiene la facultad de movilizarse automáticamente por cualquier escenario siempre y cuando el diseño de su hardware lo permita, un sistema de software que permita guiarlo es una parte importante del sistema, dicho sistema cuenta con un análisis, el cual medirá los diferentes parámetros que permitirá seleccionar el mejor algoritmo para ser parte de la arquitectura del código que será instalado en el mecanismo autónomo.

#### **3.3.2. Eficiencia.**

Es la capacidad que tiene el software para hacer buen uso de los recursos que manipula, en el sistema desarrollado la eficiencia se mide usando el número de nodos visitados y tomando en cuenta el tiempo que se demora en su resolución. El sistema desarrollado

Laberinto Virtual”, está diseñado de tal forma que genera diferentes escenarios aleatoriamente teniendo en cuenta los parámetros de ingreso como son la dificultad y la velocidad, estos como valores numéricos y como parámetro de aplicación el tipo de algoritmo que se va a usar.

### 3.3.2.1. Numero de nodos visitados.

Es el número de nodos visitados el cual cuenta los validos (Nodos parte del camino respuesta) y los nodos no validos (Nodos que no forma parte de camino respuesta).

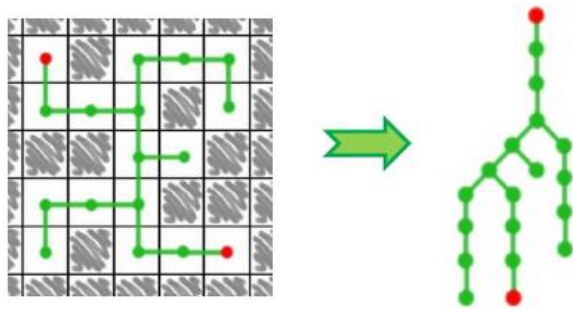


Ilustración III 1: De laberinto a nodos.

### 3.3.2.2. Camino valido (Numero que conforma el camino valido).

El número de nodos que son parte del camino solución, generados después de aplicar los algoritmos de búsqueda, en el método de la mano derecha (Right hand) no se puede llevarse a cabo un aprendizaje del camino, porque la estructura del algoritmo en si, no permite obtener una manera de memorizar la información de los nodos visitados.

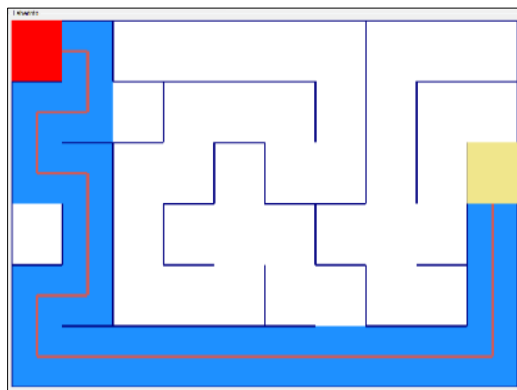


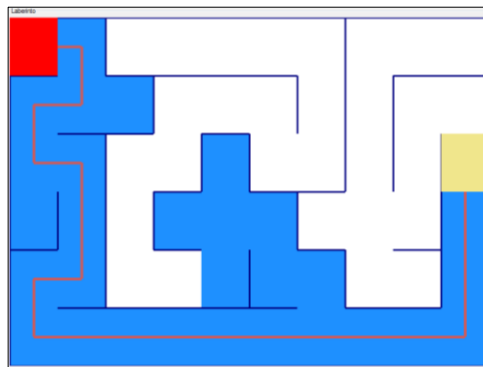
Ilustración III 2: Camino valido.

### 3.3.2.3. **Nodos resultado(Relación nodos visitados con nodos respuesta)**

Es el porcentaje que se genera al dividir los nodos camino por los nodos totales obteniendo un coeficiente parcial el cual es multiplicado por el cien por ciento de los nodos respuesta.

**Nodos totales**= $\sum Nodos i=100\%$ .

**Nodos resultado** = (Nodos Camino/ Nodos totales) \*100%.



**Ilustración III 3: Eficiencia recorrido.**

### 3.3.3. **Tiempo.**

Es el índice que mide en segundos la resolución de un laberinto.

#### 3.3.3.1. **Tiempo individual**

El tiempo de respuesta se genera una vez que el sistema busque una salida independientemente del algoritmo usado, en un menor índice de tiempo, hay mejor calidad del algoritmo o método usado.

#### 3.3.3.2. **Tiempo eficiente.**

Es el tiempo que encuentra el camino, pero sin realizar búsqueda alguna solo siguiendo la ruta que el sistema aprendió.

### 3.3.4. Aprendizaje (Inteligencia)

Como ya se vio en una descripción anterior la inteligencia es un rasgo comúnmente asociado con la capacidad de adquirir nuevos conocimientos. De esta forma, un sistema inteligente autónomo puede definirse, como aquél capaz de descubrir y registrar si una acción efectuada sobre una situación dada fue beneficiosa para lograr su objetivo, para nuestro análisis la forma de registrar si hay un adecuado aprendizaje será reconocer el camino respuesta, después de finalizada la guía y de haber sido encontrado el camino solución en el laberinto definido.

### 3.3.5. Administración de dispositivos (Uso del CPU).

Es el uso que el algoritmo hace de la CPU para cumplir con la meta de hallar el camino resultado.

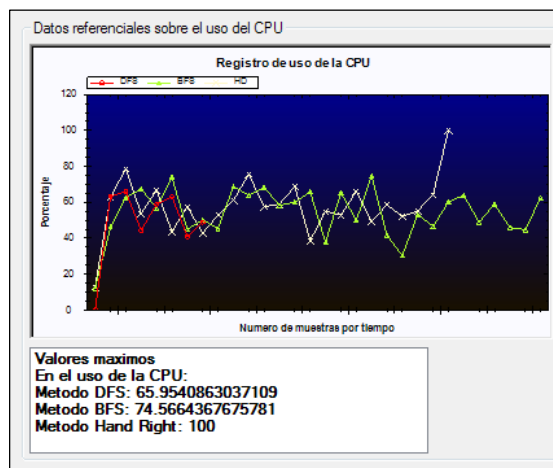


Ilustración III 4: Uso de la CPU.

#### 3.3.5.1. Valores máximos CPU.

Son los valores máximos que alcanza la CPU en toda la trayectoria de tiempo que se demora el sistema en encontrar el camino.



### 3.3.5.2. Valores promedios.

Una vez finalizada la búsqueda del camino se toma como referencia los valores promedios del uso de la CPU.

### 3.4. Representación de parámetros para el análisis de algoritmos heurísticos.

Basados en las necesidades de construir un software flexible, que le permita guiarse en diferentes escenarios hasta encontrar la salida y que aprenda como encontrar el camino. Estas necesidades se traducen en un conjunto de parámetros e índices descritos a continuación.

Tabla III 1: Parámetros algoritmos para la resolución de laberintos.

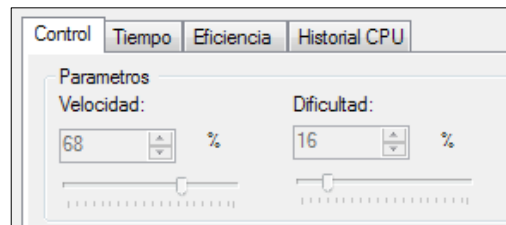
CLASIFICACION	PARAMETROS	INDICADORES
Calidad	Eficiencia	• Numero de nodos visitados
		• Numero de nodos respuesta
		• Nodos resultado (Relación nodos visitados con nodos respuesta)
	Tiempo	• Tiempo Individual
• Tiempo exacto		
Comportamiento real	Inteligencia	• Aprendizaje
Administración de recursos	Uso de la CPU	• Uso de la CPU individual
		• Promedio de uso total de la CPU.

### 3.5. Apuntes del experimento.

Para guiar y poder hacer que de esta guía aprenda un mecanismo autónomo necesitamos crear un escenario versátil y que pueda generarse de una manera aleatoria, El sistema

“Laberinto Virtual” está diseñado para crear escenarios virtuales y recorrerlos de una forma artificial simulando la guía que genera un software a una maquina autónoma.

La base del sistema está en la generación de los laberintos y la resolución de ellos, para esta parte es importante conocer el funcionamiento de la primera parte del sistema, que son los parámetros de generación y resolución, los cuales están configurados de una forma de generarlos en porcentajes de trabajos.



**Ilustración III 5: Configuración y resolución de laberintos (Laberintos virtuales).**

### 3.5.1. Inicialización del experimento.

El experimento esta descrito en un conjunto de siete experimentos individuales los cuales están descritos en la tabla a continuación.

**Tabla III 2: Datos experimentales.**

Experimento	1	2	3	4	5	6	7
Velocidad	50%	50%	50%	50%	50%	50%	50%
Dificultad	1%	10%	35%	45%	55%	65%	85%

Cada uno de los experimentos contempla los tres métodos de resolución, los métodos de backtracking (DSF y BSF) y el método de la mano derecha (Right Hand), los datos tabulados y procesados son almacenados en un registro temporal el cual registra una estadística al final de la aplicación.

### 3.5.2. Generación y resolución de laberintos.

#### 1. Experimento 1

Laberinto: Dificultad 1%, Tiempo al 50%.

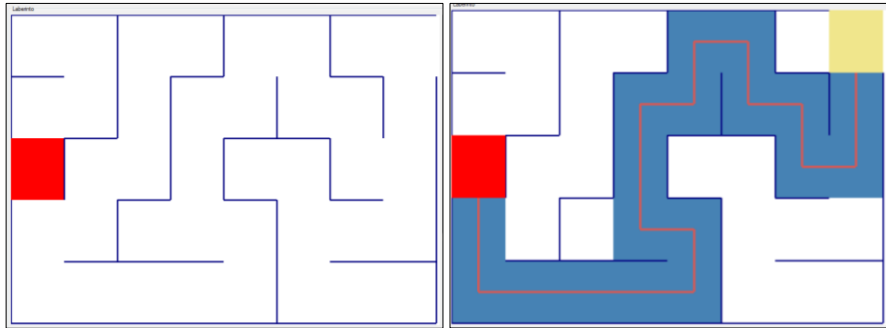


Ilustración III 6: Laberinto y camino encontrado Exp 1.

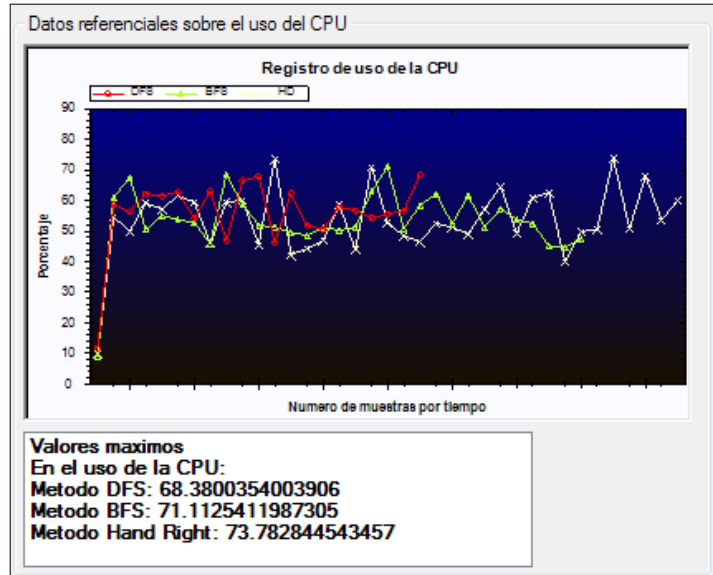


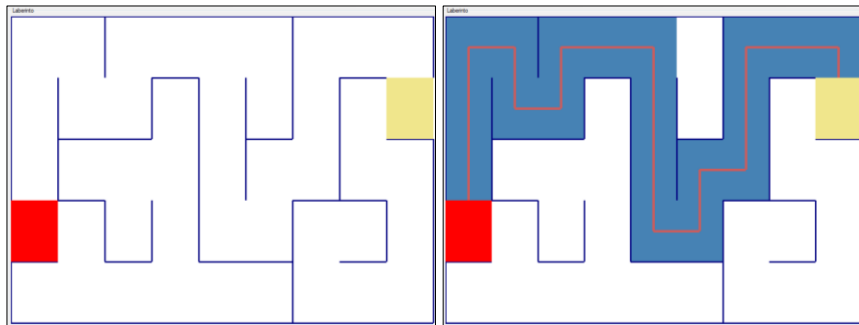
Ilustración III 7: Uso de la CPU Exp 1.

#EXP	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	TPer(s)	Vel(%)	Dif(%)
1	DFS	31	40	20	3.119	2.069	50	1
1	BFS	32	40	20	2.781	2.069	50	1
1	Hand right	28	40	0	3.559	No Ap	50	1

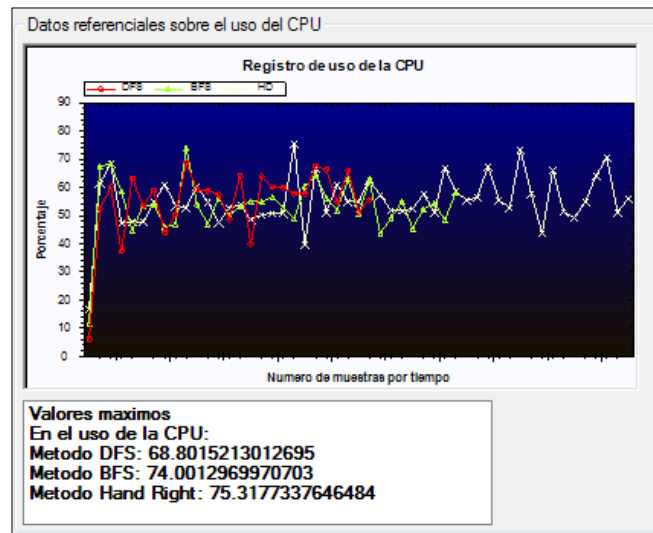
Ilustración III 8: Resultados parciales Exp 1.

## 2. Experimento 2

**Laberinto: Dificultad 10%, Tiempo al 50%.**



**Ilustración III 9: Laberinto y camino encontrado Exp 2.**



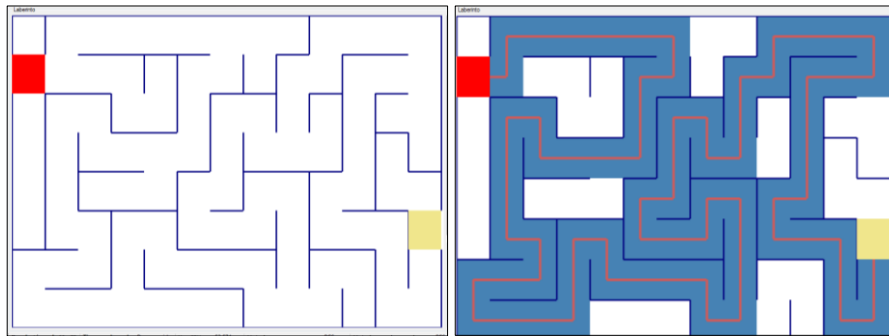
**Ilustración III 10: Uso de la CPU Exp 2.**

#	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	Vel(%)	Dif(%)	Efec	CPU%
1	BFS	36	45	21	3.324	50	10	58	53.396
2	DFS	36	45	21	4.088	50	10	58	54.969
3	Hand Right	34	45	0	4.850	50	10	61	55.333

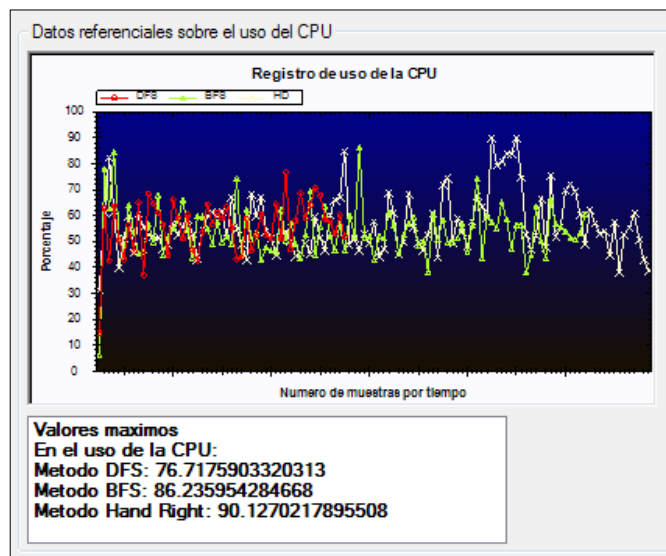
**Ilustración III 11: Resultados parciales Exp 2.**

### 3. Experimento 3

**Laberinto: Dificultad 35%, Tiempo al 50%.**



**Ilustración III 12: Laberinto y camino encontrado Exp 3.**



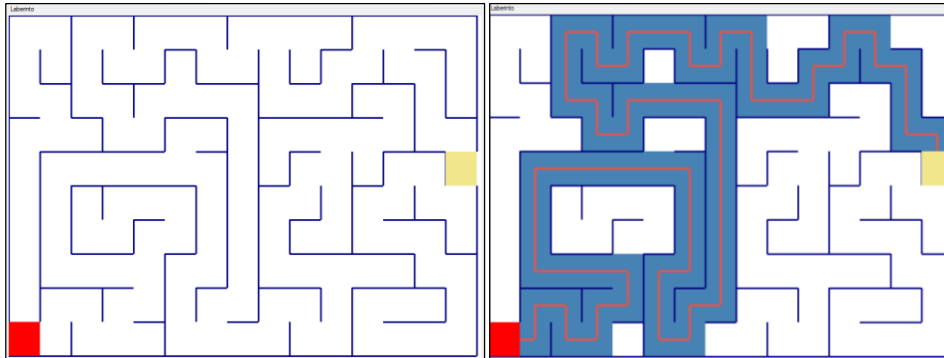
**Ilustración III 13: Uso de la CPU Exp 3.**

#	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	Vel(%)	Dif(%)	Efec	CPU%
1	DFS	90	104	73	8.477	50	35	81	55.552
2	Hand Right	91	104	0	12.211	50	35	80	56.92
3	BFS	101	104	73	10.172	50	35	72	53.938

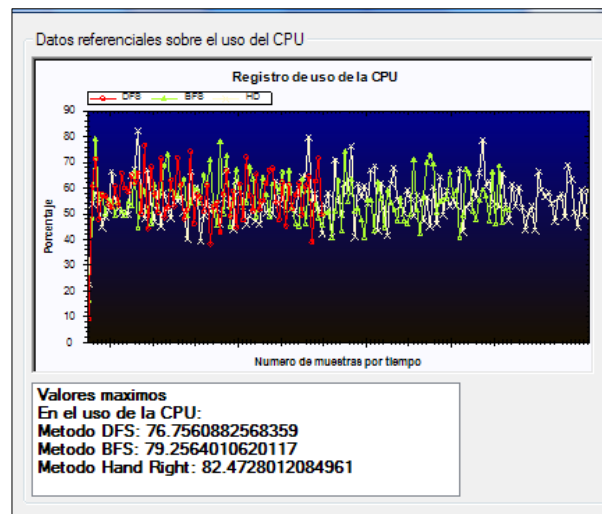
**Ilustración III 14: Resultados parciales Exp 3.**

