UNIVERSIDAD POLITECNICA DE VALENCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

PFC: Generación y Transmisión de órdenes de movimiento a un vehículo autónomo tipo QUAD

PROYECTO FINAL DE CARRERA

CARMEN ARBONA DE LA ASUNCIÓN
DIRIGIDO POR: ENRIQUE JORGE BERNABEU SOLER

A mi Director de Proyecto, Enrique Jorge Bernabeu Soler, por su dedicación y por darme la oportunidad de realizar este proyecto.

A mis padres, por el apoyo y la confianza que día tras día han ido depositando en mí dándome todos los medios y facilidades para llegar a este momento.

A mi hermana, por haber sido mi referente y modelo a seguir durante toda mi vida y de haber estado ahí siempre que he necesitado el apoyo y cariño de una hermana mayor.

A mi novio, por sus consejos, paciencia y sobre todo por haberme regalado momentos tan inolvidables durante estos años.

A mis amigos, compañeros de clase y compañera de prácticas, por haberme ayudado a superar los obstáculos que año tras año se iban presentando.

Por último, y no menos importante, a aquellas personas que, aún no pudiendo estar presentes, sé que estarían muy orgullosas de mí.

ÍNDICE

ÍNDICE DE IMÁGENES

ÍNDICE DE TABLAS

PARTE I: PROYECTO

1- RESUMEN

El objetivo principal de este Proyecto Final de Carrea es desarrollar una plataforma software que realice la conducción automatizada de un vehículo autónomo tipo Quad. Esta plataforma software transmitirá órdenes al sistema de frenado, aceleración y dirección para la conducción del citado vehículo. Esta aplicación tiene la finalidad de servir de soporte para el estudio de diferentes métodos de planificación de caminos, es decir, intenta ser una herramienta que facilite el estudio y mejoras de los distintos algoritmos de planificación.

Con este fin se ha hecho una introducción sobre el contexto actual de la robótica y en especial, sobre los vehículos móviles autónomos, grupo al que pertenece el vehículo principal de este proyecto: el Quad, el cual, lleva incorporado un sensor láser para la obtención de información de su entorno.

Este proyecto pretende ser una ayuda con el que poder comprobar visualmente los resultados obtenidos al aplicar sobre el vehículo un determinado algoritmo de planificación. Mediante esta plataforma se puede simular el funcionamiento que tendría el vehículo real al aplicarle dicho planificador.

2- PALABRAS CLAVE

- **Robótica:** "Ciencia o arte relacionada con la inteligencia artificial (para razonar) y con la ingeniería mecánica (para realizar acciones físicas sugeridas por el razonamiento)" [Asimov,1942]
- **Quad:** vehículo de cuatro ruedas, sin carrocería, con sistema de dirección mediante manillar, dotado de sillín donde el conductor va sentado a horcajadas.
- **Láser**: Dispositivo electrónico que, basado en la emisión inducida, amplifica de manera extraordinaria un haz de luz monocromático y coherente.
- **Java**: lenguaje de programación orientado a objetos (POO) desarrollado por "Sun Microsystems" a principios de los años 90. Se trata de un lenguaje fuertemente tipado y robusto.
- **Eclipse**: plataforma de desarrollo "open source" basada en Java. Actualmente está desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.
- **Cinemática Ackerman:** cinemática empleada en la mayoría de los vehículos de cuatro ruedas. Puede utilizar dos o cuatro ruedas motrices, pero únicamente utiliza dos ruedas directrices, normalmente delanteras.
- **Simulación:** técnica utilizada para representar o imitar el argumento de un problema real.

3- INTRODUCCIÓN

3.1- Descripción de los objetivos

El objetivo principal de este Proyecto Final de Carrea es desarrollar una plataforma software que permita la conducción automatizada de un vehículo autónomo tipo Quad. Esta plataforma software transmitirá órdenes al sistema de frenado, aceleración y dirección para la conducción del citado vehículo. Esta aplicación tiene la finalidad de servir de soporte para el estudio de diferentes métodos de planificación de caminos, es decir, intenta ser una herramienta que facilite el estudio y mejoras de los distintos algoritmos de planificación.

El resultado ofrecido por la aplicación deber ser por una parte visual, donde se pueda observar el recorrido seguido por el vehículo y el entorno en el que realiza dichos movimientos, y por otra parte analítico, permitiendo guardar los datos en un fichero para poder utilizarlos en otra aplicación o programa.

3.2- Motivación

Un proyecto final de carrera es un trabajo al cual se le tienen que dedicar muchas horas y de ahí a la importancia a que tema del proyecto elegido sea siempre motivador, ilusionante y del cual se puedan aprender cuántas más cosas mejor.

A la hora de elegir, tenía claro que quería un proyecto encarado hacia la informática industrial, rama sobre la cual he ido ampliando mis conocimientos a lo largo de estos cinco años y en la cual tengo decidido orientar mi vida profesional.

Este proyecto me permitía adentrarme un poco más en este mundo, conociendo algo más sobre los robots móviles y además me obligaba a profundizar en la programación de un lenguaje tan extendido e importante actualmente como es Java.

3.3- Contenido de la memoria

En esta memoria se desarrollan los pasos seguidos para la realización del Proyecto Final de Carrera. Con este motivo, he intentado darle una estructura que facilitase el entendimiento y el seguimiento de la misma. Está dividida en 5 apartados principales ordenados según la evolución del proyecto.

El primer apartado sirve para dar una visión general de los objetivos principales del proyecto, en qué consiste y la motivación del mismo. Además aporta al lector una visión general del proyecto desarrollando ítems relacionados con la robótica en general y el quad y la tecnología láser en particular con el fin de contextualizar al usuario en el trabajo desarrollado. Asimismo se explican los dos lenguajes utilizados, Java principalmente y HTML.

El segundo apartado titulado Análisis y Diseño de la herramienta contiene un estudio sobre los requerimientos funcionales y no funcionales, esto es, se fijan unos objetivos que debe cumplir la aplicación para considerarla apta para el fin para el cual ha sido diseñada. Además se incluyen un conjunto de diagramas UML con el fin de mostrar las pautas de diseño seguidas para la elaboración de la aplicación. Se han desarrollado los diagramas que se han considerado apropiados para este proyecto.

El siguiente apartado, sobre Implementación y Pruebas, trata en primer lugar, de cómo se han desarrollado las funciones claves del programa. A continuación se resume brevemente la evolución de la aplicación, se explican los ensayos prueba-error así como las mejoras que se han ido introduciendo.

Por último, se desarrolla el manual de usuario de la aplicación que pretende, como su propio nombre indica, explicar o servir de ayuda al usuario para cualquier duda que se le pueda plantear en la utilización de la misma.

4- CONTEXTO DEL PROYECTO

Este proyecto se puede clasificar dentro del ámbito de la robótica, ya que la idea principal del mismo es utilizarlo como simulación previa al objetivo principal, que es el funcionamiento de un vehículo móvil de una forma autónoma mediante la utilización de sensores.

Con este fin, se ha hecho una introducción sobre el contexto actual del mundo de la robótica, así como de su historia para conocer cuál ha sido el camino que ha llevado a la robótica a ser indispensable en el mundo actual.

Finalmente se ha hecho un breve estudio sobre dos elementos importantes que intervienen en este proyecto: el Quad y el láser.

4.1- Introducción a la Robótica

La Real Academia de la Lengua (RAE) define el término Robótica como: *"Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales."*

Se trata de una definición muy general, para un mundo tan amplio como es la robótica y es que la palabra "Robótica" puede ser definida desde varios puntos de vista:

- Con independencia respecto a la definición de "robot":
  - "*La Robótica es la conexión inteligente de la percepción a la acción*" [1]

- En base a su objetivo:
  - *"La Robótica consiste en el diseño de sistemas. Actuadores de locomoción, manipuladores, sistemas de control, sensores, fuentes de energía, software de calidad--todos estos subsistemas tienen que ser diseñados para trabajar conjuntamente en la consecución de la tarea del robot*" [2]

Para Asimov, conocido como el "Padre de la Robótica", el término Robótica se refiere "a la ciencia o arte relacionada con la inteligencia artificial (para razonar) y con la ingeniería mecánica (para realizar acciones físicas sugeridas por el razonamiento)". Esta definición data del año 1942.

Son muchas las definiciones que se han dado a lo largo de los años sobre la Robótica y por tanto, son muchos los expertos que han intentando dar con la definición de la misma. A continuación se presentan algunas de estas definiciones de diferentes autores:

- *"La robótica es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano o que requieren del uso de inteligencia. Las ciencias y tecnologías de las que deriva podrían ser: el álgebra, los autómatas programables, las máquinas de estados, la mecánica o la informática"* [3]

- *"La robótica consiste en el diseño de sistemas.Actuadores de locomoción, manipuladores, sistemas de control, sensores, fuentes de energía, software de calidad-todos estos subsistemas tienen que ser diseñados para trabajar conjuntamente en la consecución de la tarea del robot."* [4]

- *"La robótica se relaciona en si con el deseo de sintetizar algunos aspectos de la función humana mediante el uso de mecanismos, sensores, actuadores y computadoras"* [5]

4.1.1- Historia de la Robótica

A lo largo de toda la historia, el hombre siempre ha buscado facilitar y mejorar su calidad de vida construyendo o inventado artilugios que facilitasen sus actividades cotidianas. Así fue como se empezaron a crear dispositivos, denominados artefactos o máquinas, que tuvieran un movimiento sin fin y que no estuvieran controlados ni supervisados por personas.

Los antiguos egipcios unieron brazos mecánicos a las estatuas de sus dioses. Los griegos construyeron estatuas que operaban con sistemas hidráulicos, los cuales se utilizaban para fascinar a los adoradores de los templos. Personajes históricos tan importantes como Heron de Alejandría, Hsieh-Fec, Leonardo da Vinci, o von Kempelen construyeron robots en la edad media, el renacimiento y el clasicismo.

En 1738, Jacques de Vaucanson construyó uno de los autómatas más famosos de la historia, el Pato con aparato digestivo, considerado su obra maestra. Este autómata tenia más

de 400 partes móviles, y podría batir sus alas, beber agua, digerir grano e incluso defecar. Los alimentos los digería por disolución y eran conducidos por unos tubos hacia el ano, donde había un esfínter que permitía evacuarlos. Vaucanson también construyó otros autómatas, entre los que destaca El Flautista, un pastor que tocaba la flauta capaz de tocar hasta 12 melodías diferentes.

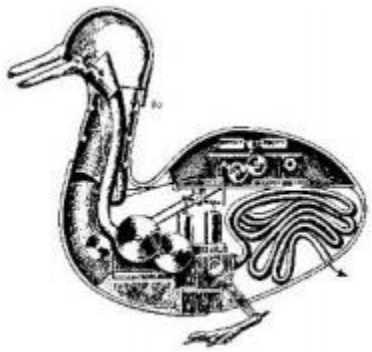


Imagen 4.1: Pato con aparato digestivo (Vaucanson)

El inicio de la robótica actual se fija a finales del siglo XVIII y principios del siglo XIX, en plena Revolución Industrial, donde se desarrollaron diversos ingenios y máquinas utilizadas principalmente en la industria textil, como por ejemplo, la máquina textil programable mediante tarjetas perforadas (precedente de las máquinas de control numérico), inventada por Joseph Jacquard en 1801.

En el siglo XIX se produjo, pues, un auge de los autómatas y se produjeron importantes avances en todas las ramas de la ingeniería.

Sin embargo no fue hasta 1920 cuando la palabra robot empezó a utilizarse. Se empleó por primera vez en 1920 en una obra de teatro llamada “Los Robots Universales de Rossum” (R.U.R.) escrita por el dramaturgo checo Karel Capek, cuya trama era: un hombre fábrica un robot, después el hombre mata al hombre. Capek utilizó la palabra checa “Robota” que significa servidumbre o trabajador forzoso, pero al hacer la traducción al inglés, “Robota” pasó a llamarse Robot.

A raíz de esta obra, han sido muchas las novelas y películas de ciencia ficción, que han seguido mostrando a los robots como máquinas destructivas y amenazadoras para el hombre.

Isaac Asimov, al cual se le conoce como el padre de la Robótica y al que se le atribuye el nacimiento del término robótica en 1942, escribió las tres leyes de la robótica, que son un conjunto de normas que la mayoría de los robots de sus novelas están diseñados para cumplir.

Estas leyes son:

1- Un robot no puede hacer daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.
2- Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la Primera Ley.
3- Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la Primera o la Segunda Ley.

Los primeros robots industriales empezaron a producirse a principios de los años 60 y estaban diseñados principalmente para realizar trabajos mecánicos difíciles y peligrosos. Las áreas donde estos robots tuvieron su aplicación fueron trabajos laboriosos y repetitivos, como la carga y descarga de hornos de fundición.

A partir de este momento, los avances en la robótica han sido espectaculares y en los últimos años se pueden encontrar robots en todas las áreas productivas industriales. Se ha evolucionado desde máquinas pesadas, rudas y limitadas a otras más rápidas, precisas y capaces de sustituir al hombre mejorando los resultados de éste. A pesar de la gran evolución que ha caracterizado a la robótica, ésta no se detiene aquí, ya que ofrece aun un amplio campo para el desarrollo y la innovación tecnológica. Una oportunidad que seguro investigadores y aficionados a los Robots no van a desaprovechar, y van a continuar diseñando robots cada vez más evolucionados, complejos y funcionales.

En la siguiente imagen se puede observar como la densidad de robots industriales aumenta años tras año a un ritmo muy alto.

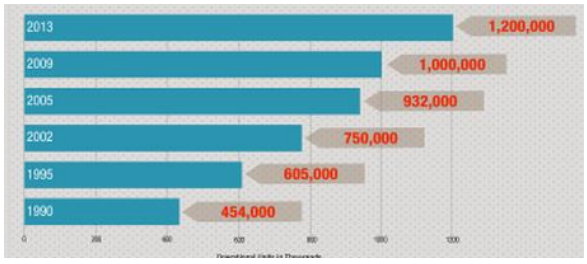


Imagen 4.2: Densidad de robots industriales

4.1.2- Robots móviles autónomos

Actualmente, el concepto de robótica ha evolucionado hacia los sistemas móviles autónomos, que son aquellos capaces de desenvolverse por sí mismos en entornos desconocidos y parcialmente cambiantes sin necesidad de supervisión.

Los robots móviles, en general, son robots con una gran capacidad de desplazamiento, basada en carros o plataformas y dotados de un sistema locomotor de tipo rodante que permite su movimiento. Siguen su camino por telemando o guiándose por la información recibida de su entorno a través de sus sensores (autónomos). Existen diferentes tipos: rodantes, andantes, reptadores, nadadores, voladores… .

La finalidad de este proyecto es la creación de una plataforma software para generación y transmisión de órdenes de movimiento a un vehículo autónomo tipo Quad que pertenece, al tipo de robots móviles rodantes. El Quad lleva incorporado un sensor láser, con el fin de proporcionar al vehículo la información sobre las distancias a los objetos que se encuentran a su alrededor y de esta forma poder evitar las colisiones con los objetos de su entorno y decidir qué camino tomar para llegar al punto destino.

4.1.2.1- Robots móviles rodantes

Son aquellos robots que se desplazan haciendo uso de ruedas, generalmente con una configuración 2+2 (configuración Ackerman) como las de un vehículo. Existen otras configuraciones como el triciclo (la configuración Ackerman se puede ver como un caso particular de ésta), configuración Oruga, entre otras. Dependiendo del terreno o el entorno es preferible utilizar una configuración u otra.

- **Configuración Ackerman**: Habitualmente solo dos de sus ruedas presentan tracción y las otras dos dirección, de forma que sea posible maniobrar el robot con un solo servomotor.

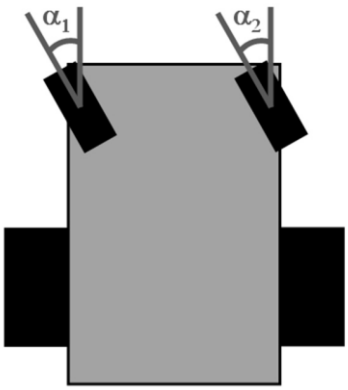


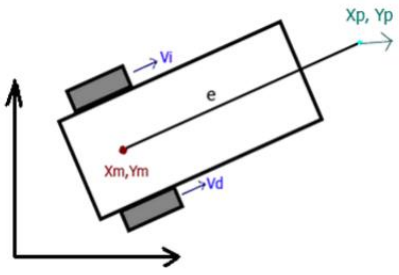
Imagen 4.3: Configuración Ackerman

4.1.2.2- Control cinemático. Estrategia del Punto descentralizado.

El control cinemático de robots móviles por punto descentralizado consiste en determinar las acciones del robot necesarias para llevarlo desde su posición actual a una posición final deseada, teniendo en cuenta las velocidades y la orientación. El punto destino está separado a una distancia e del eje de tracción del robot.

El vehículo autónomo Quad, objetivo de este proyecto, calcula el punto del camino seguido siguiente mediante esta técnica.

Los pasos a seguir desde un punto actual (xm,ym) al punto destino (xp,yp) siguiente son:



$$\begin{bmatrix} Xp \\ Yp \end{bmatrix} = \begin{bmatrix} Xm+ecos(\theta) \\ Ym+esin(\theta) \end{bmatrix}$$

Imagen 4.4: Control Cinemático de un robot móvil

De esta forma se obtiene el punto siguiente al cual tiene que avanzar el Quad, habiendo establecido previamente la inclinación de las ruedas para llegar a esa posición.

