Diseño del controlador ”fuzzy” de un robot autónomo, a través de Algoritmos Genéticos

Mario Pelegrín Ramos

8 de junio de 2014

Índice

Capítulo 1

Introducción, motivación, propósito y objetivo

1.1 Introducción y motivación

Los controladores difusos se suelen usar en muchos dispositivos y robots, ya que son sistemas que razonan de forma similar al razonamiento humano.

A veces, es difícil diseñar los controladores de un robot autónomo, para que reaccione al entorno y tome las decisiones correctas.

Normalmente cuando se diseña un controlador difuso, un experto indica la conducta que ha de tener el robot. Suele indicarla en forma de normas lógicas, como las usadas en un sistema difuso, pero es difícil que indiquen otras características del sistema, como por ejemplo las funciones de pertenencia. Para obtener estos datos del controlador, se suelen usar métodos de aprendizaje automático, como algoritmos genéticos. Esta técnica de aprendizaje crea un controlador llamado "Genetic Fuzzy Controller".

1.2 Propósito

Usar un algoritmo genético para que encuentre de forma automática, los valores de las funciones de pertenencia, del controlador difuso de un robot, que ha de evitar obstáculos.

1.3 Objetivo

Diseñar en 3D, un robot diferencial equipado con sensores, que le permitan detectar los obstáculos que tiene delante.

Diseñar en 3D, un entorno de simulación adecuado, para que el robot aprenda a evitar obstáculos.

Diseñar, para el robot, un controlador difuso que tenga modificables los parámetros de las funciones de pertenencia.

Diseñar un simulador 3D, que emule el comportamiento del robot dentro del entorno 3D.

Diseñar un algoritmo genético, que encuentre de forma automática los parámetros del controlador difuso, y le dé al robot el comportamiento correcto.

Capítulo 2

Estudio de viabilidad

2.1 Software necesario

El proyecto se puede realizar gracias a los siguientes programas:

- Solidworks: CAD para el diseño de piezas en 3D.
- SimLab: CAD para el diseño 3D, usado para convertir formatos 3D del Solidworks a formatos del Labview.
- Labview: CAD para el diseño de sistemas de control y simuladores tecnológicos.
- Labview Robotics: Paquete de herramientas de Labview para la creación y simulación de robots y sus entornos.
- Labview PID and Fuzzy Logic Toolkit: Paquete de herramientas de Labview para la creación de controladores difusos y PID.
- Microsoft Visual Estudio Express: CAD para el diseño de programas en C++ para Windows.

2.2 Estudio del coste inicial

Coste aproximado del software usado:

Solidworks	2889 €
SimLab	180 €
Labview	4820 €
Labview Robotics	2020 €
Labview PID Fuzzy	995 €
MS Visual Estudio Express	0 €
Total	10904 €

Coste aproximado del hardware usado:

1 PC AMD 4 nuclis 3.2 Ghz, 8GB DDR, HD 500GB, Monitor, etc. 800 €

Coste aproximado para iniciar el proyecto es de 11700€

Capítulo 3

Metodología

Para cumplir los objetivos se realizan varias etapas:

1 Diseño del robot y el entorno de simulación.

2 Obtener el modelo cinemático del robot.

3 Diseñar un sistema difuso base al que se le puedan asignar los valores de las funciones de pertenencia.

4 Diseñar un sistema que:

Genere controladores difusos teniendo como entrada los valores de las funciones de pertenencia.
Se pueda activar por otro programa usando una librería.

5 Diseñar un simulador cinemático del robot que:

Tenga como entrada un controlador difuso.
Tenga como salidas la distancia recorrida y la distancia al destino.
Se pueda activar por otro programa usando una librería.

6 Diseñar las clases del individuo y simulación para:

Crear individuos que contengan en sus cromosomas los parámetros de las funciones de pertenencia.
Crear un controlador difuso con los datos del individuo.
Simular el robot con un controlador difuso.

7 Diseñar el algoritmo genético:

Decidir los métodos de selección, cruce, mutación, evaluación y reinserción.
Implementar las clases que sean necesarias: población, generación, selección, etc.
Rediseñar las clases ya implementadas: individuo, simulación, etc.
Hacer pruebas parciales y totales de las clases.

8 Probar el algoritmo genético y rediseñar sus componentes, para que la población evolucione hacia un individuo, del que se obtiene un controlador, que lleva el robot al destino.

Capítulo 4

Planificación

4.1 Diseño del robot y el entorno de simulación en el Solidworks.

Tarea	Tiempo (h)	Objetivo
Búsqueda y lectura de recursos sobre robots diferenciales	12	Conocer los robots diferenciales existentes.
Estudio de los movimientos de los robots diferenciales	2	Entender cómo se mueven los robots diferenciales.
Estudio de los ejemplos robóticos del Labview.	2	Conocer entornos de simulación robóticos.
Análisis de la forma del robot diferencial	2	Decidir las formas básicas del robot a diseñar.
Búsqueda y lectura de recursos sobre diseño en Solidworks.	8	Conocer como se diseñan formas básicas y ruedas.
Diseño del cuerpo del robot	6	Crear el cuerpo del robot al que añadir ruedas.
Diseño de las ruedas del robot	6	Crear las ruedas del robot.
Escoger la forma básica del entorno de simulación.	2	Tener una idea básica de las características del entorno por donde circulará el robot.
Diseño del entorno de simulación	3	Crear el entorno de simulación.

4.2 Obtener el modelo cinemático del robot.

Tarea	Tiempo (h)	Objetivo
Búsqueda y lectura de recursos sobre cinemática de robots móviles.	20	Aprender cómo se calculan las ecuaciones cinemáticas de un robot móvil.
Calculo de las ecuaciones de cinemática directa de nuestro robot diferencial	12	Obtener las ecuaciones para aplicar una velocidad a cada rueda motriz.

4.3 Diseñar un sistema difuso base.

Tarea	Tiempo (h)	Objetivo
Lectura de recursos sobre controladores difusos.	8	Recordar la estructura y componentes de un controlador difuso
Búsqueda y lectura sobre la creación de controladores difusos en Labview con el asistente.	4	Aprender a crear sistemas difusos en Labview con el asistente de sistemas difusos.
Análisis de las variables y normas del controlador difuso.	8	Obtener las variables, los valores lingüísticos y las normas del controlador base.
Diseño del controlador difuso base.	16	Tener un sistema difuso base para la creación de controladores difusos completos.

4.4 Diseñar un sistema que cree un controlador difuso.

Tarea	Tiempo (h)	Objetivo
Lectura de recursos sobre las funciones del Labview para tratar sistemas difusos.	16	Conocer las funciones de la biblioteca del Labview para tratar sistemas difusos.
Diseño del esquema del sistema para crear el archivo del controlador difuso.	16	Obtener un sistema Labview que tenga de entrada los valores de las funciones de pertenencia y cree el archivo del controlador difuso.
Búsqueda y lectura sobre la creación de librerías a partir de los sistemas del Labview.	8	Conocer los pasos para crear la librería de un sistema Labview.
Creación de la librería dinámica a partir del sistema del Labview.	4	Obtener una librería que cree el archivo del controlador difuso.

4.5 Diseñar un simulador cinemático del robot

Tarea	Tiempo (h)	Objetivo
Búsqueda y lectura de recursos sobre simulaciones cinemáticas en el Labview.	16	Aprender a crear un simulador cinemático de un dispositivo móvil.
Diseño básico de un simulador cinemático del robot.	8	Crear un simulador que desplace el robot usando la velocidad lineal y angular que queremos que tenga.
Diseño completo del simulador cinemático del robot.	8	Modificar el simulador añadiendo el controlador difuso y para que obtenga la distancia recorrida y la distancia al destino.
Creación de la librería dinámica a partir del sistema del Labview.	4	Obtener una librería que tenga como entrada un sistema difuso, simule el robot durante un tiempo máximo y devuelva la distancia recorrida y la distancia al destino.

4.6 Diseñar las clases del individuo y su simulación

Tarea	Tiempo (h)	Objetivo
Análisis de los atributos y métodos básicos de un individuo, como también de sus cromosomas.	12	Implementar unas clases que nos permitan crear individuos validos con cromosomas de valores aleatorios.
Análisis de los atributos y métodos básicos de una simulación.	8	Implementar una clase que permita realizar la simulación de un sistema difuso.

4.7 Diseñar el algoritmo genético

Tarea	Tiempo (h)	Objetivo
Búsqueda y lectura de recursos sobre algoritmos genéticos con valores reales.	20	Conocer los métodos usados en los algoritmos genéticos con valores reales.
Análisis de los métodos de selección, cruce, mutación y reinserción para valores reales.	16	Diseñar de forma básica los métodos del algoritmo genético.
Análisis de las clases necesarias para crear el algoritmo genético	16	Diseñar de forma básica las clases del algoritmo genético.
Programación de las clases para crear el algoritmo genético.	20	Implementar la primera versión del algoritmo genético.
Probar cada una de las clases por separado y en conjunto con el resto.	20	Asegurarse que las clases están bien implementadas y que no producen ningún error inesperado.

4.8 Probar el algoritmo genético y rediseñarlo

Tarea	Tiempo (h)	Objetivo
Probar y analizar los resultados del algoritmo genético.	80	Evaluar los métodos usados en el algoritmo.
Rediseñar el algoritmo genético para mejorar los resultados que se han obtenido de las pruebas.	80	Obtener un algoritmo genético que evolucione la población hacia un individuo que lleve el robot a su destino.

Capítulo 5

Marco de trabajo y conceptos previos

5.1 Sistema difuso

Es un sistema que permite razonar y tomar decisiones, usando una base de reglas lógicas, en las que la premisa y la conclusión se cumplen en un grado de certeza.

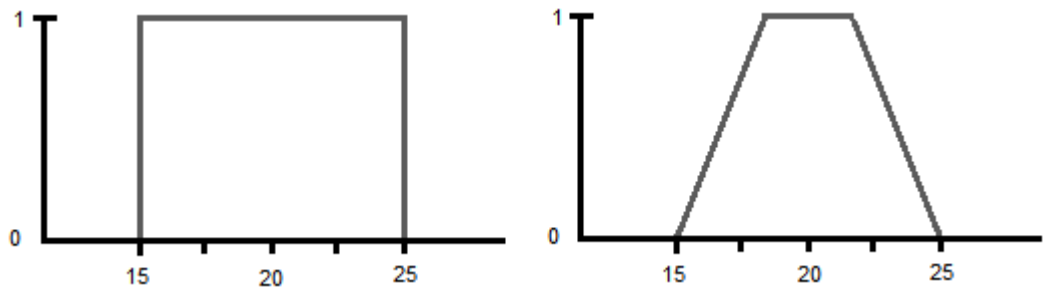
Sus reglas representan el conocimiento necesario para razonar sobre unos hechos. Por ejemplo:

Si la temperatura es templada, entonces la intensidad es media.

La temperatura es templada, se cumple en cierto grado y este grado de certeza depende de cómo se defina el hecho de estar templada.

Las variables de las reglas son variables difusas que toman valores lingüísticos. Por ejemplo la temperatura puede ser fría, templada o caliente.

Representación del valor lingüístico templado entendido como el conjunto de valores entre 15º C y 25º C:



En la izquierda el valor es booleano, en la derecha el valor es difuso.

Por ejemplo:

Si T=26ºC entonces el grado de certeza es Templada(22)=0'7.
Pero el grado de certeza de 30ºC es Templada(30)=0.
Así templada a 22º C se cumple mucho, un 70%, pero templada a 30º C se cumple nada, un 0%.

Cada valor lingüístico está definido por una función de pertenencia, que permite calcular en que porcentaje, un valor numérico de la variable pertenece a ese valor lingüístico.

Estas funciones pueden tener varias formas, en el caso anterior es trapezoidal, pero hay más formas como triangulares, de campana, etc.

Partes de un sistema difuso:

Fuzzyficador: Sistema que asigna a cada valor numérico de una variable uno o más valores lingüísticos. De otra forma, se encarga de interpretar que hechos se cumplen con la entrada de un valor numérico.