#### 4. Experimento 4

**Laberinto: Dificultad 45%, Tiempo al 50%.**



**Ilustración III 15: Laberinto y camino encontrado Exp 4.**



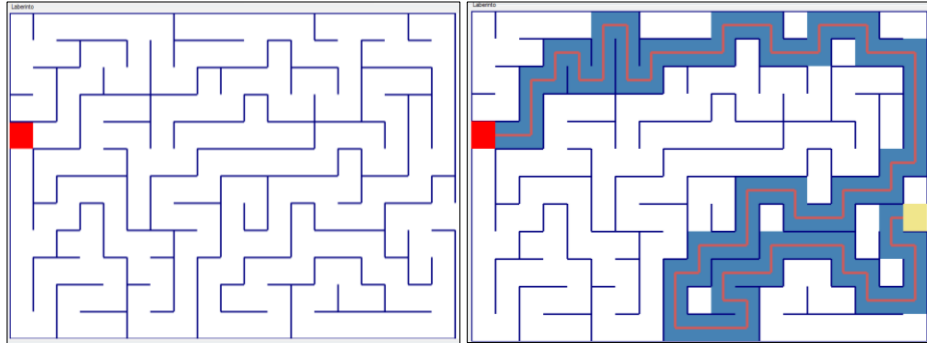
**Ilustración III 16: Uso de la CPU Exp 4.**

#	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	Vel(%)	Dif(%)	Efec	CPU%
1	BFS	130	150	68	12.164	50	45	52	55.817
2	DFS	105	150	68	10.763	50	45	64	56.727
3	Hand Right	107	150	0	14.762	50	45	63	55.284

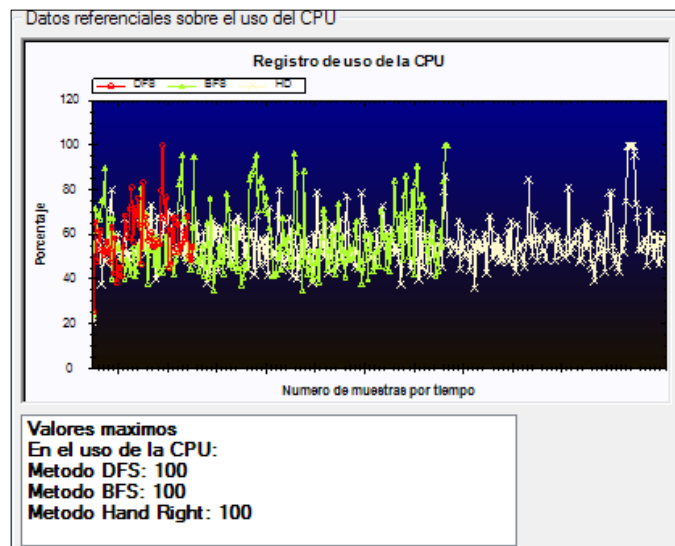
**Ilustración III 17: Resultados parciales Exp 4.**

## 5. Experimento 5

**Laberinto: Dificultad 55%, Tiempo al 50%.**



**Ilustración III 18: Laberinto y camino encontrado Exp 5.**



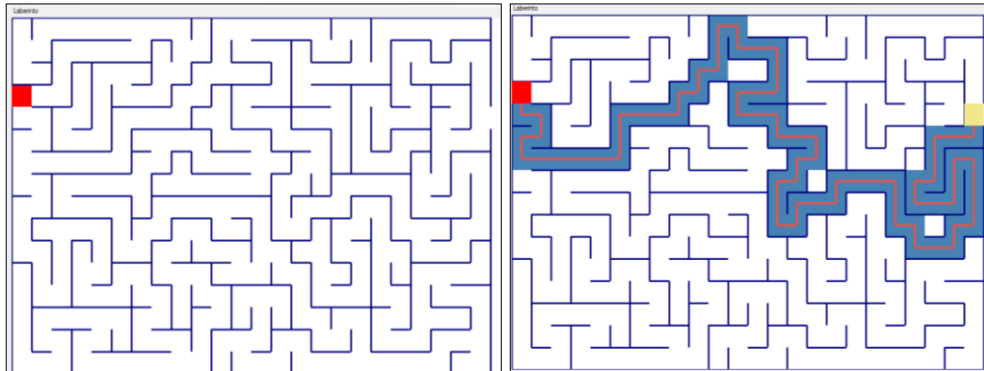
**Ilustración III 19: Uso de la CPU Exp 5.**

#	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	Vel(%)	Dif(%)	Efec	CPU%
1	DFS	104	228	76	9.650	50	55	73	58.832
2	BFS	217	228	76	19.665	50	55	35	57.658
3	Hand Right	204	228	0	35.490	50	55	37	56.273

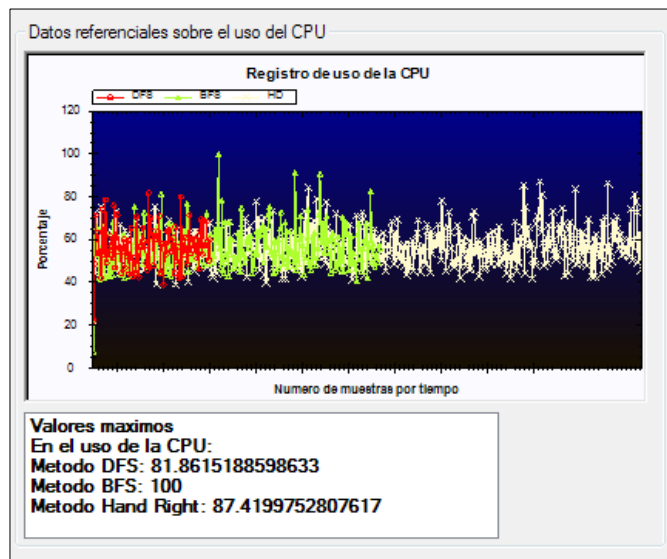
**Ilustración III 20: Resultados parciales Exp 5.**

## 6. Experimento 6

**Laberinto: Dificultad 65%, Tiempo al 50%.**



**Ilustración III 21: Laberinto y camino encontrado Exp 6**



**Ilustración III 22: Uso de la CPU Exp 6.**

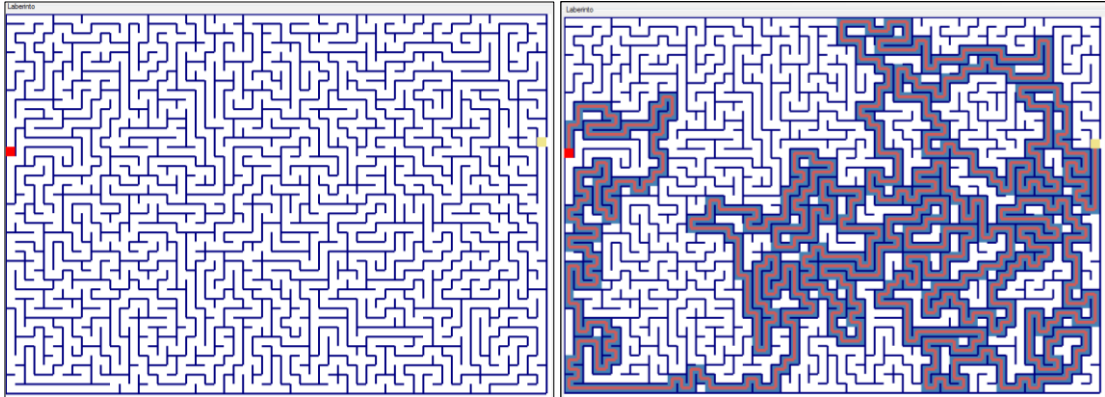
#	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	Vel(%)	Dif(%)	Efec	CPU%
1	DFS	149	384	73	16.707	50	65	48	56.611
2	BFS	268	384	73	28.507	50	65	27	55.606
3	Hand Right	279	384	0	52.077	50	65	26	56.54

**Ilustración III 23: Resultados parciales Exp 6.**

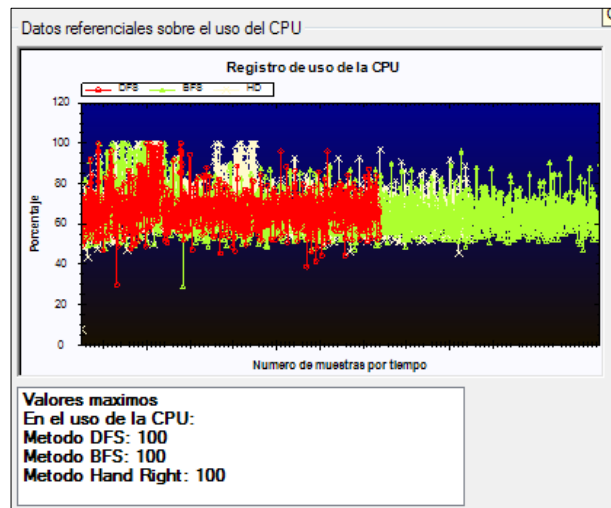


## 7. Experimento 7

**Laberinto: Dificultad 85%, Tiempo al 50%.**



**Ilustración III 24: Laberinto y camino encontrado Exp 7.**



**Ilustración III 25: Uso de la CPU Exp 7.**

#	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	Vel(%)	Dif(%)	Efec	CPU%
1	DFS	1691	2280	884	204.877	50	85	52	67.516
2	BFS	2156	2280	884	255.524	50	85	41	64.78
3	Hand Right	1212	2280	0	179.785	50	85	72	68.657

**Ilustración III 26: Resultados parciales Exp 7.**

## 8. Análisis de datos resultados y tabulación.

### 1. Tiempos de respuesta

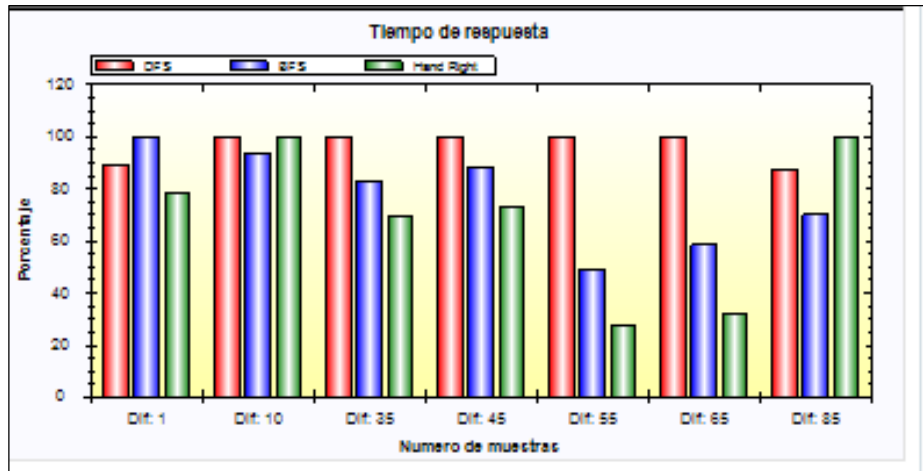


Ilustración III 27: Barras representando el rendimiento de tiempos individuales.

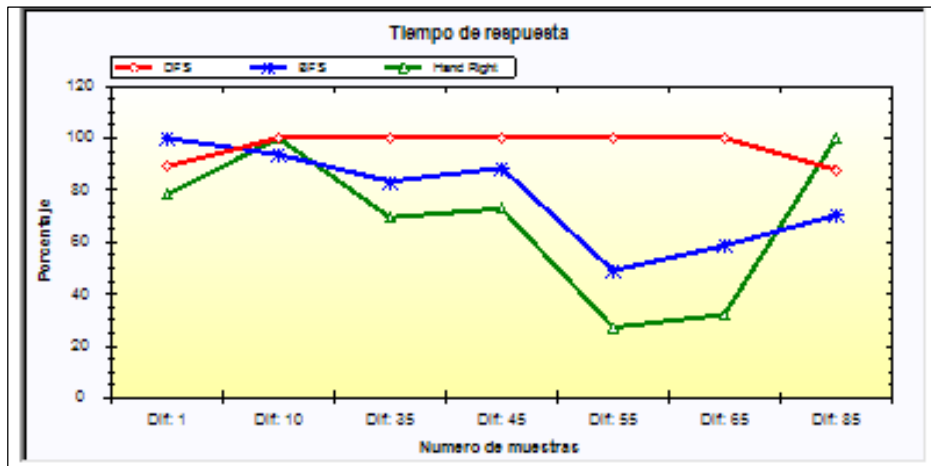


Ilustración III 28: Líneas representando el rendimiento de tiempos individuales.

### 2. Datos resultados.

**Nota:** El tiempo efectivo es el tiempo que sistema encuentra el camino sin tomar en cuenta la búsqueda, ya que el sistema lo sigue aprovechando el camino aprendido.

**Nota:** La nomenclatura de descripción de parámetros esta descrita en la primea parte de los anexos.

**Tabla III 3: Porcentajes obtenidos (Tiempos sin referencia de tiempo efectivo).**

Experimento	DFS (%)	BFS (%)	HR (%)
1	89.16	100	78.13
2	99.95	93.62	100
3	100	83.33	69.42
4	100	88.48	72.91
5	100	49.07	27.19
6	100	58.60	32.08
7	87.75	70.35	100

1.  $DFS_T = \sum DFS_i$

$$DFS_T = 89.16 + 99.95 + 100 + 100 + 100 + 100 + 87.75 = 676.86$$

$$DFS_{PT} = DFS_T / Num_{EXP}$$

$$DFS_{PT} = 676.86 / 7 = 96.69$$

2.  $BFS_T = \sum BFS_i$

$$BFS_T = 100 + 93.62 + 83.33 + 88.48 + 49.07 + 58.60 + 70.35 = 543.81$$

$$BFS_{PT} = BFS_T / Num_{EXP}$$

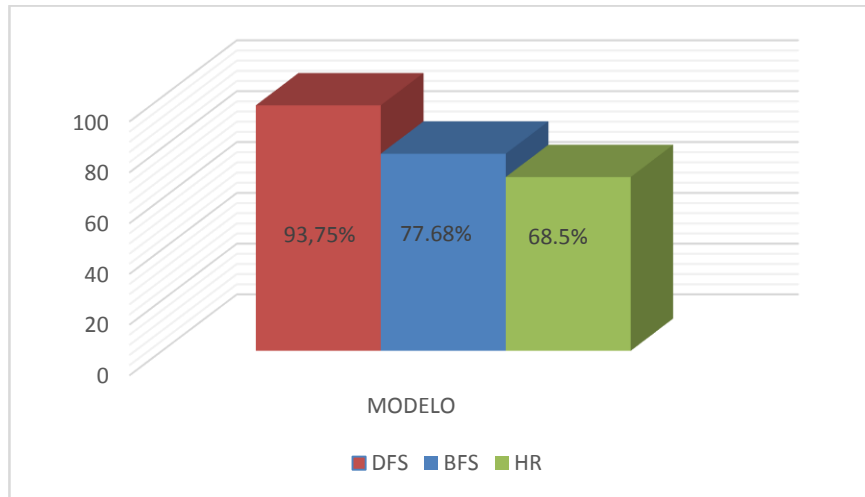
$$BFS_{PT} = 543.81 / 7 = 77.68$$

3.  $HR_T = \sum HR_i$

$$HR_T = 78.13 + 100 + 69.2 + 72.91 + 27.19 + 32.08 + 100 = 479.51$$

$$HR_{PT} = HR_T / Num_{EXP}$$

$$HR_{PT} = 479.51 / 7 = 68.50$$



**Ilustración III 29: Resultados de porcentajes totales sin tiempo efectivo.**

**Tabla III 4: Porcentajes obtenidos (Tiempos con referencia de tiempo efectivo).**

Experimento	DFS (%)	BFS (%)	HR (%)
1	66.33	74.39	58.13
2	96.26	90.16	96.30
3	88.43	73.70	61.39
4	65.80	58.22	47.98
5	80.78	39.64	21.96
6	44.92	26.32	14.41
7	45.50	32.50	56.30

$$1. \quad CDFS_T = \sum CDFS_i$$

$$CDFS_T = 66.33 + 96.26 + 88.43 + 65.8 + 80.78 + 44.92 + 45.5 = 461.02$$

$$CDFS_{PT} = CDFS_T / Num_{EXP}$$

$$CDFS_{PT} = 461.02 / 7 = 65.86$$

$$2. \quad CBFS_T = \sum CBFS_i$$

$$CBFS_T = 74.39 + 90.16 + 73.70 + 58.22 + 39.64 + 26.32 + 32.5 = 394.93$$

$$CBFS_{PT} = CBFS_T / Num_{EXP}$$

$$CBFS_{PT} = 394.93 / 7 = 56.41$$

3.  $CHR_T = \sum CHR_i$

$CHR_T = 58.13 + 96.3 + 61.39 + 47.98 + 21.96 + 14.41 + 56.3 = 356.47$

$CHR_{PT} = CHR_T / Num_{EXP}$

$CHR_{PT} = 356.47 / 7 = 52.21$

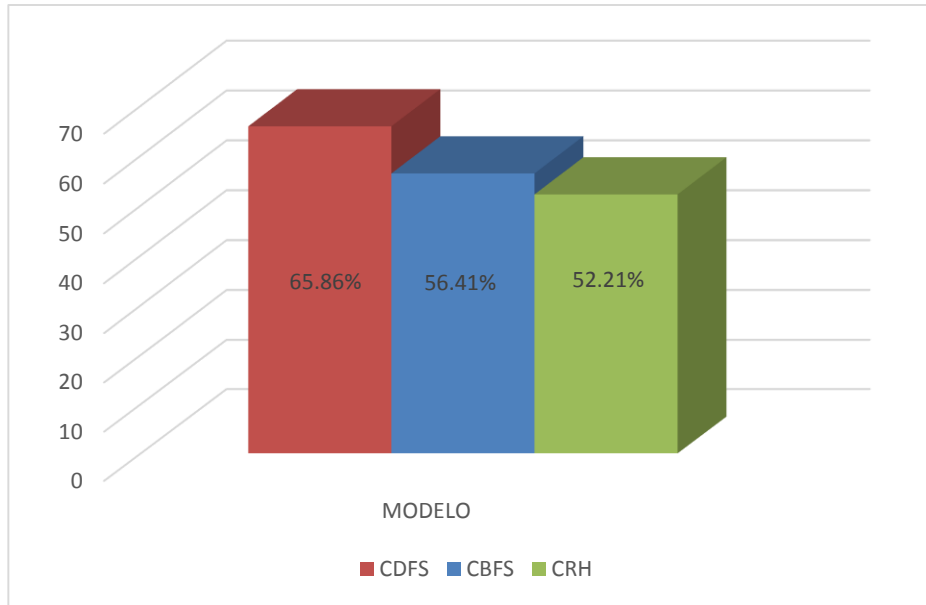


Ilustración III 30: Resultados de porcentajes totales con tiempo efectivo.