4.2- Quad

Un quad (llamado también cuadrimoto o cuatriciclo) es un vehículo de 4 ruedas (dos delante y dos detrás), sin carrocería y con un sistema de dirección mediante manillar, parecido a una moto que procede de los primeros triciclos o hart-trick, los cuales disponían de dos ruedas traseras y una delantera. Ésta configuración presentaba problemas de estabilidad y seguridad, además de la falta de propulsor.

El vehículo Quad como se conoce actualmente, empezó a aparecer a finales de los '80, principalmente en zonas agrícolas ya que a diferencia de las motocicletas, éstos eran válidos en cualquier terreno y ante condiciones climatológicas más adversas. Pocos años después empezaron a utilizarse en otros ámbitos como en el ocio y en la competición y por ello, cada día es más frecuente ver actividades que incluyen excursiones con ellos y competiciones y exhibiciones con gran afluencia de público.

Los Quads se clasifican en tres grupos, en función de las características técnicas que ofrezcan por construcción: vehículos automóviles, vehículos especiales y vehículos ciclomotores.

Imagen 4.5: Quad

En España se viene observando un enorme crecimiento en la matriculación de estos vehículos. Los primeros que se comercializaron fueron en 1994, y desde entonces, el mercado de este tipo de vehículo se ha incrementado hasta situarse en el entorno de las treinta mil unidades anuales, con un parque estimado en 2006 de más de setenta y cinco mil unidades.

4.2.1- Características

- **Potencia**: los quads llegan a cilindradas muy altas.
- **Agilidad**: permite controlar el vehículo para realizar giros con mucha facilidad.
- **Resistencia**: es una característica muy importante, ya que permite aguantar largas carreras. Es un vehículo especial, puesto que está pensado para circular por terrenos muy diversos y su aguante ha de ser enorme.
- **Ligereza**: si no tuviera esta característica sería muy difícil su manejo por lugares muy complicados.
- **Multifuncional**: Se puede circular con él por carrera, por ciudad, por terrenos arenosos, en caminos donde hay varios centímetros cúbicos de agua, etc.
- **Caja de cambios automática**: por lo general todos la usan ya que es mucho más cómodo para circular sin preocupaciones

4.2.2- Inconvenientes

- **Normativa**: su regulación no está bien definida y genera muchas dudas a los usuarios en temas tan importantes como : la licencia o permiso de conducción necesario, tipos de vías en los que se puede circular, etc.
- **Vuelcan con facilidad**: el quad aporta una falsa sensación de seguridad al conductor a pesar de que parecen mucho más estables que una motocicleta. En las curvas, las ruedas traseras tienden a levantarse y, si el piloto no es muy diestro, vuelcan con facilidad.

TODA LA NORMATIVA DE LOS QUADS

Clasificación	Características	Permiso	Placa matrícula	Uso de Casco	ITV	Vías autorizadas	Límite velocidad	Niños < 12 años
CUADRICICLO LIGERO	Máximo 350 kg.de masa en vacío y 50 c.c.	Licencia ciclomotor ó A1,A,B	Fondo amarillo, caracteres negros	SÍ (2)	NO(3)	Ni autopistas ni autovías	45 km/h	NO. Excepción: > 7 años con autorización
CUADRICICLO	Hasta 550 kg. de masa en vacío y 15 Kw	A (1), B	Fondo blanco, caracteres negros	SÍ (2)	De 4 a 10 años: bienal. Más de 10 años: anual	Todas	70 km/h	SÍ
VEHÍCULO ESPECIAL	Máquina de servicios	B	Fondo blanco, caracteres rojos	NO	De 4 a 10 años: bienal. Más de 10 años: anual	- Todas si se superan 60 km/h. Y deben hacerlo por zona especial destinada, en su defecto por arcén practicable y en defecto de ambos ocupando la parte imprescindible de la calzada.	- 70 km/h - Otros pueden tener limitaciones inferiores	SÍ

(1) Para vehículos de velocidad superior a 45 km/h o cilindrada superior a 50 cc y masa en vacío no superior a 550 kg.
(2) Salvo que cuenten con estructura de autoprotección, en cuyo caso deberán llevar cinturón de seguridad.
(3) Está prevista la aprobación de una modificación del Real Decreto que regula la inspección técnica de vehículos, por la que también estarán obligados a pasar la ITV de forma periódica.

Imagen 4.6- Normativa Vehículo Quad

4.3- Láser

Según la definición que proporciona la Real Academia Española el término láser se define como*: "Dispositivo electrónico que, basado en la emisión inducida, amplifica de manera extraordinaria un haz de luz monocromático y coherente".*

Algunas aplicaciones del Láser son:

- **Medicina**: Operaciones sin sangre, tratamientos quirúrgicos, ayudas a la cicatrización de heridas, tratamientos de piedras en el riñón, operaciones de vista, operaciones odontológicas.
- **Industria (Robótica)** : Cortado, guiado de maquinaria y robots de fabricación, mediciones de distancias precisas mediante láser.
- **Defensa**: Guiado misiles balísticos, alternativa al Radar, cegado a las tropas enemigas. En el caso del Tactical High Energy Laser se está empezando a usar el láser como destructor de blancos.
- **Ingenieria Civil**: Guiado de máquinas tuneladoras en túneles, diferentes aplicaciones en la topografía como mediciones de distancias a lugares inaccesibles o realización de un modelo digital del terreno (MDT).
- Arquitectura: catalogación de Patrimonio.
- **Arqueológico**: documentación.
- **Investigación**: Espectroscopía, Interferometría láser, LIDAR, distanciometría.
- **Desarrollos en productos comerciales:** Impresoras láser, CD, ratones ópticos, lectores de código de barras, punteros láser, termómetros, hologramas, aplicaciones en iluminación de espectáculos.
- **Tratamientos cosméticos y cirugía estética**: Tratamientos de Acné, celulitis, tratamiento de las estrías, depilación.

4.3.1- Láser en Robótica

El láser es utilizado como un sensor que es un dispositivo capaz de transformar magnitudes físicas o químicas, en magnitudes eléctricas. En concreto el láser es un sensor de distancias.

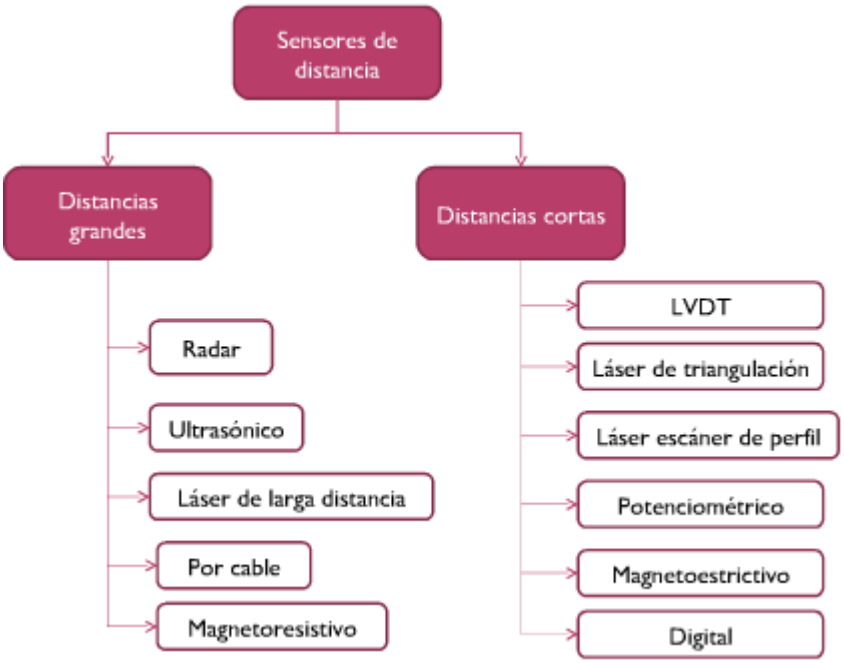


Imagen 4.7: Tipos de sensores de distancias.

4.3.1.1- Láser de Triangulación

Los láseres de triangulación proyectan un haz de luz sobre la superficie a medir, este haz es reflejado en el fotodetector del aparato con un cierto ángulo de inclinación que variará en función de la distancia medida. Utilizado para la medición de pequeñas distancias (2mm a 500 mm)

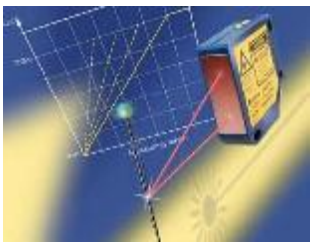


Imagen 4.8: Láser de Triangulación.

4.3.1.2- Láser de Larga Distancia

El láser envía un haz de luz con diferentes frecuencias, y compara la señal que se ha reflejado en el objeto a medir con el valor de referencia interno.

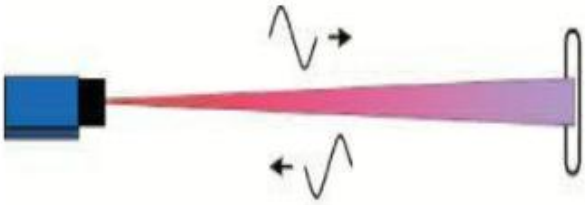


Imagen 4.9: Láser de Larga Distancia

4.3.1.3- Láser Escáner de Perfil

Basado en el principio de funcionamiento del láser de triangulación, pero proyectando un haz de luz transversal en vez de puntual, lo que permite obtener el perfil donde se está reflejando dicho haz.

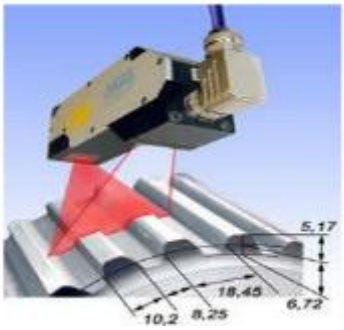


Imagen 4.10: Láser Escáner de Perfil

En este proyecto se simula el funcionamiento de este tipo de sensor láser en los que se pueden diferencias las siguientes partes:

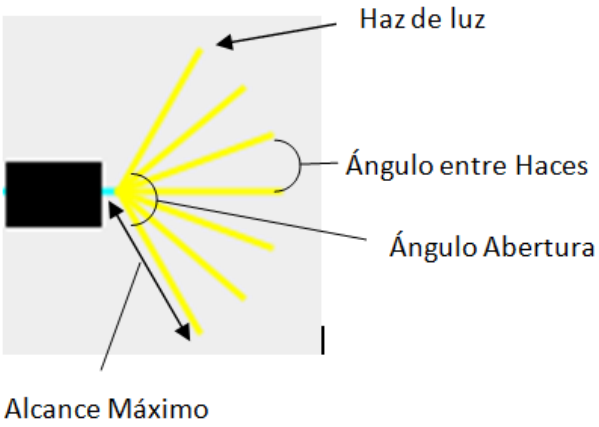


Imagen 4.11: Partes Láser Proyecto

5- EVALUACIÓN TECNOLÓGICA

Para la implementación de la aplicación, una de las primeras decisiones importantes fue la elección del lenguaje de programación puesto que tenia total libertad para escoger el tipo de lenguaje que más se adaptara a las exigencias del proyecto.

Desde el primer momento tenía claro que el lenguaje iba a ser C++ o Java, puesto que se tratan de los dos lenguajes de programación que más se han estudiado a lo largo de la carrera.

Tras estar analizando las posibilidades que ofrecían ambos lenguajes, decliné la opción de C++ debido a su manera de crear interfaces ya que, según mi opinión, con Java se obtienen resultados bastante buenos de una manera sencilla e intuitiva.

Otro punto a favor de Java, fue su portabilidad, ya que al realizarlo en Java, el programa podría ser ejecutado en otros sistemas diferentes independientemente del hardware que tengan.

Además del lenguaje de programación Java, se ha utilizado el lenguaje HTML para escribir la “Ayuda” de la aplicación.

Los principios básicos y características en las que se basan ambos lenguajes vienen detallados a continuación.

5.1- Java

Java es un lenguaje de programación orientado a objetos (POO), desarrollado por “Sun Microsystems” a principios de los años 90. Su sintaxis está basada en el lenguaje C y C++, pero con un modelo de objetos más simple ya que se eliminan herramientas de bajo nivel como por ejemplo, los punteros o la gestión dinámica de memoria. En Java el Recolector de basura (garbage collector) se encarga de destruir automáticamente los objetos de los que no haya ninguna referencia. En C++, en cambio, la memoria debe desasignarse automáticamente.

Además, se trata de un lenguaje fuertemente tipado y robusto porque realiza verificaciones tanto en tiempo de ejecución como en tiempo de compilación.

Una característica muy a destacar en este lenguaje es su portabilidad, es decir, su independencia a la plataforma en la que se ha compilado. De hecho, el axioma de Java es: "write once, run everywhere", que indica que una vez se ha escrito el programa, se puede ejecutar en cualquier dispositivo. Para ello, se compila el código fuente generándose un código conocido como "bytecode": instrucciones máquina simplificadas específicas de la plataforma Java, el cual, se ejecuta en la máquina virtual (JVM): que es un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), del dispositivo destino.

Imagen 5.1: Logotipo de Java

5.1.1- Programación Orientada a Objetos (POO)

La programación orientada a objetos es más moderna que la programación secuencial y la programación imperativa. El desarrollo Orientado a Objetos supuso una nueva forma de pensar acerca del software basándose en abstracciones que existen en el mundo real. Por ejemplo, en el caso de lenguajes orientados a objetos como C++ y Java, el elemento básico no es la función, sino el objeto. Un objeto contiene toda la información necesaria para abstraerlo: datos que describen sus atributos y operaciones que pueden realizarse sobre los mismos.

Aunque tanto C++ como Java se clasifican en el grupo de lenguajes orientados a objetos, existen diferencias entre ellos, entre las cuales, la principal podría ser que Java más que un lenguaje orientado a objetos, es un lenguaje de objetos, es decir, en Java no se puede programar sin hacer uso de al menos un objetos. Por el contrario en C++ sí.

Las características que definen la POO son principalmente:

- **Abstracción**: capacidad de separar los elementos (al menos mentalmente) para poder verlos de forma singular.

- **Encapsulación**: también conocida como ocultamiento. Permite almacenar todos los elementos de una misma entidad, al mismo nivel de abstracción. Además facilita la protección de las variables del objeto.
- **Herencia**: El mecanismo de herencia permite definir nuevas clases partiendo de otras ya existentes. Las clases que derivan de otras heredan automáticamente todo su comportamiento, pero además pueden introducir características particulares propias que las diferencian.

Y por último, los conceptos básicos que maneja la POO son:

- **Clase:** Entidad sintáctica que define las variables y los métodos que son comunes para todos los objetos de un cierto tipo. Una clase está compuesta por los atributos y los métodos.
- **Objeto:** Es una instancia de una clase creada en tiempo de ejecución, por tanto, es una estructura de datos formada por tantos campos como atributos tiene la clase.
    - El estado de un objeto viene dado por el valor de los campos.
    - Los métodos permiten consultar y modificar el estado del objeto.
- **Mensaje**: Los objetos de un programa interactúan y se comunican entre ellos por medio de mensajes. Cuando un objeto A quiere que otro objeto B ejecute una de sus funciones miembro (métodos de B), el objeto A manda un mensaje al objeto B.
- **Polimorfismo**: Posibilidad de asignar un objeto a una referencia de la misma clase o de cualquiera de las clases base (hijas). Su comportamiento vendrá determinado por la clase que se instancie, no por la referencia utilizada.

5.2- Librerías gráficas

Debido a que la finalidad del proyecto es desarrollar una plataforma software que simule la conducción automatizada de un vehículo QUAD, se ha tenido una especial atención en la interfaz de usuario para facilitar el manejo de la misma y poder observar los resultados de una forma gráfica. Para crear la GUI se han combinado algunos componentes y eventos de la librería AWT con otros de Java Swing.

5.2.1- Abstract Window Toolkit (AWT)

Abstract Window Toolkit (AWT) es una librería original de Java, que proporciona un kit de herramientas para gestionar el sistema de ventanas, gráficos e interfaces de usuario. Forma parte de versión J2SE, y es comúnmente usada en la implementación de dispositivos móviles (J2ME). El inconveniente que presenta es que las aplicaciones no se ven exactamente igual en todas las aplicaciones.

Las clases de esta librería que han sido utilizadas son:

Java.awt	Descripción
Component	Superclase abstracta de varios componentes del AWT. Un método utilizado es: public Graphics getGraphics() para obtener el contexto gráfico sobre el que dibujar figuras o componentes.
Event	Clase base de todos los eventos que se utilizan en la creación de GUIs.
Layout	Elige la mejor posición y tamaño de cada componente según el espacio disponible. Existen diferentes tipos de Layout: BorderLayout, GridLayout,GridBagLayout…
Toolkit	Superclase del Abstract Window Toolkit. En este proyecto se ha utilizado para obtener el tamaño de la pantalla del ordenador en el que se está ejecutando la aplicación.
Dimension	Gestiona el tamaño de los componentes: la altura y la anchura.
Color	Clase que representa un color
Graphics	Encapsula el contexto gráfico utilizado. Tiene una serie de métodos para dibujar figuras.