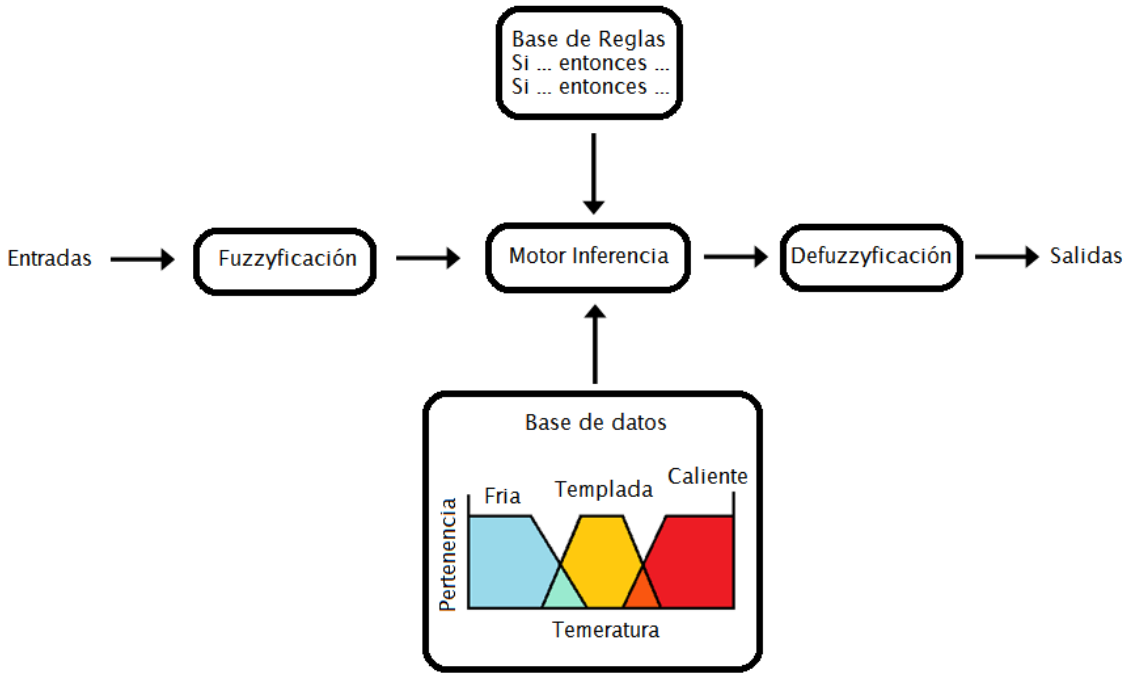
Base de datos: Contiene los datos de los valores lingüísticos de cada variable y sus funciones de pertenencia.

Base de reglas: Contiene todas las reglas que relacionan los hechos con las decisiones a tomar, es decir, los valores lingüísticos de entrada con los valores lingüísticos de salida.

Motor de inferencia: Es un sistema de control que dirige como se aplican las reglas que se cumplen en un determinado momento. Normalmente el sistema comprueba los hechos que se producen, selecciona las reglas con un grado de certeza en la premisa, y decide que reglas aplicar y el orden en que se aplican. Al aplicar una regla, se asigna el valor de certeza de los valores lingüísticos de salida, en función del grado de certeza de los valores lingüísticos de entrada.

Defuzzyficador: Sistema que realiza el proceso de asignar un valor numérico a un valor lingüístico de salida, en función del grado de certeza que tiene el valor lingüístico.

Diagrama de bloques:



5.2 Controlador difuso tipo Mandami

Es un sistema de control compuesto por:

Un conjunto de variables difusas y sus valores lingüísticos.
Un conjunto de reglas lógicas que relacionan las variables de entrada con las de salida, y que permiten controlar las variables de salida, en función de los valores de las variables de entrada.
Un método de fusificar las variables de entrada.
Unos métodos para los operadores lógicos de las reglas.
Un método de defusificar las variables de salida.

5.2.1 Funcionamiento de un controlador difuso:

Toma los valores de las variables de entrada y obtiene los valores lingüísticos de entrada.

Selecciona las reglas que tienen ese valor lingüístico de entrada. Aplica las reglas y calcula el grado de certeza de los valores lingüísticos de salida, usando los métodos de los operadores lógicos.

Agrega los grados de certeza de los valores lingüísticos de salida y defusifica esa unión, obteniendo un valor numérico para las variables de salida.

5.2.2 Ejemplo de controlador difuso para un calentador de agua:

Variable entrada: Temperatura del Agua T(x)
Valores lingüísticos: fría, templada, caliente

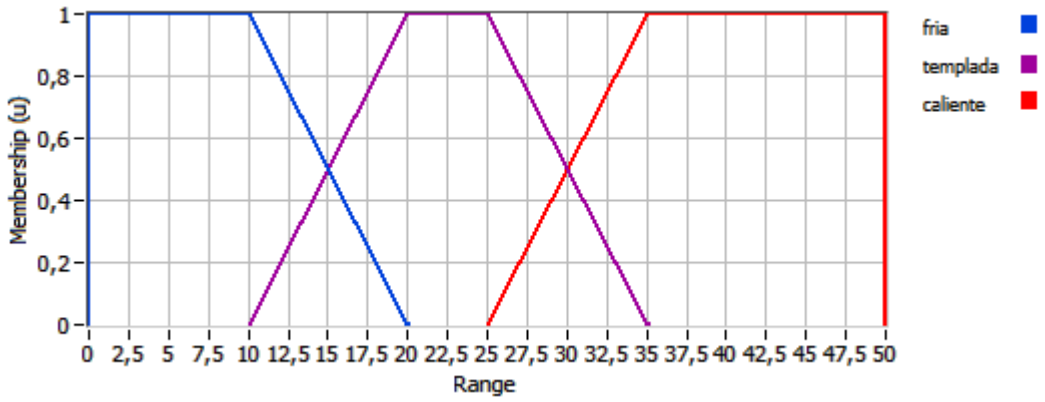
Variable salida: Intensidad del calefactor I(x)
Valores lingüísticos: baja, media, alta

Reglas lógicas:
R1 Si la temperatura es fría entonces la intensidad es alta.
R2 Si la temperatura es templada, la intensidad es media.
R3 Si la temperatura es caliente, la intensidad es baja.

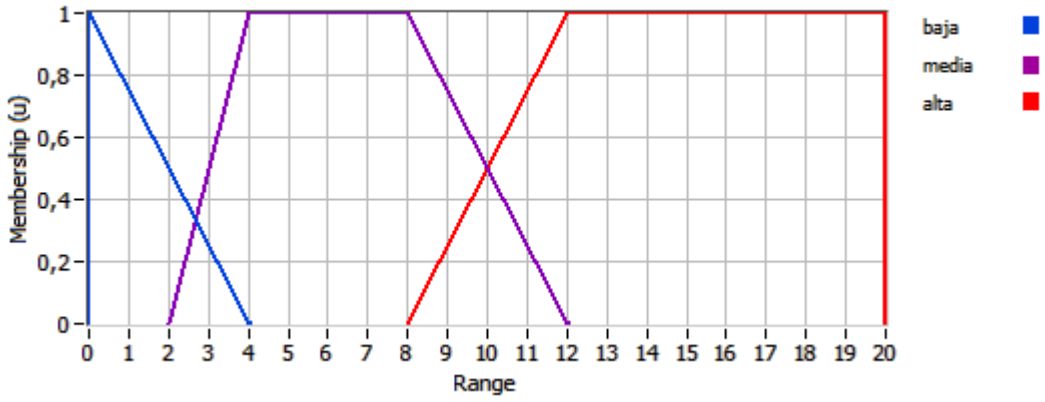
Cada valor lingüístico tiene su función de pertenencia que indica el grado de certeza.

Esquema de las funciones de pertenencia para la variable temperatura:

Temperatura



Intensidad



Ejemplo de control si la Temperatura es 15ºC:

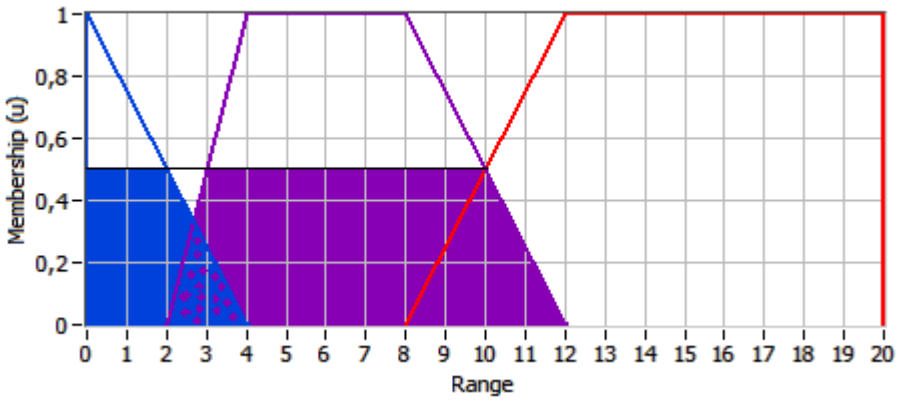
Fusificación:

fría(15)=0.5
templada(15)=0.5
caliente(15)=0
Los valores lingüísticos de entrada son fría y templada.

Razonamiento:

Se cumplen las reglas R1 y R2.

Agregación de las salidas debidas a las reglas R1 y R2:



Defusificación por el centro de gravedad aproximado:

$$\mathrm{COG} = \frac{\sum_{x=a}^{b} \mu_A(x)x}{\sum_{x=a}^{b} \mu_A(x)}$$

Intensidad=(0(0.5) + 1(0.5) + 2(0.5) + 3(0.5) + 4(0.5) + 5(0.5) + 6(0.5) + 7(0.5) + 8(0.5) + 9(0.5) + 10(0.5) + 11(0.25)) / (11(0.5) + 0.25)

Intensidad=30.25/5.75=5.26

5.3 Algoritmo Genético

Son sistemas que imitando la evolución genética de los individuos de una población, se usan para encontrar soluciones óptimas a algunos problemas.

En la evolución intervienen los métodos de selección, cruce, mutación y reinserción, similares a los que se dan en la evolución natural.

Ventajas de usar la evolución para resolver problemas:

En los problemas que tienen un espacio de soluciones muy grande, en donde un algoritmo de búsqueda exhaustivo no es recomendable, la evolución puede encontrar una buena solución en un tiempo aceptable.
Como se aplica a varios individuos a la vez, la búsqueda de una solución se hace de forma paralela.

Inconvenientes de usar la evolución para resolver problemas:

Un inconveniente importante es el parentesco de los individuos, que ocurre si son muy parecidos y hay poca diversidad en los genes.
La causa principal es la adaptación rápida de la población, debido a que los mejores individuos se escogen de forma frecuente para crear hijos.
Esto provoca encontrar soluciones poco óptimas, ya que la ventana de soluciones exploradas es reducida.

Para que la evolución llegue a una solución bastante óptima, se ha de mantener un equilibrio entre: la velocidad de adaptación, la pérdida de diversidad en los genes y la variedad en la selección de los padres. Esto se consigue equilibrando la explotación y exploración genética.

La explotación es la capacidad del algoritmo en mezclar las características genéticas que tienen los individuos.

La exploración es la capacidad del algoritmo para generar nuevas características genéticas en los individuos.

5.3.1 Elementos de un algoritmo genético:

Individuo: cada uno de los elementos que forma la población, que contiene una posible solución en su cromosoma.

Cromosoma: cadena de información, en formato alfanumérico, que representa una solución al problema.

Gen: cada uno de los valores que tiene el cromosoma.

Alelo: cada uno de los valores que puede tomar un gen. Suelen ser valores binarios, caracteres, números enteros o reales.

Método de selección: método que establece como se cogen un grupo de individuos de la población para que sean los padres de la siguiente generación de hijos.

Método de cruce o recombinación: método que establece como se forman los cromosomas de los hijos partiendo de los cromosomas de los padres.

Método de mutación: método que establece como se muta el cromosoma de un individuo. Es necesario para que la evolución tenga un mayor rango de exploración y evite problemas de parentesco.

Método o función de adaptación: método que nos permite asignar un grado de adaptación a cada individuo de la población.

Método de reinserción: método que establece como se introducen los hijos en la población.

5.3.2 Funcionamiento básico de un algoritmo genético:

Se crea una población inicial de individuos, con sus cromosomas inicializados de forma aleatoria.

Se evalúan los individuos de la población inicial con la función de adaptación.

Hasta obtener un individuo deseado o una población con unas características:

1) Se seleccionan los padres de la nueva generación de hijos.
2) Se generan los hijos usando los métodos de cruce y mutación.
3) Se evalúan los hijos con la función de adaptación.
4) Se reinsertan los hijos en la población, sustituyendo a algunos de los padres en caso necesario.

Para evolucionar la población, se suele dejar vivos los mejores individuos, o sustituir toda la población por los mejores hijos.

Hay varias opciones para finalizar el algoritmo, como por ejemplo:

Por contador de generaciones:
Establecer un máximo de generaciones que puede evolucionar.