3. Eficiencia.

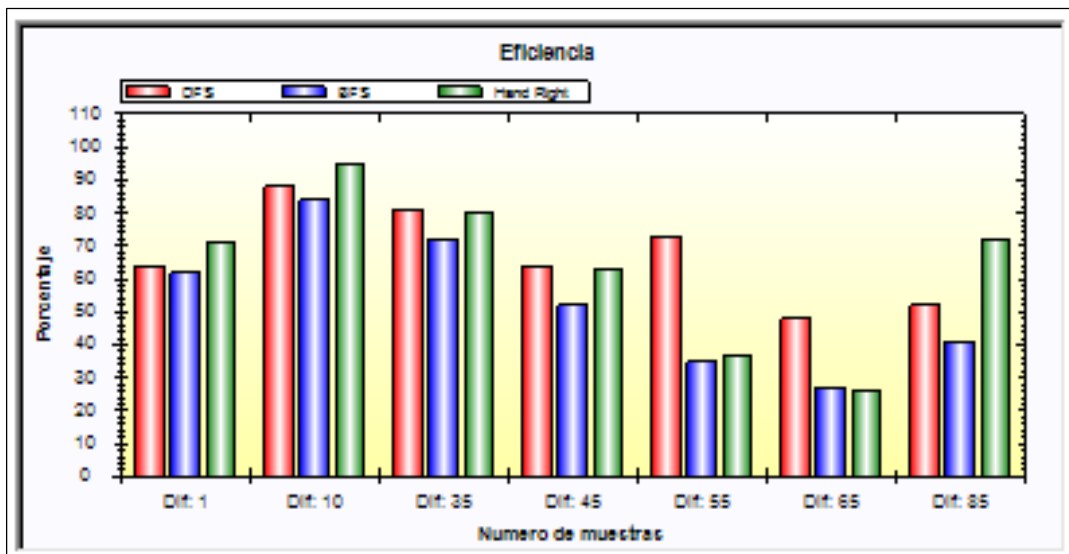


Ilustración III 31: Barras representando la eficiencia individual.

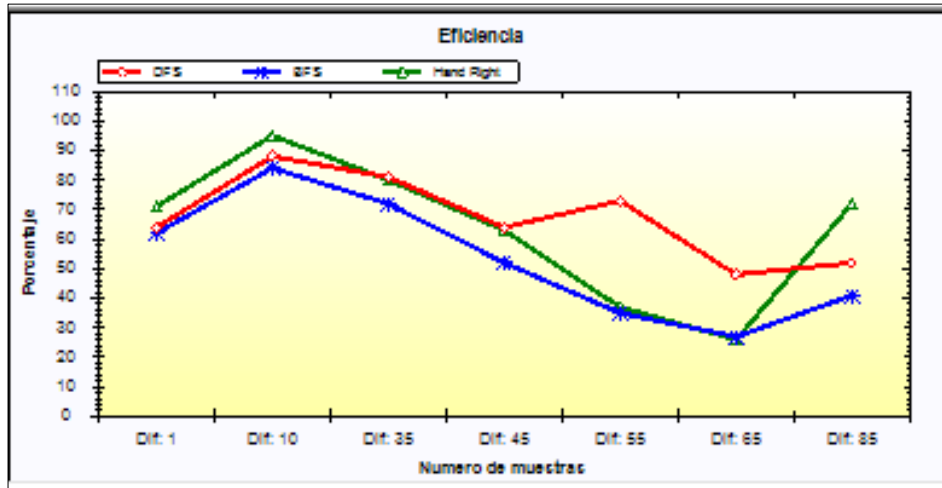


Ilustración III 32: Líneas representando la eficiencia individual.

Tabla III 5: Porcentajes obtenidos (Eficiencia).

Experimento	DFS (%)	BFS (%)	HR (%)
1	64	62	71
2	88	84	95
3	81	72	80
4	64	52	63
5	73	35	37
6	48	27	26
7	52	41	72

$$1. \quad EDFS_T = \sum EDFS_i$$

$$EDFS_T = 64 + 88 + 81 + 64 + 73 + 48 + 52 = 470$$

$$EDFS_{PT} = EDFS_T / Num_{EXP}$$

$$EDFS_{PT} = 470 / 7 = 67.14$$

$$2. \quad EBFS_T = \sum EBFS_i$$

$$EBFS_T = 62 + 84 + 72 + 52 + 35 + 27 + 41 = 373$$

$$EBFS_{PT} = EBFS_T / Num_{EXP}$$

$$EBFS_{PT} = 373 / 7 = 53.28$$

3.  $EHR_T = \sum EHR_i$

$EHR_T = 71 + 95 + 80 + 63 + 37 + 26 + 72 = 444$

$EHR_{PT} = EHR_T / Num_{EXP}$

$EHR_{PT} = 444 / 7 = 63.42$

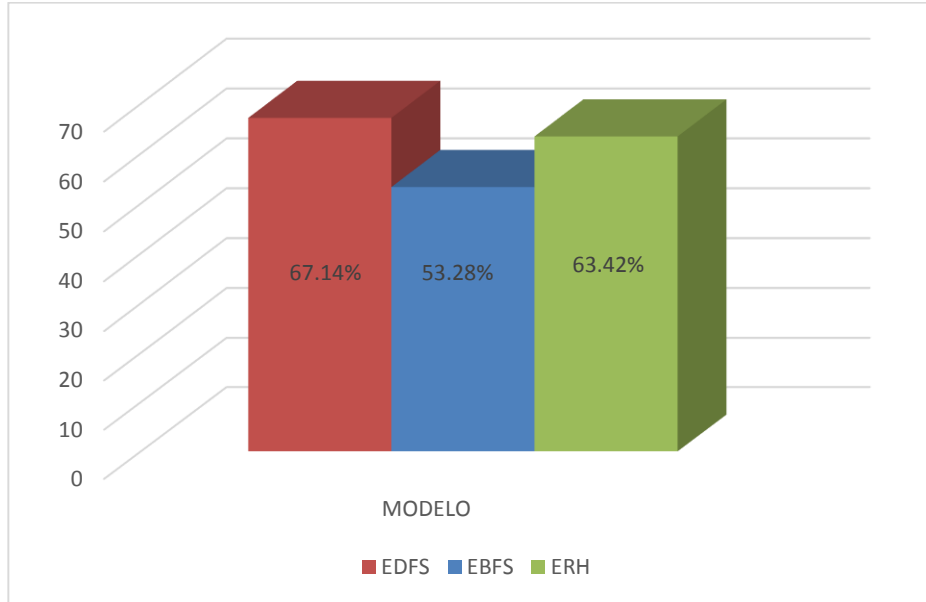


Ilustración III 33: Porcentajes totales de la eficiencia al encontrar la solución.

4. Administración de recursos (Uso de la CPU)

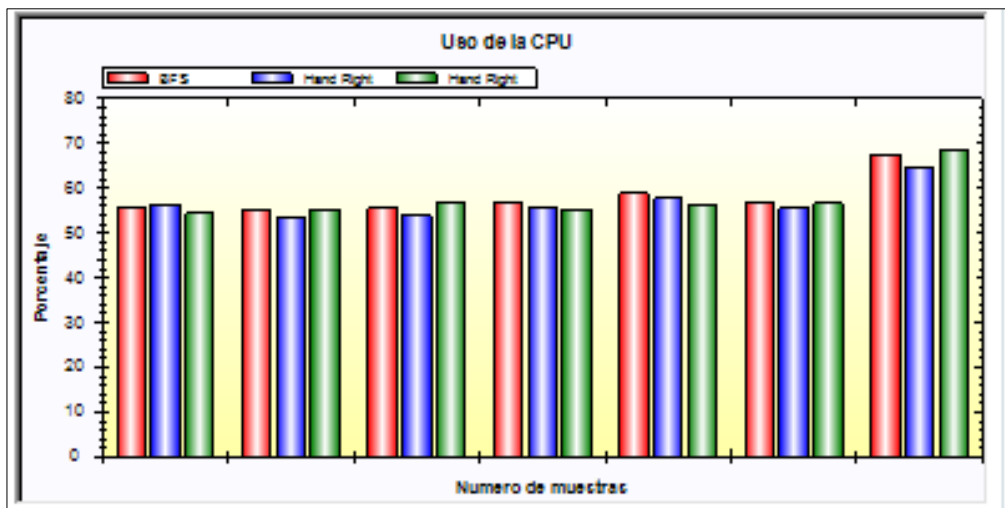


Ilustración III 34: Barras representando el uso de la CPU.

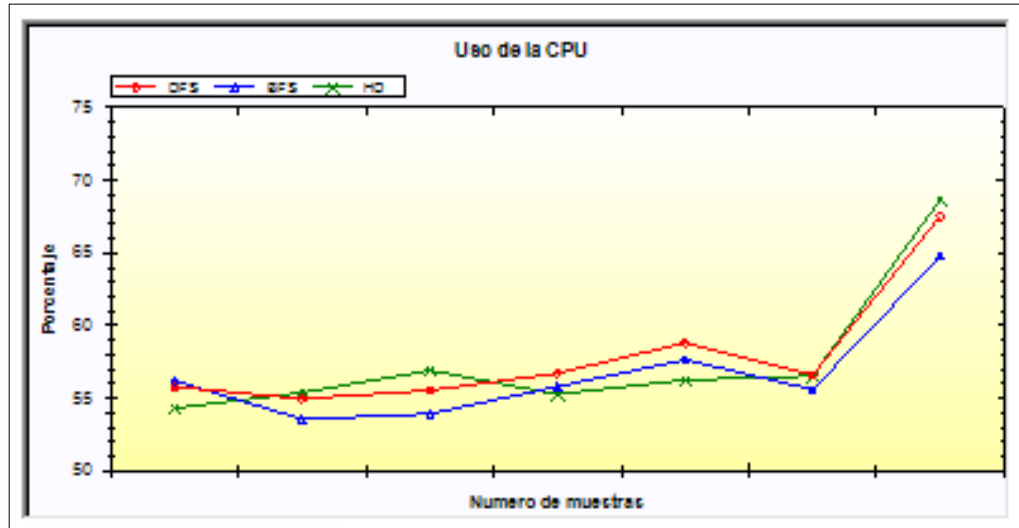


Ilustración III 35: Líneas representando el uso de la CPU.

Tabla III 6: Porcentajes obtenidos (Uso de la CPU).

Experimento	DFS (%)	BFS (%)	HR (%)
1	55	56	54
2	54	53	55
3	55	53	56
4	56	55	55
5	58	57	56
6	56	55	56
7	67	64	68

$$1. \quad UDFS_T = \sum UDFS_i$$

$$UDFS_T = 55 + 54 + 55 + 56 + 58 + 56 + 67 = 401$$

$$UDFS_{PT} = UDFS_T / Num_{EXP}$$

$$UDFS_{PT} = 401 / 7 = 57.28$$

$$2. \quad UBFS_T = \sum UBFS_i$$

$$UBFS_T = 56 + 53 + 53 + 55 + 57 + 57 + 55 + 64 = 405$$

$$UBFS_{PT} = UBFS_T / Num_{EXP}$$

$$UBFS_{PT} = 405 / 7 = 64.28$$



3.  $UHR_T = \sum UHR_i$

$UHR_T = 54 + 55 + 56 + 55 + 56 + 56 + 68 = 400$

$UHR_{PT} = CHR_T / Num_{EXP}$

$UHR_{PT} = 400 / 7 = 57.14$

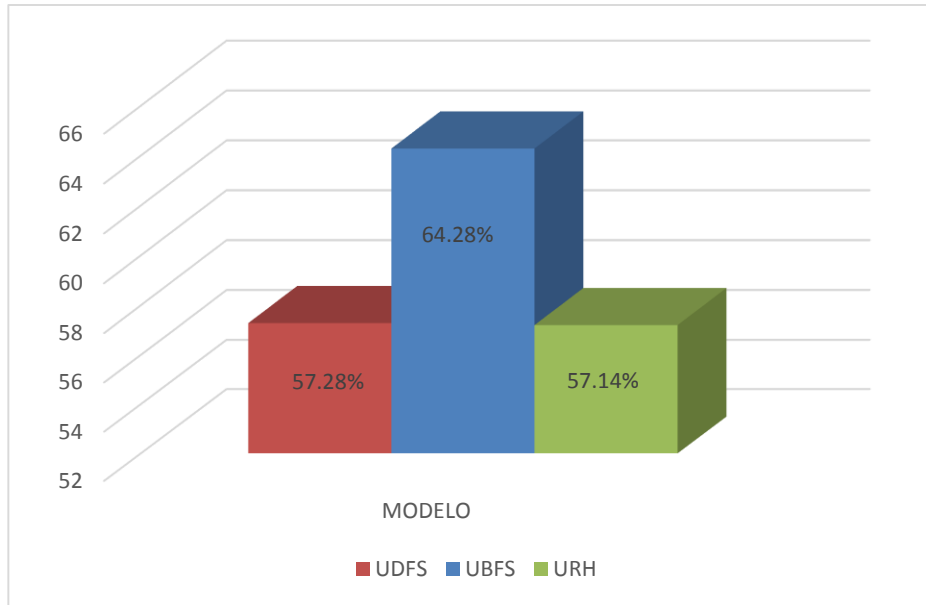


Ilustración III 36: Porcentajes totales del uso de la CPU.

### 3.6.4. Representación de Resultados

#### 3.6.4.1. Análisis Heurístico

Los datos resultados de los indicadores con sus respectivos índices se encuentran en un cuadro de análisis entre los algoritmos heurísticos y el método tradicional, cuyas pruebas de análisis se desarrollaron en diferentes escenarios virtualizados llevados a cabo en el software didáctico “Laberinto Virtual”.

La calificación para cada parámetro se determinara de acuerdo a la escala que se mostrara a continuación, lo cual nos permitirá determinar qué tipo de algoritmo se acopla mejor al objetivo de guiar a un mecanismo autónomo.

**Tabla III 7: Escala de valoración cualitativa y cuantitativa para los indicadores**

Valor Cualitativo		Valor Cuantitativo	Valor Representativo
Insuficiente	No Adecuado	1 a 10	☺
Parcial	Poco Adecuado	11 a 14	☺☺
Suficiente	Adecuado	15 a 17	☺☺☺
Excelente	Muy Adecuado	18 a 20	☺☺☺☺

### 3.6.4.1.1. Calidad

La calidad se define para nuestro experimento con el índice que califica el desenvolvimiento del mecanismo o en este caso del mecanismo simulado, validando el tiempo y la eficiencia que ese resultado de la medición del porcentaje relacionado con los caminos fallidos con respecto al camino solución.

### 3.6.4.1.2. Parámetro 1: Eficiencia.

El sistema mide la eficiencia a través del número de nodos que visita en el transcurso de la simulación, mientras menor es el número de nodos relacionados entre el número de nodos del camino encontrado y el total de nodos visitados el sistema lo califica con una mayor puntuación en el porcentaje de eficacia.

**Tabla III 8: Valoración para indicadores (Eficiencia).**

Indicadores	DFS		BFS		HR	
	Val Rep.	Val Cal.	Val Rep.	Val Cal.	Val Rep.	Val Cal.
Numero de nodos visitados.	☺☺☺	Adecuado	☺	No Adecuado	☺	No Adecuado
Numero de nodos respuesta.	☺☺☺☺	Muy Adecuado	☺☺☺☺	Muy Adecuado	☺☺	Poco Adecuado
Eficiencia.	☺☺☺☺	Muy Adecuado	☺☺☺	Adecuado	☺☺	Poco Adecuado

### 3.6.4.1.3. Parámetro 2: Tiempo.

El sistema mide el tiempo en que resuelve el problema planteado desglosando en su unidad máxima y mínima para dar un porcentaje de eficiencia en tiempo, también puede incluirse como un referente importante el ingreso de un tiempo llamado tiempo efectivo, el cual mide cuanto se demora el sistema en seguir un camino ya almacenado anteriormente.

**Tabla III 9: Valoración para indicadores (Tiempos).**

Indicadores	DFS		BFS		HR	
	Val Rep.	Val Cal.	Val Rep.	Val Cal.	Val Rep.	Val Cal.
Tiempo individual	☺☺☺	Adecuado	☺☺	Poco Adecuado	☺☺☺	Adecuado
Tiempo exacto	☺☺☺☺	Muy Adecuado	☺☺☺☺	Muy Adecuado	☺	No Adecuado

### 3.6.4.2. Comportamiento real

Es el comportamiento que acoge un sistema autónomo en distintos escenarios.

#### 3.6.4.2.1. Parámetro 3: Aprendizaje

El sistema tiene que aprender de la guía que realiza, dicho aprendizaje esta limitado por el tipo de algoritmo que se utilizó, pues por la características ya descritas anteriormente hay algoritmos que permiten el aprendizaje y otros no, dependiendo de las características individuales de cada uno de ellos.

**Tabla III 10: Valoración para indicadores (Tiempos).**

Indicadores	DFS		BFS		HR	
	Val Rep.	Val Cal.	Val Rep.	Val Cal.	Val Rep.	Val Cal.
Aprendizaje	☺☺☺☺	Muy Adecuado	☺☺☺☺	Muy Adecuado	☺	No Adecuado

### 3.6.4.3. Administración de recursos.

Es la medición del uso de la CPU que hace cada uno de los algoritmos.

#### 3.6.4.3.1. Parámetro 4: Uso de la CPU:

El sistema mide el porcentaje de uso de la CPU durante la ejecución del problema de búsqueda, la lectura de los porcentajes en promedio del uso de la CPU se almacenan temporalmente, una vez concluido el trabajo del sistema las variables toma un nuevo valor eliminando valores individuales anteriores.

**Tabla III 11: Valoración para indicadores (Uso de la CPU).**

Indicadores	DFS		BFS		HR	
	Val Rep.	Val Cal.	Val Rep.	Val Cal.	Val Rep.	Val Cal.
Uso de la CPU(Individual)	☺☺☺	<b>Adecuado</b>	☺☺	<b>Poco Adecuado</b>	☺☺	<b>Poco Adecuado</b>
Uso de la CPU(Total)	☺☺☺☺	<b>Muy Adecuado</b>	☺☺☺	<b>Adecuado</b>	☺☺	<b>Poco Adecuado</b>

### 3.6.4.4. Resultados finales

Los resultados finales son las tabulaciones de los diferentes resultados cada uno de ellos evaluados y calificados, la cuantificación de todos los resultados en conjunto nos dará una mejor apreciación sobre el final del análisis de la aplicación de un algoritmo heurístico frente a uno que no sigue una trayectoria heurística.

Tabla III 12: Valoración para indicadores establecidos.

CLASIFICACION	PARAMETROS	INDICADORES	DFS	BFS	HR
Calidad	Eficiencia	• Numero de nodos visitados	16	9	8
		• Numero de nodos respuesta	19	19	12
		• Nodos resultado	19	16	12
	Tiempo	• Tiempo Individual	16	13	16
		• Tiempo exacto	19	19	1
Comportamiento real	Inteligencia	• Aprendizaje	20	20	1
Administración	Uso de la CPU	• Uso de la CPU individual	17	11	11
		• Promedio de uso total de la CPU.	19	17	12

### 3.6.5. Interpretación.

**Eficiencia:** Luego de la tabulación de resultados vemos como el algoritmo DFS presenta ventajas frente a los demás métodos, una de las desventajas del método BFS es la ramificación continua de sus nodos, la misma que lo hace inadecuado para la aplicación en un mecanismo autónomo, el modelo de la mano derecha HR es el modelo no heurístico el cual tiene una desventaja leve ante los demás métodos, el principal es que el sistema toma una referencia en cuanto a caminos ya recorridos de los métodos ya sea DFS o BFS, así que podemos concluir que estas medidas finales no son reales. Sin embargo.