Tabla 5.1: Clases de la Librería java.awt

5.2.2- Java Swing

Swing es una biblioteca gráfica para Java. En ella se puede encontrar la mayoría de los widgets o componentes necesarios en una GUI, tales como botones, paneles, editores de texto, menú, checkbox, etc. Esta librería surgió como mejora de la librería AWT (de hecho es una extensión y no un sustituto), solucionando el problema de la portabilidad puesto que permite que las aplicaciones tengan el mismo comportamiento, independientemente de la plataforma en la que se ejecuten.

Características principales:

- Independencia de plataforma.
- Extensibilidad: los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto.
- Orientada a componentes.
- Customizable: se pueden configurar todos los parámetros de los componentes (bordes, colores, tamaños, márgenes,…).

Las clases que se pueden encontrar en la biblioteca Swing son:

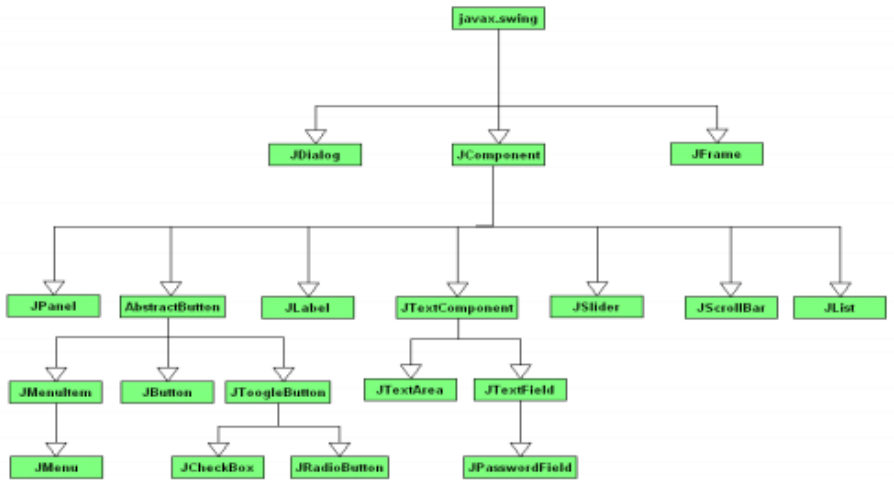


Imagen 5.2: Clases del paquete java.swing

Para el desarrollo de la interfaz de usuario se ha utilizado la mayoría de estas clases, entre las cuales destaca JFrame que es la clase empleada para representar la ventana principal de la aplicación y las ventanas secundarias de modificación de datos. A estas ventanas se les

han añadido distintos componentes (botones, áreas de texto, etiquetas …) hasta conformar la ventana con el aspecto deseado.

5.3- Entorno de programación

Como cualquier proyecto donde se tenga que compilar y desarrollar software, es importante pensar en que entorno se desea escribir, compilar, depurar y ejecutar el código. Para realizar este proyecto, una vez decidido que el lenguaje de programación iba a ser Java, tenía dos opciones: Eclipse o NetBeans. Finalmente me decanté por Eclipse puesto que era un entorno conocido y utilizado anteriormente y se ajustaba a las exigencias de esta interfaz.

5.3.1- Eclipse

Eclipse es una plataforma de desarrollo "open source" basada en Java. Fue desarrollado por IBM, el cual, puso a disposición de los usuarios su código fuente. Actualmente, Eclipse está desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

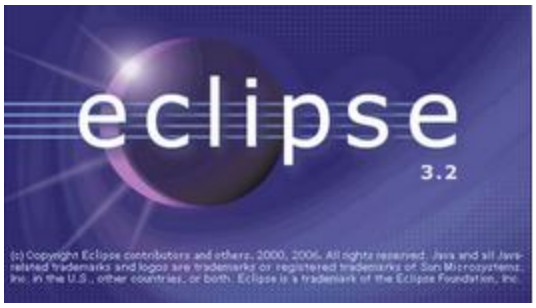


Imagen 5.3:Logotipo Eclipse

Para realizar la depuración de código he utilizado la versión *Eclipse Helios,* que es la versión 3.6 de Eclipse y que entre otras características interesantes destaca el gran manejo y facilidad para ver el contenido de las variables en el debug, facilitando la depuración y corrección del código.

Para realizar la interfaz de la aplicación he utilizado la versión de Eclipse 3.2 que fue la utilizada en la asignatura de Ingeniería de la Programación. Para poder realizar la interfaz se debe instalar la clase Visual Class en Eclipse.

En la página web http://www.eclipse.org/, en la sección Downloads, se pueden encontrar todas las versiones disponibles para descargar, manuales para su funcionamiento y los plugins para ampliar el funcionamiento de Eclipse, como por ejemplo el Visual Class.

5.4- HTML

HTML, de HyperText Markup Language , es un lenguaje utilizado principalmente para la creación de páginas web. No se trata de un lenguaje de programación como C++, Java, etc., sino que es más bien, un sistema de etiquetas, cerradas por corchetes angulares (<,>).

Otra característica que presenta HTML, es que no utiliza ningún compilador y por tanto si existe algún error de sintaxis éste no lo detectará y se visualizará en la forma como éste lo entienda. El entorno para trabajar en este lenguaje puede ser cualquier procesador de texto y el documento que se cree se debe guardar con la extensión .htm o .html.

Se puede describir la estructura, el contenido y el formato que se desee, mediante las etiquetas adecuadas predeterminadas.

PARTE II: ANÁLISIS Y DISEÑO DE LA HERRAMIENTA

6- ESPECIFICACIÓN *DE REQUERIMENTOS*

6.1- Introducción

La especificación de requerimientos es la fase de análisis del ciclo de vida de un programa donde se debe describir lo más detalladamente posible todo lo que se espera de ese software.

6.1.1- Propósito

El propósito de este apartado es definir cuáles son los requerimientos que debe tener un programa de simulación de generación y transmisión de movimiento a un vehículo autónomo tipo Quad. Este apartado va dirigido a aquellas personas interesadas en conocer y utilizar el producto.

6.1.2- Referencias

Para la redacción de este apartado se han consultado los siguientes documentos:

- Guía de elaboración de ERS según el IEEE830.
- Transparencias de la asignatura Ingeniería de Requerimientos.

6.1.3- Visión global

A continuación, se detallan las funcionalidades básicas de la aplicación (requerimientos funcionales) así como otras características que no dependen del proyecto (requerimientos no funcionales).

6.2- Interfaz de usuario

El diseño y la usabilidad de la interfaz gráfica de usuario (GUI) tiene mucha importancia en este proyecto ya que debe ser una herramienta para facilitar el estudio de los movimientos

de los vehículos móviles autónomos y por tanto los resultados se deben visualizar de una manera rápida y fácilmente entendibles. Va dirigida principalmente a personas del ámbito de la robótica interesados en la conducción automatizada de vehículos autónomos.

6.3- Requerimientos funcionales

Los requerimientos funcionales describen, como su nombre indica, la funcionalidad o servicios que se espera que provea el sistema.

A continuación se especifican los requisitos funcionales considerados para que la aplicación cumpla los objetivos:

6.3.1- Configuración del Quad

- **REQ1**: Modificar características del Quad

La aplicación debe permitir cambiar los valores del vehículo tales como el ancho, largo y ángulo máximo de giro ya que debe ser válida para distintas configuraciones de vehículo.

- **REQ2**: Modificar características del láser

Se debe poder cambiar los valores de los parámetros que caracterizan al láser como el ángulo de abertura, alcance máximo, ángulo entre haces, etc.

6.3.2- Configuración del mapa

- **REQ3**: Insertar Borde

Cuando se realizan las pruebas sobre un vehículo real se llevan a cabo en un terreno delimitado y éste no se debe salir de él. Por tanto en la simulación se tiene que delimitar también. Este “terreno simulado” es en realidad un obstáculo con una forma rectangular o cuadrada.

- **REQ4**: Insertar obstáculos

Con el fin de dificultar el camino del Quad para comprobar el funcionamiento del láser y del propio vehículo, se deben poder insertar tantos obstáculos como el usuario crea conveniente. Éstos pueden ser circulares o cuadrados con el tamaño y la inclinación deseada.

- **REQ5**: Modificar obstáculos

Ya sea porque a la hora de insertar los parámetros se ha tenido algún error o bien porque se quiere cambiar el tamaño o posición por algún motivo, el programa debe proporcionar la posibilidad de modificar los obstáculos sin necesidad de repetir el proceso.

- **REQ6**: Escalar la vista de dibujo

Se debe poder realizar una simulación sobre una superficie cuadrada (o rectangular) de cualquier tamaño y, por ello, para obtener una vista adecuada, la aplicación debe tener los recursos necesarios para escalar el dibujo hasta obtener una representación adecuada.

6.3.3- Configuración de la simulación

- **REQ7**: Establecer posiciones iniciales

Un buen estudio del funcionamiento de este tipo de vehículos requiere muchas pruebas con trayectorias distintas, por tanto, se debe poder cambiar el punto de salida y el punto de meta.

- **REQ8**: Establecer la Inclinación inicial de las ruedas del Quad

Por el mismo motivo que en el REQ7, se debe permitir que el Quad inicie su camino con distintas inclinaciones de sus ruedas.

- **REQ9**: Establecer velocidad inicial

Cualquier vehículo puede funcionar a distintas velocidades , por ello, se debe poder cambiar este valor

- **REQ10**: Control de la aceleración

Durante la conducción la velocidad no se mantiene constante ya que hay que tener en cuenta la aceleración y el frenado (aceleración negativa). La aplicación debe permitir el control de esta aceleración (cambiar su valor) durante la ejecución de la simulación.

6.3.4- Simulación

- **REQ11**: Visualizar el mapa completo

Para configurar correctamente el mapa, se debe poder visualizar en todo momento los obstáculos que se han añadido, así como las posiciones iniciales.

- **REQ12**: Visualizar el recorrido

Una vez iniciada la simulación la aplicación debe mostrar la trayectoria realizada por el Quad posición a posición.

- **REQ13**: Vista analítica de los resultados

Además de los resultados gráficos, se deben mostrar las resultados de una forma más analítica para poder comprender de una manera más exacta el comportamiento del Quad.

- **REQ14**: Comprobar Información Láser

Se debe poder visualizar los datos obtenidos por el láser en la posición en la que se encuentra en ese momento de la simulación el Quad.

6.3.5- Control Simulación

- **REQ15**: Iniciar la simulación (START)

Se debe poder iniciar la simulación

- **REQ16**: Parar la simulación (STOP)

Se debe poder parar la simulación en todo momento, cuando ha llegado a la meta pero también aunque no haya llegado.

- **REQ17**: Detener momentáneamente la simulación (PARAR)

Con el fin de llevar un mayor control sobre la simulación, ésta se debe poder parar momentáneamente con el fin de cambiar la aceleración, visualizar los datos obtenidos por el láser …

- **REQ18**: Reanudar la simulación (REANUDAR)

Al igual que se debe poder pausar la simulación, se debe poder reanudarla en la misma posición en la que se paró.

6.3.6- Otras

- **REQ19**: Guardar Simulación

El programa debe aportar la opción de guardar los datos utilizados en la simulación (configuración del quad y de los obstáculos) con el fin de poder utilizar esa misma disposición del mapa en otro momento sin necesidad de volver a configurarlo de nuevo.

- **REQ20**: Abrir Simulación

Se deben de poder abrir las configuraciones ya guardadas que cumplan con la estructura del archivo requerida por el programa.

- **REQ21**: Generar Fichero de datos

El usuario deberá tener la posibilidad de guardar todos los resultados obtenidos en la simulación en un fichero con un formato adecuado para su utilización en otras aplicaciones como Matlab.

- **REQ22**: Ayuda

El programa debe tener una opción para desplegar un tutorial que explique el uso de la aplicación.

- **REQ23**: Salir

El programa debe tener una opción para salir del programa.

6.4- Requerimientos no funcionales

Los requerimientos no funcionales son aquellos requerimientos que no se refieren directamente a las funciones específicas que entre el sistema, sino a las propiedades emergentes de éste.

A continuación se especifican los requisitos no funcionales:

- **Fiabilidad**: La fiabilidad de la aplicación depende del funcionamiento que haga el usuario de la misma.
- **Modificabilidad**: El programa se puede modificar a partir del código fuente con el fin de ampliar o mejorar la funcionalidad de la aplicación.
- **Dependencia de otras partes**: No tiene dependencias que otras aplicaciones.
- **Portabilidad**: La aplicación es portable ya que funciona bajo la máquina virtual de Java (JVM).
- **Tiempo de respuesta**: Viene determinado por la capacidad del procesador donde se ejecuta la aplicación.

7- DIAGRAMAS UML

UML es lenguaje de propósito general para el modelado orientado a objetos y se utiliza para visualizar, especificar, construir y documentar un sistema. Nació debido a la necesidad de una notación uniforme. Es pues, una notación para describir sistemas (no es un método).

Imagen 5.1: Logotipo UML

Existen una cantidad de diagramas de tipos diferentes (en UML 2.0, hay 13 diferentes), que se pueden clasificar según sean Diagramas de Estructura, Diagramas de Comportamiento o Diagramas de Interacción.

Diagramas de Estructura	Diagramas de clases
	Diagramas de componentes
	Diagramas de objetos
	Diagramas de estructura compuesta (UML 2.0)
	Diagrama de despliegue
	Diagrama de paquetes
Diagramas de Comportamiento	Diagramas de actividades
	Diagramas de casos de uso
	Diagrama de estados
Diagramas de Interacción (subtipo de diagramas de comportamiento)	Diagrama de secuencia
	Diagrama de comunicación
	Diagrama de tiempos
	Diagrama global de interacciones

Tabla 7.1- Clasificación diagramas UML

Para este proyecto se han realizado los diagramas marcados en la Tabla 5.1. Para su realización se ha empleado la herramienta MyEclipse9.

7.1- Diagrama de clases

Muestra las clases del sistema y las relaciones entre las mismas. Sirven para reflejar la estructura estática del sistema.

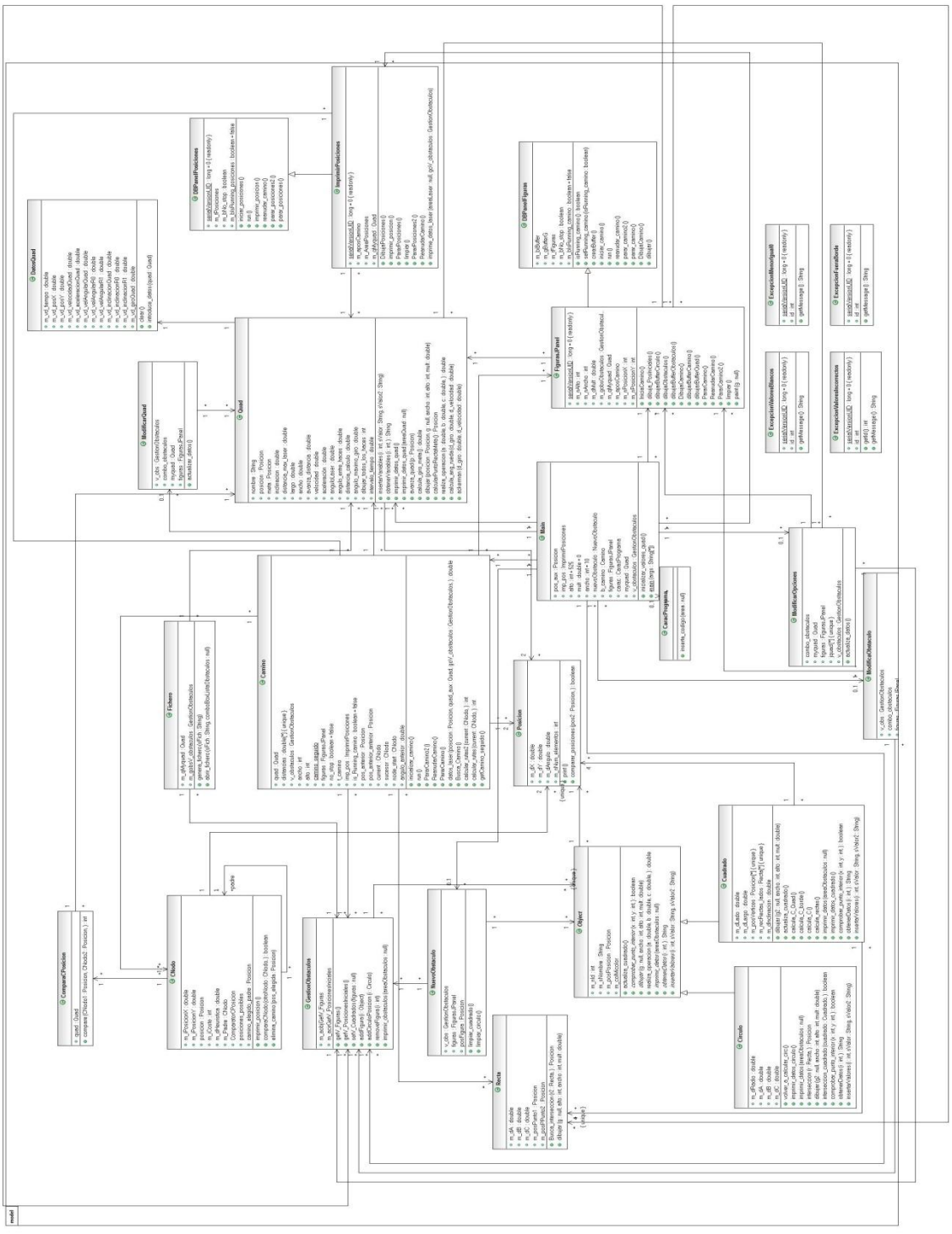


Imagen 7.1. Diagrama de Clases

En el diagrama de clases se han omitido los getters y setters de cada la clase por motivos de espacio.