Por contador de estancamiento:
Establecer un máximo de generaciones donde la población no produce ningún individuo mejor.

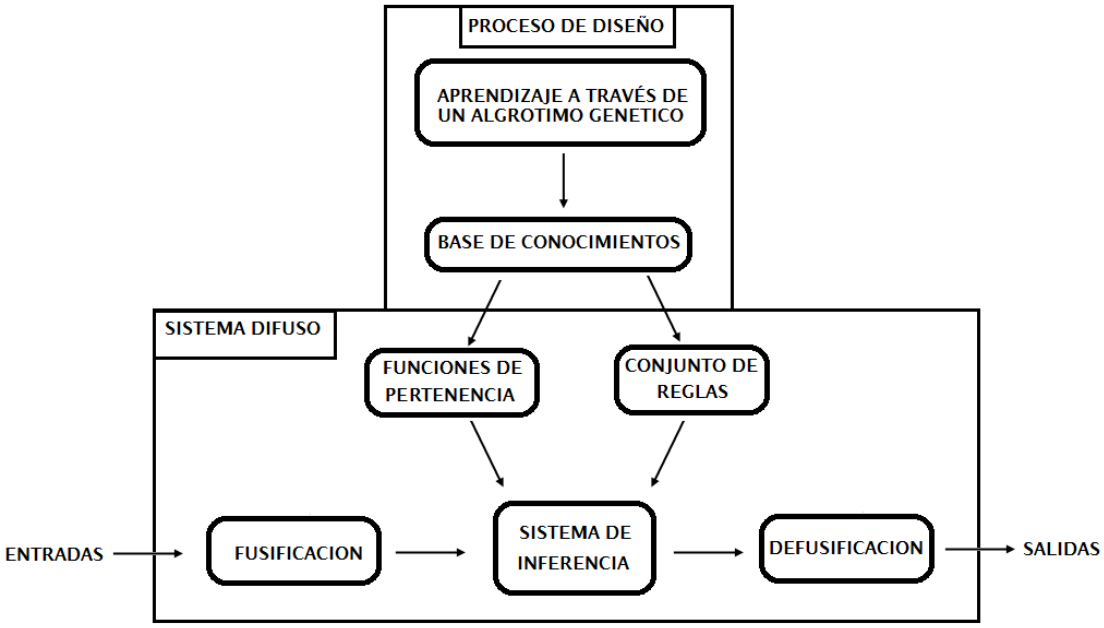
Encontrar uno o varios individuos con un grado de adaptación.

Encontrar una población con una adaptación media deseada.

5.4 Genetic Fuzzy Controller

Es la fusión de un sistema difuso y un algoritmo evolutivo, en donde el algoritmo evolutivo se usa en la fase de diseño del sistema difuso, para aprender u optimizar algún componente del sistema difuso.

En los sistemas difusos basados en reglas, como los del tipo Mandami, el algoritmo genético se usa para aprender u optimizar las normas lógicas, los valores lingüísticos y las funciones de pertenencia.



Cuando el sistema difuso, tiene definidas el conjunto de reglas y sus valores lingüísticos, el algoritmo optimiza las funciones de pertenencia.

Cuando el conjunto de reglas no está definido, se establecen un conjunto de valores lingüísticos, distribuidos de manera uniforme, para que el algoritmo aprenda un conjunto de reglas.

En la optimización de las funciones de pertenencia, los cromosomas de un individuo contienen la forma de la función y como se reparten éstas en el rango de cada variable difusa.

En el aprendizaje de las reglas, los cromosomas de un individuo contienen, o bien una regla o bien el conjunto de reglas. Esta información puede ser la de las matrices o tablas de decisión que se usan para generar el conjunto de reglas.

Cuando el cromosoma de un individuo representa una regla, el algoritmo busca una población, que sea una solución óptima al conjunto de reglas. Es el llamado método Michigan.

Cuando el cromosoma de un individuo representa el conjunto de reglas, se busca un individuo que sea una solución óptima al conjunto de reglas. Es el llamado método Pittsburgh.

5.4.1 Optimización de las funciones de pertenencia de un sistema difuso tipo Mandami:

El sistema contiene un conjunto de reglas con unos valores lingüísticos, para los que se quiere obtener unas funciones de pertenencia. Estas funciones han de conseguir que el sistema proporcione unos valores de salida asociados a unos valores de entrada.
Para obtener las funciones, se codifican las funciones en el cromosoma de un individuo y se crea un algoritmo genético que busque una solución al problema.

Características del algoritmo de optimización:

Se crea el conjunto de datos de entreno, donde un valor de entrada tiene asociada una salida deseada.

Se diseña la codificación de las funciones de pertenencia en el cromosoma de un individuo, teniendo en cuenta las restricciones de la distribución de los valores lingüísticos en el cromosoma.

Se crea un método de simulación del individuo, que crea un sistema difuso con su cromosoma, introduce las entradas del conjunto de entreno, y obtiene las salidas del sistema.

Se crea una función de adaptación, que asigna un mayor grado de adaptación, a aquellos individuos que tienen menor error, respecto a las salidas del conjunto de entreno.

Se diseñan los operadores de cruce y mutación de los individuos, que mezcla y mutan los valores lingüísticos. El operador de cruce, suele mezclar los mismos valores lingüísticos de los cromosomas, ya que pocas veces tiene sentido mezclar valores lingüísticos diferentes.

El resto de métodos del algoritmo genético son los que se suelen usar de forma genérica, como por ejemplo una selección de padres por ruleta, y sustituir peores padres por mejores hijos.

5.4.2 Codificación de las funciones de pertenencia triangulares y trapezoides:

Si las funciones de pertenencia son triangulares o trapezoides se pueden codificar usando reales. Cada función se puede especificar con un grupo de 3 a 4 puntos, que tienen unas restricciones de orden y que han de estar dentro de un rango.

Por ejemplo:

Una función trapezoide, codificada por 4 puntos a,b,c,d.

En donde: **a** es el punto izquierdo, **b** el centro izquierdo, **c** el centro derecho y **d** el punto derecho.

Las restricciones de orden son que a<=b<=c<=d.

Podemos restringir la posición de la función en el rango de la variable X si $a>=X_0$ y $d<=X_n$

Este tipo de codificación de las funciones de pertenencia han sido usadas por autores como Cordón y Herrera (1997) y Herrera, Lozano y Verdegay (1995).

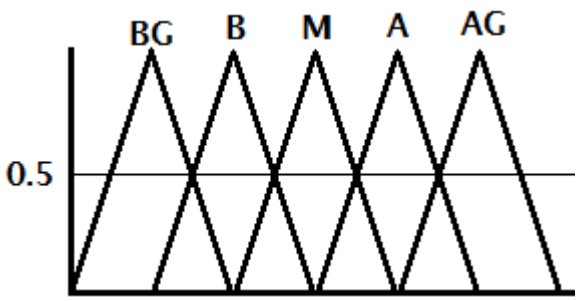
5.4.3 Restricciones de las funciones de pertenencia:

<u>Restricción del grado de pertenencia:</u>

Cuando se quiere que la suma del grado de pertenencia de cualquier valor de la variable sea igual a un valor. Si queremos que sea 1:

$$\sum_{i=1}^{n} \mu_i(x) = 1 \qquad \forall x \in X$$

Esto hace que las funciones se crucen a la mitad de sus rampas.



Este caso dónde la suma de los grados de pertenencia es 1, es llamado partición exhaustiva.

Restricción de orden entre los valores lingüísticos:

Para mantener el orden entre una función y la función del valor lingüístico siguiente, entre los puntos que definen las funciones, ha de existir una determinada distancia.

Restricción de completitud:

Para que cualquier valor x de una variable X, tenga como mínimo una función que le asigne un grado de pertenencia superior a 0:

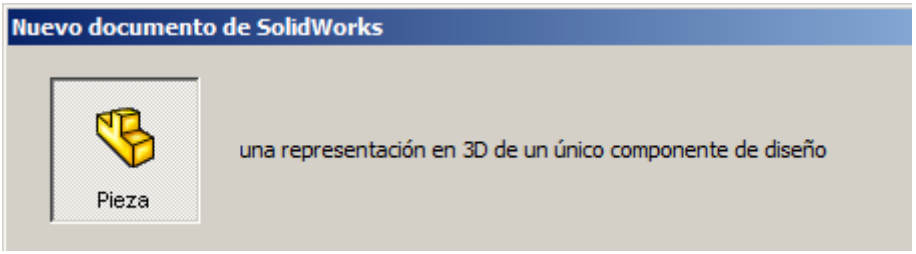
$$\forall x \in X, \exists i \; tal \; que \; \mu_i(x) \geq k \;\; i \;\; k > 0$$

Si se usa esta restricción en las variables de entrada, todos los valores de entrada pertenecen como mínimo a un valor lingüístico.

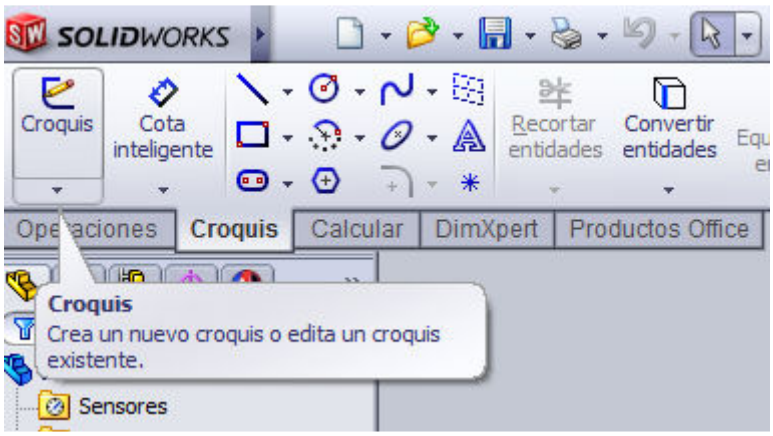
5.5 Entorno Solidworks

El Solidworks es un editor 3D que nos permite diseñar piezas a través de su geometría y crear grosores de materia usando técnicas de extrusión y cuerpos de revolución.

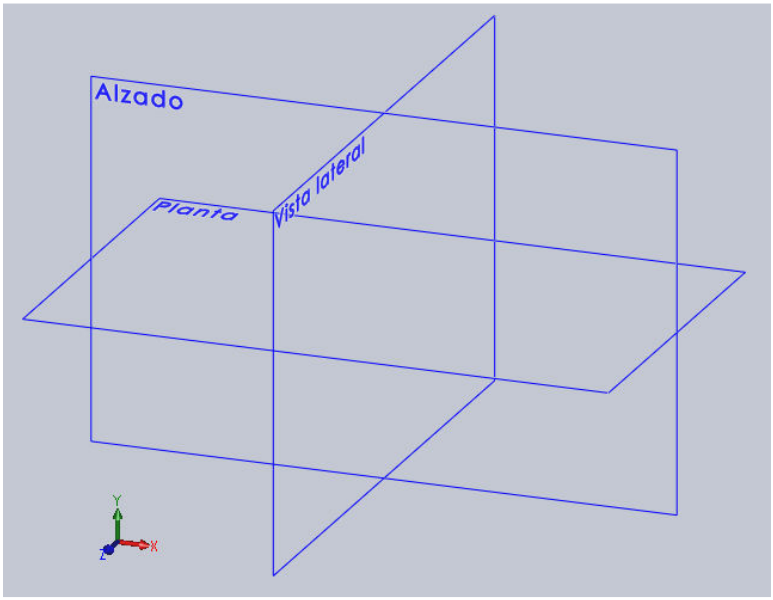
Para crear una pieza vamos a "Nuevo", sale un menú y escogemos Pieza:



Ahora ya se puede crear nuestro croquis geométrico para crear la pieza:

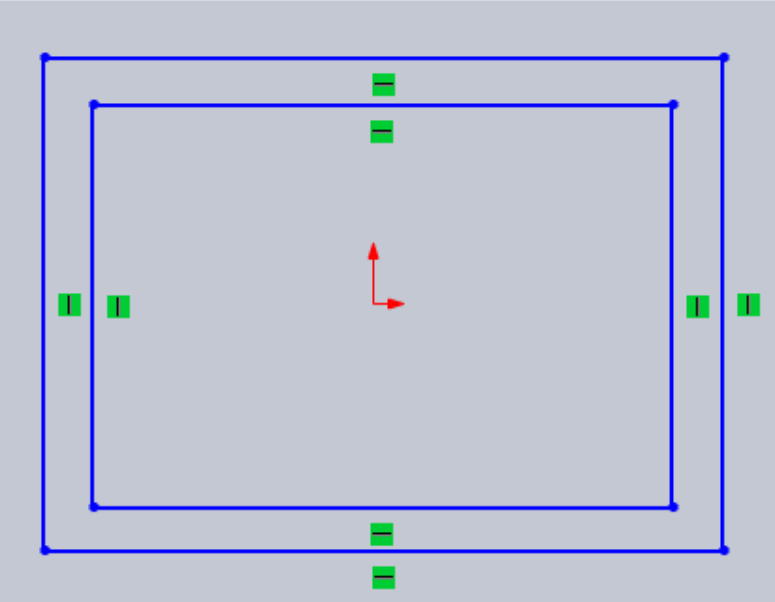


Se selecciona el plano donde se quiere hacer la pieza:

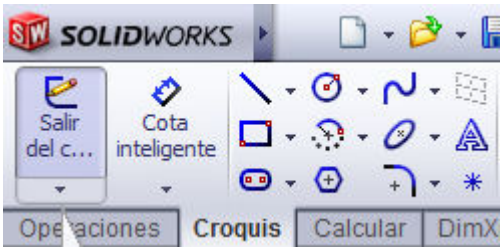


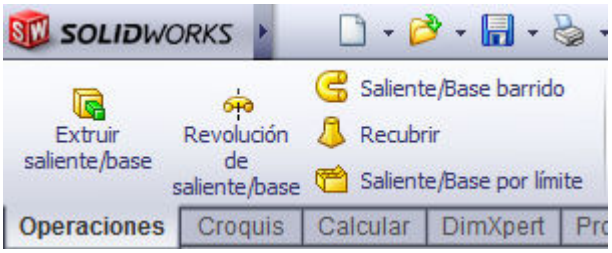
Para crear un croquis geométrico hay varias herramientas, que permiten crear líneas, puntos, rectángulos, circunferencias, elipses, polígonos, etc.

Se crea por ejemplo el croquis para una caja, con dos rectángulos:



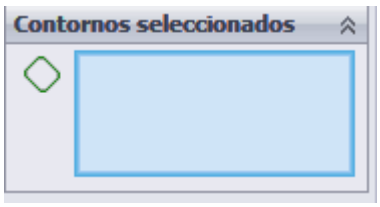
Cuando se ha creado el croquis geométrico, para añadirle grosor y crear la pieza, se sale de él, se va al menú de operaciones y escogemos una.



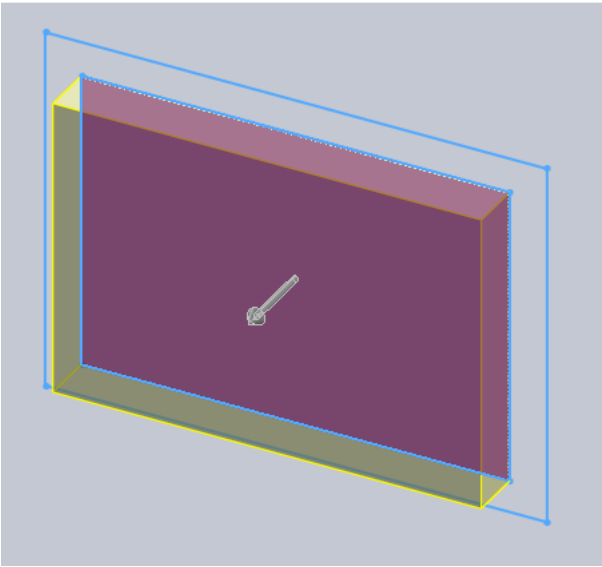


En este caso se escoge extruir un grosor fino en la base y un grosor mayor en las paredes:

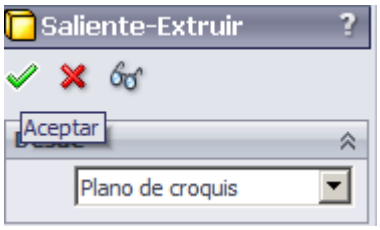
Dentro de la opción de extruir, se va a la zona inferioir izquierda de la pantalla a contornos seleccionados:



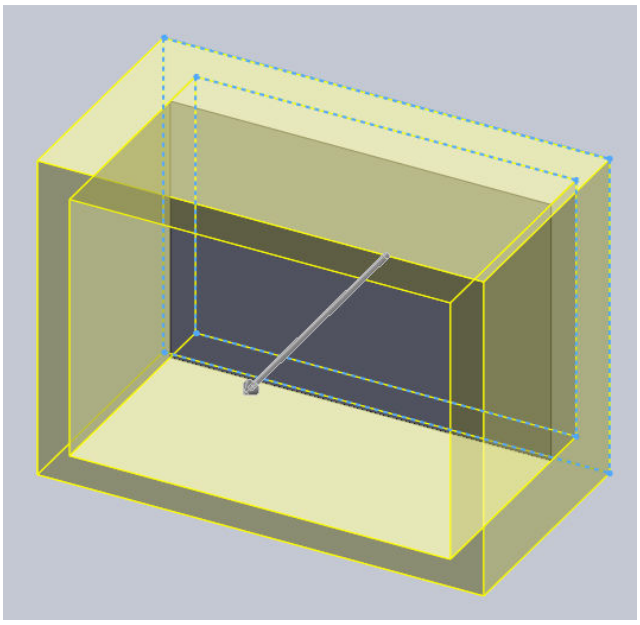
Se clica una vez en el recuadro azul y luego se selecciona la zona del croquis a extruir:



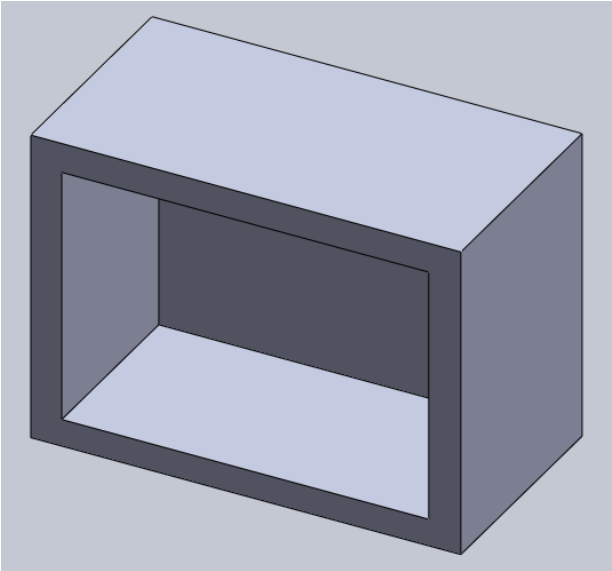
En el menú de la izquierda, se puede cambiar el grosor y la dirección de extrusión. Cuando se obtiene el grosor deseado se da en aceptar.



Ahora se vuelve seleccionar el croquis en el árbol y de la misma forma se crea un grosor en las paredes de la caja:



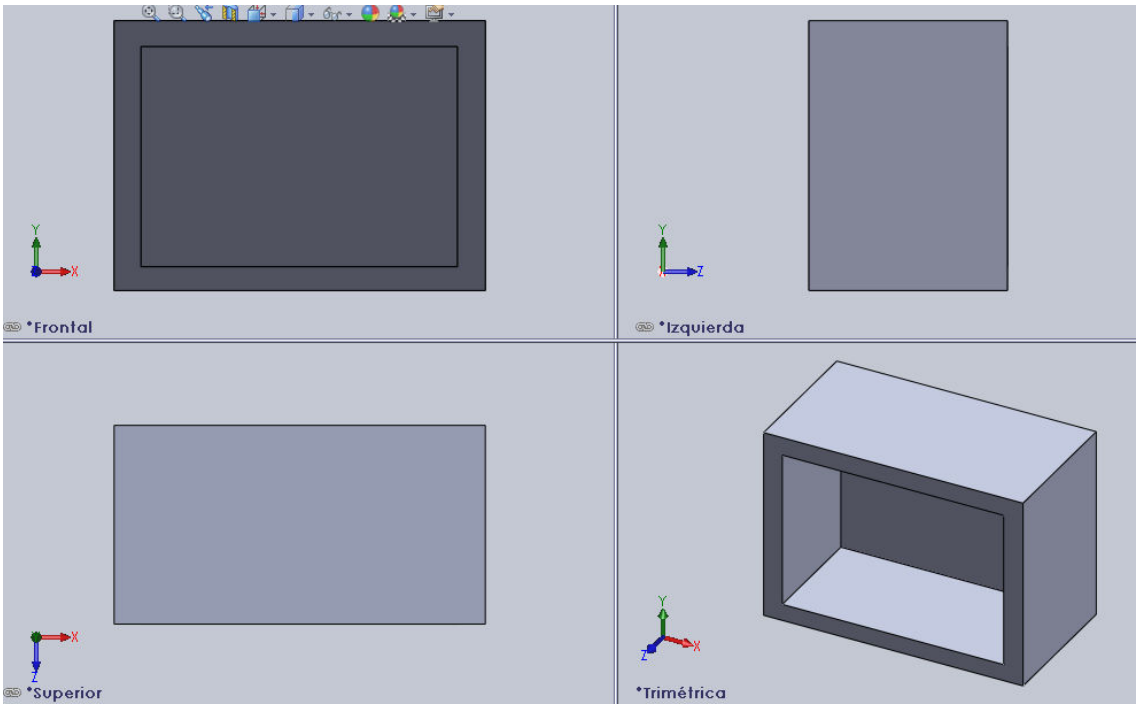
Se acepta la extrusión y ya tenemos la caja que se quería crear.



Una herramienta muy útil es el cambio de vista que está en la parte superior de la pantalla:

Los cubos centrales permiten cambiar de orientación la vista y cambiar entre las vistas de malla y solido.

Incluso sacar varios puntos de vista a la vez:



5.6 Entorno Labview

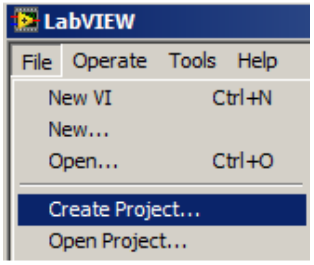
El Labview acrónimo de (Laboratory Virtual Instrumentation Engineering Workbench) es una plataforma que permite el diseño y desarrollo de sistemas de ingeniería.

Emula el comportamiento de dispositivos e instrumentos, que se pueden unir o conectar mediante la programación gráfica, creando diagramas de bloques.

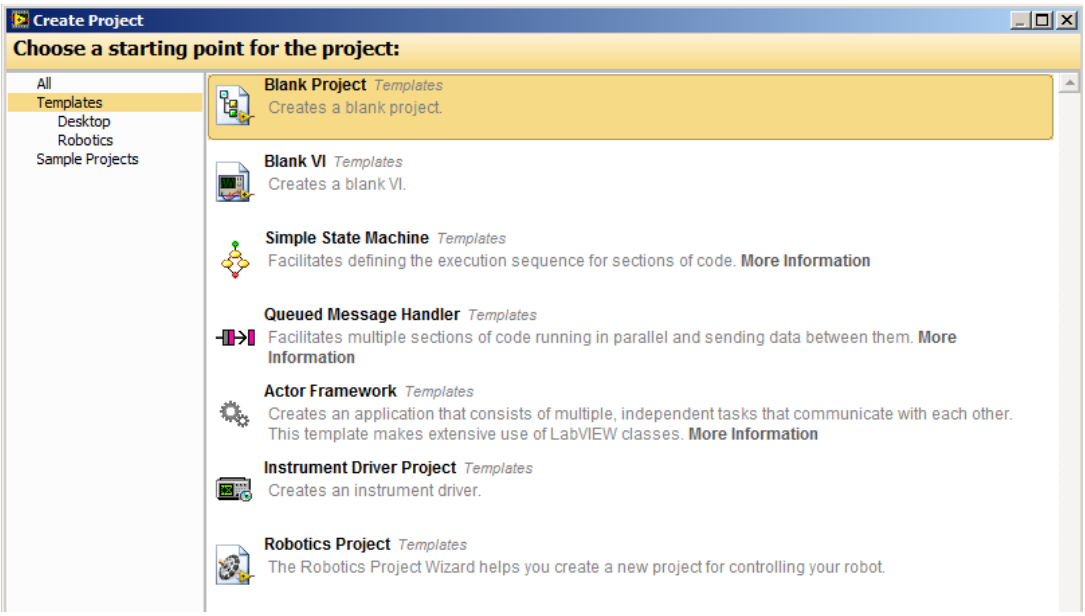
Su principal ventaja es, que acelera y abarata el desarrollo de sistemas, ya que permite a los ingenieros, enfrentarse a los problemas que surgen durante el desarrollo de prototipos, permitiendo realizar pruebas de hardware, software, control, simulaciones, etc.