**Tiempo:** Los tiempos individuales mostrados como resultado final muestran una ventaja del algoritmo DFS frente al algoritmo BFS, pero tiene un desempeño similar al método no heurístico, el cual tiene una muy baja calificación en el tiempo efectivo ya que este método no cuenta con esta característica, así que una vez más podemos ver que el

algoritmo DFS tiene una ventaja sobre los demás métodos para ser aplicados en un mecanismo real. **Por su parte.**

**Inteligencia:** Con el análisis observamos que los algoritmos heurísticos tiene la capacidad de almacenar caminos, a diferencia del no heurístico el cual no tiene la capacidad de almacenar coordenadas que simulan el camino encontrado, por su característica de choque y reacción el método de la mano derecha (Hand Right), no registra un trayecto que pueda ser almacena para que luego pueda ser usado como memoria y de esta manera dar una guía nueva al mecanismo pero basándose en lo aprendido. **Así mismo.**

**Uso de la CPU:** Una de la principales preocupaciones a la hora de tratar de instalar un software en una maquina autónoma, es el procesamiento que este utilizara, al final del experimento observamos como resultado, que las más altas calificaciones en uso de la CPU lo obtiene el algoritmo DFS el cual nos indica que el uso de la CPU por parte de este algoritmo es menor que el de los demás algoritmos en principal medida con el algoritmo BFS, que por sus características ya estudiadas, absorbe demasiado procesamiento por parte de la CPU.

### 3.6.6. Calificación

**Calculo de los porcentajes.**

$$C_{DFS} = \sum DFSi$$

$$C_{BFS} = \sum BFSi$$

$$C_{HR} = \sum HR i$$

$$P_{XX} = \left( \frac{C_{XX}}{Ct} \right) * 100\%$$

$$C_{DFS}: 16 + 19 + 19 + 16 + 19 + 17 + 19 + 20 = 145$$

$$C_{BFS}: 9 + 19 + 16 + 13 + 19 + 11 + 17 + 20 = 124$$

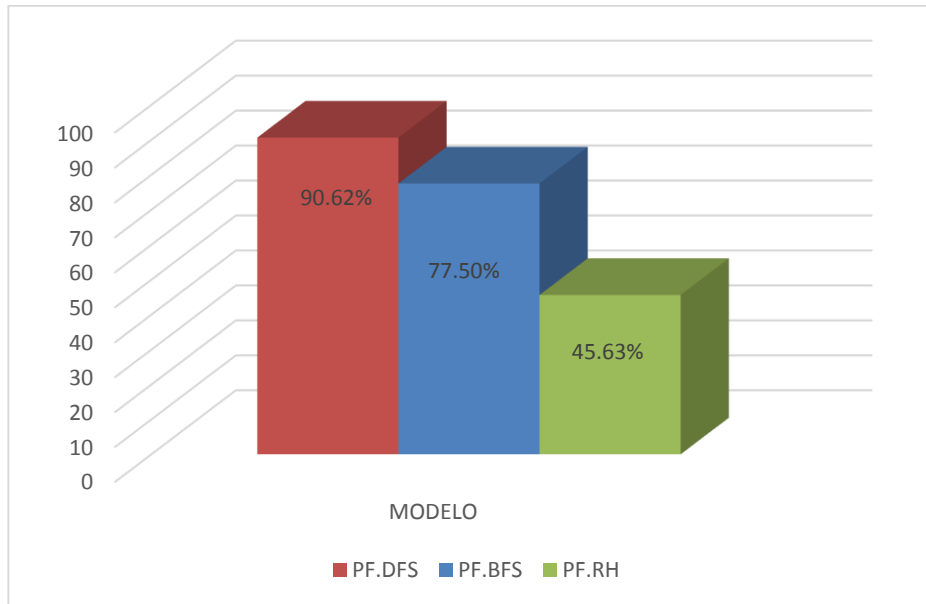
$$C_{HR}: 8 + 12 + 12 + 16 + 1 + 11 + 12 + 1 = 73$$

$$C_t: 20 + 20 + 20 + 20 + 20 + 20 + 20 + 20 = 160$$

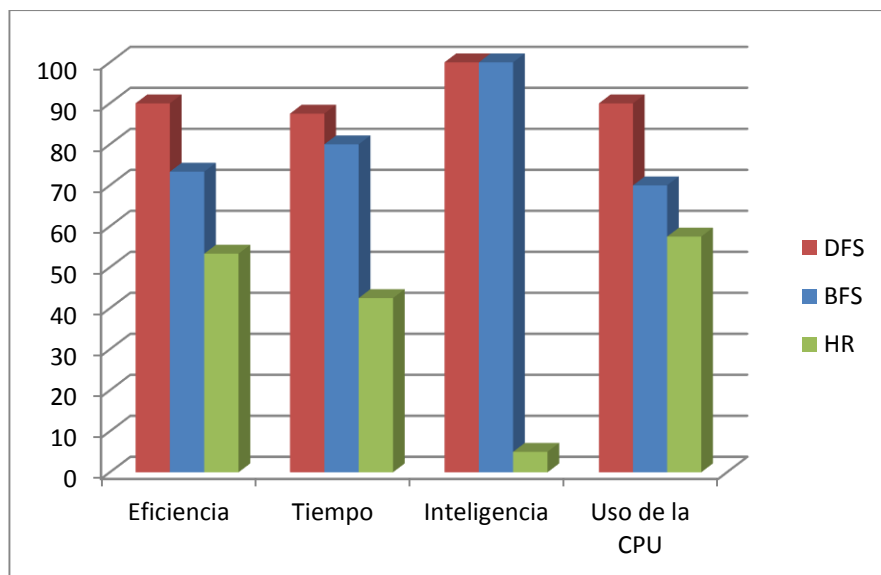
$$P_{DFS} = \left( \frac{C_{DFS}}{C_t} \right) * 100\% = 90.62$$

$$P_{BFS} = \left( \frac{C_{BFS}}{C_t} \right) * 100\% = 77.5$$

$$P_{HR} = \left( \frac{C_{HR}}{C_t} \right) * 100\% = 45.63$$



**Ilustración III 37: Grafica de resultados**



**Ilustración III 38: Grafica general de resultados**

**Tabla III 13: Resultados generales.**

	<b>Eficiencia</b>	<b>Tiempo</b>	<b>Inteligencia</b>	<b>Uso de la CPU</b>
<b>DFS</b>	90	87.5	100	90
<b>BFS</b>	73.3	80	100	70
<b>HR</b>	53.3	42.5	5	57.5

### **3.6.7. Interpretación**

Con la tabulación de los resultados obtenidos en el análisis de algoritmos heurísticos frente a un no heurístico, se da a conocer de una manera global el comportamiento de cada uno de los algoritmos escogidos para usarlo como arquitectura principal en el desarrollo de aplicaciones orientadas en la guía y aprendizaje de mecanismos autónomos.

El simulador diseñado como herramienta didáctica, virtualizó diferentes escenarios los cuales sirvieron para obtener los datos que fueron tabulados dentro de parámetros descritos anteriormente, la eficiencia y los tiempos de respuesta son una parte muy importante a la hora de escoger un adecuado algoritmo de aplicación para el desarrollo de un software inteligente.

La versatilidad del sistema recae en cumplir una estandarización tanto por parte de los mecanismos en los que serán instalados este tipo de sistemas, como en la herramienta que se use para el desarrollo de la misma.

Frente a los diferentes escenarios DFS demostró tener una ventaja ante los otros algoritmos usados para el análisis, demostrando que un algoritmo heurístico si es una forma adecuada de dar guía e inteligencia a un mecanismo autónomo.



### **3.6.8. Análisis de Resultados**

En los diferentes datos tabulados obtenemos una ventaja de desenvolvimiento en el algoritmo DFS, su eficacia muestra una ventaja en la guía para busca un camino adecuado con el menor gasto de recursos posible, el tiempo que toma el sistema en resolver un laberinto es una parte importante del análisis, porque mientras lo resuelva en menos tiempo existe menos uso de procesamiento, con menos recursos usados más probabilidad de instalarlo en un mecanismo que por sí mismo pueda auto guiarse y aprender a medida que vaya avanzando.

Por los datos graficados y tabulados podemos determinar que el análisis concluye como adecuado el uso de algoritmos heurísticos frente a los de choque y reacción, y que por las características propias del algoritmo DFS, es la mejor opción a la hora de elegirlo como base algorítmica para el desarrollo de sistemas inteligentes orientados a la guía y aprendizaje en mecanismos autónomos.

**CAPITULO IV:**  
**ALGORITMOS HEURÍSTICOS APLICADOS EN EL APRENDIZAJE Y GUÍA**  
**DE SISTEMAS AUTÓNOMOS (OBTENCIÓN DE UN MODELO DE**  
**PLANIFICACIÓN ADECUADO).**

**4.1. Algoritmos de Planificación**

Los desarrollos producidos en las últimas décadas en campos como la robótica, la inteligencia artificial o la teoría de control han despertado un interés creciente por los problemas de planificación. [15]

Siguiendo el excelente trabajo de revisión sobre algoritmos de planificación realizado por La Valle, podría decirse que en robótica la planificación estaba inicialmente relacionada con problemas tales como el movimiento de un piano dentro de una casa, de una habitación a otra, sin golpear contra ningún objeto. Actualmente, el tipo de problemas abordado ha crecido e incluye aspectos tales como la dinámica del problema, incertidumbres o la presencia de múltiples robots. En Inteligencia artificial, originalmente se entendía por planificación la búsqueda de una secuencia de operadores lógicos que transformaran un estado inicial en un estado final deseado. En la actualidad el problema se ha extendido e incluye ideas de teoría de la decisión, como cadenas de decisión de Markov, información imperfecta de estados, y teoría de juegos. La teoría de control tradicionalmente ha estado relacionada con problemas de estabilidad, realimentación y optimización, pero hay también un interés creciente en el diseño de algoritmos que permitan obtener trayectorias en lazo abierto para sistemas no lineales. [15]

A medida que se ha ido ampliando el área de interés en cada uno de los campos, se ha producido un intercambio de conocimientos y técnicas entre ellos, que permite establecer una base de conocimiento común. [15]

Desde un punto de vista general, podemos descomponer un problema de planificación en los siguientes elementos:

**Estados.** Un problema de planificación supone siempre la existencia de un espacio de estados, que configura todas las posibles situaciones en que pueda encontrarse el sistema objeto de estudio, por esta razón recibe también el nombre de espacio de configuración. Un estado puede venir representado por la posición en el espacio y la orientación de un móvil, las orientaciones de las articulaciones de un brazo robótico, o las posiciones de las fichas en un tablero de ajedrez. En general, según el sistema de que se trate o el grado de definición del mismo que sea preciso manejar, podemos encontrar espacios de estados discretos (finitos o infinitos) y espacios de estados continuos. En muchas aplicaciones el tamaño del espacio de estados es demasiado grande para poder representarlo explícitamente. Pero en cualquier caso, supone un componente necesario para la formulación de un problema de planificación. [15]

**Tiempo.** Cualquier problema de planificación, supone una secuencia de decisiones que deben aplicarse en el tiempo. En un problema concreto, puede ser necesario incluirlo explícitamente, como sucede en los problemas de planificación de movimiento, o tan solo implícitamente, reflejando el hecho de que las acciones tomadas deben ser sucesivas. Como en el caso de los estados, el tiempo puede ser discreto o continuo, en este último caso, podemos considerar la toma de decisiones como una función continua que evoluciona en el tiempo. [15]

**Acciones.** Un plan genera acciones que manipulan los estados. Desde un punto de vista de control es más frecuente hablar en términos de controladores y entradas al sistema. Cuando se formula un plan, es necesario conocer cómo las acciones modifican los estados. Habitualmente esto se expresa como una función de los estados para el caso de sistemas de tiempo discreto y mediante ecuaciones diferenciales para sistemas de tiempo

continuo. En este último caso, es bastante frecuente en los problemas de planificación tratar de obtener una trayectoria en el espacio de estados integrando las ecuaciones diferenciales. [15]

**Estado inicial y estado objetivo.** Un problema de planificación supone la existencia de un estado inicial en que se encuentra el sistema y un estado o un conjunto de estados finales a los que se desea llevar el sistema. [15]

**Criterio.** Un problema de planificación lleva habitualmente asociado un criterio con el que se quiere hacer que el sistema evolucione desde el estado inicial hasta el estado objetivo. [15]

Generalmente, se distinguen dos tipos de criterios,

- **Viabilidad.** Se busca un plan que permita llegar al estado objetivo, con independencia de la eficiencia de dicho plan. Es decir el único criterio es que el plan sea realizable. [15]
- **Optimizar.** Se busca un plan viable que además de alcanzar el estado objetivo, sea óptimo desde el punto de vista de su desarrollo en algún sentido; minimizar el tiempo empleado, la energía consumida, etc. [15]

#### 4.1.1. Planificación de movimiento

Se entiende por planificación de movimiento en general la planificación en espacios de estado continuos. Por otro lado, y en el ámbito de la robótica y del movimiento de vehículos autónomos se entiende la planificación del movimiento de dichos objetos dentro del mundo 2D o 3D, con posibles obstáculos, en el que se encuentran inmersos. No es difícil entender que la segunda acepción está relacionada con la primera, puesto que el espacio real en que se mueven los vehículos puede constituir parte de su espacio de configuraciones, o al menos será posible encontrar una ley de transformación más o menos complicada entre ambos espacios. La dimensión completa del espacio de estados coincide con el número de grados de libertad que tenga el vehículo o robot. Por supuesto, para este tipo de problemas no es posible encontrar una definición explícita de los estados

del sistema puesto que se trata de un espacio de estados infinito e incontable. En este contexto, un plan puede describirse como una trayectoria continua en el espacio de configuraciones. [15]

Una práctica habitual a la hora de tratar con problemas de planificación de movimiento es hacerle discreto al problema. Se construye un modelo discreto a partir del sistema continuo y se planifica sobre el modelo discreto. Así por ejemplo para el caso de la búsqueda de una trayectoria del movimiento de un vehículo se descompone ésta en puntos de paso, cada uno de ellos se considera como un estado discreto, y se planifican las trayectorias entre estados discretos. Este método simplifica la búsqueda de soluciones viables, siempre y cuando los puntos de paso elegidos como estados discretos sean alcanzables, y exista algún modo de obtener las acciones, en general funciones de los estados, que definen la transición de un estado a otro. [15]

Como se indicó en el apartado anterior, desde el punto de vista de la planificación es posible buscar soluciones óptimas o soluciones simplemente viables. En muchos problemas de planificación de movimiento, la búsqueda de soluciones viables es ya de por sí complicada. En cualquier caso, la discretización del problema que acabamos de describir, puede hacer que se simplifique dicha búsqueda en la medida en que obtener una trayectoria entre los nuevos estados discretos sea sencillo. Si éste es el caso, la dificultad se traslada a la discretización del espacio de búsqueda. Cuando se está buscando una solución óptima el problema se complica puesto que analíticamente supone el cálculo de un extremo para la función de coste que se pretende minimizar. En general, el camino a seguir suele ser entonces la búsqueda entre las posibles soluciones en el espacio de configuración. Esta búsqueda puede ser sistemática: se prueban todos los casos posibles hasta dar con el óptimo. Es lo que se conoce como planificación combinatoria de movimiento, y por razones obvias generalmente solo es aplicable en situaciones concretas donde el número de posibles combinaciones es limitado. La segunda vía es la que se conoce en términos generales como planificación de movimiento basada en muestreo. En este caso no se exploran todas las posibilidades sistemáticamente. Se prueban algunas de ellas, es decir se muestrea el espacio de soluciones, y se busca un método heurístico que permita converger hacia la solución óptima a partir de las muestras analizadas. Por último

indicar que ambos métodos pueden también emplearse para obtener soluciones viables cuando no es sencillo o posible obtener directamente la trayectoria que une dos estados consecutivos. [15]

Los tipos de planificación descritos hasta ahora, dan lugar a lo que se conoce como planificación en lazo abierto. Es decir, no hay ningún mecanismo que permita controlar los resultados del plan, una vez puesto en marcha y corregir posibles desviaciones. Un paso más en el estudio de los métodos de planificación lo constituye lo que se conoce como planificación de movimiento realimentada. En este caso, se estudia la evolución del sistema en el espacio de estados como fruto de la planificación, y en función de dicha evolución se modifica o reconstruye el plan elaborado. Este tipo de planificación es necesaria en presencia de incertidumbres. [15]

#### **4.1.2. La planificación como un problema de búsqueda en un espacio de estados**

Dada la descripción de un problema de planificación a partir de estados iniciales, estados objetivos y un conjunto de operaciones o acciones disponibles, su solución podría ser vista como un problema de búsqueda, en el cual empezando por el estado inicial y, aplicando las acciones disponibles, una por una, e ir obteniendo nuevos estados se logre llegar a un estado que satisface la descripción del objetivo. [15]

La planificación es, básicamente, un problema de búsqueda en un espacio de estados, definido por:

- Un conjunto de estados, normalmente representados en lógica de predicados.
- Unas funciones de transición: denominadas operadores, similares conceptualmente a las reglas de los sistemas de producción.
- Un estado inicial, que debe pertenecer al conjunto de estados válidos y es el estado de partida.
- Un estado final o meta, que también debe pertenecer al conjunto de estados válidos y es el estado al que se desea llegar.

Requiere encontrar una secuencia eficiente de acciones que conducen a un sistema desde un estado inicial hasta un estado objetivo. Desde el punto de vista de agentes inteligentes, el campo de la planificación busca construir algoritmos de control que permitan a un agente sintetizar una secuencia de acciones que le lleve a alcanzar sus objetivos. [15]

Para lograr esto, solo se necesitaría además de la descripción del problema, un algoritmo de búsqueda de los normalmente utilizados en cualquier curso de Inteligencia artificial. Se podría denominar un planificador por espacio de estados, puesto que la búsqueda se realiza a través de un espacio de estados posibles como el mostrado a continuación.

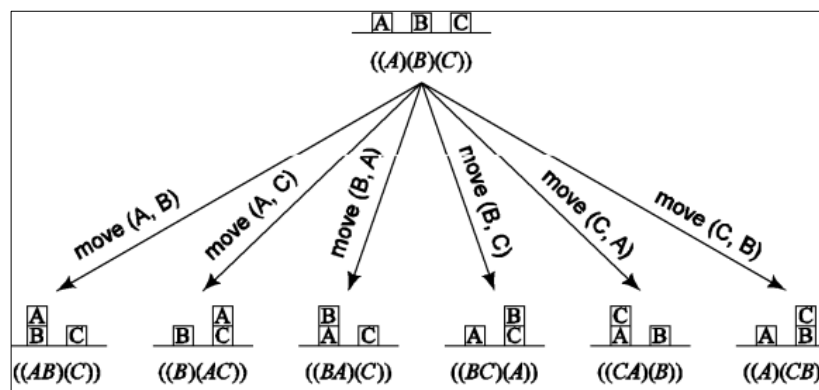


Ilustración IV 1: Espacio de estados

En este espacio de estados, cada nodo representa una posible descripción del mundo circundante y cada arco que une dos nodos representa la aplicación de una acción. De esta forma, el camino que se busca hasta el estado meta constituye, realmente, el plan de acción deseado para resolver el problema.