En el Apéndice A se explica la función de cada una de las clase

7.2- Diagrama de paquetes

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos.

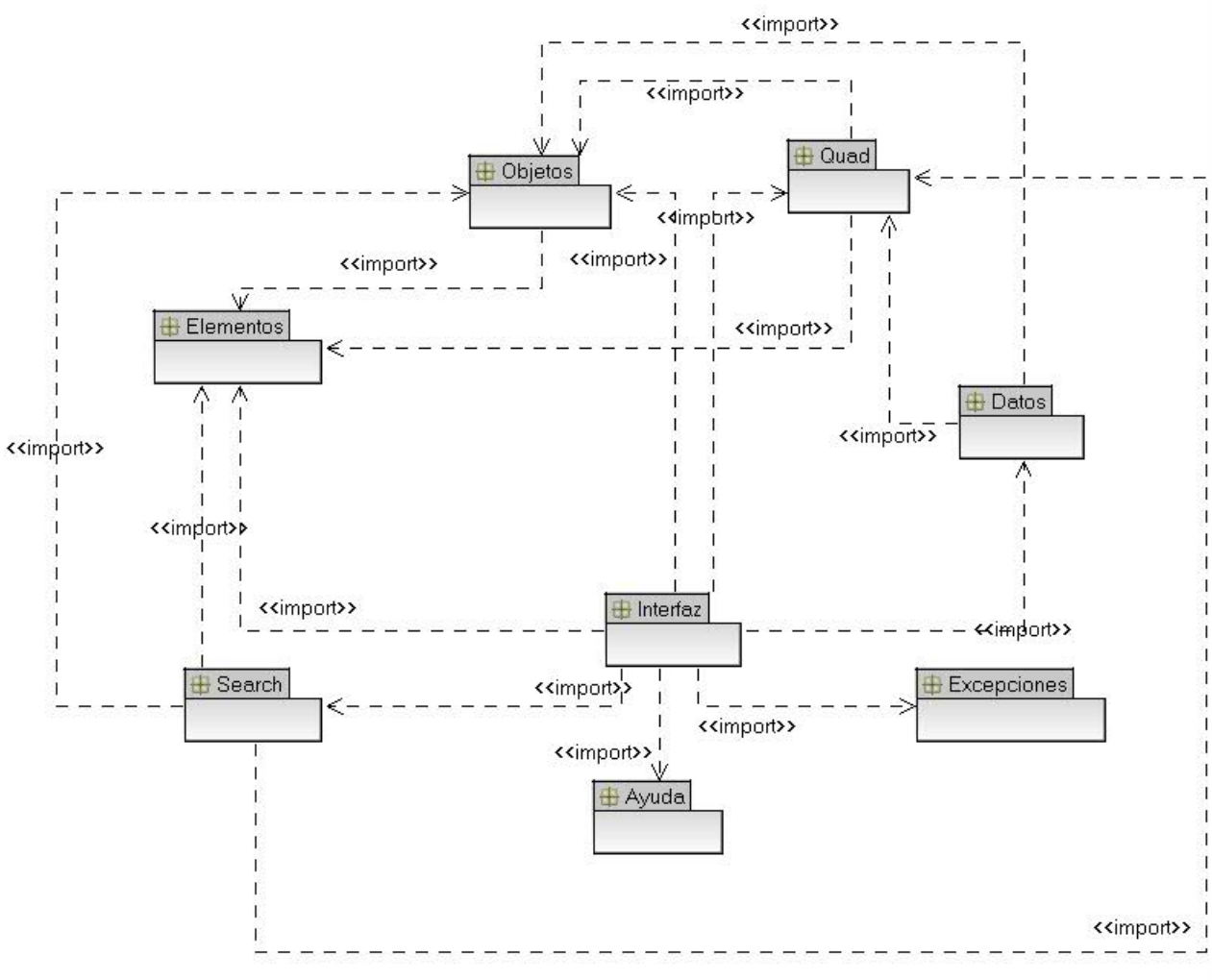


Imagen 7.2. Diagrama de Paquetes

7.5-Diagramas Casos de Uso

Los casos de uso son una técnica para especificar el comportamiento de un sistema. Se trata por tanto de una representación de la secuencia de acciones entre un sistema y sus actores, respondiendo a un evento que inicia uno de los actores principales.

7.5.1- Actores

Existe un único actor que es el usuario de la aplicación.

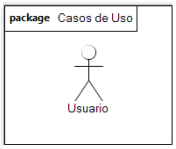


Imagen 7.3- Casos de Uso: Actores

7.5.2- Casos de uso

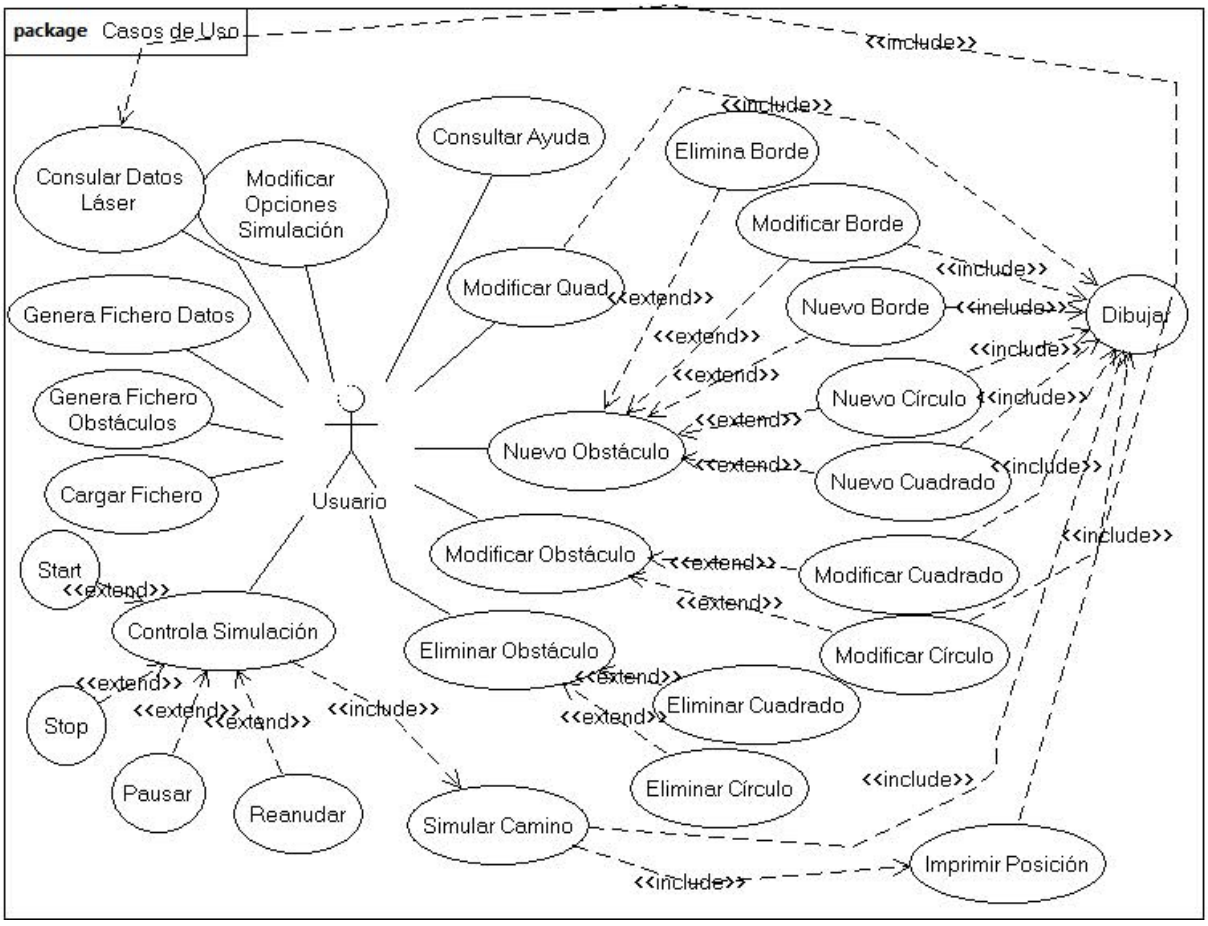


Imagen 7.4- Casos de Uso

A continuación se describen los casos de uso en profundidad:

7.5.1.1- Cargar Fichero

CASO DE USO:	Cargar Fichero
Actores	Usuario
Descripción	Se carga en la aplicación una configuración guardada anteriormente.
USUARIO	**APLICACIÓN**
1- Click en "Abrir" 3- Seleccionar archivo y "Abrir"	2- Mostrar ventana para examinar 4- Se carga la configuración guardada

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	El usuario cancela la operación.	Cierra la venta de Abrir Archivo
#2	En 3. El usuario selecciona un fichero con un formato no válido	La aplicación informa con un mensaje de error y cierra la ventana de Abrir Archivo.

7.5.1.2- Genera Fichero Obstáculos

CASO DE USO:	Genera Fichero Obstáculos
Actores	Usuario
Descripción	Se guarda la configuración actual en un fichero.
USUARIO	**APLICACIÓN**
1- Click en "Guardar Figuras" 3- Seleccionar ubicación, introducir el nombre y "Guardar"	2- Mostrar ventana para elegir ubicación 4- Crear el fichero

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	El usuario cancela la operación.	Cierra la venta de Guardar Fichero

#2	En 3. El usuario selecciona un fichero con un formato no válido	La aplicación informa con un mensaje de error y cierra la ventana de Abrir Archivo.

7.5.1.3- Genera Fichero Datos

CASO DE USO:	Genera Fichero Datos	
Actores	Usuario	
Descripción	Se guardan los resultados obtenidos en cada instante de tiempo de la simulación en un fichero.	
USUARIO	**APLICACIÓN**	
1- Click en “Crear Fichero Datos” 3- Seleccionar ubicación, introducir el nombre y “Guardar”	2- Mostrar ventana para elegir ubicación 4- Crear el fichero	
CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	El usuario cancela la operación.	Cierra la venta de Guardar Fichero
#2	En 3. El usuario selecciona un fichero con un formato no válido	La aplicación informa con un mensaje de error y cierra la ventana de Abrir Archivo.

7.5.1.4- Consultar Ayuda

CASO DE USO:	Consultar Ayuda
Actores	Usuario
Descripción	Muestra el manual de usuario de la aplicación
USUARIO	**APLICACIÓN**
1- Click en “Ayuda”	2- Mostrar la ventana de Ayuda

7.5.1.5- Modificar Quad

CASO DE USO:	Modificar Quad	
Actores	Usuario	
Descripción	Se encarga de visualizar o modificar los datos del Quad	
USUARIO	**APLICACIÓN**	
1- Click en "Modificar" (del bloque II: Datos Quad) 3- El usuario modifica los campos que desee y confirma los cambios.	2- Mostrar ventana para Modificar dichos valores 4- El sistema actualiza la información.	
CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	En 3. El usuario introduce valores incorrectos o deja espacios en blanco.	El sistema muestra un error conforme los datos introducidos son erróneos (si son números) o mantiene los valores anteriores correctos en el caso de introducir valores no numéricos.
#2	En 1. El usuario pulsa sobre el botón "Modificar Quad" del bloque I: Menú	El sistema muestra la ventana Modificar Quad.

7.5.1.6- Nuevo Borde

CASO DE USO:	Nuevo Borde
Actores	Usuario
Descripción	Se encarga de introducir el borde que limita la superficie a simular.
USUARIO	**APLICACIÓN**
1- Click en "Borde" (del bloque III: Obstáculos)	2- Mostrar ventana Borde

3- El usuario introduce los valores en los campos oportunos y pulsa en “Añadir”.	4- El sistema introduce y dibuja el nuevo borde

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	En 3. El usuario cancela y/o cierra la ventana.	El sistema no almacena ningún borde.
#2	En 3. El usuario introduce valores incorrectos o deja espacios en blanco.	El sistema introduce por defecto el valor de 100.0

7.5.1.7- Modificar Borde

CASO DE USO:	Modificar Borde
Actores	Usuario
Descripción	Se encarga de introducir el borde que limita la superficie a simular.
Precondiciones	Que esté almacenado un borde en el sistema.
USUARIO	**APLICACIÓN**
1- Click en “Borde” (del bloque III: Obstáculos)	2- Mostrar ventana Borde
3- El usuario modifica los campos que desee y confirma los cambios pulsando en el botón “Modificar”	4- El sistema modifica y dibuja el borde modificado

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	En 3. El usuario cancela y/o cierra la ventana.	El sistema no modifica el borde y mantiene el mismo.
#2	En 3. El usuario introduce valores incorrectos o deja espacios en blanco.	El sistema mantiene el valor correcto anterior de ese campo

7.5.1.8- Eliminar Borde

CASO DE USO:	Eliminar Borde
Actores	Usuario
Descripción	Se encarga de eliminar el borde que limita la superficie a simular.
Precondiciones	Que no haya ningún otro obstáculo que no sea el borde.
USUARIO	**APLICACIÓN**
1- Click en "Borde" (del bloque III: Obstáculos) 3- El usuario pulsa en "Eliminar".	2- Mostrar ventana Borde 4- El sistema elimina el borde y redibuja el mapa vacío.

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	En 3. El usuario cancela y/o cierra la ventana.	El sistema no elimina el borde y mantiene el mismo.

7.5.1.9- Nuevo Obstáculo

CASO DE USO:	Nuevo Obstáculo
Actores	Usuario
Descripción	Permite introducir nuevos obstáculos.
Precondiciones	Que exista un borde.
USUARIO	**APLICACIÓN**
1- Click en "Nuevo" (del bloque III: Obstáculos) 3- El usuario selecciona la pestaña Círculo o Cuadrado según el elemento que desea introducir. Tras rellenar todos los campos oportunos el usuario pulsa en "Añadir".	2- Mostrar ventana Nuevo Obstáculo 4- El sistema introduce y dibuja el nuevo obstáculo

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN

#1	En 3. El usuario cancela y/o cierra la ventana.	El sistema no introduce el obstáculo.
#2	En 3. El usuario introduce valores incorrectos o deja espacios en blanco.	Dependiendo del error, el sistema muestra mensajes de error o escribe un 1.0.
#3	En 1. El usuario pulsa sobre el bloque VI: Gráfica en la posición donde desea introducir el obstáculo	El sistema muestra la ventana Nuevo Obstáculo estableciendo como centro del obstáculo las coordenadas x,y sobre las que ha hecho clic el usuario
#4	En 1. El usuario pulsa sobre el botón "Nuevo Obstáculo" del bloque I: Menú	El sistema muestra la ventana Nuevo Obstáculo.

7.5.1.10- Modificar Obstáculo

CASO DE USO:	Modificar Obstáculo
Actores	Usuario
Descripción	Permite Modificar el valor del obstáculo seleccionado.
USUARIO	**APLICACIÓN**
1- Selecciona en el combobox el nombre del obstáculo que desea cambiar y clic en "Modificar" (del bloque III: Obstáculos)	2- Mostrar ventana Modificar Obstáculo
3- El usuario modifica los campos que desee y confirma los cambios pulsando en el botón "Modificar"	4- El sistema modifica el obstáculo y dibuja el obstáculo modificado.

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	En 3. El usuario cancela y/o cierra la ventana.	El sistema no modifica el obstáculo.

#2	En 3. El usuario introduce valores incorrectos o deja espacios en blanco.	Dependiendo del error, el sistema muestra mensajes de error o escribe un 1.0.
#3	En 1. El usuario pulsa sobre el bloque VI: Gráfica en la posición donde se encuentra el obstáculo a modificar	El sistema muestra la ventana Modificar Obstáculo

7.5.1.11- Eliminar obstáculo

CASO DE USO:	Eliminar Obstáculo	
Actores	Usuario	
Descripción	Permite eliminar el obstáculo seleccionado.	
USUARIO	**APLICACIÓN**	
1- Selecciona en el combobox el nombre del obstáculo que desea eliminar y clic en "Modificar" (del bloque III: Obstáculos)	2- Mostrar ventana Modificar Obstáculo	
3- El usuario pulsa en el botón "Eliminar"	4- El sistema elimina el obstáculo y dibuja el mapa sin el obstáculo eliminado.	
CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	En 3. El usuario cancela y/o cierra la ventana.	El sistema no elimina obstáculo.
#2	En 1. El usuario pulsa sobre el bloque VI: Gráfica en la posición donde se encuentra el obstáculo a eliminar	El sistema muestra la ventana Modificar Obstáculo

7.5.1.12- Modificar Opciones Simulación

CASO DE USO:	Modificar Opciones Simulación
Actores	Usuario
Descripción	Se encarga de modificar las opciones de la

	simulación tales como la velocidad, aceleración, posiciones iniciales…
USUARIO	**APLICACIÓN**
1- Click en "Modificar" (del bloque IV: Opciones Simulación) 3- El usuario modifica los campos que desee y confirma los cambios.	2- Mostrar ventana para Modificar dichos valores 4- El sistema actualiza la información y redibuja las posiciones actualizadas.

CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	En 3. El usuario introduce datos incorrectos.	El sistema muestra un error conforme los datos introducidos son erróneos (si son números) o mantiene los valores anteriores correctos en el caso de introducir valores no numéricos.
#2	En 1. El usuario pulsa sobre el botón "Modificar Opciones" del bloque I: Menú	El sistema muestra la ventana Modificar Opciones.

7.5.1.13- Consulta Datos Láser

CASO DE USO:	Consulta Datos Láser
Actores	Usuario
Descripción	Permite que el usuario pueda comprobar los datos devueltos por el láser para una posición y mapa dados.
Precondiciones	No haber empezado la simulación o estar pausada.
USUARIO	**APLICACIÓN**
1- Click en "Obtener Datos Láser" (del bloque V: Resultados Simulación en la pestaña Láser)	2- El sistema realiza una llamada a la función " datos_láser" e imprime los datos obtenidos.

7.5.1.14- Start

CASO DE USO:	Start	
Actores	Usuario	
Descripción	Se encarga de iniciar la simulación.	
USUARIO	**APLICACIÓN**	
1- Click en "Start" (del bloque VII: Control Simulación)	2- El sistema inicia la búsqueda del camino desde el punto inicial hasta el punto meta. Para cada punto realiza llamadas a dibujar(para las figuras y el camino ya calculado), a imprimirPosicion(imprime las posiciones por las que pasa el quad) y a datos_láser para obtener los nuevos valores devueltos por el sensor.	
CAMINOS ALTERNATIVOS		
Número	USUARIO	APLICACIÓN
#1	La posición inicial coincide con la posición final	El sistema avisa de esta situación.

7.5.1.15- Stop

CASO DE USO:	Stop
Actores	Usuario
Descripción	Se encarga de detener la simulación.
USUARIO	**APLICACIÓN**
1- Click en "Stop" (del bloque VII: Control Simulación)	2- El sistema detiene por completo la búsqueda del camino parando por tanto, la impresión de posiciones y el dibujo de las mismas.

7.5.1.16- Pausar

CASO DE USO:	Pausar
Actores	Usuario
Descripción	Se encarga de pausar momentáneamente la simulación.
USUARIO	**APLICACIÓN**
1- Click en “Pause” (del bloque VII: Control Simulación)	2- El sistema detiene momentáneamente la búsqueda del camino parando por tanto, la impresión de posiciones y el dibujo de las mismas.

7.5.1.17- Reanudar

CASO DE USO:	Reanudar
Actores	Usuario
Descripción	Se encarga de reanudar la simulación.
USUARIO	**APLICACIÓN**
1- Click en “Pause” (del bloque VII: Control Simulación)	2- El sistema reanuda la búsqueda del camino, continuando por tanto, la impresión de posiciones y el dibujo de las mismas.

PARTE III: IMPLEMENTACIÓN

8- IMPLEMENTACIÓN

8.1- Introducción

Como se ha comentado en el apartado 4 (Evolución Tecnológica), esta aplicación se ha implementado en el lenguaje de programación Java y mediante el entorno de programación Eclipse.

Tras realizar el análisis y diseño de la aplicación el siguiente paso consistía en implementarlo, en plasmar todas las ideas y conseguir el resultado deseado, es decir, en poner en práctica todo el estudio previo realizado

La finalidad de este apartado es explicar cómo ha ido evolucionando la aplicación y cómo se han resuelto e implementado los principales problemas que planteaba el desarrollo de este software. El código fuente de estas funciones explicadas se pueden encontrar en el Apéndice B.

8.2- Evolución

Antes de empezar a visualizar los primeros resultados de la elaboración de un proyecto, se debe pasar por distintas fases hasta que se llega a la última y más deseada, el resultado final.

La primera fase de este proyecto fue la elaboración de la función que simulaba el funcionamiento de un sensor láser. Este programa no interactuaba con el usuario ni tenía interfaz gráfica, simplemente devolvía por pantalla las distancias calculadas según los obstáculos que se introducían previamente en el código.

Una vez que la función ya devolvía los datos correctamente, empecé a crear la interfaz gráfica del usuario, buscando que fuera lo más intuitiva y sencilla de manejar posible. Por ello, la idea fue dividir la pantalla principal en dos, por un lado que permitierar ver los valores de los parámetros introducidos y por otra parte, la visualización del camino seguido por el quad.

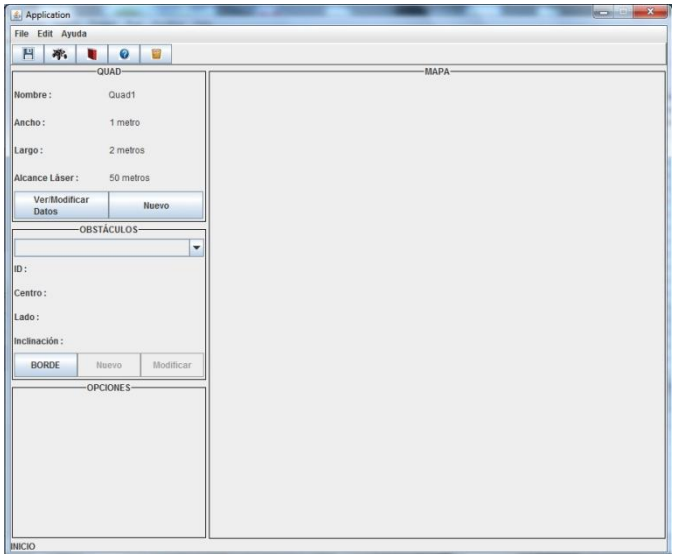


Imagen 8.1- Primera implementación

Tras conseguir que se dibujaran los obstáculos correctamente y poder realizar operaciones sobre ellos como modificar, eliminar, etc empecé a centrarme más en cómo obtener el movimiento del Quad y la planificación del camino. Para poder observar mejor los resultados obtenidos se añadió un área de texto donde se mostraban las posiciones calculadas por el planificador. Además añadí una Ayuda donde poder consultar el manual de usuario de la aplicación. Tras algunos cambios, el resultado obtenido era el siguiente:

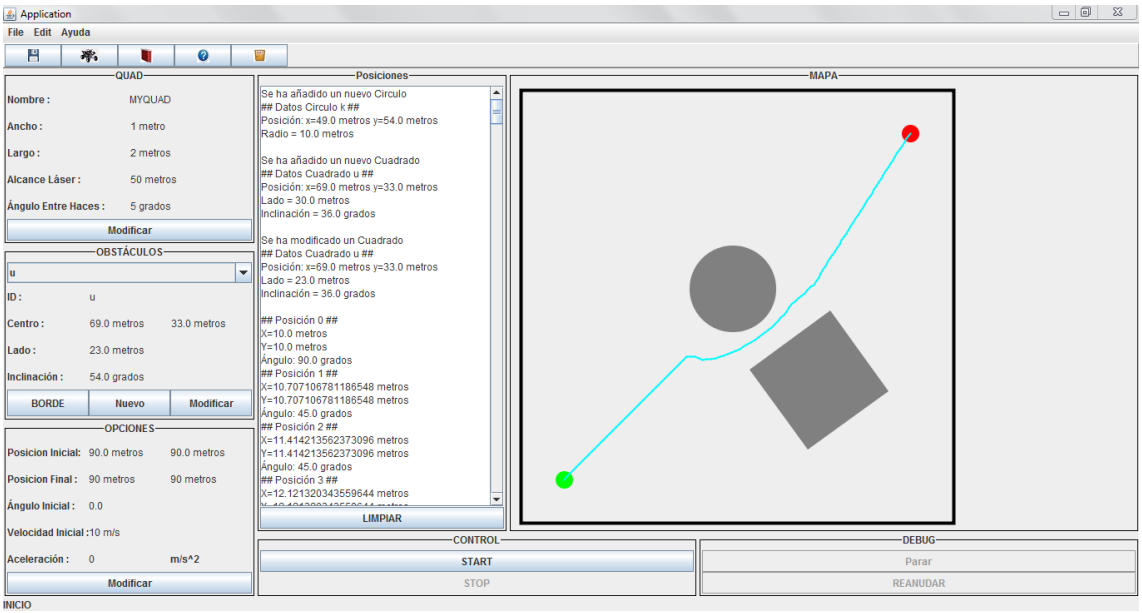


Imagen 8.2- Segunda implementación

Finalmente, y pensando siempre en hacer una aplicación que permita un buen estudio de planificación de caminos, añadí otra área de texto desde donde se pudiera obtener los datos devueltos por el láser desde la posición en las que se encuentra en ese momento el quad.

En este momento me di cuenta que la forma de calcular el movimiento del Quad no era correcto y por tanto había que cambiarlo. Debido a que el quad es un vehículo con configuración Ackerman, la forma de calcular los ángulos y velocidad de las ruedas es mediante la cinemática Ackerman.

Por último, se ha considerado conveniente que la aplicación permitiera guardar los resultados en un fichero con un formato adecuado para poder visualizar los resultados en otro programa como por ejemplo Matlab.

El aspecto final de la aplicación es:

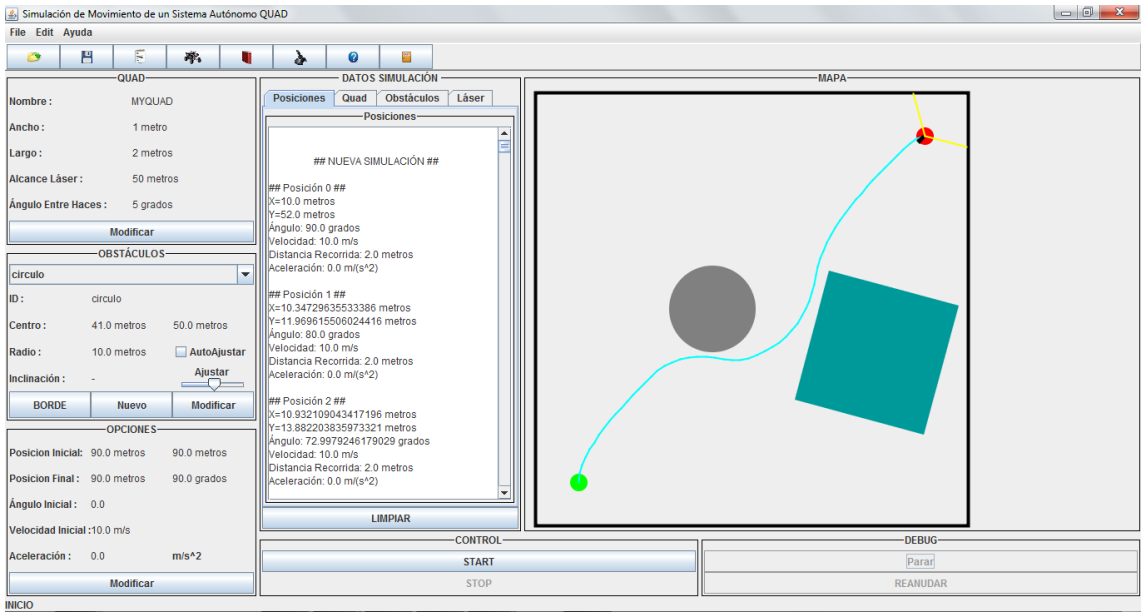


Imagen 8.3- Implementación final

8.2- Obtención de los Datos del Láser

Esta función tiene como objetivo simular el funcionamiento del láser incorporado en el Quad, es decir, debe devolver información sobre los objetos que se encuentran alrededor del vehículo. Esta información son las distancias calculadas desde la posición del láser hasta el objeto encontrado o, en el caso de no encontrar obstáculo, la distancia máxima de lectura del láser.

Los valores propios del láser como ángulo de abertura, ángulo entre haces, alcance máximo, etc. vienen definidos por defecto, pero el usuario puede decidir cambiarlos si así lo desea.

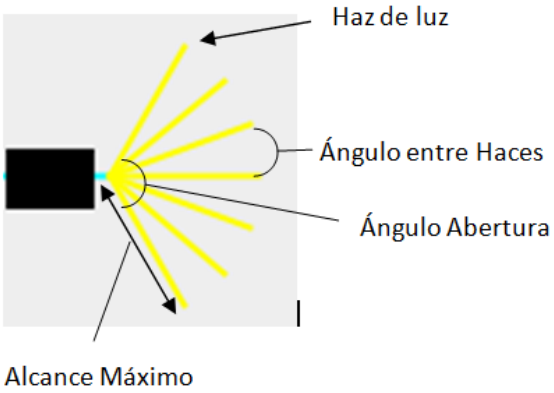


Imagen 8.4- Variables Láser

Esta función es llamada por el planificador del camino para comprobar la situación alrededor del vehículo y de esa forma decidir la siguiente posición.

La posición de láser depende de la posición del Quad, y del mismo modo, la inclinación de éste también afecta al láser.

La idea principal es la de recorrer todo los haces del sensor y comprobar si interceptan con algún obstáculo. De ser así, se calcula esa distancia.

Con este fin, cada haz es implementado como una recta y la forma de comprobar si existen obstáculos cercanos es mediante intersecciones: para los obstáculos con forma circular se comprueba si existe intersección entre el haz y la circunferencia. Análogamente para los

cuadrados, se calculan las intersecciones entre el haz y las cuatro rectas que forman el cuadrado.

Para comprobar el funcionamiento de esta función se ha utilizado el programa AUTOCAD debido a su facilidad para dibujar en 2D y rapidez para acotar. De esta forma se han ido analizando, corrigiendo y comprobando los resultados obtenidos por el programa

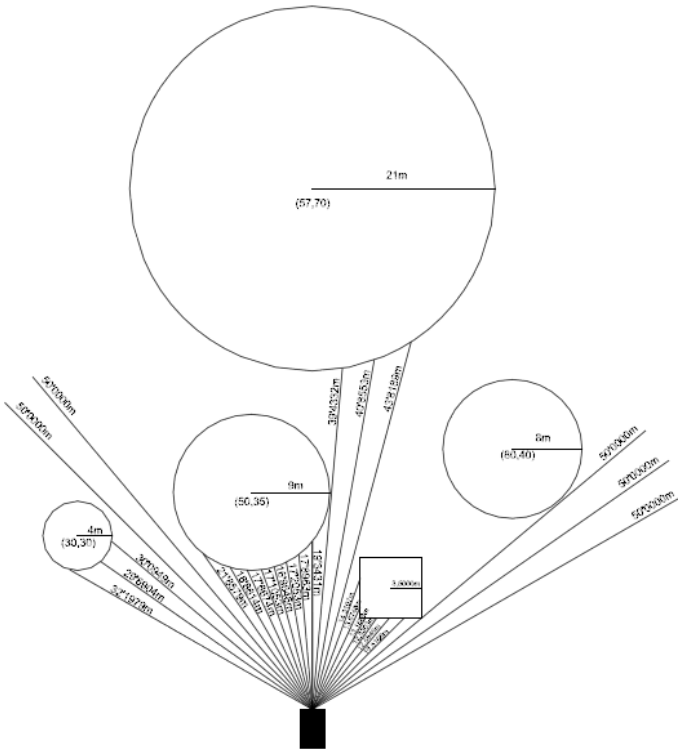


Imagen 8.5- Ejemplo Mapa

8.3- Movimiento. Cinemática Ackerman.

El Quad es un vehículo con una configuración 2+2 es decir, dos ruedas delanteras para la dirección y dos ruedas traseras de tracción. Esta configuración, conocida como "Configuración Ackerman", está pensada para evitar el derrape de las ruedas.

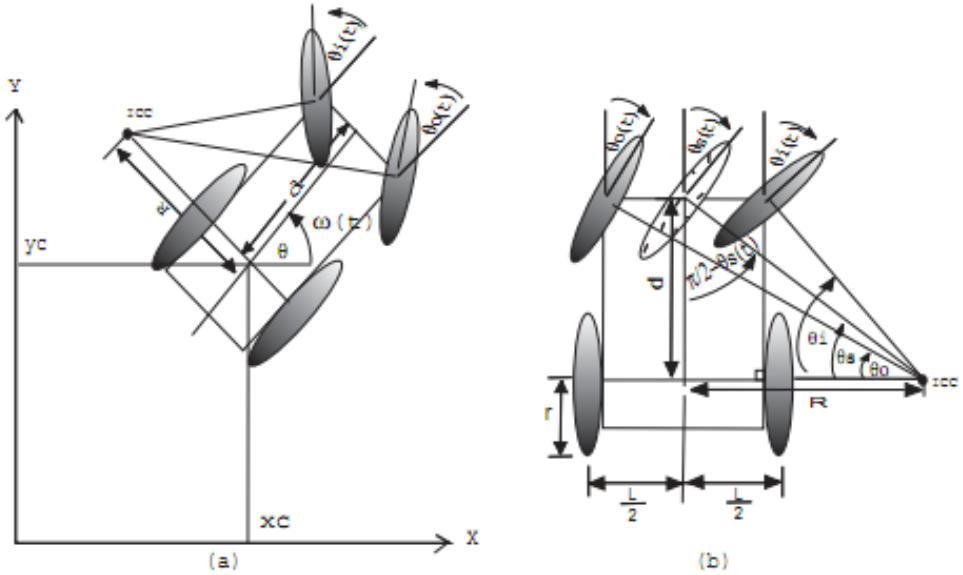


Imagen 8.6-Configuración Ackerman

Como se muestra en la Imagen 6.3, el objetivo es que el vehículo gire en torno a un único punto, el centro instantáneo de rotación o CIR (ICC). Para ello, la rueda delantera interior a la curva se rota un ángulo mayor que la interior, a fin de evitar que las ruedas patinen.