Otra ventaja importante es, que está constantemente evolucionando y van añadiendo dispositivos e instrumentos de muchos campos: electrónica, mecánica, matemática, informática, robótica, inteligencia artificial, etc.

Para crear un proyecto nuevo, se clica en File->New->Create Project

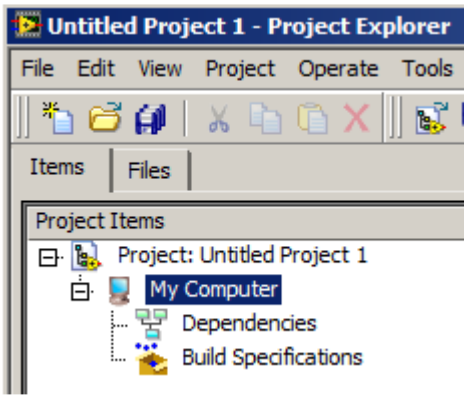


Sale un menú donde se escoge un tipo de proyecto:



Se crea un proyecto en blanco al que añadir sistemas, también llamados Instrumentos Virtuales.

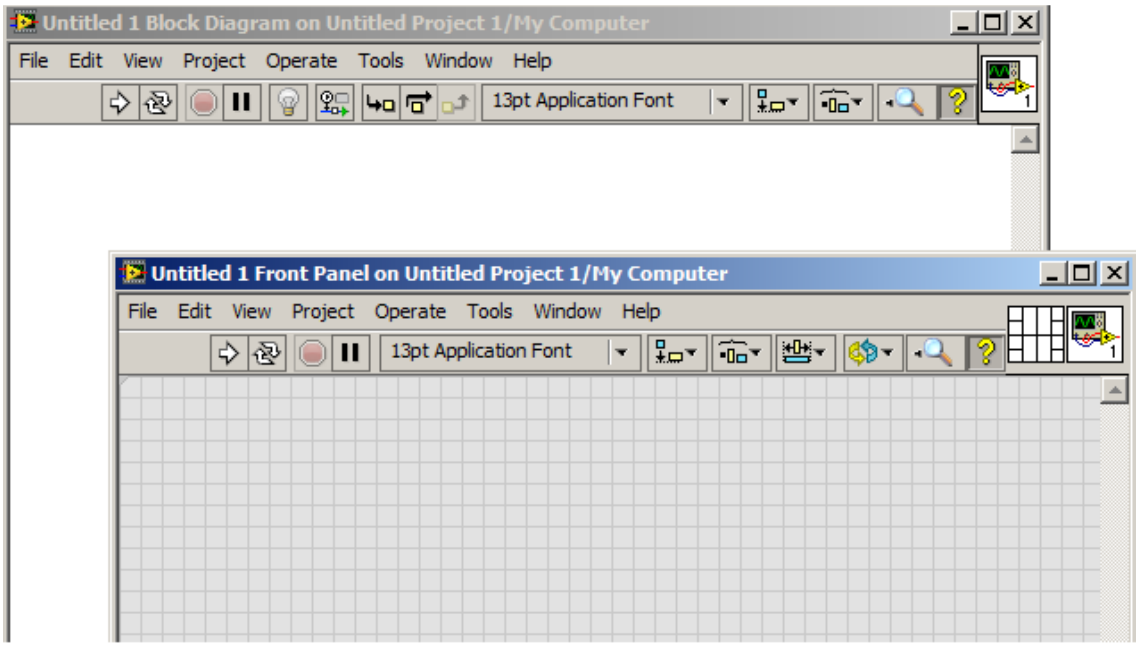
En el menú anterior se escoge Blank Project y él muestra el explorador del proyecto:



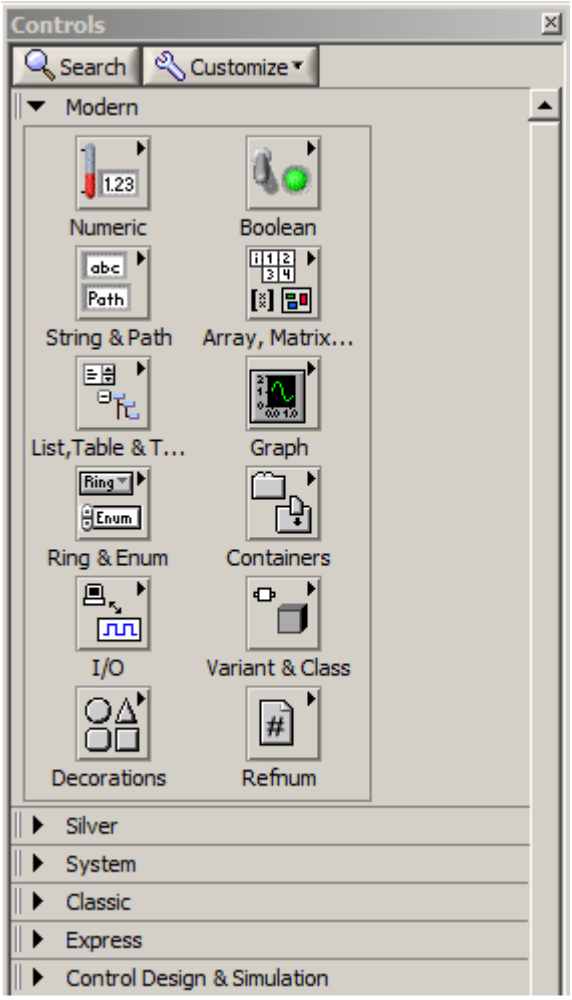
Para añadir un sistema, se clica en File->New VI y muestra dos ventanas:

Una es el panel frontal del sistema con la que se puede interactuar.

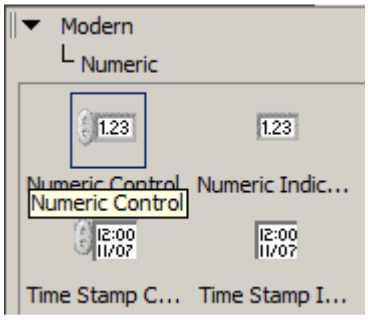
La otra es el diagrama de bloques donde se construye el sistema.



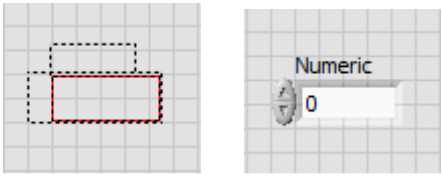
Para añadir botones, indicadores y otros elementos interactivos en el panel frontal, se clica con el botón derecho del ratón y él muestra la librería de controles, ordenada por familias:



Se escoge una, por ejemplo un control numérico:



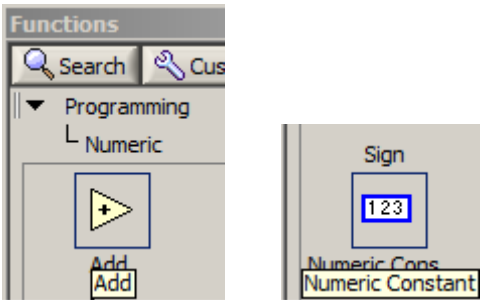
Se clica encima del icono, él muestra una mano, la colocamos en un lugar del panel frontal, y al volver a clicar, él la suelta en esa posición.



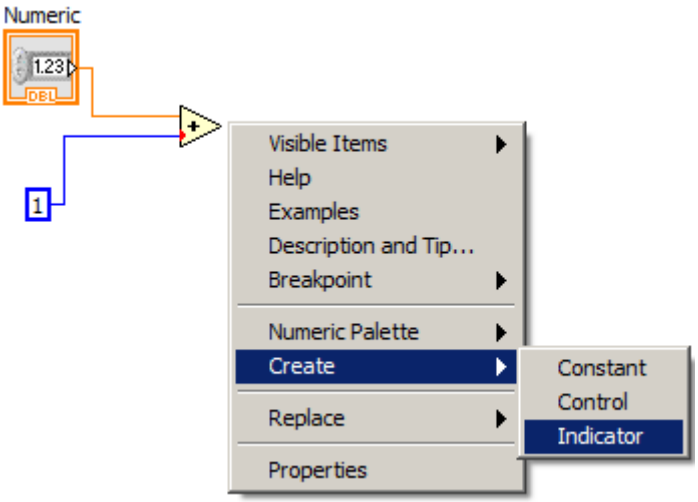
Si se va al diagrama de bloques, se ha añadido un dispositivo con el mismo nombre:

Ahora en el diagrama de bloques, se va a sumar una unidad y a mostrar el resultado en un indicador en el panel frontal.

Se va a la librería y se añade un bloque de suma y una constante:

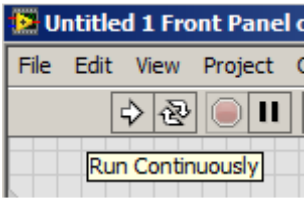


Se enlaza todo en el diagrama y al clicar con el botón derecho sobre la salida de la suma, creamos un indicador:

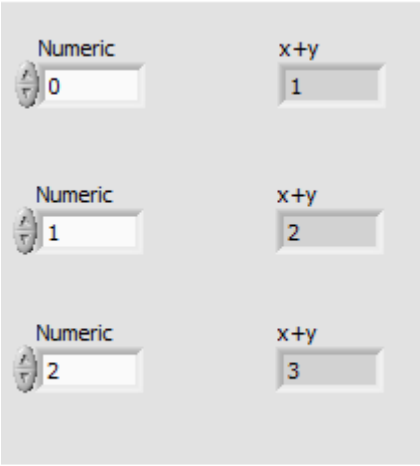


En el panel frontal tenemos lo siguiente:

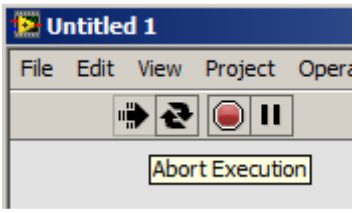
Para ejecutar el sistema, se va a la barra de herramientas y se clica en las flechas Run Continously:



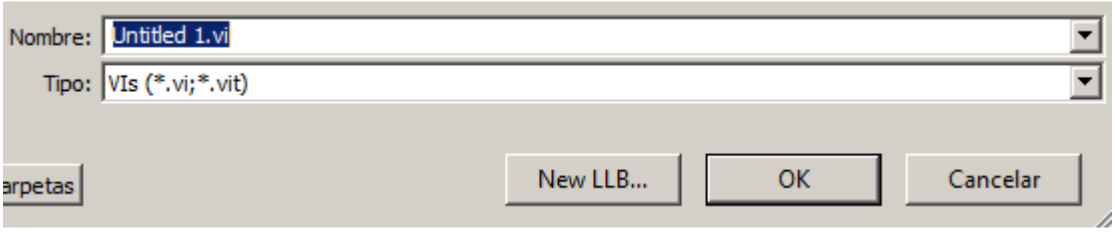
Ahora cada vez que se cambia de valor el control numérico, cambia también el resultado:



Paramos la ejecución del sistema dándole al icono del círculo rojo:



Para guardar nuestro sistema, clicamos en File->Save as, se introduce el nombre del sistema y damos en OK:



Capítulo 6

Requisitos del sistema

El sistema tiene que aprender y optimizar las funciones de pertenencia del controlador del robot, para que llegue al destino evitando los obstáculos que se encuentre por el camino.

Para esto se han de cumplir los siguientes pasos:

- El robot ha de poder detectar los obstáculos que tiene en frente, y obtener su distancia y ángulo, respecto a la orientación del robot.

- El robot ha de estar controlado por un controlador difuso, tipo Mandami, que se cargará en el robot a través de un archivo.

- Se necesita un sistema que genere archivos de sistemas difusos, teniendo de entrada los valores de las funciones de pertenencia.

- El simulador cinemático ha de emular el movimiento del robot en el entorno de simulación. Lo simulará durante un tiempo máximo o hasta que llegue al destino. Al final de la simulación indicará la distancia recorrida por el robot, si ha llegado, si se le ha agotado el tiempo y a qué distancia se ha quedado del destino.

- Los individuos del algoritmo genético, han de tener en sus cromosomas, la información de las funciones de pertenencia del sistema difuso.

- La función de adaptación tiene que realizar la simulación de un individuo, y asignarle un grado de adaptación en función del recorrido del robot.

- El algoritmo ha de evolucionar la población para obtener un individuo, que en la simulación el robot llegue al destino.