Estos planes obtenidos como resultado de una búsqueda en un espacio de estados son planes totalmente ordenados de acciones, en el sentido de que dos acciones se encuentran ordenadas entre sí formando una única secuencia. [15]

Este tipo de planificador, también se podría clasificar como un planificador por progresión puesto que la búsqueda se desarrolla hacia delante desde el estado inicial hasta un estado meta. El principal problema de este método de construcción hacia delante es que el factor de ramificación, es decir, el número promedio de elecciones que se presentan

en cada paso, puede llegar a ser muy elevado, lo que hace que el proceso de búsqueda pueda ser ineficiente. [15]

Una forma de ganar en eficiencia en un proceso de búsqueda en espacio de estados, consiste en construir este proceso de búsqueda en el sentido inverso, es decir, comenzando por el estado meta hasta el estado inicial. Basándose en la suposición de que el estado meta del problema puede contener un menor número de literales que el necesario para describir un estado completo, como el estado inicial, entonces este proceso de búsqueda hacia atrás tendrá un factor de ramificación menor y, por tanto, la búsqueda será más eficiente. Un algoritmo de planificación basado en un proceso de búsqueda como este se podría calificar como un planificador por regresión. [15]

Los algoritmos para resolver modelos de estados son los llamados algoritmos de búsqueda que exploran total o parcialmente el estado de espacios tratando de encontrar un camino (óptimo) que lleve de  $s_0$  a algún  $s \in SG$ . [15]

Se distingue entre:

**1. Búsqueda ciega o algoritmos de fuerza bruta:** la meta tiene un papel pasivo en la búsqueda. Por ejemplo, búsqueda en profundidad (LIFO), en búsqueda de anchura, bidireccional, y otros. [16]

**Nota:** La búsqueda por fuerza bruta no se debe confundir con backtracking, método que descarta un gran número de conjuntos de soluciones, sin enumerar explícitamente cada una de las mismas. [16]

**1. Algoritmos de búsqueda heurística:** se utiliza una función  $h(s)$  que estima la “distancia” (coste) desde el estado  $s$  hasta  $SG$  para guiar la búsqueda. Por ejemplo, búsqueda en escalada, técnicas de mejor-primero ( $A^*$ ,  $IDA^*$ ). [16]



#### **4.1.2.1. Métodos de planificación**

El proceso de planificación puede especificarse como un proceso de búsqueda en un espacio de estados, por lo que se estudiarán las diferentes formas de realizar este tipo particular de búsqueda. [17]

Tradicionalmente, esta búsqueda se ha venido realizando hacia atrás. Es decir, partiendo de las metas, estado final, hasta alcanzar el estado inicial. Una vez hecho esto, los operadores que se han ejecutado, por orden, conformarán el plan, resultado del proceso de planificación. La búsqueda se suele realizar hacia atrás debido a que es menor el factor de ramificación partiendo de las metas que partiendo del estado inicial. [17]

La tema (D, I, O) expresa implícitamente un problema de búsqueda en el espacio que define D, y cuya exploración (de una forma más o menos inteligente) debe encontrar los sucesivos pasos del plan solución. [17]

##### **4.1.2.1.1. Búsqueda de planes hacia delante.**

Dado el estado inicial, y el estado final, se van aplicando operadores para llegar desde el estado inicial al estado final. [17]

Un problema de planificación se puede plantear como un problema de espacio de estados:

- Estados descritos mediante listas de literales.
- Operadores como listas de precondiciones, adición y borrado.
- Función es-estado-final descrita por un objetivo

La búsqueda de planes podrá hacerse usando los algoritmos de búsqueda ya vistos anteriormente: anchura, profundidad, primero el mejor, A\*, DFS, BFS, etc.

El uso de la representación STRIPS permite el uso de heurísticas independientes del dominio. Por ejemplo, el número de literales en el objetivo que quedan por satisfacer en un estado. [17]

La búsqueda hacia adelante en un problema de planificación:

- Es completa (si el conjunto de objetos es finitos)
- Es ineficiente en la práctica (por las razones ya expuestas)

El formalismo lógico para representar los estados, resuelve en cierta medida:

- El problema del marco
- La obtención de heurísticas independientes del dominio

Sin embargo, persisten algunos problemas

- El gran factor de ramificación (consideración de acciones irrelevantes)
- La búsqueda del plan de manera secuencial
- Planes no jerarquicos.

**4.1.2.1.2. Búsqueda de planes hacia atrás.** Partimos del estado final e intentamos llegar al estado inicial.

Pero para luego dar una solución necesitamos aplicar unos operadores inversos. [17]

- Comienzo en un objetivo
- En cada estado se generan todos los posibles predecesores
- Finalizar cuando se alcanza un objetivo que es cierto en el estado inicial

**Predecesor de un objetivo G:**

- Mediante operador O y sustitución  $\theta$  es cualquier estado E cuyo sucesor al aplicar O y  $\theta$  satisface G.
- Relevante: Al menos, en la lista de adición del operador está uno de los literales de G
- Consistente: En la lista de borrado de G no aparece ningún literal de G

**Ventajas:**

- Generalmente más eficiente: menos ramificación.
- Posibilidad de aplicar heurísticas independientes del dominio (proximidad al estado inicial).

#### 4.2. Determinación de un método adecuado de planificación.

Para el análisis de los métodos heurísticos se ha tomado como referencia un conjunto de definiciones de la publicación “Técnicas de IA para problemas de planificación” [17] desarrollada por Césari Matilde del Departamento de Ciencias Informática e Ingeniería de la Universidad de Washington, ya descritas anteriormente, de las cuales se ha seleccionado las características de cada uno de los métodos de planificación para ser valorados, de acuerdo con las necesidades del experimento resultado, una vez comprobado la eficacia del método DFS frente a los demás planteados, la búsqueda de una adecuada planificación será valorada por sus características dependiendo también de la aplicabilidad en el algoritmo con mejor resultado en el anterior análisis.

##### 4.2.1. Valoración de un modelo adecuado de planificación.

En cada método se valora sus características independientemente como adecuado o no adecuado, explotando al máximo las funcionalidades ya mencionadas de cada una de las características ya expuestas.

- **Validación de características individuales:** En este índice se valorizará las características individuales de los métodos ya descritos.

Tabla IV 1: Descripción de valores.

Valoración		
Implementación/20	Valoración Cualitativa	Representación
1 a 5	No Adecuado	☹
6 a 10	Poco Adecuado	☹☹
11 a 15	Adecuado	☹☹☹
16 a 20	Muy Adecuado	☹☹☹☹

**Tabla IV 2: Valoraciones individuales (Método búsqueda hacia delante).**

Método (Búsqueda de planes hacia delante).	Calificación	Valoración/20
1. Solución completa (Existe respuesta): Buscará un resultado en caso de haberla.	☺☺☺☺	20
2. Grado de dificultad alto para ser aplicado en una aplicación real.	☺	4
3. Heurística independiente del dominio: La aplicación de una lógica heurística es independiente a este tipo de planificación.	☺☺☺	15
4. Ramificación de búsqueda: Abre camino constantemente para encontrar la solución o la recompensa.	☺☺☺	12
5. Aplicación de la búsqueda secuencial: Sigue una búsqueda ordenada uno a continuación de otro.	☺☺☺☺	19
6. Planificación no jerárquica (No existen niveles de resultados alcanzados y tampoco niveles de planificación).	☺	5
<b>Total:</b>		<b>75</b>

**Tabla IV 3: Valoraciones individuales (Método búsqueda hacia atrás).**

Método (Búsqueda de planes hacia atrás).	Calificación	Valoración/20
1. Comienzo desde el estado final el cual contiene la recompensa a uno inicial.	☺	5
2. Finaliza cuando obtiene un estado inicial	☺☺	8
3. Ramificación mínima pues contiene registros de una ruta ya planteada.	☺☺☺☺	19
4. Planificación no jerárquica (No existen niveles de resultados alcanzados y tampoco niveles de planificación).	☺	5
5. Posibilidad de aplicar heurísticas independientes del dominio (proximidad al estado inicial).	☺☺☺	15
<b>Total:</b>		<b>52</b>

Para la realización de la comparación se utilizara la siguiente nomenclatura:

*X* = Representa el puntaje obtenido por el método hacia delante.

*Y* = Representa el puntaje obtenido por el método hacia atrás.

*Cdj* = Representa el puntaje alcanzado por el método hacia delante en suma.

*Crór* = Representa el puntaje alcanzado por el método hacia atrás en suma.

*Ct* = Representa el puntaje por el cual serán evaluados los parámetros.

*Pdj* = Calificación porcentual obtenida por el método hacia delante en suma.

*Prór* = Calificación porcentual obtenida por el método hacia atrás en suma.

Las fórmulas que se utilizarán en el proceso del análisis comparativo son las siguientes:

$$Cdj = \sum X$$

$$Cdj = 20 + 4 + 15 + 12 + 19 + 5 = 75$$

$$Ct = 20 + 20 + 20 + 20 + 20 + 20 = 120$$

$$Crór = \sum Y$$

$$Crór = 5 + 8 + 19 + 5 + 15 = 52$$

$$Ct = 20 + 20 + 20 + 20 + 20 = 100$$

$$Pdj = \left(\frac{Cdj}{Ct}\right) * 100\%$$

$$Pdj = \left(\frac{75}{120}\right) * 100\%$$

$$Pdj = 62.5$$

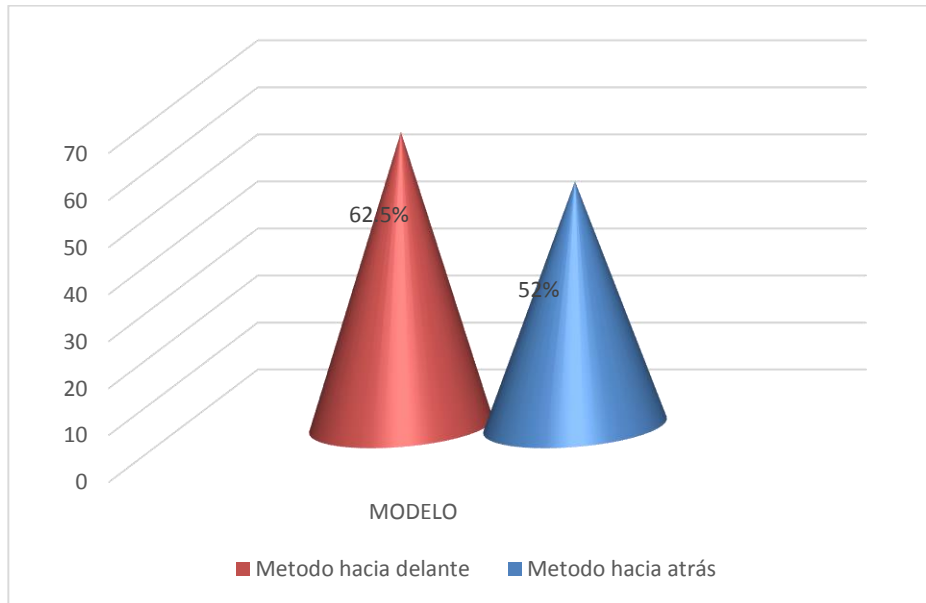
$$Prór = \left(\frac{Crór}{Ct}\right) * 100\%$$

$$Prór = \left(\frac{52}{100}\right) * 100\%$$

$$Prór = 52$$

**Tabla IV 4: Ponderación de resultados (Total).**

Métodos calificados	Puntajes obtenidos	Equivalencia
Búsqueda de planes hacia delante.	75	62.5%
Búsqueda de planes hacia atrás.	52	52%



**Ilustración IV 2: Resultados de porcentajes totales obtenidos del análisis de métodos de planificación.**

#### 4.2.2. Interpretación

**Solución.** El modelo de planificación Búsqueda hacia adelante encontrara una solución si la hay, dicho resultado no depende de la complejidad del escenario, se extenderán caminos hasta encontrar la salida o la recompensa.

**Selección heurística independiente del dominio.** Se podrá elegir un algoritmo heurístico independiente del modelo de planificación búsqueda hacia adelante, siempre y cuando no contradiga con la funcionalidad del mismo.

**Ramificación de búsqueda.** Abre caminos de búsqueda constantemente hasta encontrar una respuesta (TRUE) o alguna recompensa, una característica importante para la aplicación del método de planificación búsqueda hacia adelante es la guía y generación de un camino hasta encontrar una solución.

**Busque ordenada.** EL método de planificación búsqueda hacia adelante nos permite generar una búsqueda ordenada lo que permite controlar una memoria activa en un dispositivo autónomo.

#### **4.2.3. Análisis de resultados.**

Los datos obtenidos nos demuestra que un método de planificación adecuado para aplicar en el método heurístico DFS, el cual resulto como la mejor opción entre los métodos heurísticos y no heurístico analizados en el capítulo anterior, es la planificación de búsqueda hacia delante la cual cumple con las características requeridas por el experimento, como resultado final obtenemos un solo conjunto de resultados entre los algoritmos propuesto el algoritmo DFS como mejor algoritmo heurístico para búsqueda y aprendizaje, y la búsqueda hacia delante como un adecuado método de planificación.

### 4.3.Comprobación de Hipótesis.

*La hipótesis planteada es:*

**H1:** El análisis de los algoritmos heurísticos permitirá aplicar una adecuada planificación para el aprendizaje y guía de mecanismos autónomos.

#### 4.3.1. Resultados de mejor algoritmo en la búsqueda y aprendizaje.

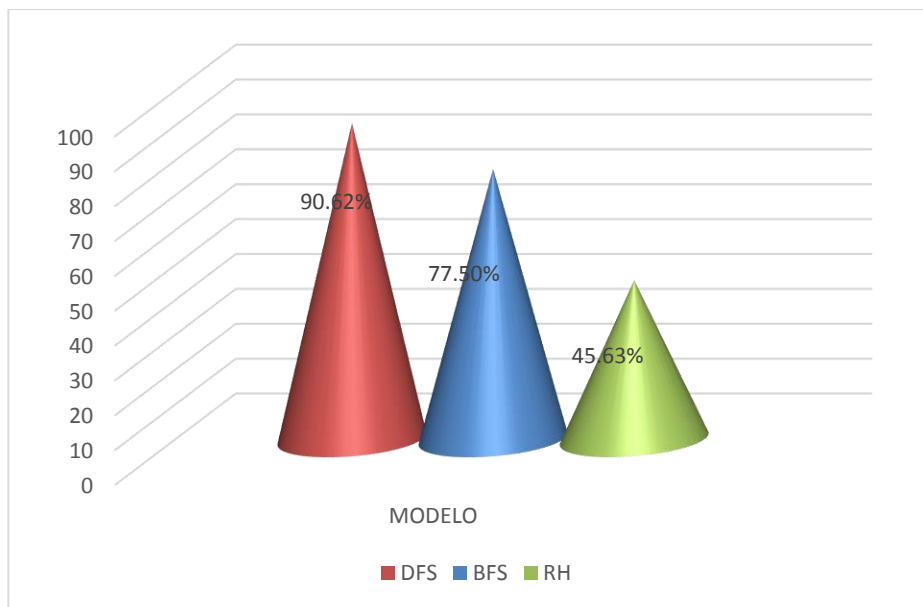
**Tabla IV 5: Resultados de porcentajes totales (Cumplir hipótesis).**

CLASIFICACION	PARAMETROS	INDICADORES	DFS	BFS	HR
Calidad	Eficiencia	• Numero de nodos visitados	16	9	8
		• Numero de nodos respuesta	19	19	12
		• Nodos resultado	19	16	12
	Tiempo	• Tiempo Individual	16	13	16
		• Tiempo exacto	19	19	1
Comportamiento real	Inteligencia	• Aprendizaje	20	20	1
Administración	Uso de la CPU	• Uso de la CPU individual	17	11	11
		• Promedio de uso total de la CPU.	19	17	12
Totales:			145	124	73



**Tabla IV 6: Resultados totales (Cumplir hipótesis).**

TOTAL		
	Valor	Porcentaje
DFS	145	90.62
BFS	124	77.50
HR	73	45.63



**Ilustración IV 3: Grafica de resultados final del análisis.**

#### **4.3.2. Resultado del análisis heurístico aplicado al aprendizaje y guía de mecanismos autónomos.**

Haciendo una referencia a los datos obtenidos en la tabla IV 5 obtenemos como resultado que los algoritmos heurísticos presentan una ventaja a la hora de ser usado como algoritmos base en el diseño de software para sistemas inteligentes que sean autónomos, con soporte en los resultados obtenidos en el capítulo IV, seleccionando un adecuado método de planificación, concluyo, que el análisis de los algoritmos heurísticos permitió encontrar un adecuado método de planificación para el aprendizaje y guía de mecanismos autónomos.

## **CAPÍTULO V:**

### **5.1. DESARROLLO DEL SISTEMA DIDÁCTICO “LABERINTO VIRTUAL”.**

El Aprendizaje Automático es una disciplina dentro de los Sistemas Inteligentes cuya importancia ha quedado establecida desde hace dos décadas.

De las variadas formas que se han explorado, la que mejor imita el aprendizaje humano es la que toma en los sistemas con aprendizaje por observación y descubrimiento.

El principio de que los programas de aprendizaje permiten comprobar teorías de aprendizaje y define los sistemas de producción adaptativos como sistemas que aprenden nuevas reglas basadas en lo que han observado, agregando estas nuevas reglas al sistema de producción existente.

Los sistemas autónomos operan el aprendizaje y planificación mediante la observación de las consecuencias de ejecutar acciones planeadas en un ambiente.

Los resultados observados en el actual análisis nos permiten enfocarnos en el desarrollo de un sistema que aproveche al máximo la investigación y desarrollar un sistema didáctico con la capacidad de generar múltiples escenarios y que sea capaz de resolverlos usando un algoritmo seleccionado por el usuario.

Este sistema está dirigido al uso en laboratorios para la simulación de laberintos, para conocer de cerca la importancia de aprovechar una investigación de este tipo y

aprovecharla al máximo en la elaboración de sistema que provean de inteligencia a mecanismos autónomos.

El estudio y resultado de la investigación está destinada a fortalecer las bases de la enseñanza acerca de las inferencias lógicas dentro de la inteligencia artificial en la materia de Bases del Conocimiento en la Escuela de Ingeniería en Sistemas de la ESPOCH.

## **5.2. Ingeniería de la Información**

La metodología usada para el desarrollo de la aplicación es SCRUM, una metodología que permite la gestión y desarrollo de software basada en un proceso iterativo e incremental utilizado comúnmente en entornos basados en el desarrollo ágil de software.

El modelo de desarrollo Scrum es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto.

Scrum, más que una metodología de desarrollo software, es una forma de auto-gestión del equipo de programadores. El grupo de programadores decide cómo hacer sus tareas y cuánto van a tardar en ello (Sprint). Scrum ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro.

## **5.3. Definición del ámbito.**

“Laberinto Virtual” es un sistema didáctico que permite medir las características ya estudiadas de los algoritmos heurísticos frente a los no heurísticos, la mayor parte de aplicaciones estos dirigidos al mejoramiento de los sistemas hardware, pero no se da la misma importancia al desarrollo de sistemas software, los mismos que son los encargados de proveer de inteligencia al mecanismo independiente, la interpretación de datos por parte del sistema no será compleja pero será lo suficientemente didáctica para demostrar las diferencias de los diferentes comportamientos entre sistemas con una arquitectura de adaptabilidad frente a otro que no se adapta solo reacciona por fuerza bruta.