La distancia desde la posición del vehículo hasta este centro CIR se calcula según la siguiente Ecuación:

$$radio = largo * \tan\left(\frac{\pi}{2} - giro\right)$$

siendo el giro, el ángulo de curvatura del volante, es decir, el giro que se quiere realizar sobre el vehículo.

A partir del CIR y de las dimensiones del Quad, es decir, del ancho y del largo, se pueden deducir las siguientes ecuaciones (Ecuación 1 y 2) que relaciona la dirección de las dos ruedas delanteras.

$$\cot\left(\theta_i(t)\right) = \frac{R - \frac{ancho}{2}}{largo} \quad (1)$$

$$\cot\left(\theta_o(t)\right) = \frac{R + \frac{ancho}{2}}{largo} \quad (2)$$

Una vez resuelto el problema de la dirección es necesario abordar el tema de la velocidad. La velocidad del vehículo (v) es la velocidad que poseen las ruedas de tracción, que son las ruedas traseras de la configuración Ackerman. Entonces la velocidad, Vs, que es la velocidad de la rueda imaginaria (la cual tiene una inclinación igual al giro que se quiere realizar) es calculada mediante la siguiente ecuación:

$$Vs = \frac{v}{cos(\theta_s(t))} \quad (3)$$

La nueva inclinación se calcula sumándole a la actual el ángulo girado por el vehículo, es decir, la velocidad angular del mismo. Esta velocidad se calcula según la Ecuación 4:

$$\omega(t) = \frac{Vs(t) * sin\ (giro(t))}{ancho} \quad (4)$$

El cálculo de las velocidades angulares de las dos ruedas delanteras se calcula de un modo análogo al calculado, primero obteniendo las velocidades v_i y v_o y a partir de éstas las velocidades angulares ω_i y ω_o .

Por último, la nueva posición (x,y) del vehículo viene dada por la Ecuación 5:

$$\begin{bmatrix} x(t+\Delta t) \\ y(t+\Delta t \end{bmatrix} = \begin{bmatrix} x(t) + v(t) * \Delta t * \cos\left(\theta(t)\right) \\ y(t) + v(t) * \Delta t * \sin\left(\theta(t)\right) \end{bmatrix} \quad (5)$$

Por ejemplo , si se quiere girar 45 grados a la derecha con un Quad de dimensiones largo=2 y ancho=1, se obtendría un resultado como el que se puede observar en la Imagen 6.4.

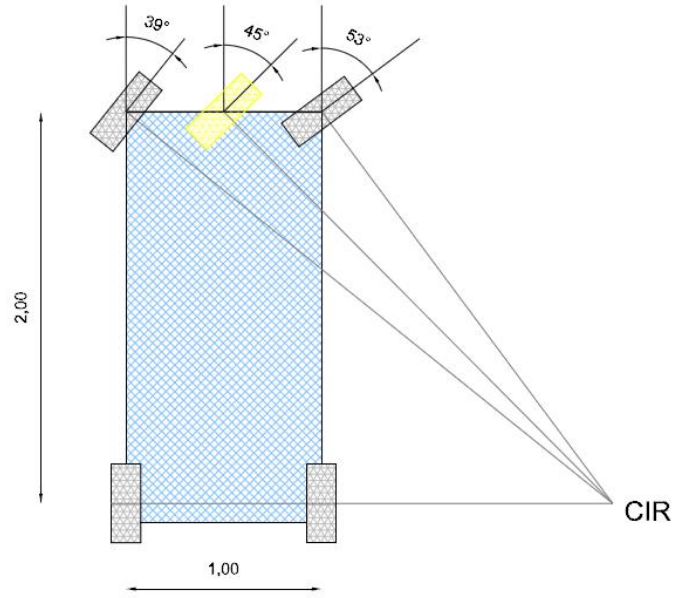


Imagen 8.7- Ejemplo Movimiento Quad

PARTE IV: MANUAL DE USUARIO

9- INTRODUCCIÓN

Este Manual de Usuario tiene como objetivo explicar el funcionamiento de la aplicación "Simulación de Movimiento de un Sistema Autónomo QUAD" y guiar al usuario a través de todas las opciones que ofrece el programa para conseguir el resultado deseado.

9.1- Descripción general de la Aplicación

La finalidad de este proyecto es la creación de una plataforma software que simule la generación y transmisión de órdenes de movimiento a un vehículo autónomo tipo Quad.

Éste Quad lleva incorporado un sensor láser, con el fin de proporcionar al vehículo la información sobre las distancias a los objetos que se encuentran a su alrededor y de esta forma poder evitar las colisiones con los objetos de su entorno y decidir qué camino tomar para llegar al punto destino.

Por tanto el objetivo de esta aplicación es la simulación del funcionamiento del sensor láser, el cual, le proporciona al Quad una serie de valores que el vehículo debe ser capaz de interpretar y decidir, a partir de estos valores, cual es el siguiente movimiento a realizar para llegar a la posición final.

El requisito hardware mínimo indispensable para poder lanzar la aplicación es tener instalada alguna de las versiones de la máquina virtual de Java (JVM) más recientes.

9.2- Pantalla inicial de la aplicación

Tras hacer doble clic sobre el icono de la aplicación, aparece la pantalla principal que tiene el siguiente aspecto:

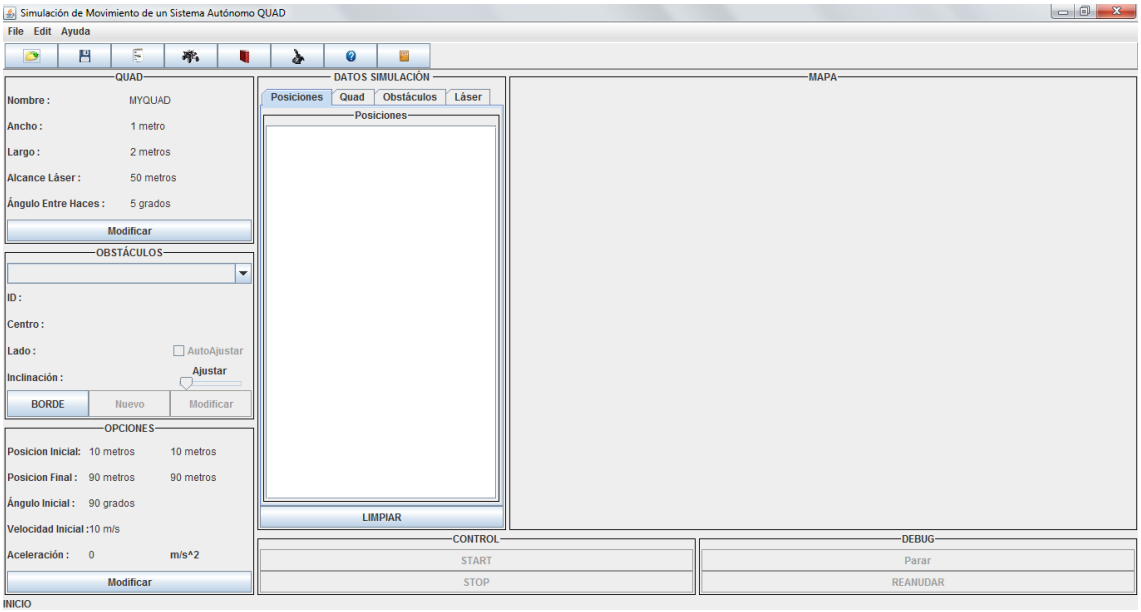


Imagen 9.1: Pantalla principal de la aplicación

Como se puede apreciar en la Imagen 8.1 la interfaz está dividida en siete bloques con el fin de facilitar la comprensión y el manejo de la misma. Se busca que el usuario que está realizando la simulación tenga siempre visible las características más importantes de la misma, es decir, que pueda comprobar el resultado de la simulación (la gráfica) mediante los valores utilizados sin necesidad de ir abriendo nuevas ventanas, facilitando de esta manera, el estudio del resultado obtenido.

Imagen 9.2: Estructura de la pantalla principal.

A continuación se explicarán cada una de las partes, su utilidad y su funcionamiento.

9.3- Bloque I: Menú

El menú está formado por una serie de botones que nos permiten acceder de una manera más rápida a las ventanas desde donde se pueden modificar los valores del Quad, de los obstáculos y de las opciones de la simulación. Mediante los botones Abrir y Guardar se puede salvar y abrir la configuración de una antigua simulación. Además incluye una Ayuda desde donde se puede leer el funcionamiento de la aplicación. Por último se encuentra el botón Salir, que sirve para abandonar la aplicación.

Imagen 9.3: Bloque I: Menú

9.4- Bloque II: Datos Quad

En este bloque se puede apreciar el valor de los parámetros del Quad.

Al iniciar la aplicación, el Quad tiene ya establecidos valores por defecto en todos sus parámetros para facilitar al usuario el no tener que ir introduciendo cada vez que ejecuta la aplicación todos los valores, y de esta forma, simplemente tiene que modificar aquellos valores que le interesan.

Debido a temas de espacio, no se pueden mostrar todos los parámetros en la pantalla principal, limitándose únicamente a 5. Para poder modificar o ver el resto de valores hay que hacer clic en modificar y nos aparecerá una nueva ventana con los valores actuales del Quad.

Otra forma de acceder a esta ventana es mediante el botón  del bloqueI: Menú.

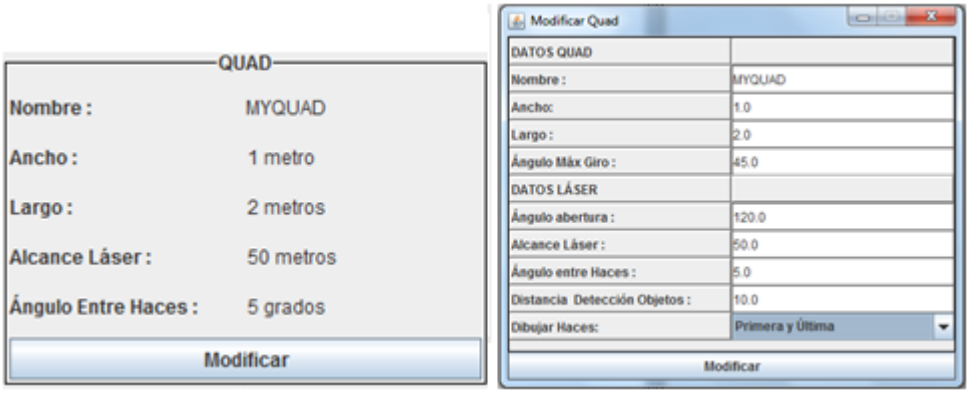


Imagen 9.4: Bloque II: Datos Quad

El significado de los distintos parámetros son:

NOMBRE	EXPLICACIÓN
DATOS QUAD	
Nombre :	Nombre del Quad.
Ancho :	Medida Ancho del Quad.
Largo :	Medida Largo del Quad.
Ángulo Máx. Giro :	Ángulo Máximo de Giro.
DATOS LÁSER	
Ángulo obertura :	Máxima obertura del Láser.
Alcance Láser :	Alcance máximo del Láser. Por defecto 50.
Ángulo entre Haces :	Distancia en grados entre los distintos haces del Láser. Este ángulo tiene que ser menor y múltiplo del ángulo de obertura.
Distancia Detección Objetos :	Distancia utilizada por el Quad para la detección de obstáculos. Si encuentra objetos a una distancia menor que ésta, empieza a reaccionar. Este valor debe ser menor que el Alcance del Láser.
Dibujar Haces :	Se utiliza para el dibujado de las Haces. Se puede seleccionar “La Primera y Última”, “Todas” o “Ninguna”.

Tabla 9.1: Parámetros Datos Quad

Todos los parámetros, a parte de las restricciones mencionadas, deben cumplir la condición de ser mayores de 0. En el caso de que alguna de las restricciones no se cumplan, el programa avisa o se queda con el valor anterior válido

9.5- Bloque III: Datos Obstáculos

En este bloque se muestran los obstáculos insertados en el mapa con el fin de obstaculizar el paso del vehículo.

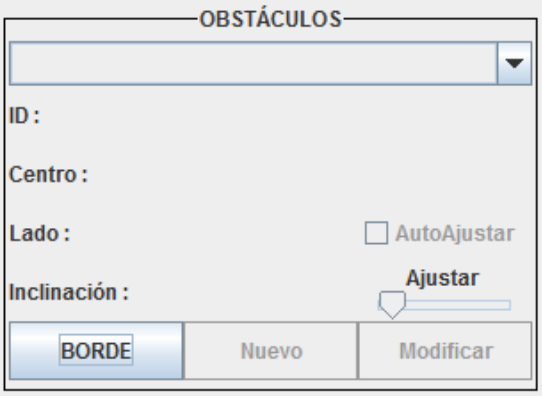


Imagen 9.5: Bloque III: Datos Obstáculos

Cuando se arranca la aplicación, no hay ningún obstáculo por defecto y por tanto es el propio usuario el que debe introducir tantos obstáculos como desee. El primero de ellos debe ser el borde, es decir, los límites del terreno que queremos utilizar para hacer las pruebas.

Al pulsar el botón “Borde” nos aparece una nueva ventana para la introducción de los datos. Por tratarse del Borde la mayoría de las variables están ya predefinidas, como es el caso del nombre, del Punto Inicial y la Inclinación. El usuario debe elegir si desea un terreno con una forma cuadrada o rectangular. Según esta elección se deberá introducir únicamente el ancho (para los cuadrados) o el ancho y el largo (para los rectángulos).

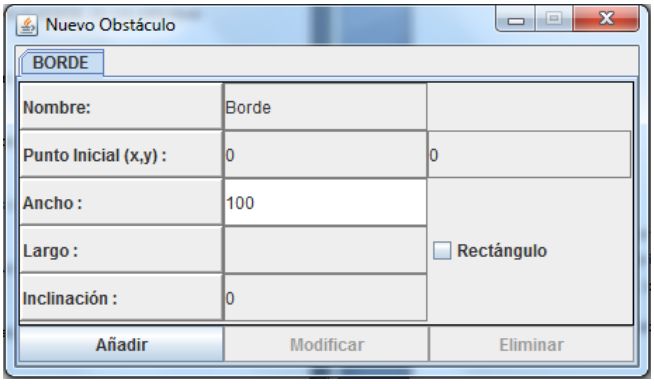


Imagen 9.6: Bloque III: Borde

Si se desea modificar o incluso eliminar el borde, se debe acceder de nuevo a esta ventana y pulsar sobre el botón correspondiente.

Como se puede apreciar en la Imagen 8.5 (Bloque II: Datos Obstáculos), existe una opción que se llama AutoAjustar y cuya función no es más que ajustar el tamaño del borde introducido al tamaño del panel donde se dibuja la gráfica. Por defecto el tamaño del panel está adecuado para valores de Ancho=100, pero con esta opción, se pueden introducir bordes de cualquier tamaño pudiéndolos visualizar correctamente en la pantalla. Se ha añadido además, una slider para ajustar manualmente esta vista, en el caso que el usuario así lo desee.

Una vez introducido el borde, se pueden ya insertar los obstáculos. Éstos pueden ser cuadrados o circulares. Para insertar un nuevo obstáculo y por tanto acceder a la ventana de Añadir Obstáculo existen tres formas para hacerlo.

1- Mediante el botón “Nuevo” del bloque: Datos Obstáculos.
2- Mediante el botón  del bloque: Menú.
3- Haciendo clic con el botón izquierdo sobre el bloque: Gráfica. Este modo se diferencia de los dos anteriores en que se guarda la posición donde se ha clicado como centro del obstáculo a introducir, facilitando así la inserción de obstáculos.

La ventana “Nuevo Obstáculo” está dividida en dos pestañas, según si se desea introducir un Círculo o un Cuadrado. Según esta elección se deben introducir los parámetros correspondientes y pulsar en Añadir.

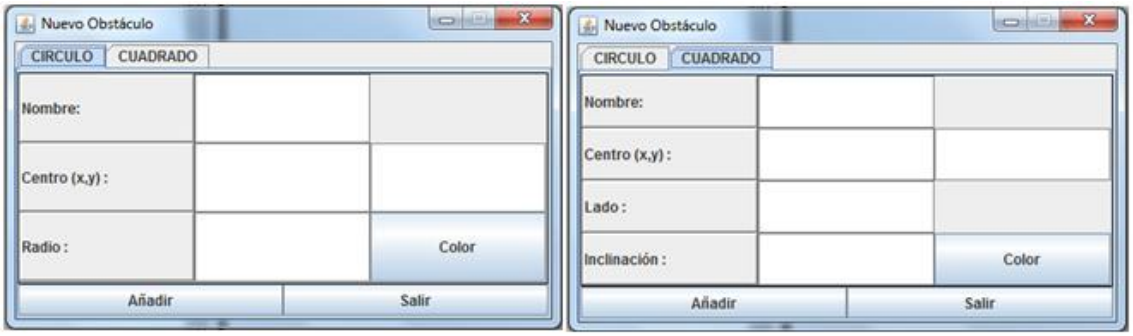


Imagen 9.7: Bloque III: Nuevo Obstáculo

A medida que se van introduciendo los nuevos obstáculos, se van almacenando en un ComboBox. Al seleccionar el ítem de un obstáculo, se mostrarán sus propiedades en el bloqueIII: Datos Obstáculos de la Pantalla Principal.