- El sistema ha de ser de tipo batch, y tiene que informar de la evolución de la población, para analizar su comportamiento y realizar los reajustes necesarios.

Capítulo 7

Estudios y decisiones

El sistema se ha desarrollado con el siguiente software:

- CAD en 3D de Solidworks:

  Permite crear piezas en 3D de una forma sencilla y rápida, con herramientas de dibujo y extrusión de grosores.
  Permite guardar las piezas en varios formatos compatibles con otros programas.

- Labview y Labview Robotics de National Instruments:

  Permite la creación de sistemas virtuales electrónicos, mecánicos, robóticos, de inteligencia artificial, etc. que ayudan en el desarrollo de proyectos, en el diseño de sistemas y de prototipos.

  Los sistemas creados en Labview se pueden exportar a librerías dinámicas de Windows, que se pueden usar en software programado en C o C++.

- Visual Estudio Express de Microsoft:

  Permite desarrollar software programado en C++ para entorno Windows, de una forma fácil, ya que aporta un conjunto de herramientas útiles para el programador, como por ejemplo: comprobación de errores de código, localización de métodos, localización de llamadas a métodos, compilación pulsando un botón, depurador, inserción de puntos de parada, etc.

Capítulo 8

Análisis y diseño del sistema

En este capítulo se analiza cómo ha de ser el sistema y sus componentes para cumplir con los objetivos.

El sistema ha de ser un ejecutable tipo *batch,* que optimice el sistema de control de un robot móvil usando un algoritmo genético. El sistema tiene que ser capaz de aprender una parte del controlador: las funciones de pertenencia del controlador difuso.

El sistema necesita:

Un robot y un entorno de simulación por donde moverse.
Un modelo cinemático del robot para simular sus movimientos.
Un sistema difuso que controle el robot.
Un simulador que emule el robot dentro del entorno.
Un algoritmo genético que optimice el robot.

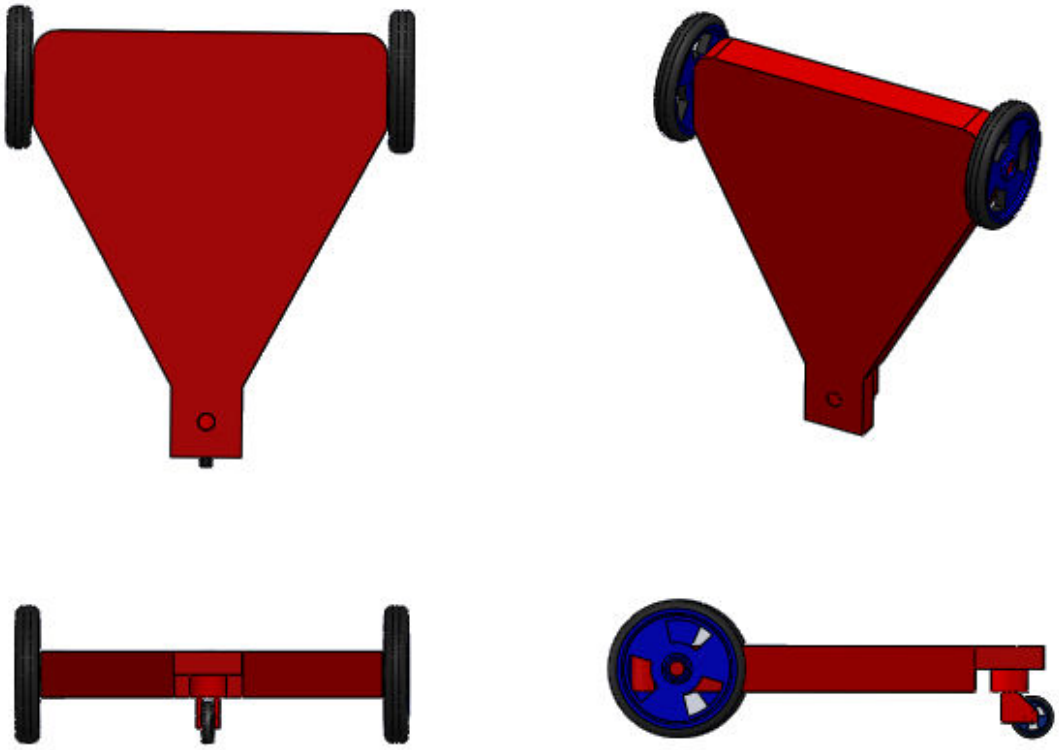
8.1 Análisis y diseño del robot y el entorno de simulación.

El robot ha de ser un robot simple y pequeño que sea estable y pueda ir adelante, atrás, girar a derecha e izquierda.

Se ha diseñado un robot diferencial con ruedas motrices en la parte frontal y una rueda caster en parte posterior.

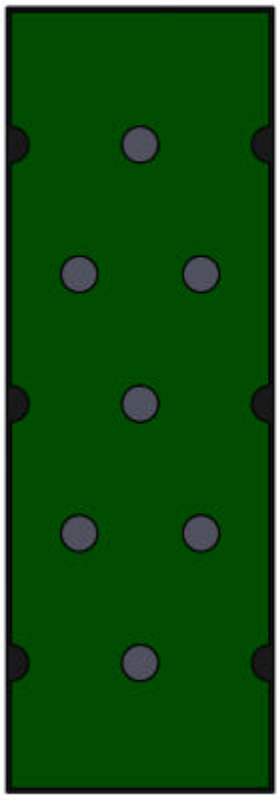
Tiene un cuerpo con forma de triangular, tipo isósceles, que mide 18 cm de largo y 17 cm de ancho.

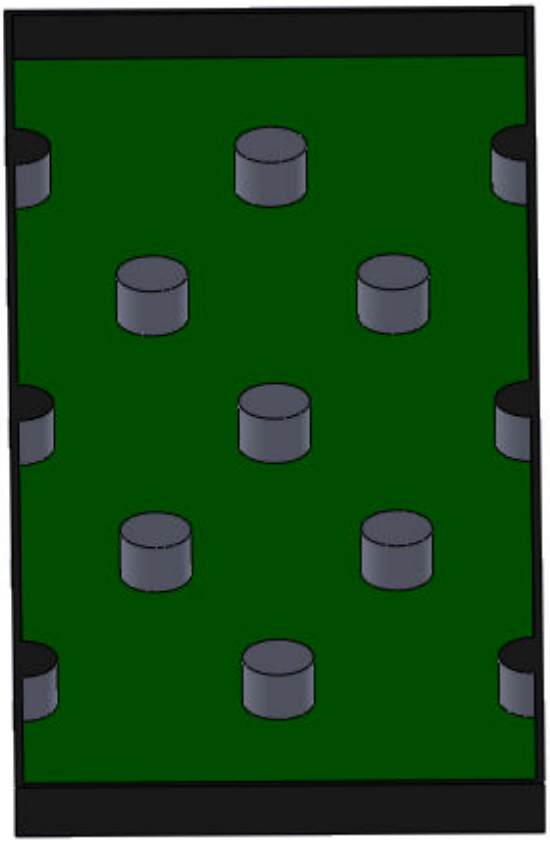
La distancia entre las ruedas motrices es de 16 cm.
La distancia entre el eje de las ruedas motrices y el eje de rotación del grupo caster es de 16 cm.
Las ruedas delanteras tienen una altura de 6 cm y un grosor de 1 cm.



El entorno de simulación ha de ser un espacio cerrado con forma de caja y con obstáculos que el robot pueda salvar a izquierda y derecha.

Como el robot mide unos 20x20 cm, el entorno es de 2 m de ancho por 6 m de largo, así podremos poner obstáculos por todo el escenario, dejando entre ellos una distancia mínima de 40 cm para que el robot los pueda evitar.





Los obstáculos tienen forma de columnas redondas, separadas entre sí por unos 60 cm. La distancia más corta en el escenario es de 40 cm y está entre las columnas y la pared.

Recorrido del robot por el escenario:

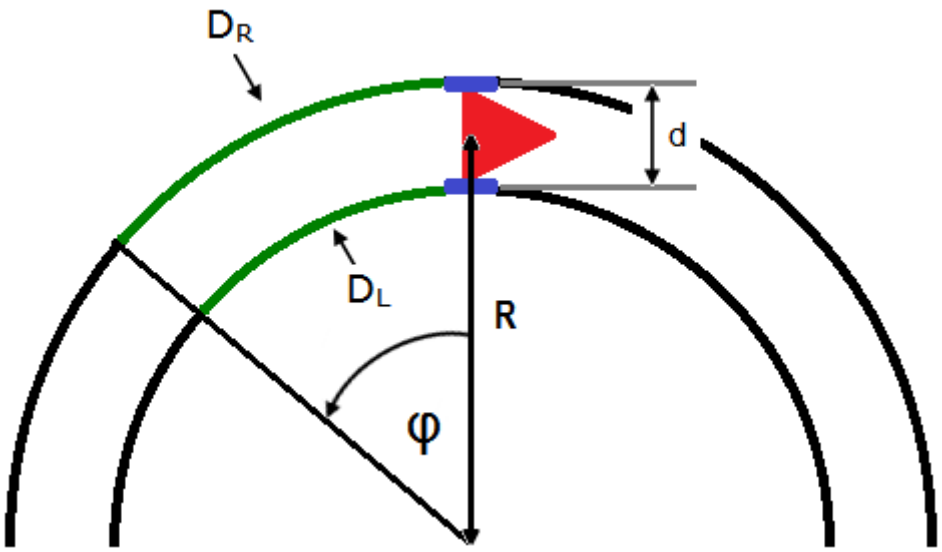
El robot saldrá del centro del lado inferior y ha de llegar al centro del lado superior del escenario. Para hacer este recorrido tendrá que girar varias veces a derecha e izquierda.

Si D es derecha, I es Izquierda unas posibles soluciones son:

D, I, D, I, D, I
I, D, I, D, I, D

8.2 Análisis y diseño del modelo cinemático del robot.

Modelo cinemático de un robot diferencial:



La distancia recorrida por el robot en cada rueda es:

$D_R=\varphi(R + d/2)$ φ es el ángulo recorrido por el robot
$D_L= \varphi(R - d/2)$ φ es el ángulo recorrido por el robot

La distancia recorrida por el centro del robot es:

$Dc= \varphi \cdot R$ φ es el ángulo recorrido por el robot

Si el robot se mueve con una velocidad angular W, entonces en el punto de las ruedas tenemos que la velocidad lineal es:

$V_R = W(R + d/2)$ velocidad rueda derecha
$V_L = W(R - d/2)$ velocidad rueda izquierda

Desde el punto de vista de cada rueda, la velocidad lineal depende de su velocidad de giro y de su perímetro:

$V_R=2\pi r \cdot w_R$ r es el radio, w la velocidad angular
$V_L=2\pi r \cdot w_L$ r es el radio, w la velocidad angular

La velocidad en el centro del robot es:

$V_C = W \cdot R$

El objetivo es calcular la velocidad de giro de cada rueda en función de la velocidad lineal y angular del robot.

Se pone la velocidad angular y lineal del robot en función de las velocidades de cada rueda:

$V_R + V_L = W(R + d/2 + R - d/2) = W \cdot 2R$; $W \cdot R = (V_R + V_L)/2$;
$Vc = (V_R + V_L)/2$;

$V_R - V_L = W(R + d/2 - R + d/2) = W \cdot d$; $W \cdot d = (V_R - V_L)$;
$W = (V_R - V_L)/d$;

Se crea el siguiente sistema de ecuaciones:

$V_R + V_L = 2 \cdot Vc$
$V_R - V_L = W \cdot d$

Si sumamos:
$2V_R = 2Vc + Wd$;
$V_R = Vc + W(d/2)$

Si restamos:
$2V_L = 2Vc - W \cdot d$;
$V_L = Vc - W(d/2)$

La velocidad lineal de cada rueda, en función de Vc y W es:

$V_R = Vc + W(d/2)$
$V_L = Vc - W(d/2)$

Para saber que velocidad de giro necesita cada rueda:

$2\pi r \cdot w_R = Vc + W(d/2)$
$2\pi r \cdot w_L = Vc - W(d/2)$

$$w_R = \frac{Vc + W(d/2)}{2\pi r} \quad [rad/s]$$

$$w_L = \frac{Vc - W(d/2)}{2\pi r} \quad [rad/s]$$

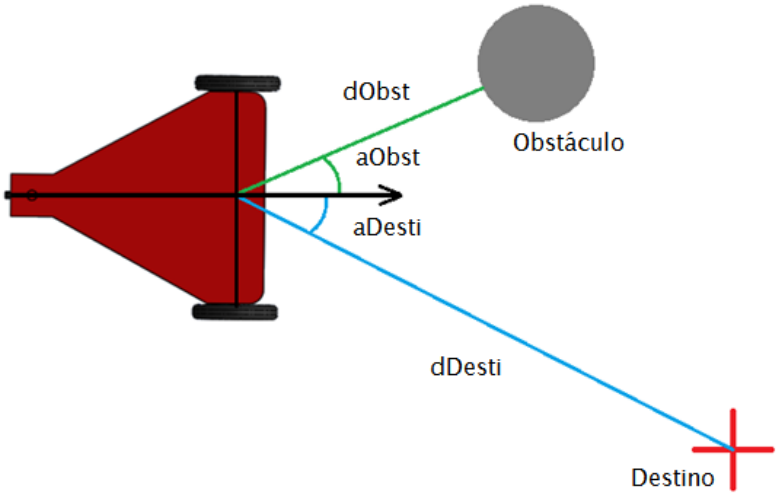
Comprobaciones:

Si el robot va recto y no gira, W=0 y entonces la ruedas llevan la misma velocidad, Vc.