El sistema contempla una programación modular con planificación a un crecimiento según mejore la investigación a futuro.

El sistema tiene que desempeñar las siguientes funciones:

1. Generar un laberinto independientemente de los ya generados anteriormente.
2. Cada laberinto se genera con la aplicación de un algoritmo específico y de un porcentaje en velocidad y dificultad definido por el usuario.
3. Se podrán llevar un control de los nodos involucrados en la resolución y del número de nodos generados.
4. La resolución del laberinto tendrá un punto de salida y un punto de llegada que serán creadas aleatoriamente.
5. El tiempo esta simulado a la realidad, esto quiere decir que es independiente a los eventos de los botones que están involucrados en el funcionamiento del sistema.
6. Se presentara estadísticas obtenidas solo de los experimentos almacenados temporalmente por parte del usuario.
7. Por referencia, el tiempo dado por el algoritmo no heurístico estará condicionado a los tiempos generados por los algoritmos heurísticos.
8. La memoria de ejecución es volátil, por tal motivo el historial de funcionamiento del CPU está disponible solo en el final de cada experimento.
9. El tiempo efectivo es opcional tanto para la estadística final como para las estadísticas independientes.

### **5.3.1. Estudio de Factibilidad**

#### **5.3.1.1. Factibilidad Técnica**

- **Recurso Humano**

Existe el recurso humano capacitado para realizar la administración del sistema, obteniendo una operación garantizada y uso garantizado.

- **Recurso Hardware**
  - ✓ Intel Core 2 Duo 2.1Ghz
  - ✓ DVD-RW
  - ✓ Teclado, Mouse, Monitor
- **Recurso Software**
  - ✓ *Sistema Operativo:* Microsoft Windows
  - ✓ *Tecnología de Visual Studio C#.*
- **Recurso Humano Requerido**

**Tabla V 1: Recurso Humano Requerido**

Función	Formación
<b>Jefe de Proyecto</b>	Ingeniería en Sistemas
<b>Equipo de desarrolladores</b>	Estudiante de Ingeniería en Sistemas
<b>Administrador de Hardware</b>	Ingeniería en Sistemas, Ingeniería Electrónica
<b>Diseñador</b>	Estudiante de Ingeniería en Sistemas

### **Factibilidad Operativa**

#### **Recurso Humano**

El recurso humano que participará del sistema son:

- Usuarios directos

Los usuarios directos a capacitar para el manejo del sistema son:

#### **Personal a Capacitar**

**Tabla V 2: Personal a Capacitar**

Nombre	Función
<b>U. Administrador</b>	Instructores de las cátedras que utilizaran el sistema desarrollado.
<b>U. Estudiante</b>	Estudiantes de las diferentes cátedras relacionadas con el estudio de la inteligencia artificial.

### Factibilidad Económica

El tiempo de duración de este proyecto será de 12 meses

### Costos

Tabla V 3:I Costo de Desarrollo

<b>Costos de desarrollo</b>		
<b>Costo personal (mensual)</b>		
<b>Jefe de proyecto y desarrollador</b>	\$800	\$9.600
<b>Diseñador</b>	\$160	\$3.200
<b>Total Costo Personal</b>		<b>\$12.800</b>
<b>Costo de hardware y software</b>		
<b>Costos Hardware</b>		
<b>Computadoras</b>	\$1200	\$2400
<b>Sistema NXT</b>	\$400	\$400
	<b>Total Costo Hardware</b>	<b>\$2800</b>
<b>Costo Software</b>		
<b>Internet</b>	\$400	
	<b>Total Costo Software</b>	<b>\$400</b>
<b>Costos varios</b>		
<b>Suministros</b>	\$300	
<b>Alimentación</b>	\$800	
<b>Papel A4</b>	\$10	
	<b>Total Costos Varios</b>	<b>\$1.210</b>
<b>COSTO TOTAL DE DESARROLLO</b>		<b>\$17210</b>

### Análisis Costo Beneficio

Los beneficios que se podrá obtener con la utilización del sistema desarrollado son los siguientes:

- Permitirá realizar un análisis heurístico en el proceso de dar inteligencia a un mecanismo autónomo.

- Se podrá obtener datos resultados de la aplicación de ciertos algoritmos de una forma virtual y se podrá obtener resultados de los diferentes experimentos.
- Mediante el sistema se puede generar resultados estadísticos que ayudarán a la toma de decisiones a los desarrolladores de sistemas inteligentes.

#### **5.4. Desarrollo del sistema “Laberinto virtual”**

El sistema “Laberinto virtual” está pensado para simular el desenvolvimiento de un mecanismo autónomo en un ambiente virtual, descrito como laberintos, cada laberinto generado tiene una descripción particular, con un grado de dificultad el cual puede ser manipulado por parte del usuario, cada uno de los laberintos generados se le conoce como experimento el cual para ser almacenado como estadística temporal necesita ser ejecutado por los tres algoritmos disponibles en la simulación.

El sistema nos muestra las ventajas de usar un algoritmo heurístico o no heurístico, nos permite rastrear las principales características cuantificadas y valoradas en escalas medibles y comparables, en la vida real simular un algoritmo de una magnitud muy grande sería una de las experiencias más demoradas y con valores no reales o muy pocos confiables, el sistema tiene varias funciones ya definidas anteriormente.

##### **1. Generar un laberinto independientemente de los ya generados anteriormente.**

La generación de los laberintos es por medio de los dos tipos de algoritmos backtrading (DFS, BFS), una vez seleccionado, el algoritmo a aplicarlo se selecciona la dificultad y la velocidad, esta acción nos entrega un numero de nodos y una forma aleatoria de diferentes caminos del laberinto.

Código de generación:

```
private void generadorlaberintoBusquedaProfunda(Celdas[,] arr, int anchura, int altura)
{
    Stack<Celdas> stack = new Stack<Celdas>();
    Random random = new Random();
    Celdas location = arr[this.aleatorio.Next(altura), this.aleatorio.Next(anchura)];
```

```
stack.Push(location);
    while (stack.Count > 0)
    {
        List<Point> neighbours = this.getNeighbours(arr, location, anchura, altura);
        if (neighbours.Count > 0)
        {
            Point temp = neighbours[random.Next(neighbours.Count)];
            this.localizacionActualGen = temp;
            this.GolpeaPared(arr, ref location, ref arr[temp.Y, temp.X]);
            stack.Push(location);
            location = arr[temp.Y, temp.X];
        }
        else
        {
            location = stack.Pop();
        }
        Thread.SpinWait(this.Sleep * sleepPeriod);
    }
    this.creaInicioFinLaberinto(this.laberinto);
}

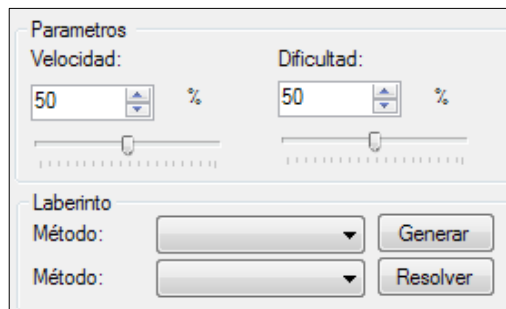
private void generadorLaberintoBusquedaEnAnchura(Celdas[,] arrLaberinto, int anchura, int altura)
{
    Queue<Celdas> queue = new Queue<Celdas>();
    Random random = new Random();
    Celdas location = arrLaberinto[this.aleatorio.Next(altura), this.aleatorio.Next(anchura)];
    queue.Enqueue(location);
    while (queue.Count > 0)
    {
        List<Point> neighbours = this.getNeighbours(arrLaberinto, location, anchura, altura);
        if (neighbours.Count > 0)
        {
            Point temp = neighbours[random.Next(neighbours.Count)];
            this.localizacionActualGen = temp;
            this.GolpeaPared(arrLaberinto, ref location, ref arrLaberinto[temp.Y, temp.X]);
            queue.Enqueue(location);
            location = arrLaberinto[temp.Y, temp.X];
        }
        else
        {
            location = queue.Dequeue();
        }
    }
}
```



```
}  
Thread.SpinWait(this.Sleep * sleepPeriod);  
}  
this.creaInicioFinLaberinto(this.laberinto);  
}
```

**2. Cada laberinto se genera con la aplicación de un algoritmo específico y de un porcentaje en velocidad y dificultad definido por el usuario.**

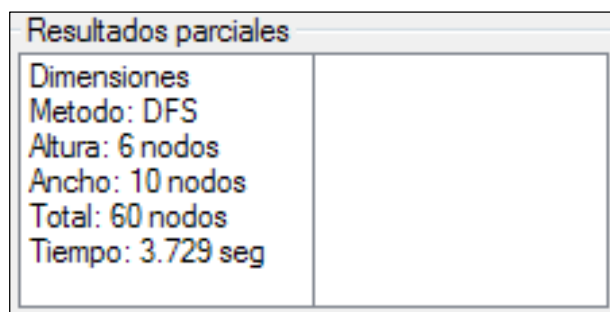
Los laberintos generados tienen que tener la capacidad de ser administrables tanto en su velocidad de generación como en la dificultad.



**Ilustración V 1: Parámetros para la generación de laberintos.**

**3. Se podrán llevar un control de los nodos involucrados en la resolución y del número de nodos generados.**

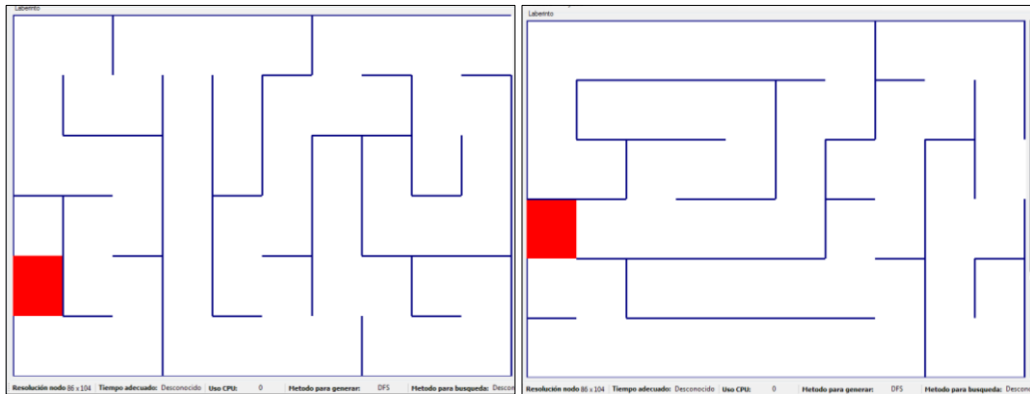
El número de los nodos involucrados dependen de la configuración inicial, la misma que puede ser manipulada al inicio por el usuario.



**Ilustración V 2: Descripción de la cantidad de nodos generados.**

**4. La resolución del laberinto tendrá un punto de salida y un punto de llegada que serán creadas aleatoriamente.**

Las salidas e ingresos de los laberintos son creados aleatoriamente, así cuentan con el mismo grado de dificultad y con la misma velocidad la generación de los laberintos cambia de una forma aleatoria.



**Ilustración V 3: Ejemplo de salidas de laberintos generados.**

**5. El tiempo esta simulado a la realidad, esto quiere decir que es independiente a los eventos de los botones que están involucrados en el funcionamiento del sistema.**

El tiempo es una de las medidas más importantes para la estadística final, la misma esta generada por medio de una clase la cual se activa automáticamente en el momento de la ejecución y se activa mientras la actividad se lleve a cabo, no estableciendo un tiempo no definido estrictamente.

**Código de la clase de control de tiempo:**

```
public class MicroTimer
{
    public delegate void MicroTimerElapsedEventHandler(
        object sender,
        MicroTimerEventArgs timerEventArgs);
    public event MicroTimerElapsedEventHandler MicroTimerElapsed;

    System.Threading.Thread _threadTimer = null;
    long _ignoreEventIfLateBy = long.MaxValue;
    long _timerIntervalInMicroSec = 0;
```

```
bool _stopTimer = true;

public MicroTimer()
{
}

public MicroTimer(long timerIntervalInMicroseconds)
{
    Interval = timerIntervalInMicroseconds;
}

public void Start()
{
    if (Enabled || Interval <= 0)
    {
        return;
    }
    _stopTimer = false;
    System.Threading.ThreadStart threadStart = delegate()
    {
        NotificationTimer(ref _timerIntervalInMicroSec,
            ref _ignoreEventIfLateBy,
            ref _stopTimer);
    };

    _threadTimer = new System.Threading.Thread(threadStart);
    _threadTimer.Priority = System.Threading.ThreadPriority.Highest;
    _threadTimer.Start();
}

public void Stop()
{
    _stopTimer = true;
}

public void StopAndWait()
{
    StopAndWait(System.Threading.Timeout.Infinite);
}

public bool StopAndWait(int timeoutInMilliSec)
{
    _stopTimer = true;
    if (!Enabled || _threadTimer.ManagedThreadId ==
        System.Threading.Thread.CurrentThread.ManagedThreadId)
```

```
{  
    return true;  
}  
return _threadTimer.Join(timeoutInMilliSec);  
}  
  
public void Abort()  
{  
    _stopTimer = true;  
  
    if (Enabled)  
    {  
        _threadTimer.Abort();  
    }  
}  
}
```

**6. Se presentara estadísticas obtenidas solo de los experimentos almacenados temporalmente por parte del usuario.**

Las estadísticas son la forma de analizar los resultados y cuantificar los valores obtenidos, cada uno de los experimentos consta de tres generaciones dos que son de los algoritmos no heurísticos y el no heurístico.

#	Algoritmo	N.Vis	N.Tot	N.Cam	Tiem(s)	Vel(%)	Dif(%)	Efec	CPU%
1	BFS	36	45	21	3.324	50	10	58	53.396
2	DFS	36	45	21	4.088	50	10	58	54.969
3	Hand Right	34	45	0	4.850	50	10	61	55.333

**Ilustración V 4: Ejemplo de estadística almacenada temporalmente.**

**7. Por referencia, el tiempo dado por el algoritmo no heurístico estará condicionado a los tiempos generados por los algoritmos heurísticos.**

Los algoritmos no heurísticos no tiene un control de los nodos que visitan por esta razón no se puede tener un numero de nodos de referencia para obtener las estadísticas necesarias, por tal motivo se tomado como referencia el número de nodos generados en

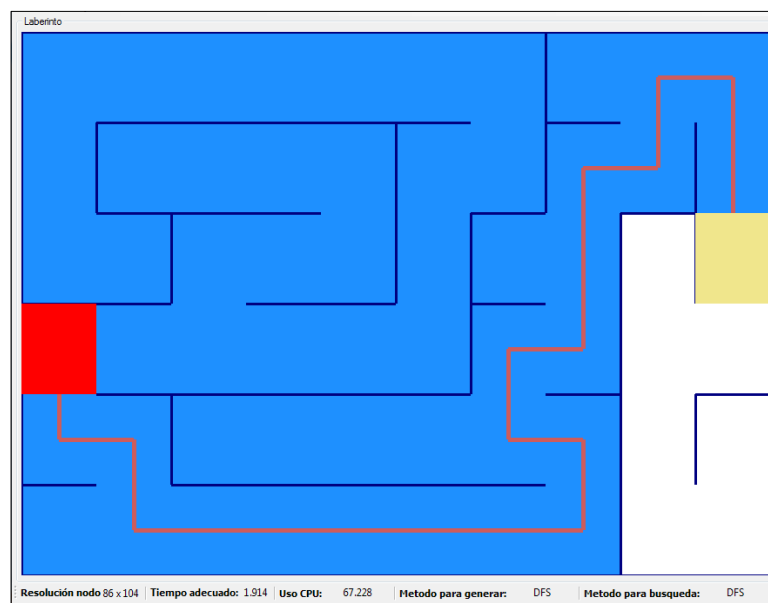
las ejecuciones de los otros algoritmos heurísticos, los cuales si tienen un número de nodos respuesta. El problema en este tipo de cuantificación de datos es que los datos obtenidos por parte del algoritmo no heurístico, no son del todo reales y que posiblemente den datos que solo sirvan con valores de comparación en el sistema.

**8. La memoria de ejecución es volátil, por tal motivo el historial de funcionamiento del CPU está disponible solo en el final de cada experimento.**

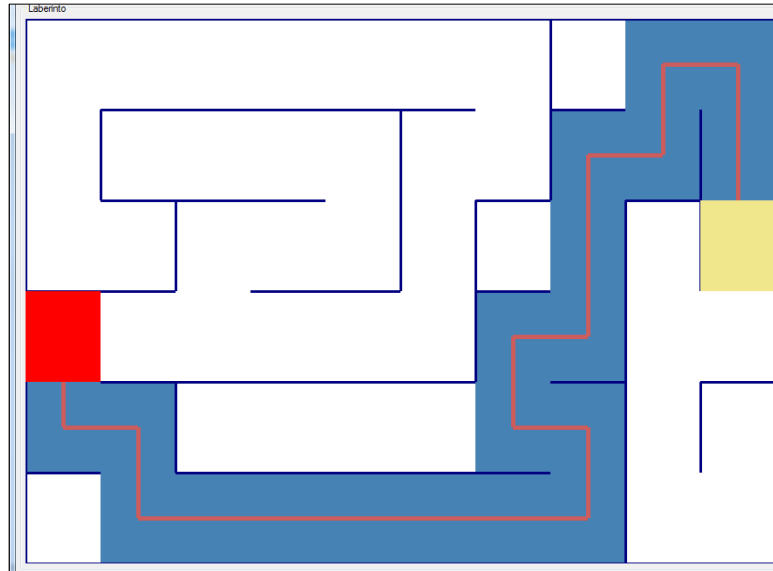
Cada experimento usa el CPU de una forma diferente, el objetivo es mediante un estudio verificar cual es el adecuado, así que tomando en cuenta una tasa promedio y los rangos máximos de la gráfica generada por el uso del procesamiento central, podemos medir cual es un algoritmo adecuado para aplicación en un mecanismo autónomo.

**9. El tiempo efectivo es opcional tanto para la estadística final como para las estadísticas independientes.**

El tiempo al que hemos denominado tiempo efectivo es el tiempo que toma atravesar el laberinto sin necesidad de una búsqueda de camino porque el camino está memorizado y lo que hace el sistema es guiarse con las coordenadas almacenadas.



**Ilustración V 5: Ejemplo en que el camino es encontrado normalmente.**



**Ilustración V 6: Ejemplo en que el camino es encontrado usando la memoria del sistema.**

### **5.5. Diseño y desarrollo de las clases contenedoras de los algoritmos estudiados.**

Como parte de las especificaciones iniciales, el análisis consiste en ver la mejor alternativa a la hora de diseñar el software que proveerá de inteligencia a un mecanismo autónomo, el algoritmo que resulto el más óptimo es el algoritmo DFS (Búsqueda en profundidad) que es una especialización del algoritmo backtraining.

## **6. Resumen.**

El sistema hace uso de la clase desarrollada, la clase “laberinto.cls” la misma que consta de los métodos que son parte de nuestra investigación, como por ejemplo los métodos backtraining (DFS y BFS) y el método no heurístico hand right (HR Mano derecha), el sistema tiene la capacidad de crecer ingresando nuevos método se puede y de esta manera otros algoritmo para el estudio comparativo.