Al igual que en el borde, en los obstáculos también se pueden hacer cambios accediendo a la ventana de "Modificar Obstáculo". Para ello hay que seleccionar su ítem (el nombre del objeto) y pulsar en Modificar, o bien, pulsar con el botón izquierdo del ratón sobre el objeto en el Bloque:Gráfica. Esta ventana permite también eliminar el objeto.

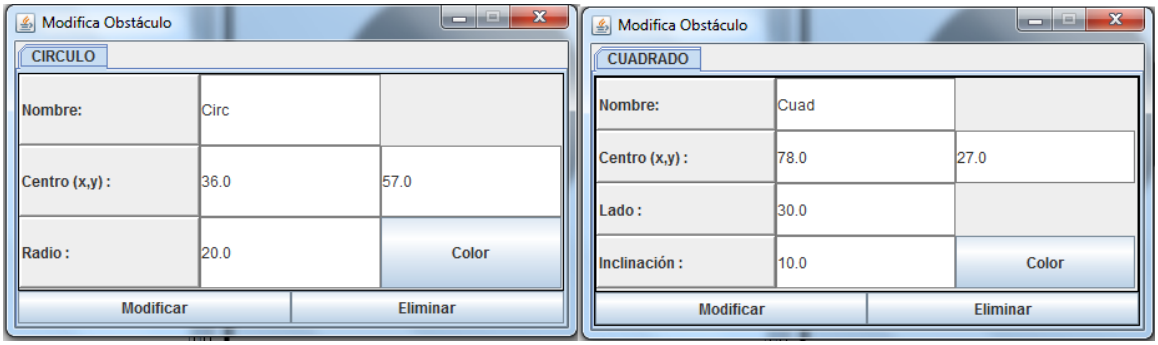


Imagen 9.8: Bloque III: Modifica Obstáculo

9.6- Bloque IV: Opciones Simulación

Este bloque está pensado para englobar en él aquellos parámetros que no son características propias del quad, sino más bien, para aquellos parámetros que varían las características de la simulación, como por ejemplo, la posición inicial, final, velocidad...

Al igual que con los datos del Quad, al iniciar la aplicación, todas estas opciones de simulación tienen valores por defecto y será el usuario el que decida si cambiarlos o mantenerlos. Para hacer alguna modificación se debe pulsar en “Modificar”

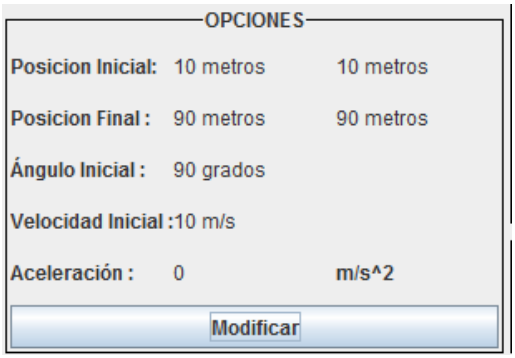


Imagen 9.9: Bloque IV: Opciones Simulación

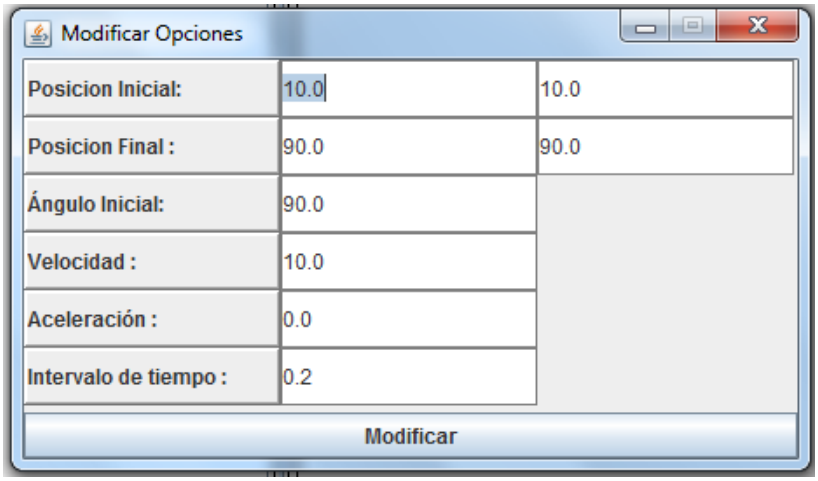


Imagen 9.10: Bloque IV: Modificar Opciones Simulación

El significado de los distintos parámetros son:

NOMBRE	EXPLICACIÓN
Posición inicial :	Primera posición del camino.
Posición final :	Posición meta.
Ángulo inicial :	Inclinación inicial de las ruedas del Quad.
Velocidad :	Velocidad inicial del Quad.
Aceleración :	Aceleración del Quad en ese momento. Se puede modificar durante la ejecución de la simulación.

Intervalo de tiempo	Intervalo de tiempo entre el cálculo de una posición a otra.

Tabla 9.2: Parámetros Opciones SImulación

Otra forma de introducir la posición inicial y final es mediante el ratón en el bloque VI: Gráfica, ya que basta con pulsar con el botón derecho en el punto donde queremos y saldrá un menú desplegable para elegir entre ambas posiciones.

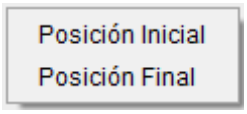


Imagen 9.12: Bloque IV: Modificar Posición inicial y final

9.7- Bloque V: Resultados Simulación

Observando el dibujo de la simulación, se puede apreciar ya si los resultados son correctos o no, pero no sabemos con exactitud los puntos por los que pasa o cuáles son los valores que devuelve el láser en ese punto. Por ello se ha diseñado este bloque, cuya intención es proporcionar al usuario un resultado más analítico para que pueda interpretar mejor los movimientos del Quad.

En el bloque se pueden observar 4 pestañas, con 4 áreas de texto diferentes.

- **Posiciones**: Se van imprimiendo las posiciones actuales del Quad, es decir, se muestran todas las posiciones por las que pasa el Quad desde la posición inicial hasta la posición meta. Se muestra además, la inclinación de las ruedas, la velocidad y la aceleración en ese momento.
- **Quad**: se muestran las características del Quad.
- **Obstáculos**: se muestran todos los obstáculos del mapa actual junto con sus características (posición, centro, lado...).
- **Láser**: esta pestaña tiene la intención de mostrar los resultados que devuelve el láser en esa posición e inclinación en la que se encuentra el Quad. Para ello, primero hay que poner en Pause la simulación y pulsar “Obtener Datos Láser”.

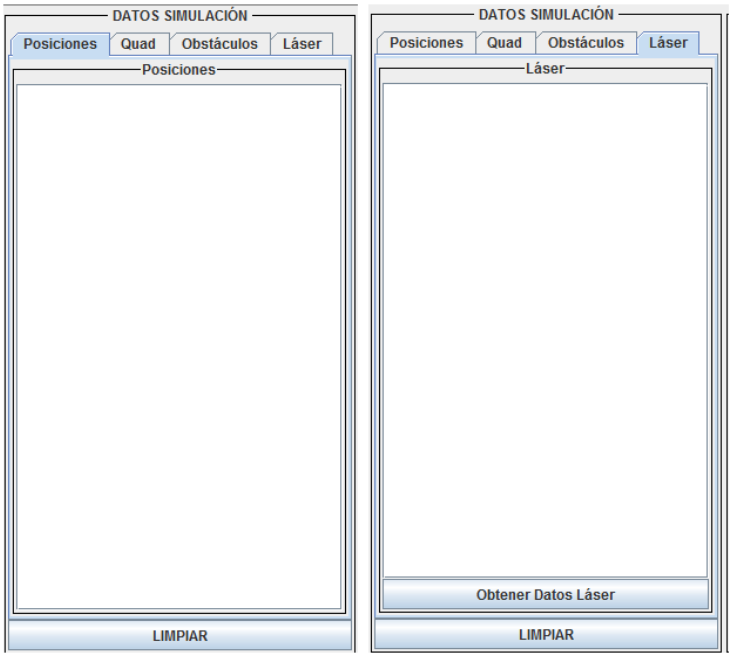


Imagen 9.13: Bloque V: Resultados Simulación

Por último, el botón Limpiar sirve para vaciar las 4 áreas de texto (Posiciones, Quad, Obstáculos y Láser), en el caso que el usuario así lo desee.

9.8- Bloque VI: Gráfica

Este bloque, quizás uno de los más importantes, permite visualizar los resultados de la simulación. Como ya se ha comentado antes, desde este bloque se pueden insertar nuevos obstáculos, modificarlos, eliminarlos y cambiar las posiciones de partida y meta.

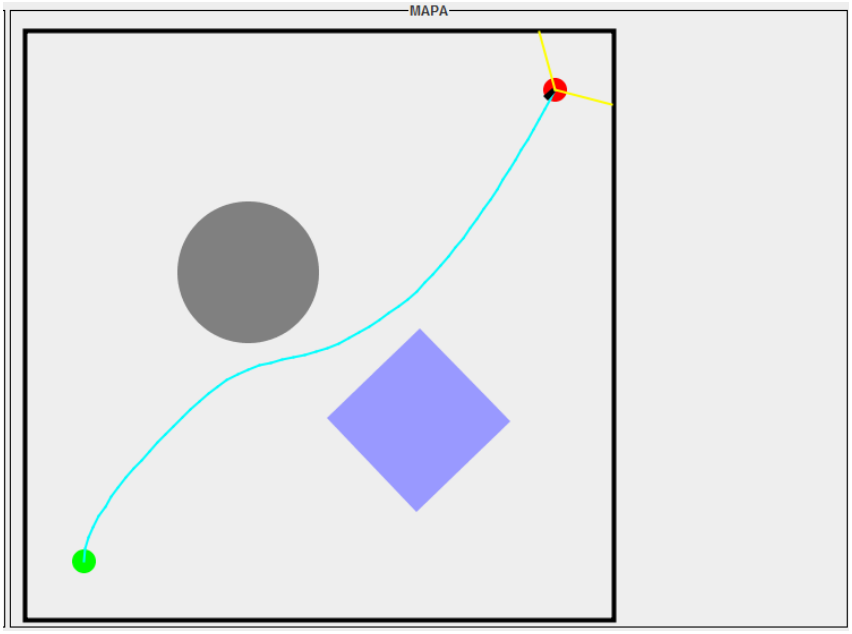
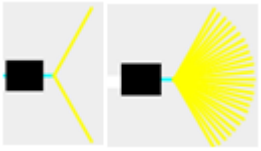


Imagen 9.14: Bloque VI: Gráfica

- La posición inicial se representa mediante un círculo verde.
- La posición final se representa mediante un círculo rojo.
- El Quad se representa mediante un rectángulo negro con las medidas (ancho y largo) que se han establecido.
- Los haces del láser se representan mediante líneas amarillas



- Los obstáculos se representan con el color elegido para cada objeto.
- El camino se representa mediante una polilínea azul.

9.9- Bloque VII: Control Simulación

Esta parte de la aplicación está dividida en dos: por un lado encontramos los botones de control (Start y Stop) y por otro, los botones de debug (Parar, Reanudar).

Imagen 9.15: Bloque VII: Control Simulación.

Según el instante de la simulación los botones estarán activados o desactivados. Las funciones de cada uno son:

- **START**: Sirve para empezar la simulación y estará desactivado durante la duración de la misma.
- **STOP**: sirve para finalizar la simulación, tanto como si se ha llegado a la posición final o si se desea finalizarla antes.
- **PARAR**: sirve para detener momentáneamente la ejecución, por si se desea ver las posiciones con mayor detenimiento, los valores devueltos por el láser…etc. Además cuando la simulación se encuentra en este punto, se puede modificar la aceleración del vehículo cambiando simplemente el valor en el Bloque IV: Opciones Simulación.
- **REANUDAR**: Permite continuar la simulación en el mismo punto en el que se había detenido.

CONCLUSIONES

Las conclusiones obtenidas al finalizar el proyecto, a grosso modo, han sido que la programación de cualquier aplicación, ya sea con el fin que sea, es tan interesante como complejo y que aunque parece que estás analizando e implementando una gran cantidad de cosas, en realidad, siempre hay mucho por aprender y que nunca se sabe lo suficiente.

Al empezar el proyecto me había fijado una serie de objetivos y que a medida que se fueron cumpliendo, éstos fueron aumentando y mejorando con el fin de obtener el mejor resultado posible. Durante la elaboración del mismo he llegado a varias conclusiones de las que destaco:

- El objetivo personal de crear una aplicación funcional y útil que pudiera ser utilizada en proyectos futuros.
- Lo importante que es realizar una buena planificación para la elaboración de las distintas fases de un proyecto.
- Fundamental es también realizar un buen análisis del problema y utilizar las herramientas y lenguajes adecuados para evitar futuros problemas.
- Cuánto más clara y sencilla sea la interfaz, más información útil aportará al usuario.
- Es importante el punto de vista de más gente, ajena a este proyecto, ya que su opinión se tuvo en cuenta a la hora de modificar algunos aspectos que podrían no ser claros para ellos.

En resumen, aunque el camino de elaboración de un Proyecto Final de Carrera implica dedicarle muchas horas y trabajo, y por supuesto superar las dificultades que se van presentando, mi valoración personal, una vez superado este camino, es más que positiva.

Son muchos los posibles trabajos futuros que se podrían realizar tras éste realizado, pero siguiendo la línea en la cual se encuentra la aplicación, se podría implementar el movimiento de obstáculos con el fin de darle más realidad al objetivo final que es la generación de movimiento de un vehículo autónomo tipo Quad en un entorno real.

BIBLIOGRAFÍA

[1] Programación de Threads en Java

http://eisc.univalle.edu.co/materias/Programacion_Interactiva/lecturas/Lectura8.pdf

[2] Enciclopedia libre
http://es.wikipedia.org/wiki/Wikipedia:Portada

[3]Apuntes asignatura Ingeniería de la Programación

[4]Página web de Eclipse
http://www.eclipse.org/

[5] Información sobre Quads
http://www.dgt.es/revista/archivo/pdf/num176-2006-quads.pdf

[6]Historia de la Robótica
http://robotiica.blogspot.com/2007/10/historia-de-la-robtica.html

[7]Real Academia Española
http://www.rae.es/rae.html

[8]Programación en Java
http://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java

REFERENCIAS Y FUENTES

[1] Michael Brady and Richard Paul, editors. Robotics Research: The First International Symposium. The MIT Press, Cambridge MA, 1984

[2][Joseph L. Jones and Anita M. Flynn. Mobile robots: Inspirations to implementation. A K Peters Ltd, 1993]

[3] Macchiavello, 2008

[4] Joseph L. Jones and Anita M. Flynn,1983

[5] Craig y John, 2006

PARTE V: APÉNDICES

APÉNDICE A: DESCRIPCIÓN DE CLASES

A continuación se explica la función de cada una de clases que forman la aplicación. Los paquetes que contienen las clases están ordenados por orden alfabético:

Paquete Ayuda

- **Ayuda:** Esta clase se encarga de mostrar el manual de usuario desde el botón de ayuda de la aplicación. Según la aclaración que se desee se mostrará una información u otra.

Paquete Datos

- **Fichero**: Mediante esta clase se pueden abrir y guardar ficheros, es decir, se pueden importar ficheros, con la estructura adecuada, para abrir modelos de mapas previamente guardados. Asimismo se pueden generar dos tipos de ficheros, uno donde se almacene la configuración actual (obstáculos, estado del quad, etc) y otro donde se registre el valor de las variables calculadas en la ejecución de la aplicación.

Paquete Elementos

- **Posición**: Esta clase se utiliza para representar un punto mediante sus coordenadas (x,y). Además tiene dos parámetros más utilizadas para la planificación del camino.
- **Recta**: Esta clase se utiliza para representar una recta mediante su forma general Ax+By+C=0. La recta es creada a partir de dos puntos pertenecientes a la misma.

Paquete Excepciones

- **ExcepcionMenorIgual0**: excepción utilizada para indicar que el radio o el lado de los obstáculos no puede ser menor o igual a 0.
- **ExcepciónValoresBlancos**: esta excepción salta cuando se ha dejado alguna casilla de los parámetros a introducir en blanco.
- **ExcepciónValoresIncorrectos**: esta excepción es utilizada para comprobar que las distintas restricciones entre valores se cumplen .