Si el robot gira a la izquierda W>0, la derecha gira más que la izquierda.
Si el robot gira a la derecha W<0, la izquierda gira más que la derecha.

8.3 Análisis y diseño del controlador difuso.

Primero se analizan y diseñan las variables difusas del controlador.
Así que se analiza que necesita para evitar un obstáculo y llegar al destino.
Relaciones de posición entre el robot, un obstáculo y un destino:



El controlador tiene como entradas:

La distancia al destino.
El ángulo relativo al destino, respecto de la orientación del robot.
La distancia al obstáculo.
El ángulo relativo al obstáculo, respecto de la orientación del robot.

El controlador tiene como salidas:

La velocidad lineal del robot.
La velocidad angular del robot.

Rango de las variables:

Distancia al destino de 0 a 10 m
Ángulo al destino de -180º a 180º
Distancia al obstáculo de 0 a 3 m
Ángulo al obstáculo de -90º a 90º
Velocidad lineal del robot de 0 a 1 m/s
Velocidad angular del robot de -180º a 180º grados/s

Valores lingüísticos de las variables:

En la mayoría de controladores que se han consultado, las variables tenían unos pocos valores lingüísticos, indicando cantidades como nada, poco, mucho, etc. Otra característica es que cuando se necesita un valor que está centrado alrededor del 0, se suele usar una función triangular.

Siguiendo esos ejemplos se decide que:

> En las distancias se usen 3 valores trapezoides.
> Para los ángulos y la velocidad angular se usen 5 valores, 1 triangular en el centro y el resto trapezoides.
> Para la velocidad lineal se usen 4 valores, 1 triangular y el resto trapezoides. 3 como en las distancias podrían ser pocos y es clave controlar bien la velocidad del robot.

Valores:

> DistanciaDestino=[zero, pequeña, grande]
> AnguloDestino=[negGran, negativo, zero, positivo, posGran]
> DistanciaObstaculo=[zero, pequeña, grande]
> ÁnguloObstaculo=[negGran, negativo, zero, positivo, posGran]
> VelocidadLineal=[zero, pequeña, mediana, grande]
> VelocidadAngular=[negGran, negativo, zero, positivo, posGran]

Las funciones de pertenencia pueden tener muchas formas, pero para simplificar se usan las más simples, la triangular y la trapezoide.
Si Δ indica triangular y Π indica trapezoide:

Variable	Valor Lingüístico	Forma Función
dDesti	[Z, P, G]	[Π, Π, Π]
aDesti	[NG, N, Z, P, PG]	[Π, Π, Δ , Π, Π]
dObst	[Z, P, G]	[Π, Π, Π]
aObst	[NG, N, Z, P, PG]	[Π, Π, Δ , Π, Π]
VelLin	[Z, P, M, G]	[Δ , Π, Π, Π]
VelAngular	[NG, N, Z, P, PG]	[Π, Π, Δ , Π, Π]

Diseño de las normas del controlador:

Para diseñar el conjunto de normas primero se analiza la reacción básica del robot a los obstáculos y al destino.

El robot ha de reaccionar al obstáculo más cercano, ya que es el primero que tendría que evitar.
Si el obstáculo está más lejos que el destino, ignorará el obstáculo, pero si el obstáculo está más cerca que el destino, intentará evitarlo.

El controlador trata los obstáculos en dos casos:

A) El obstáculo está a una distancia menor o igual a la del destino. En este caso tiene que priorizar su atención en el obstáculo e intentar evitarlo. Se puede dirigir al destino siempre que esté en otra dirección que el obstáculo.

B) El obstáculo está a más distancia que el destino. En este caso tiene que ignorar el obstáculo y dirigirse hacia el destino. Para simplificar este caso, se puede corregir la distancia devuelta por el sistema sensor y situar el obstáculo a la distancia máxima del sensor, así un obstáculo detrás del destino siempre estará muy lejos para el robot.

Para diseñar el conjunto de normas se han realizado estos pasos:

Se crean las variables con los valores lingüísticos.
Se distribuyen de forma intuitiva las funciones dentro del rango de cada variable. La distribución ha de ser bastante uniforme y simétrica.
Al principio se usan un conjunto de reglas bastante general, con pocas normas y giros suaves, con las que seguramente el robot no esquivará la mayoría de los obstáculos.
A base de razonar y probar el robot, se va aumentando la intensidad de los giros y se tratan más casos de forma específica, hasta que consigue esquivar bastantes obstáculos.

A continuación se detallan las reglas finales que se han obtenido para el controlador *fuzzy* del robot.

8.3.1 Reglas para la Velocidad Angular:

Caso 1: El destino está muy lejos

dDesti=Grande & dObst=Grande
(El obstáculo está muy lejos, se va hacia el destino)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	NG	NG	Z	PG	PG
N	NG	NG	Z	PG	PG
Z	NG	NG	Z	PG	PG
P	NG	NG	Z	PG	PG
PG	NG	NG	Z	PG	PG

dDesti=Grande & dObst=Pequeño
(Está cerca del obstáculo, si está en dirección al destino, lo evita)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	Z	Z	Z	PG	PG
N	PG	PG	P	PG	PG
Z	NG	NG	P	PG	PG
P	NG	NG	N	NG	NG
PG	NG	NG	Z	Z	Z

dDesti=Grande & dObst=Zero
(Está muy cerca del obstáculo, si está en dirección al destino, lo evita mucho)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	P	P	Z	PG	PG
N	PG	PG	P	PG	PG
Z	NG	NG	PG	PG	PG
P	NG	NG	N	NG	NG
PG	NG	NG	Z	N	N

Caso 2: El destino está cerca

dDesti=Pequeña & dObst= Grande
(El obstáculo está muy lejos, se va hacia el destino)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	NG	NG	Z	PG	PG
N	NG	NG	Z	PG	PG
Z	NG	NG	Z	PG	PG
P	NG	NG	Z	PG	PG
PG	NG	NG	Z	PG	PG

dDesti= Pequeña & dObst= Pequeña
(Probablemente está más cerca del obstáculo que del destino, lo evita)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	Z	Z	Z	PG	PG
N	PG	PG	P	PG	PG
Z	NG	NG	P	PG	PG
P	NG	NG	N	NG	NG
PG	NG	NG	Z	Z	Z

dDesti= Pequeña & dObst=Zero
(Está muy cerca del obstáculo, lo evita mucho)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	P	P	Z	PG	PG
N	PG	PG	P	PG	PG
Z	NG	NG	PG	PG	PG
P	NG	NG	N	NG	NG
PG	NG	NG	Z	N	N

Caso 3: El destino está muy cerca

dDesti=Zero & dObst=Grande
(Probablemente está más cerca del destino que del obstáculo, va hacia el destino)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	NG	NG	Z	PG	PG
N	NG	NG	Z	PG	PG
Z	NG	NG	Z	PG	PG
P	NG	NG	Z	PG	PG
PG	NG	NG	Z	PG	PG

dDesti=Zero & dObst=Petit
(Probablemente está a la misma distancia del destino y del obstáculo, si está en la misma dirección del destino, lo evita)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	Z	Z	Z	P	PG
N	P	P	P	P	PG
Z	NG	N	P	P	PG
P	NG	N	N	N	N
PG	NG	N	Z	Z	Z

dDesti=Zero & dObst=Zero
(Probablemente está a la misma distancia del destino que del obstáculo, si está en la misma dirección del destino, lo evita mucho)

W=

aObst \ aDesti	NG	N	Z	P	PG
NG	P	P	Z	PG	PG
N	PG	PG	P	PG	PG
Z	NG	NG	PG	PG	PG
P	NG	NG	N	NG	NG
PG	NG	NG	Z	N	N

8.3.2 Reglas para la Velocidad Lineal:

Caso 1: El destino está muy lejos

dDesti=Grande & dObst=Grande
(Probablemente está más cerca del obstáculo que del destino, si están en direcciones separadas la velocidad puede ser más alta)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	M	M	G	G	G
N	P	P	M	G	G
Z	M	M	P	M	M
P	G	G	M	P	P
PG	G	G	G	M	M

dDesti=Grande & dObst=Pequeña
(Está más cerca del obstáculo, así que si está en medio reducimos velocidad)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	P	P	M	G	G
N	Z	Z	P	M	M
Z	M	P	Z	P	M
P	M	M	P	Z	Z
PG	G	G	M	P	P

dDesti=Grande & dObst=Zero
(Está muy cerca del obstáculo, así que reducimos velocidad)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	Z	Z	P	M	M
N	Z	Z	Z	P	P
Z	P	Z	Z	Z	P
P	P	P	Z	Z	Z
PG	M	M	P	Z	Z

Caso 2: El destino está cerca

dDesti=Pequeño & dObst=gran
(Probablemente está más cerca del obstáculo que del destino, si están en direcciones separadas la velocidad puede ser más alta)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	P	P	M	M	M
N	P	P	M	M	M
Z	M	M	P	M	M
P	M	M	M	P	P
PG	M	M	M	P	P

dDesti=Pequeño & dObst=Pequeño
(Probablemente está más cerca del obstáculo que del destino, así que reducimos velocidad)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	P	P	M	M	M
N	Z	Z	P	M	M
Z	M	P	Z	P	M
P	M	M	P	Z	Z
PG	M	M	M	P	P

dDesti=Pequeño & dObst=Zero
(Está muy cerca del obstáculo)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	Z	Z	P	M	M
N	Z	Z	Z	P	P
Z	P	Z	Z	Z	P
P	P	P	Z	Z	Z
PG	M	M	P	Z	Z

Caso 3: El destino está muy cerca

dDesti=Zero & dObst=Gran
(Está más cerca del destino que del obstáculo o tenemos un obstáculo detrás del destino)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	P	P	M	P	P
N	P	P	M	P	P
Z	P	P	P	P	P
P	P	P	M	P	P
PG	P	P	M	P	P

dDesti=Zero & dObst=petit
(Está a la misma distancia del destino que del obstáculo)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	Z	Z	P	P	P
N	Z	Z	P	P	P
Z	P	Z	Z	Z	P
P	P	P	P	Z	Z
PG	P	P	P	Z	Z

dDesti=Zero & dObst=Zero
(Está muy cerca del obstáculo)

V=

aObst \ aDesti	NG	N	Z	P	PG
NG	Z	Z	P	Z	Z
N	Z	Z	Z	Z	Z
Z	Z	Z	Z	Z	Z
P	Z	Z	Z	Z	Z
PG	Z	Z	P	Z	Z

8.4 Análisis y diseño del simulador cinemático

El simulador cinemático tiene que ser capaz de emular el comportamiento del robot dentro del entorno, es decir, hacer que salga de un origen, indicarle un destino y usando el controlador, intente llegar al destino evitando los obstáculos.
La simulación ha de tener un tiempo máximo, ya que hay la posibilidad que el robot no sea capaz de llegar al destino.
Al final de la simulación tendrá que proporcionar: la distancia recorrida por el robot, si ha llegado o no, y la distancia a la que tiene el destino. Como es muy difícil que el robot llegue a una posición exacta, se usa un margen de tolerancia, es decir, se considera que ha llegado, si se queda a una distancia muy corta de la posición de destino.