El sistema es modular, cada módulo consta de características operativas independientes, de tal modo que si se ingresa nuevos método solo necesitaríamos describirlos en la parte visual del proyecto para poder estudiarlos y aplicarlos.

**Nota:** El código generado se encuentra en Anexos 2, presentados como parte de tesis.

## 7. Análisis del Sistema

### 7.1. Definir Casos de Uso esenciales en formato extendido

#### CASO DE USO ADMINISTRACION

Tabla V 4: Caso de Uso Autenticación

<b>IDENTIFICAR CASO DE USO:</b>	CU_Administracion
<b>NOMBRE DEL CASO DE USO</b>	Administración de experimentos
<b>ACTORES</b>	Estudiantes y Docentes
<b>PROPÓSITO</b>	Presentar el funcionamiento del sistema desarrollado
<b>VISIÓN GENERAL</b>	Definir el funcionamiento básico del sistema y los pasos para realizar las medidas experimentales
<b>TIPO</b>	Primario, real y expandido
<b>REFERENCIAS</b>	Requerimientos
<b>CURSO TÍPICO DE EVENTOS</b>	
<b>ACCIÓN DE LOS ACTORES</b>	<b>RESPUESTA DEL SISTEMA</b>
1. Cuando los usuarios ingresan a manipular el sistema.	2. Ingresa directamente no hace falta identificación.
3. El usuario ingresa los parámetros requeridos	4. Solicita parámetros de ingresos
	5. Genera el laberinto de acuerdo con los parámetros y el algoritmo seleccionado.
<b>CURSOS ALTERNATIVOS</b>	
2.1.- El sistema no presenta pantalla para ingreso de datos	

Mensaje de error

4.1.- “Genere nuevo laberinto”

4.2.- Tiempo no almacenado

Mensaje de advertencia

4.3.- “Desea registra en memoria temporal”

4.4.- “Ingrese tiempo eficiente”



Ilustración V 7: Diagrama caso de Uso

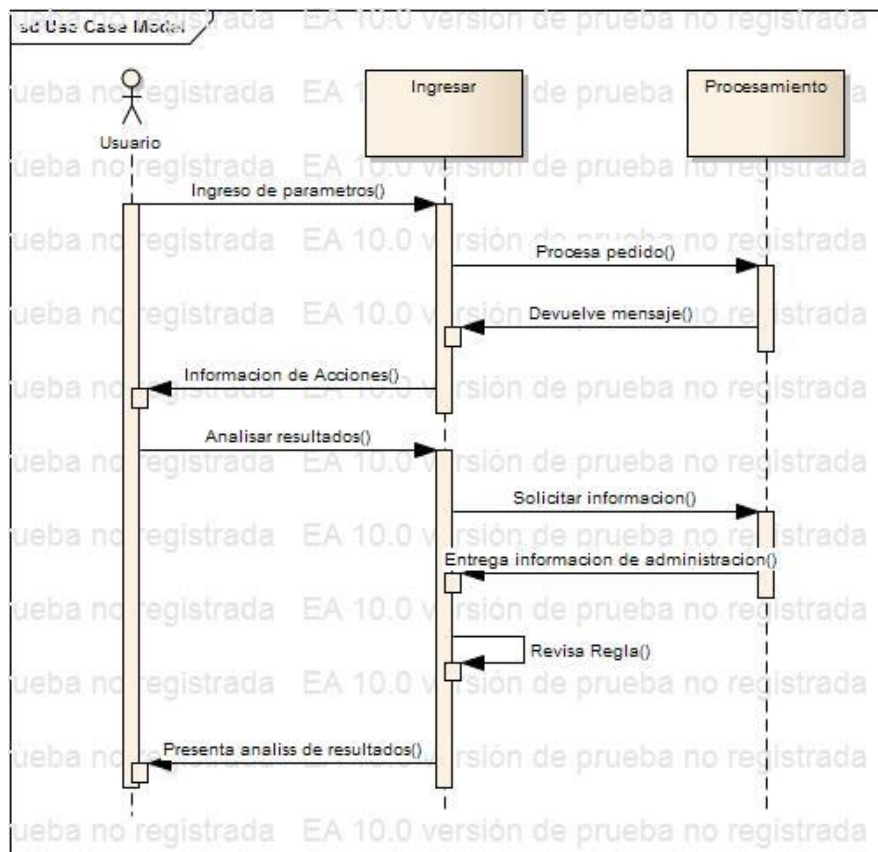
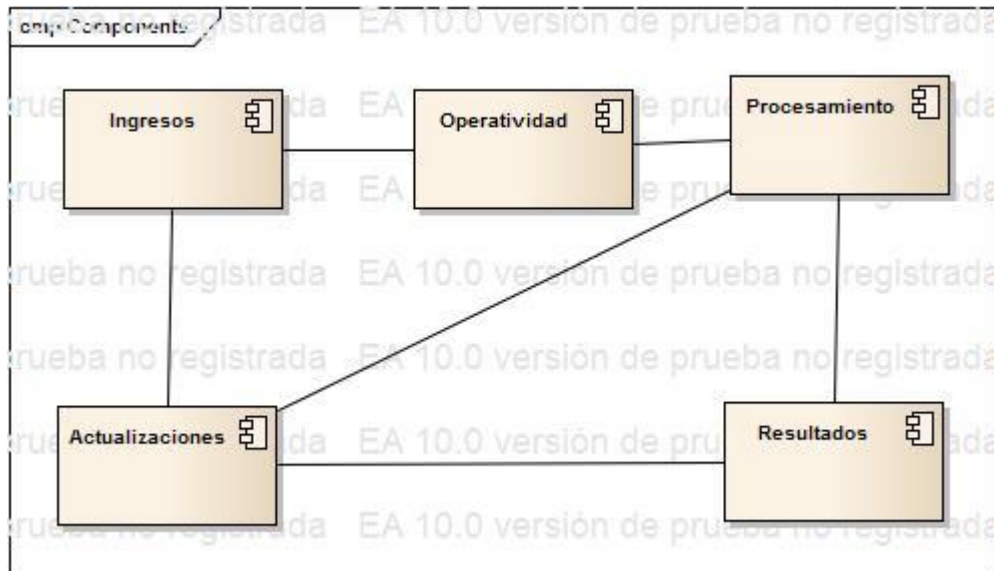


Ilustración V 8: Diagrama de secuencia.





**Ilustración V 9: Diagrama de componentes.**

## **CAPÍTULO VI: IMPLEMENTACIÓN DEL ALGORITMO PARA LA GUÍA Y APRENDIZAJE EN UN MECANISMO AUTÓNOMO.**

### **6.1. Implementación.**

Una vez analizados los diferentes algoritmos propuesto el siguiente paso es verificar la efectividad instalándolo en un mecanismo autónomo, para esta parte del experimento usaremos, el lenguaje de programación *robotC*.

### **6.2. RobotC**

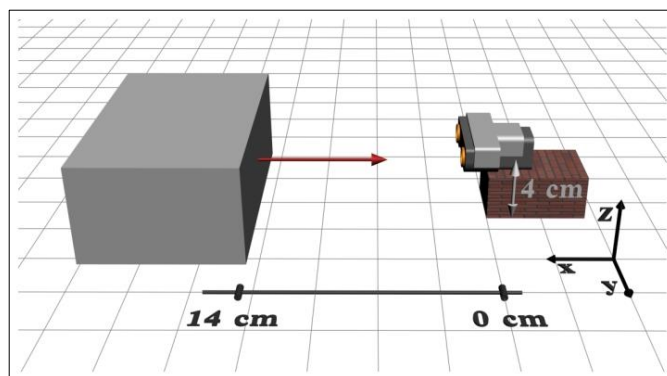
RobotC es un lenguaje para robótica educacional basado en C, con un entorno de desarrollo bastante sencillo. RobotC da soporte a diversas plataformas de robótica, entre ellas a LEGO Mindstorms NXT. El objetivo de esta nueva serie de artículos sobre RobotC es acercar a la gente a este entorno, que nos permite crear programas más complejos que NXT-G con bastante facilidad. [18]

### **6.3. Descripción.**

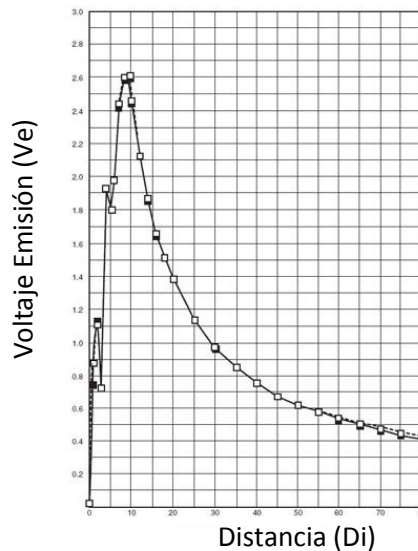
Una vez analizado la eficiencia de un algoritmo heurístico, se dispuso a implementar lo en un mecanismo LEGO Mindstorms NXT ya definido anteriormente, para que analice una salida y comprobar su aprendizaje, se desarrolló un escenario con dos posibles salidas. El sistema desarrollado tiene la capacidad de proveer al mecanismo de una forma

de guiarse y una vez que este encontrado el camino la capacidad de encontrar el camino sin necesidad de volver a buscar un camino de salida.

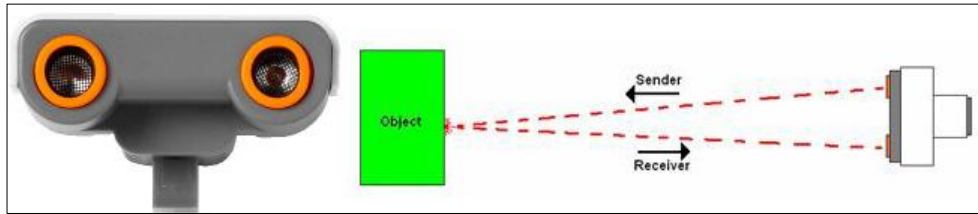
El escenario tiene una disposición de 4x3 en donde el autómata se desarrollara, el escenario cuenta con una gama de colores y una especie de guía, el motivo es que en la vida real el autómata tendrá la capacidad de explotar al máximo sus periféricos de entrada, en este caso los ingreso son un sensor de color y un sensor ultrasónico para medir la distancia.



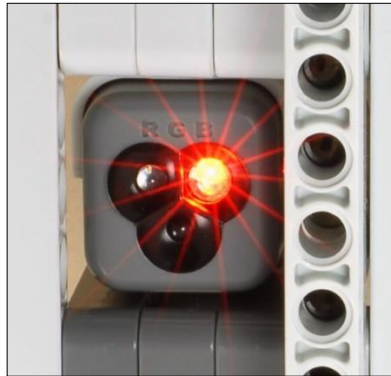
**Ilustración VI 1: Sensor ultrasónico.**



**Ilustración VI 2: Rangos en la medición de varias distancias (Cuerpo a 10cm).**



**Ilustración VI 3: Funcionamiento del sensor**



**Ilustración VI 4: Sensor de color.**

#### **6.4. Desarrollo experimental.**

En nuestro proyecto, trabajamos con un agente bien informado de lo que pasa también conocido como agente autónomo o mecanismo autónomo con capacidad de aprendizaje.

A continuación describimos nuestro autónomo.

El agente con memoria además de tratar de alcanzar el objetivo en base a una tabla de percepción acción.

Tiene en cuenta los siguientes elementos:

- Estado
- Cómo evoluciona el mundo
- Lo que hacen mis acciones

Agente está constituido por:

- Sensores
- Actuadores
- Metas

Percepciones: el agente tendrá un sensor que identificara el estado en que se encuentra y comunicara al robot la instrucción que debería tomar.

Acciones: Las acciones emprendidas por el agente son:

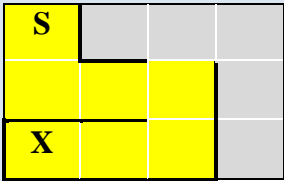
- Avanzar derecha
- Avanzar izquierda
- Avanzar adelante
- Avanzar giro total.

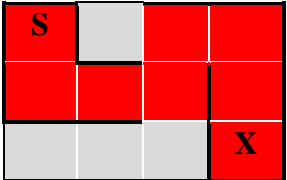
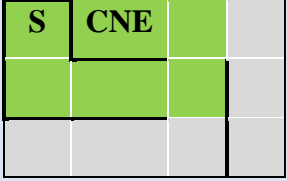
La representación de las acciones mediante el movimiento que permite el robot desarrollado.

**Tabla VI 1: Acciones tomadas por el autómata.**

Traducción maquina	Acción	Símbolo grafico
<b>AA</b>	Avanza Adelante	Camina hacia a delante
<b>AI</b>	Avanza Izquierda	Gira a la Izquierda
<b>AD</b>	Avanza Derecha	Gira a la derecha
<b>AG</b>	Avanza Giro Total	Cuando no allá camino realiza un giro de 180 grados.

**Tabla VI 2: Descripciones de posibles caminos.**

Grafico	Descripción	Costo (# Nodos)	Costo Total (# Nodos)	Porcentaje de éxito (%)
	Aquí vemos el primer camino solución, el desplazamiento que tiene que hacer el autómata es directo y no	<b>7</b>	<b>7</b>	<b>100</b>

	encuentra desafío por la forma base del algoritmo implementado.			
	En el segundo camino solución hay un regreso porque el algoritmo ingresa a buscar el camino al primer callejón.	<b>8</b>	<b>11</b>	<b>73</b>
	En el siguiente camino no existe memoria porque este es una muestra que n hay solución.	<b>0</b>	<b>0</b>	<b>0</b>

### 6.5. Características del autómata

Nuestro agente inteligente, tiene las siguientes características:

Autonomía, reactivo y proactivo.

- La autonomía, entienda como, la capacidad del agente de tomar sus propias decisiones basado más en sus experiencias que en su conocimiento dado por el programador.
- El agente solucionador de laberintos, almacena sus experiencias en forma de reglas e infiere sus conocimiento, utilizando el encadenamiento adelante.
- El agente es reactivo, ya que una vez modificado el entorno cambia de posición en el escenario.
- Este inmediatamente genera un proceso nuevo para la solución el problema.
- El agente es proactivo, pues aprovecha cada oportunidad presentada en el ambiente para lograr sus objetivos.



**Ilustración VI 5: Agente o mecanismo autónomo.**

## **6.6. Medida de rendimiento**

Nuestro agente, utiliza como medida de rendimiento, un valor de 1 si logra avanzar un cuadro en el escenario en forma segura. Recibirá una penalización de 0 en una variable “*memoria*” en caso de estar encerrado y emitirá un sonido de encierro, en este caso el agente está diciendo que ha caído en un camino sin salida, en el momento que encuentre su recompensa (en este caso es la detección del color negro) la variable “*memoria*” cambiara a un uno lógico.



**Ilustración VI 6: Agente o mecanismo autónomo usado para el experimento.**

## 6.7. Implementación

El algoritmo general para la implementación del agente de reflejo simple es:

**Tabla VI 3: Algoritmo general.**

<b>Algoritmo general del agente.</b>
<b>1. Leer archivo de configuración inicial (.ini)</b>
<b>2. Generar ambiente aleatorio</b>
<b>3. Iniciar agente</b>
<b>4. Si condición de parada = 1.</b>
<b>a. Termina</b>
<b>b. Si no ir a 5</b>
<b>5. Leer percepciones del estado actual</b>
<b>6. Realizar acción según tabla. Ir a 4</b>

**Tabla VI 4: Algoritmo general con memoria activa.**

<b>Algoritmo general con función de la memoria.</b>
<b>1. Leer archivo de configuración inicial (.ini)</b>
<b>2. Generar ambiente aleatorio</b>
<b>3. Iniciar agente</b>
<b>4. Si condición de parada = 1.</b>
<b>a. Termina</b>
<b>b. Si no ir a 5</b>
<b>5. Leer percepciones del estado actual</b>
<b>6. Realizar acción según reglas</b>
<b>7. Calcular estadísticos</b>
<b>8. Actualizar estado interno. Ir a 4.</b>

La implementación del algoritmo DFS el cual después el análisis resulto como mejor opción, está realizado con código C elaborado en la herramienta RobotC.