Paquete Interfaz

- **DBPanelFiguras**: Esta clase implementa un hilo de ejecución paralelo a la aplicación que se encarga de dibujar los obstáculos y el camino que va siguiendo el quad.
- **DBPanelPosiciones**: Esta clase implementa otro hilo de ejecución paralelo a la aplicación y tiene la función de mostrar la posición y otras variables del quad en ese instante de tiempo.
- **FigurasJPanel**:Clase que extiende a DBPanelFiguras y es el que interacciona con la clase camino para dibujar el camino. Es el intermediario entre DBPanelFiguras y el resto de clases que solicitan sus métodos.
- **ImprimirPosiciones**: Clase que extiende a DBPanelPosiciones y es el que interacciona con la clase camino para mostrar los datos del quad en ese momento. Es el intermediario entre DBPanelPosiciones y el resto de clases que solicitan sus métodos. Mediante esta clase también se muestran los valores devueltos por el sensor láser en la posición en la que se encuentra en ese instante que el usuario lo solicita.
- **Main**: Esta clase contiene el main() de la aplicación. Implementa además, la interfaz gráfica de usuario con todas sus funcionalidades: abrir archivo, modificar opciones, añadir obstáculos.. A partir de esta interfaz principal se pueden abrir las distintas ventanas que completan la totalidad de la interfaz de usuario.
- **ModificaObstaculo:** Esta clase implementa la ventana desde donde se pueden modificar los distintos parámetros que definen al obstáculo seleccionado.
- **ModificarOpciones:** Esta clase implementa la ventana desde donde se pueden modificar las opciones de simulación como la velocidad, la aceleración, la posición inicia y final, etc.
- **ModificarQuad:** Esta clase implementa la ventana desde donde se pueden modificar las opciones que definen al quad como largo, ancho, ángulo de giro. También se pueden modificar las características de su sensor láser (ángulo de abertura, ángulo entre haces, etc.).
- **NuevoObstaculo:** Esta clase implementa la ventana desde donde se puede añadir el borde en primer lugar, y después el número de obstáculos que el usuario estime oportuno, ya sean cuadrados o círculos.

Paquete Objects

- **Circulo**: Clase que hereda de Object y se utiliza para representar un círculo mediante la ecuación de la circunferencia $(x-a)^2+(y-b)^2=radio^2$. Entre los distintos métodos destaca el de hallar los puntos intersección con una recta.
- **Cuadrado**: Clase que hereda, al igual que Círculo, de Object y representa un cuadrado (o rectángulo, en el caso del borde). Esta representación viene dada por un vector de cuatros rectas que representan los lados, y otro vector de cuatro posiciones que representan los vértices. Se puede también indicar la inclinación del mismo.
- **GestionObstaculos**: clase utilizada para llevar el control de los obstáculos insertados mediante un ArrayList de objetos. Además también controla los dos círculos que se utilizan para representar el inicio y el fin del recorrido.
- **Object**: clase abstracta que representa un obstáculo en sí. Permite diferenciar entre los dos tipos de obstáculos existentes mediante un id (0 para círculos, 1 para cuadrados).

Paquete Quad

- **DatosQuad**: clase utilizada para almacenar los resultados que se van obteniendo en cada intervalo de tiempo.
- **Quad**: Mediante esta clase se representa la configuración del vehículo utilizado en la aplicación, definiendo además, las características del sensor láser que lleva incorporado. Tiene una gran cantidad de métodos entre los que se puede destacar el de ackerman, que es el encargado del movimiento del quad, es decir, de calcular la cinemática utilizada para su desplazamiento a partir del ángulo de giro y la velocidad.

Paquete Search

- **Camino**: esta clase implementa un hilo de ejecución paralelo a la aplicación que se encarga de calcular el camino que debe seguir el quad. Mediante el método

Buscar_camino va calculando cada posición según los datos obtenidos por el método datos_láser. Una vez se calcula el nuevo punto se llama a la función ackerman del quad para realizar el movimiento y a las clases de FigurasJPanel y ImprimirPosicion para que actualicen tanto el dibujo como los datos que se muestran en los paneles.

- **CNodo**: cada posición por la que ha pasado el quad se representa como un Nodo para llevar un seguimiento de este camino. Las posiciones calculadas como posibles futuras posiciones a partir del nodo actual se almacenan en una cola de prioridad. Este orden lo impone la clase ComparaCPosicion.
- **ComparaCPosicion**: esta clase es utilizada para ordenar la cola de prioridad de las posiciones a las que se puede acceder a partir de la posición actual. Este orden viene dado por la distancia a la meta y por el número de haces del láser libres, es decir, que no interceptan con ningún obstáculo.

APÉNDICE B: CÓDIGO FUENTE

Obtención de los Datos del Láser

```
// ----------------------------------------------------------------------------
/*
 * Nombre: datos_laser Descripción: Función que calcula los datos obtenidos
 * por el láser en la posición actual
 * Argumentos: double posicion: Posición
 * del Láser (situado en el Quad) desde la cual se quieren hallar los datos
 * del láser quad quad_aux: quad sobre el cual se encuentra el láser
 * GestionObstaculos goV_obstaculos: Vector donde se encuentran almacenados
 * todos los obstáculos del mapa actual Valor devuelto: el vector de
 * distancias. El tamaño de este vector depende del número de haces del
 * láser.
 */

public double[] datos_laser(Posicion posicion, Quad quad_aux, GestionObstaculos goV_obstaculos) {

        // se calcula el ángulo inicial y final del láser en ese instante
        double angulo_inicial = quad_aux.getInclinacion()+ quad_aux.getAnguloLaser() / 2;
        // inclinación
        double angulo_final = angulo_inicial - quad_aux.getAnguloLaser();

        // vector distancias
        distancias = new double[(int) ((quad_aux.getAnguloLaser()) / quad_aux
                                .getAngulo_entre_haces()) + 1];// lista para

        // se inicializan todos los elementos del vector al alcance máx del láser
        for (int i = 0; i < (int) (((quad_aux.getAnguloLaser()) / quad_aux
                                .getAngulo_entre_haces()) + 1); i++) {
                distancias[i] = quad_aux.getDistancia_max_laser();

                }
        // se recorre todo el vector de obstáculos
        for (int i = 0; i < goV_obstaculos.getV_Figuras().size(); i++) {
           // para cada obstáculo se comprueba si alguno de los haces
          // intercepta con dicho obstáculo
          if (goV_obstaculos.getV_Figuras().get(i).getId() == 0)// es un
          // circulo
          {
                Circulo c1 = (Circulo) goV_obstaculos.getV_Figuras().get(i);
                int j1 = -1;

                for (double i1 = angulo_inicial; i1 >= angulo_final; i1 -= quad_aux
                                .getAngulo_entre_haces()) {
                        j1++;
                        Posicion[] a = new Posicion[2];
                        Posicion p1 = new Posicion();
                        // punto calculado sobre el haz con ángulo i1 del láser
                        p1.setX(quad_aux.getPosicion().getX()
```

```
                        + ((quad_aux.getPosicion().getX() + quad_aux
                                .getAvanza_distancia()) - quad_aux
                                .getPosicion().getX())
                        * Math.cos(Math.toRadians(-i1)));
                p1.setY(quad_aux.getPosicion().getY()
                        + ((quad_aux.getPosicion().getX() - quad_aux
                                .getAvanza_distancia()) - quad_aux
                                .getPosicion().getX())
                        * Math.sin(Math.toRadians(-i1)));

                Recta angu = new Recta(quad_aux.getPosicion(), p1); // recta
                // creada entre el punto posicion del quad
                // y el punto de la recta del haz del láser

                // punto intersección de la recta con el círculo
                a = (c1).interseccion(angu);
                if (a[0].getX() != -1 && a[0].getY() != -1) {
                        Double distMax1 = Math.sqrt((Math.pow(a[0].getX()
                        - quad_aux.getPosicion().getX(), 2))+ (Math.pow(a[0].getY()
                        - quad_aux.getPosicion().getY(), 2)));

                        Double distMax2 = Math.sqrt((Math.pow(a[1].getX()
                        - quad_aux.getPosicion().getX(), 2))+ (Math.pow(a[1].getY()
                        - quad_aux.getPosicion().getY(), 2)));

                        // si la distancia del punto p1 al punto interseccion es
                        // mayor que el punto del quad a la interseccion
                        // distancia=50

                        if (Math.sqrt((Math.pow(a[0].getX() - p1.getX(), 2))
                        + (Math.pow(a[0].getY() - p1.getY(), 2))) > distMax1)
                                distMax1 = quad_aux.getDistancia_max_laser();

                        if (Math.sqrt((Math.pow(a[1].getX() - p1.getX(), 2))
                        + (Math.pow(a[1].getY() - p1.getY(), 2))) > distMax2)
                                distMax2 = quad_aux.getDistancia_max_laser();

                        if (distMax1.isNaN())
                                distMax1 = quad_aux.getDistancia_max_laser();
                        if (distMax2.isNaN())
                                distMax2 = quad_aux.getDistancia_max_laser();
                        if (distMax2 < distMax1)
                                distMax1 = distMax2;
                        if (distancias[j1] > distMax1)
                                distancias[j1] = (double) distMax1;

                        }
                }

        } else {

                Cuadrado c1 = (Cuadrado) goV_obstaculos.getV_Figuras().get(i);
                // ////////////////////CUADRADOS//////////////////////

                // ////////////////////////////////
```

```
                    // ahora analizamos el cuadrado

                    // / C1 -------C2
                    // / | |
                    // / | |
                    // / | |
                    // / C4 -------C3
                    // /

                    int j1 = -1;
                    double i1 = 0;
                    for (i1 = angulo_inicial; i1 >= angulo_final; i1 -= quad_aux
                                    .getAngulo_entre_haces()) {
                            j1++;
                            boolean arriba = false;
                            boolean derecha = false;
                            Posicion p1 = new Posicion();

                            p1.setX(quad_aux.getPosicion().getX()
                            + ((quad_aux.getPosicion().getX() + 1) - quad_aux
                            .getPosicion().getX())
                                            * Math.cos(Math.toRadians(-i1)));
                            p1.setY(quad_aux.getPosicion().getY()
                            + ((quad_aux.getPosicion().getX() - 1) - quad_aux
                                    .getPosicion().getX())
                                            * Math.sin(Math.toRadians(-i1)));

                            if (p1.getY() > quad_aux.getPosicion().getY())
                                    arriba = true;
                            else
                                    arriba = false;

                            if (p1.getX() > quad_aux.getPosicion().getX())
                                    derecha = true;
                            else
                                    derecha = false;

                            Recta angu = new Recta(quad_aux.getPosicion(), p1); // recta
                            // creada entre el punto posicion del quad
                            // y el punto de la recta del haz del láser

for (int n_recta = 0; n_recta < 4; n_recta++) { // buscar  intersecciones con las 4 rectas del
                                                    // cuadrado
                    Recta r1 = ((Cuadrado) c1).getRectas_lados(n_recta);
                    Posicion inters = r1.Busca_interseccion(angu);
                    if (inters.getX() != -1 && (inters.getY() == ((Cuadrado) c1)
                                    .getVertices(n_recta).getY()&& inters.getX() == ((Cuadrado) c1)
                                    .getVertices(n_recta).getX()|| inters.getY() == ((Cuadrado) c1)
                                    .getVertices((n_recta + 1) % 4).getY()&& inters.getX() == ((Cuadrado)
                                    c1).getVertices((n_recta + 1) % 4).getX()|| inters.getY() > ((Cuadrado)
                                    c1).getVertices(n_recta).getY()&& inters.getY() < ((Cuadrado) c1)
                                    .getVertices((n_recta + 1) % 4).getY()|| inters.getY() < ((Cuadrado) c1)
                                    .getVertices(n_recta).getY()&& inters.getY() > ((Cuadrado) c1)
                                    .getVertices((n_recta + 1) % 4).getY()|| inters.getX() > ((Cuadrado)
                                    c1).getVertices(n_recta).getX()&& inters.getX() < ((Cuadrado) c1)
```

```
                                .getVertices((n_recta + 1) % 4).getX() || inters.getX() < ((Cuadrado)
                                 c1).getVertices(n_recta).getX()&& inters.getX() > ((Cuadrado) c1)
                                .getVertices((n_recta + 1) % 4).getX())) {
                                        if (!derecha&& (inters.getX() < quad_aux.getPosicion()
                                           .getX() && (i1 != 90))|| derecha&& inters.getX() >
                                            quad_aux.getPosicion().getX()&& (i1 != 270)|| arriba &&
                                            inters.getY() > quad_aux.getPosicion().getY()&& (i1 !=
                                            180)|| !arriba && inters.getY() < quad_aux.getPosicion()
                                           .getY() && (i1 != 0)) {
                                            // la intersección está en el cuadrado
                                            Double lado_AC = Math.sqrt((Math.pow(inters.getX()
                                              - quad_aux.getPosicion().getX(), 2)) +
                                               (Math.pow(inters.getY()- quad_aux.getPosicion()
                                                  .getY(), 2)));
                                            if (lado_AC.isNaN())
                                                lado_AC = quad_aux.getDistancia_max_laser();
                                            if (lado_AC < distancias[j1])
                                                distancias[j1] = (double) lado_AC;

                                                        }
                                                }
                                        }

                                }
                        }
                }// fin while

                return distancias;
        }

//-------------------------------------------------------------------------------------
```

Movimiento. Cinemática Ackerman.

```
// ----------------------------------------------------------------------------
/*
 * Nombre: calcula_ang_ruedas
 * Descripción: Función que calcula los ángulos de giro y velocidades de las ruedas delanteras
 * Argumentos:
 *     double d_giro: giro realizado por el volante del vehículo
 *     double d_velocidad: velocidad que lleva el quad en ese instante
 * Valor devuelto: no devuelve ningún valor
 */

public void calcula_ang_ruedas(double d_giro,double d_velocidad){

        double radio=largo*(Math.tan(Math.PI/2-((d_giro))));

        double d_angulo_interior = 0;
        double d_angulo_exterior = 0;


        if (d_giro != 0) // Si no se desplaza en línea recta
        {

          if(d_giro<0){
              d_angulo_exterior = Math.atan(largo/(Math.abs(radio)+(ancho/2)))*180/Math.PI;

              d_angulo_interior =  Math.atan(largo/(Math.abs(radio)-(ancho/2)))*180/Math.PI;

              double d_vtexterior=d_velocidad/Math.cos(Math.abs(d_angulo_exterior*Math.PI/180));
              doublé d_vtinterior= d_velocidad/Math.cos(Math.abs(d_angulo_interior*Math.PI/180));

              this.m_vd_velocidadesRuedas[0]=
                  (d_vtexterior*Math.sin(d_angulo_exterior*Math.PI/180) )/ ancho;

               m_vd_velocidadesRuedas[1]= (d_vtinterior*Math.sin(d_angulo_interior*Math.PI/180) )/
                  ancho;

             this.m_vd_angulosRuedas[0]=d_angulo_exterior;
             this.m_vd_angulosRuedas[1]=d_angulo_interior;
             this.m_vd_angulosRuedas[2]=d_giro;

          }
          else{ //giro>0
              d_angulo_exterior = Math.atan(largo/(Math.abs(radio)+(ancho/2)))*180/Math.PI;
                                  d_angulo_interior =  Math.atan(largo/(Math.abs(radio)-
                                  (ancho/2)))*180/Math.PI;

              double d_vtexterior=d_velocidad/Math.cos(Math.abs(d_angulo_exterior*Math.PI/180));
              double d_vtinterior=d_velocidad/Math.cos(Math.abs(d_angulo_interior*Math.PI/180));
              this.m_vd_velocidadesRuedas[1]=
             (d_vtexterior*Math.sin(d_angulo_exterior*Math.PI/180) )/ ancho;
              this.m_vd_velocidadesRuedas[0]=
             (d_vtinterior*Math.sin(d_angulo_interior*Math.PI/180) )/ ancho;
```

```
                this.m_vd_angulosRuedas[1]=d_angulo_exterior;
                this.m_vd_angulosRuedas[0]=d_angulo_interior;
                this.m_vd_angulosRuedas[2]=d_giro;
                    }

            }
            else{ //linea recta
                        this.m_vd_angulosRuedas[0]=0;
                        this.m_vd_angulosRuedas[1]=0;
                        this.m_vd_velocidadesRuedas[1]=this.m_vd_velocidadesRuedas[2];
                        this.m_vd_velocidadesRuedas[0]=this.m_vd_velocidadesRuedas[2];

            }
}
            // -------------------------------------------------------------
            /*
             * Nombre: ackerman
             * Descripción: Función que calcula la cinemática ackerman para obtener la posición siguiente
             * Argumentos:
             *     double d_giro: giro realizado por el volante del vehículo
             *     double d_velocidad: velocidad que lleva el quad en ese instante
             * Valor devuelto: no devuelve ningún valor
             */

            public void ackerman( double d_giro, double d_velocidad){

            //          0--------1
            //          -       -
            //          -       -
            //          2--------3

                        m_d_giroActual=d_giro;
                        //paso radianes
                        d_giro=d_giro*Math.PI/180;

                                //velocidad de la rueda imaginaria delantera
                        double d_vt=d_velocidad/Math.cos(Math.abs(d_giro));
                         //CALCULAR Ángulos Ruedas

                                this.m_d_velAngular=(d_vt*Math.sin(d_giro) )/ ancho;
                                this.m_vd_velocidadesRuedas[2]=m_d_velAngular;
                                 calcula_ang_ruedas(d_giro,d_velocidad);

                                this.setInclinacion(m_d_velAngular+this.getInclinacion());
                                double d_x=this.getPosicion().getX()+
                                  d_velocidad*this.intervalo_tiempo*Math.cos(this.inclinacion*Math.PI/180);
                                double d_y=this.getPosicion().getY()+d_velocidad*this.intervalo_tiempo
                                *Math.sin(this.inclinacion*Math.PI/180);

                                Posicion p=new Posicion(d_x,d_y);
                                p.setAngulo(this.getInclinacion());
                                this.setPosicion(p);
            }
//          ------------------------------------------------------------------------------
```