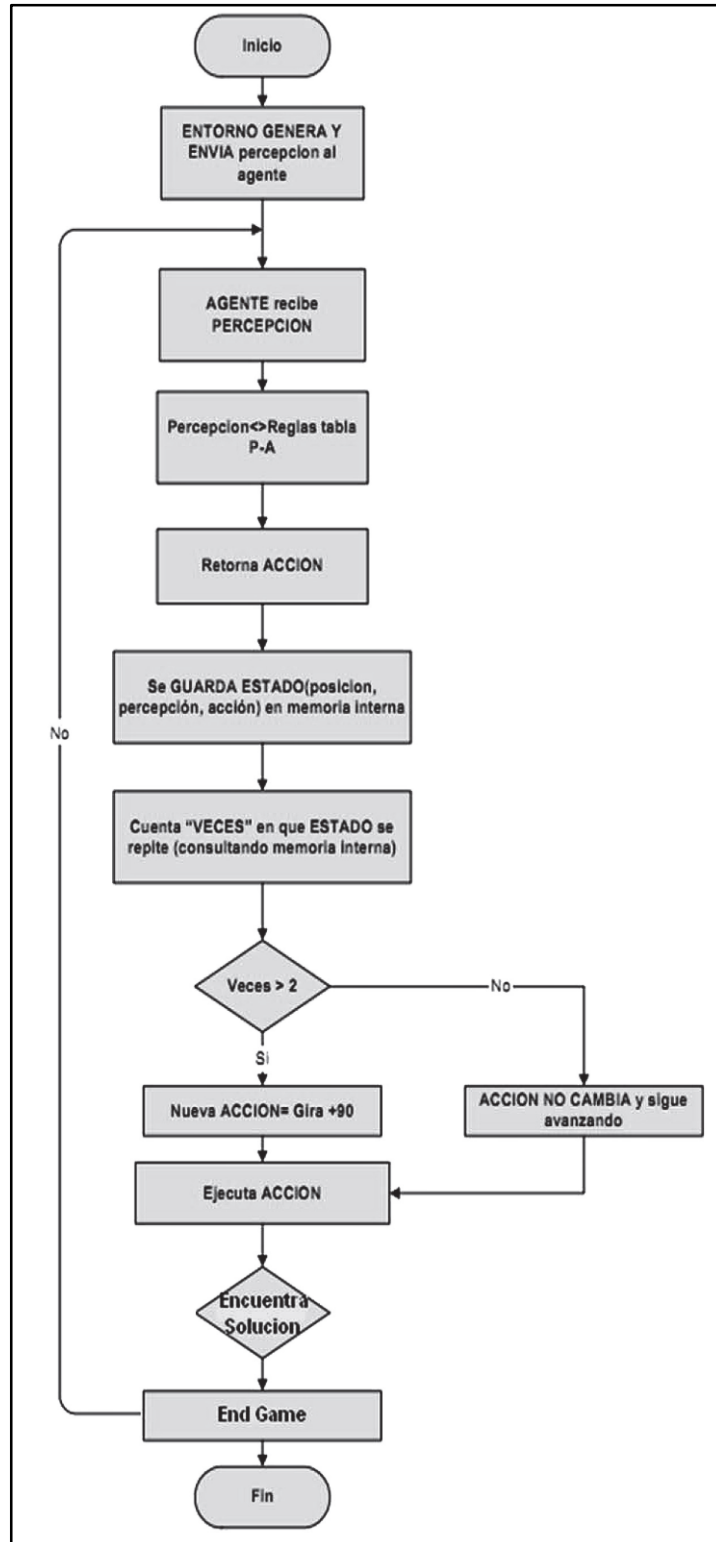
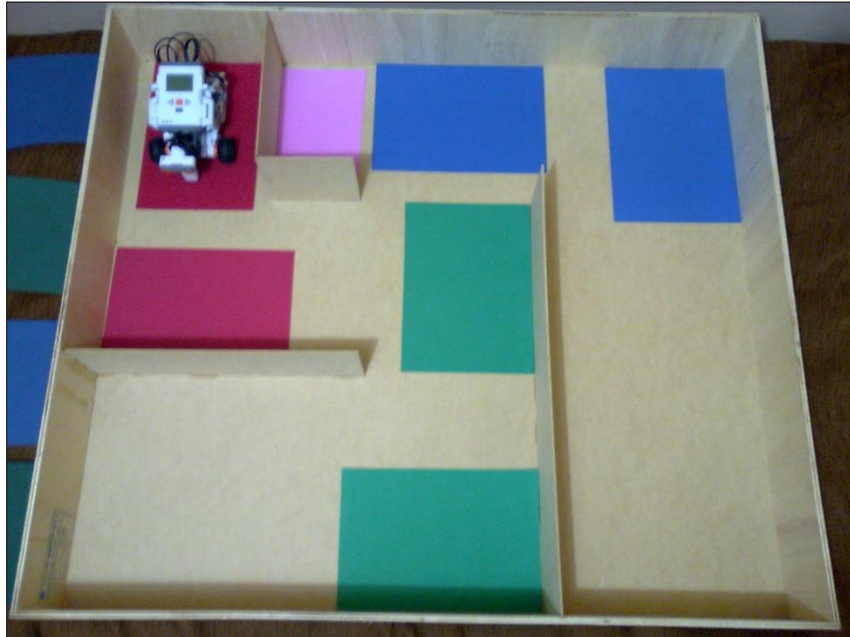
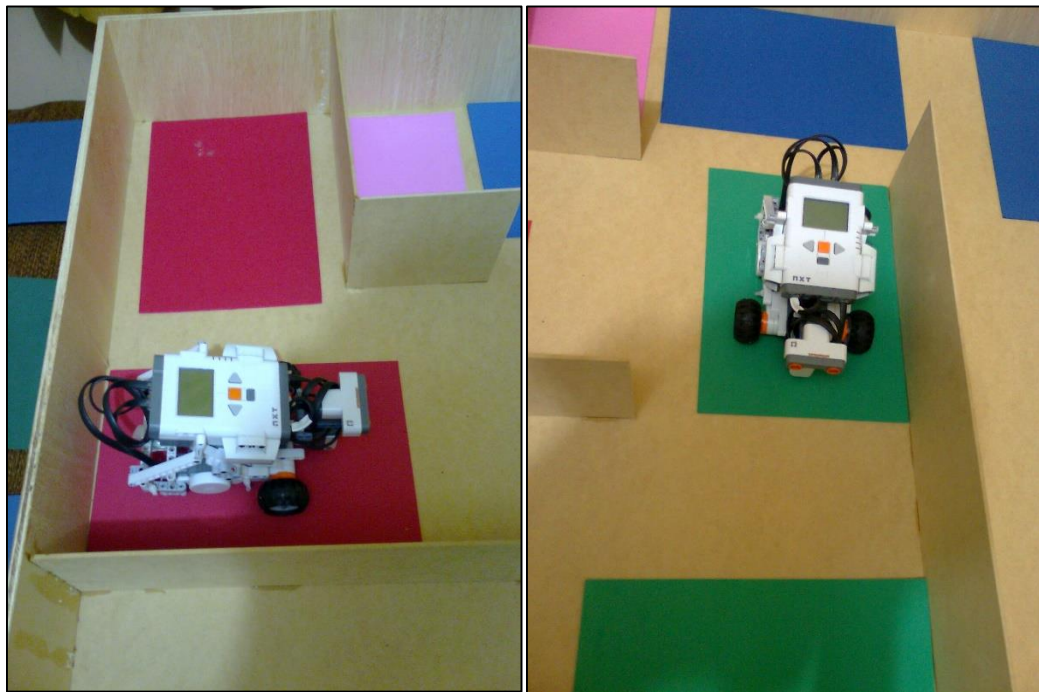


Ilustración VI 7: Diagrama general del agente.

## 7. Funcionamiento del código implementado



**Ilustración VI 8: Autómata en el inicio del escenario planteado.**



**Ilustración VI 9: Primer obstáculo a sortear.**



**Ilustración VI 10: Autómata encontrando la salida.**

El código representa el desarrollo de un sistema con recursividad infinita cuyo único final será cuando encuentre una solución, esta característica viene del uso del algoritmo DFS como base de la arquitectura la cual da la capacidad de aprender y guiarse en el escenario propuesto.

### **7.1. Descripción.**

Una vez instalado el sistema software en el autómata los agentes empiezan a funcionar e interactuar con el escenario propuesto, los colores en el laberinto son guías para memorizar, el sistema parte reconociendo el color rojo como color de inicio, y los colores azul y verde como colores de guía, el final es reconocido con el color negro.

## **8. Implementación y Pruebas**

### **8.1. Definición de estándares de Programación**

Para realizar la codificación del sistema se han definido los siguientes estándares:

- El nombre de las clases se encuentran con las iniciales cls al inicio.
- El nombre de las variables tienen las iniciales var al inicio.

#### **8.1.1. Pruebas Unitarias**

Para asegurar el correcto funcionamiento del Sistema se han probado sus métodos de forma independiente, enviando datos de entrada desde el código, para luego obtener los a través de los diferentes métodos para realizar distintas mediciones. Se han probado especialmente todas las funciones para validaciones de datos.

#### **8.1.2. Pruebas de Módulos y del Sistema**

Las pruebas finales consistieron en verificar que la información ingresada se vea inmediatamente reflejada en los diferentes experimentos, esto sirve para comprobar que el procesamiento funciona correctamente.

Se provocaron errores intencionales para verificar el correcto funcionamiento del sistema, así como de las funciones de validación de datos, como por ejemplo:

- Realizar demasiado procesamiento
- Eliminar experimentos

Tratar de ingresar parámetros fuera de los márgenes correctos, como tratar de ingresar nuevos datos luego de registrarlos en la memoria temporal, tratar de eliminar experimentos antes de ingresar uno no registrado.

## CONCLUSIONES

1. El experimento presento mejores resultados en un ámbito con menores obstáculos, por otra parte gracias a las características propias del algoritmo aplicado en la guía y el modelo de planificación seleccionado, una de las características principales que adopta un mecanismo autónomo con un sistema inteligente desarrollado con base en el presente resultado, es si existe un camino solución lo encontrara no importando la complejidad y mientras lo busca, el sistema ira aprendiendo el camino.
2. El uso de la memoria del agente es indispensable para evitar que el agente llegue a ciclos repetitivos infinitos y sobre todo para dar al mecanismo autónomo independencia en su trabajo, tal objetivo se ve reflejado en los resultados del parámetro inteligencia, dando como muy adecuado a los algoritmos backtraking con un 100% de adaptación a la memorización frente al 0% de adaptación obtenido por el algoritmo no heurístico.
3. El uso de reglas de como forma de representación del conocimiento es la más adecuada para este tipo de agente ya que el objetivo es llegar a la meta o recompensa siendo indiferente del costo computacional y teniendo en cuenta la planificación adecuada para dicho objetivo, resultado “muy adecuada” la planificación de búsqueda hacia delante con un 62,5% frente al 52% obtenido en el análisis por el método de planificación de búsqueda “hacia atrás” con un 25%.
4. El análisis realizado demostró que la hipótesis se cumple con éxito, los algoritmos heurísticos son una forma adecuada para dar inteligencia a los mecanismos autónomos, DFS es un algoritmo que permite versatilidad en la guía y en el aprendizaje, obteniendo un porcentaje total de un 90,62% frente al 45,63% obtenido por HR.
5. Una vez instalado el sistema en el agente autómatas demostró gran desenvoltura en el escenario escogido, el parámetro administración del CPU con 90% obtenido por el algoritmo DFS, le permite al mecanismo aprovechar al máximo las conexiones que existen entre los agentes y la base del mecanismo que es la CPU del sistema.

6. El hombre ha buscado una forma de aprovechar al máximo todos los recursos a su disposición, cuando la informática dio a luz una nueva forma de procesar la información, los saltos hacia depender de aparatos informáticos fueron cada vez más grandes, al punto de depender de este tipo de mecanismos y cuando hablamos de inteligencia artificial siempre se calcula en cómo desarrollar los más sofisticados sistemas hardware, el desarrollo de un sistema software con la capacidad de explotar al máximo la forma de dar inteligencia al mecanismo autónomo, es la única forma de llegar a desarrollar un sistema que pueda aprender, y así cumplir con el sueño que la humanidad ha tenido desde que dio a luz el primer sistema autónomo.

## RECOMENDACIONES

- ✓ Los avances tecnológicos en el área de Sistemas y Electrónica van de la mano, trabajar en grupo para intercambiar experiencias será una gran forma de obtener grandes avances no solo con materiales didácticos sino también con estudios aplicables y publicables.
- ✓ Para aplicar los resultados obtenidos en un análisis como el realizado en la presente investigación, debemos tomar muy en cuenta las características del dispositivo en el que vamos a instalar el sistema software desarrollado, de lo contrario podríamos enfrentarnos a fallas imprevistas, ya que las mismas serán fallas muy ajenas al software.
- ✓ La mayoría de las herramientas que se utiliza para el desarrollo de sistemas inteligentes están orientados a las grandes empresas, por tal motivo tienen un costo de utilización, no siempre a un precio accesible para la didáctica académica, una investigación mucho más exhaustiva en este campo de la tecnología, en politécnicas y universidades como la nuestra, mejoraría el acceso a este tipo de tecnología en nuestra localidad, estudiantes y docentes podrían hacer uso de herramientas generadas por investigaciones nacionales y de esta manera aprovechar al máximo de las enormes ventajas de la inteligencia artificial.
- ✓ La inteligencia artificial nació como una forma de imitar la inteligencia humana, por tal motivo las investigaciones en este campo deberían basarse en como el cerebro humano ve el mundo exterior, aunque el cerebro es una máquina perfecta, tiende a facilitar las cosas para aprender y analizar los problemas a los que se enfrenta día a día, muchos autores proponen el uso de una máquina matemática para que aprenda de un forma numérica, de esta manera reconocer el mundo exterior a pura lógica matemática, aunque siempre ha resultado ser un buen método hay que recordar que el cerebro humano se ubica y procesa todo por medio de objetos.

## RESUMEN

La investigación sobre el análisis de los algoritmos heurísticos orientado a la guía y aprendizaje de mecanismos autónomos, permite la selección de una base adecuada para el desarrollo de un sistema que provea de inteligencia y autonomía en robots autónomos, simulaciones y experimentos llevados a cabo en la Facultad de Electrónica de la Escuela Superior Politécnica de Chimborazo.

La presente investigación de análisis se sustenta en el método científico, con técnicas de observación directa y experimentación mediante softwares de simulación desarrollados en Visual Studio.net (C#) e implementaciones en mecanismos autónomos NXT 2.0, Robot C.

El análisis de algoritmos principal tópico en la investigación, que fue integrado en Robot C y su respectiva simulación en Visual Studio .net, para generar un software con base en un algoritmo heurístico, permitiendo que el mecanismo aprenda de una forma rápida fiable y óptima, dando como resultado cuantitativo, obtenido de la simulación de escenarios, con un 45% de efectividad de los algoritmos no heurísticos frente al 95% de los algoritmos heurísticos.

Los resultados obtenidos en la simulación, permiten concluir que el desarrollo de sistemas inteligentes usando un algoritmo heurístico es ideal para la automatización del aprendizaje en sistemas autónomos obteniendo un 45% de efectividad con respecto a los algoritmos no heurísticos, al fin de desarrollar un software inteligente que permita ser instalados en los mecanismos autónomos desarrollados en la Escuela de ingeniería Electrónica en la Escuela Superior Politécnica de Chimborazo.

En conclusión, el uso de un algoritmo heurístico ofrece una base sólida para generar sistemas software que permitan a los mecanismos autónomos obtener inteligencia y guiarse de una manera efectiva en cualquier tipo de escenario.

Se recomienda a los estudiantes y docentes en la cátedra de Inteligencia Artificial en la Facultad de Informática y Electrónica, el uso de algoritmos heurísticos para sus experimentos, este tipo de algoritmo es versátil a la hora de aprender nueva información y óptima al momento de guiar los movimientos en mecanismos independientes.



## ABSTRACT

The current research on the analysis of the heuristic algorithms oriented to the guidance and learning of autonomous mechanisms, make possible the selection of an appropriate basis for the development of a system that provides intelligence and autonomy in self-sufficient robots, simulations and experiments conducted in the faculty of Electronics belonging to "Escuela Superior Politécnica de Chimborazo".

This research of analysis is based on the scientific method, with direct observation techniques and experimentation by means of simulation software, developed in Visual Studio.net (C#) and implementations in autonomous mechanisms NXT 2.0 with Robot C.

The Analysis of algorithms, main topic in this research, consisted in designing algorithms which were integrated with Robot C and their respective simulation; in Visual Studio.net in order to generate a software based on a heuristic algorithm, enabling the mechanism to learn in a fast, reliable and effective way giving as quantitative result, obtained in the simulation scenarios 45% of effectiveness of non-heuristic algorithms compared with 95% of the heuristic algorithms.

The outcomes collected from the simulation, support the conclusion that the development of intelligent systems by using a heuristic algorithm, is ideal for the automation of learning autonomous systems; obtaining a 45% de effectiveness regarding to non-heuristic algorithms, to develop an intelligent software, which allows to install them into the autonomous mechanisms, developed in the "Escuela de Ingeniería Electrónica (Electronic Engineering)" at "Escuela Superior Politécnica de Chimborazo".

To conclude, the use of a heuristic algorithm provides a strong foundation to create software systems that enable the autonomous mechanisms to achieve intelligence and be guided effectively in any sort of scenario.

It is recommended to students and teachers of Artificial Intelligence at the "Facultad de Informática y Electrónica", to use heuristic algorithms for conducting their experiments, such kind of algorithm is versatile when learning new information and it is optimal at the moment of guiding the movements in independent mechanisms.

## **BIBLIOGRAFÍA**

- [1]. **ALBERINO, S.**, Definiciones de Inteligencia artificial.,  
Buenos Aires-Argentina., Universidad de Buenos Aires.,  
Ebook.secyt.frba.utn.er/gia/inteligencia\_artificial.htm.
- [2]. **INTELIGENCIA ARTIFICIAL.**  
[http://es.wikipedia.org/wiki/Inteligencia\\_Artificial](http://es.wikipedia.org/wiki/Inteligencia_Artificial)  
2013-12-01.
- [3]. **NAIOUF, M.**, Modelo de interoperabilidad para sistemas autónomos en  
entornos distribuidos., Buenos Aires-Argentina., Universidad de Buenos  
Aires., Laboratorio de Sistemas Inteligentes., Facultad de Ingeniería., 2007., Pp  
2-8.
- [4]. **FORO DE INTELIGENCIA ARTIFICIAL.**  
[http://taringa.net/es.foro.org/234456/Que\\_es\\_la\\_heuristica](http://taringa.net/es.foro.org/234456/Que_es_la_heuristica)  
2013-11-21.
- [5]. **BANDERAS, F.**, Aplicación de procedimientos heurísticos orientados a la  
resolución de problemas matemáticos.,  
Nuevo León - México., Universidad Autónoma de Nuevo León., Facultad de  
Ciencias Física – Matemáticas., 1999., Pp 120-230.

- [6]. **FLORES, E.**, Gnoseología de la heurística.,  
2013-02-02.  
Ebook [arquepoetica.azc.uam.mx/heuristica.htm](http://arquepoetica.azc.uam.mx/heuristica.htm)
- [7]. **CORDOVA, M.**, Análisis Heurístico.,  
2013-05-05.  
Ebook [librosmaravillosos.com/enelereinodelingenio/capitulo16.html](http://librosmaravillosos.com/enelereinodelingenio/capitulo16.html).
- [8]. **GUILLEN, P.**, Resolución de laberintos.,  
2013-06-02.  
Ebook [pier.guillen.com.mx/alghortim/09-busqueda/09-4-caminoLaberinto.htm](http://pier.guillen.com.mx/alghortim/09-busqueda/09-4-caminoLaberinto.htm).
- [9]. **CASAS, C.**, Esquema Algorítmico del Backtracking.,  
2013-11-11.  
Ebook [uls.edu.sv/index.php?option=com\\_phocadownload&view=category&download=103:esquema-algoritmico-back-tranking.html](http://uls.edu.sv/index.php?option=com_phocadownload&view=category&download=103:esquema-algoritmico-back-tranking.html).
- [10]. **FLEIFEL, F.**, Robots Autónomos y Aprendizaje por refuerzo.,  
Ebook [fleifel.net/robotsaprendizaje.php](http://fleifel.net/robotsaprendizaje.php)  
2013-05-07.
- [11]. **APRENDIZAJE AUTOMATICO BASADO EN INTERCAMBIO DE OPERADORES EN SISTEMAS INTELIGENTES AUONOMOS.**, Buenos Aires-Argentina., Universidad de Buenos Aires., Facultad de Ingeniería., 2001.  
Ebook [librosfree.com/estuditesis.ar/aprendizaje\\$.automatico.php](http://librosfree.com/estuditesis.ar/aprendizaje$.automatico.php).

**[12]. OPERADORES MINDSTORM.**

[http://es.wikipedia.org/wiki/lego\\_mindstorm](http://es.wikipedia.org/wiki/lego_mindstorm)

2013-06-08.

**[13]. SENSORS.**

[http://roble.pntic.mec.es/amoc0048/nxt/dosmarcos/sensor\\_de\\_color.htm](http://roble.pntic.mec.es/amoc0048/nxt/dosmarcos/sensor_de_color.htm)

2013-06-08.

**[14]. C SHAP.**

[http://es.wikipedia.org/wiki/C\\_Sharp](http://es.wikipedia.org/wiki/C_Sharp)

2013-12-02.

**[15]. BENITEZ, J.,** Planificación de maniobras para barcos autónomos.,  
Madrid-España., Universidad Complutense de Madrid., Facultad de  
Informática., 2009., Pp 34-99.

**[16]. ALGORITMOS DE FUERZA BRUTA.**

[http://es.wikipedia.org/wiki/C3%baSquedda\\_de\\_fuerza\\_bruta](http://es.wikipedia.org/wiki/C3%baSquedda_de_fuerza_bruta)

2013-07-01.

**[17]. MATILDE, C.,** Técnicas de IA (Inteligencia Artificial) para problemas de  
planificación.,

Washington-USA., Universidad de Washington ., Departamento de Ciencias  
Informática e Ingeniería., 2010., Pp 3-24.

**[18]. MANTIS, G.,** Manejo básico de LCD y botones en RotobC.,

Ebook.blog.electricbricks.com/es/2012/03/tutorial-robot-lego-  
mindstorm#5090.

2013-07-02.