Pasos que ha de realizar el simulador:

- Iniciar los sistemas sensores del robot.
- Iniciar los sistemas motrices del robot.
- Configurar el controlador con un sistema fuzzy.
- Hasta finalizar la simulación:
  - Aplicar el modelo cinemático y calcular la velocidad de las ruedas.
  - Actualizar las velocidades de las ruedas motrices.
  - Calcular la distancia que ha recorrido respecto a la iteración anterior y sumarla a la distancia recorrida.
  - Hacer un barrido de los obstáculos que tiene alrededor.
  - Calcular las variables de entrada del controlador:
    - Distancia al destino.
    - Ángulo al destino respecto el frontal del robot.
    - Distancia del obstáculo más cercano.
    - Ángulo al obstáculo más cercano.
  - Aplicar las variables de entrada al controlador.
  - Obtener las variables de salida del controlador, y actualizar la velocidad lineal y angular que debería llevar el robot.
  - Comprobar si hemos llegado o se ha acabado el tiempo.

Interfaz del simulador:

- Ha de tener la opción de cargar el sistema de control.
- Ha de tener la opción de mostrar la simulación en un entorno 3D.
- Ha de mostrar si ha llegado, si se ha agotado el tiempo, la distancia recorrida y la distancia al destino.

8.5 Análisis y diseño del Algoritmo Genético

Este proceso se ha hecho en dos fases:

En la primera fase se necesita desarrollar las clases Cromosoma, Individuo y Simulación, por dos motivos:

> Primero para enlazar y comprobar el funcionamiento de las librerías.
> Segundo para evaluar y asignar un grado de adaptación a un individuo en función de su simulación.

En la segunda fase, se analizar y diseña cómo será el algoritmo principal, la población y los métodos evolutivos.

8.5.1 Fase 1: Análisis y diseño de las clases que enlazan con las librerías.

8.5.1.1 La clase Cromosoma

Definición de un cromosoma:

> Es la representación como cadena genética, de las funciones de pertenencia de una variable del sistema difuso.
> Se tendrá un tipo de cromosoma por cada variable del controlador. El cromosoma de tipo A,B,C, etc.
> Un cromosoma será una tabla de números reales, en donde cada número se usa para definir una función de pertenencia. Por ejemplo una función triangular estará representada por tres números: extremo izquierdo, centro y extremo derecho.
> Cada posición de la tabla del cromosoma será un gen, que se podrá cruzar o mutar con otro gen.

Los cromosomas se pueden agrupar, ya que hay que tienen la misma estructura:

> Tipo A,C,F son los cromosomas aDesti, aObst, VelAng.
> Tienen la estructura [Π, Π, Δ, Π, Π]
>
> Tipo B,D son los cromosomas dDesti, dObst.
> Tienen la estructura [Π, Π, Π]
>
> Tipo E es el cromosoma de la variable VelLin.
> Tiene la estructura [Δ, Π, Π, Π]

Para inicializar un cromosoma se han de tener en cuenta unas condiciones de validez:

> Se tiene que mantener la estructura de las funciones de pertenencia de cada variable.
>
> Cada valor lingüístico se ha de cruzar o tener un punto coincidente con el valor lingüístico anterior y posterior, para que todos los valores numéricos de las variables, pertenezcan al menos a un valor lingüístico, a una función de pertenencia.

Si no se cumplen estas condiciones de validez:

> Por una banda, no se estaría codificando bien la estructura de las funciones de pertenencia de cada variable.
>
> Por otra banda, existirían entradas sin *fuzzyficar*, que no producirían salidas en el controlador y dejarían el robot parado sin velocidad.

Por ejemplo, si tenemos un cromosoma con 3 valores lingüísticos con funciones de pertenencia de tipo triangular, un cromosoma válido seria:

Funcion	Fun1			Fun2			Fun3		
Cromosoma (Valor de definición)	0	2	4	3	5	7	6	8	10
Posicion	0	1	2	3	4	5	6	7	8

Para crear un cromosoma de forma aleatoria y válida:

En la primera posición, va el mínimo del rango de la variable y en la última, va el máximo del rango. Por ejemplo [0,12]
Se cuenta cuantos valores del cromosoma han de estar en serie creciente. En el caso anterior serian las posiciones 1,4,7
Se calculan n valores aleatorios dentro del rango de valores del cromosoma. Por ejemplo: (6, 3, 9)
Se ordenan de forma creciente, (3, 6, 9), y se asignan en las posiciones de la serie creciente.

Cromosoma	0	3			6			9	12

Se calculan el resto de posiciones del cromosoma, entre los valores ya asignados, para que el cromosoma sea válido.
Por ejemplo:

C[2]=AleatorioEntre(3,6)=5
C[3]=AleatorioEntre(3,5)=4
C[5]=AleatorioEntre(6,9)=8
C[6]=AleatorioEntre(6,8)=7

El cromosoma final es:

Cromosoma	0	3	5	4	6	8	7	9	12

- Condiciones de validez para los cromosomas de tipo A,C,F:

Estructura [Π, Π, Δ, Π, Π]:

	Π	Π	Δ	Π	Π
Gen	a,b,c,d	e,f,g,h	i, j, k	l, m, n, o	p, q, r, s
Posición	0,1,2,3	4,5,6,7	8,9,10	11,12,13,14	15,16,17,18

Condiciones de validez:

a=min i a<=b i b<=c i c<=d
d<=g i d>=e
b<=e
e<=f i f<=g i g<=h
h<=j i h>=i
f<=i
i<=j i j<=k
k<=n i k>=l
j<=l
l<=m i m<=n i n<=o
o<=r i o>=p
m<=p
p<=q i q<=r i r<=s i s=max

Condiciones de inicialización:

c[0]=min i c[18]=max
min<=c[1]<=c[2]<=c[5]<=c[6]<=c[9]<=c[12]<=c[13]<=c[16]<=c[17]<=max
c[2]<=c[3]<=c[6]
c[4]<=c[3] // cruce 1
c[1]<=c[4]<=c[5]
c[6]<=c[7]<=c[9]
c[8]<=c[7] // cruce 2
c[5]<=c[8]<=c[9]
c[9]<=c[10]<=c[13]
c[11]<=c[10] // cruce 3
c[9]<=c[11]<=c[12]
c[13]<=c[14]<=c[17]
c[15]<=c[14] // cruce 4
c[12]<=c[15]<=c[16]

La función de inicialización de tipo A,C,F realiza estos pasos:

1) Calcula una tabla de 9 aleatorios.
2) Los ordena de menor a mayor.
3) Asigna los 9 valores a las posiciones [1,2,5,6,9,12,13,16,17].
4) El resto de valores son un valor aleatorio entre los que ya están.
   Por ejemplo:
   c[3]=aleatorio(c[2],c[6])
   El caso del valor de una posición de cruce es algo especial.
   Si tenemos dos trapezoides [a,b,c,d] y [e,f,g,h], el valor de cruce de **e** tiene que ser menor que **f** y a la vez menor a **d**, así que se mira cual de los dos, **d** y **f,** es menor.
   Con posiciones sería:
   Como c[4]<=c[3] y c[4]<=c[5]
   Si c[3]<=c[5] entonces c[4]=aleatorioEntre(c[1],c[3])
   si c[5]<c[3] entonces c[4]=aleatorioEntre(c[1],c[5])

- Condiciones de validez para los de tipo B,D:

Estructura [Π, Π, Π]:

	Π	Π	Π
Gen	a,b,c,d	e,f,g,h	i, j, k, l
Posicion	0,1,2,3	4,5,6,7	8,9,10,11

Condiciones de validez:

a=min i a<=b i b<=c i c<=d
d<=g i d>=e
b<=e i e<=f i f<=g i g<=h
h<=k i h>=i
f<=i
i<=j i j<=k i k<=l i l=max

Condiciones de inicialización:

c[0]=min i c[11]=max
min<=c[1]<=c[2]<=c[5]<=c[6]<=c[9]<=c[10]<=max
c[2]<=c[3]<=c[6]
c[4]<=c[3] // cruce 1
c[1]<=c[4]<=c[5]
c[6]<=c[7]<=c[10]
c[8]<=c[7] // cruce 2
c[5]<=c[8]<=c[9]

La función de inicialización de tipo B,D ha de realizar estos pasos:

1) Crea una tabla de 6 aleatorios.
2) Los ordena de menor a mayor.
3) Asigna los 6 valores a las posiciones [1,2,5,6,9,10].
4) El resto de valores son un valor aleatorio entre los que tenemos.
   Por ejemplo:
      c[3]=aleatorio(c[2],c[6])
   En los valores de cruce se mira cual de los dos valores es menor.
   Por ejemplo:
      En c[4]=aleatorioEntre(c[1],c[3]) si c[3]<=c[5]
      En c[4]=aleatorioEntre(c[1],c[5]) si c[3]>c[5]

- Condiciones de validez para los de tipo E:

Estructura ATTT:

	Δ	Π	Π	Π
Gen	a,b,c	d,e, f,g	h,i, j, k	l , m, n, o
Posicion	0,1,2	3,4,5,6	7,8,9,10	11,12,13,14

Condiciones de validez:

a=min i a=b i b<=c
c<=f
b<=d i d<=c
d<=e i e<=f i f<=g
g<=j
e<=h i h<=g
h<=i i i<=j i j<=k
k<=n
i<=l i l<=k
l<=m i m<=n i n<=o i o=max

Condiciones de inicialización:

c[0]=min i c[14]=max
min<=c[1]<=c[4]<=c[5]<=c[8]<=c[9]<=c[12]<=c[13]<=max
c[1]<=c[2]<=c[5]
c[3]<=c[2] // cruce 1
c[1]<=c[3]<=c[4]
c[5]<=c[6]<=c[9]
c[7]<=c[6] // cruce 2
c[4]<=c[7]<=c[8]
c[9]<=c[10]<=c[13]
c[11]<=c[10] // cruce 3
c[8]<=c[11]<=c[12]

La función de inicialización de tipo E ha de realizar estos pasos:

1) Crea una tabla de 7 aleatorios.
2) Los ordena de menor a mayor.
3) Asigna los 7 valores a las posiciones [1,4,5,8,9,12,13].
4) El resto de valores son un valor aleatorio entre los que tenemos.
   Por ejemplo:
      c[2]=aleatorio(c[1],c[5])
   En los valores de cruce se mira cual de los dos valores es menor.
   Por ejemplo:
      En c[3]=aleatorioEntre(c[1],c[2]) si c[2]<=c[4]
      En c[3]=aleatorioEntre(c[1],c[4]) si c[2]>c[4]

La clase cromosoma ha de permitir:

Crear un cromosoma.
Inicializarlo de forma aleatoria.
Verificar que es válido.
Mostrar el cromosoma

Clase Cromosoma

Usa clase TablaReal
Usa clase Aleatorio

Constantes:

LONCA:=19	MINA:=-180	MAXA:=180	[º]
LONCB:=12	MINB:=0	MAXB:=1000	[cm]
LONCC:=19	MINC:=-90	MAXC:=90	[º]
LONCD:=12	MIND:=0	MAXD:=300	[cm]
LONCE:=15	MINE:=0	MAXE:=100	[cm]
LONCF:=19	MINF:=-180	MAXF:=180	[º]

Datos:

Cadena : TablaReal.
Tipo : caracter

Métodos:

CreaCromosoma(t : caracter)
// Crea un cromosoma de tipo t y la cadena a vacía.
Inicializa()
// Llena la cadena del cromosoma según su tipo.
LlenaCromosomaACF() retorna booleano
// Llena un cromosoma de tipo ACF aleatorio y válido.
LlenaCromosomaBD() retorna booleano
// Llena un cromosoma de tipo BD aleatorio y válido.
LlenaCromosomaE() retorna booleano
// Llena un cromosoma de tipo E aleatorio y válido.
EsValido() retorna booleano
// Comprueba según su tipo si el cromosoma es válido
GetCadena() retorna cromosoma
// Devuelve la cadena de reales del cromosoma
GetTipo() retorna caracter
// Devuelve el tipo del cromosoma
Mostrar()
// Muestra el tipo y la cadena del cromosoma

Métodos privados:

EsValidoACF() retorna booleano
// Comprueba si el cromosoma de tipo A, C o F es válido
EsValidoBD() retorna booleano
// Comprueba si el cromosoma de tipo B o D es válido
EsValidoE() retorna booleano
// Comprueba si el cromosoma de tipo E es válido

Fclase

Clase TablaReal

Datos:

tabla : tabla de real
longitud : entero
numElem : entero

Métodos:

CreaTablaReal(nElem:entero)
// Crea una tabla de reales con sitio para nElem
InsertaOrdenado(n:real) retorna booleano
// Si hay espacio, inserta n manteniendo el orden creciente.
Mostrar()
// Muestra la tabla de reales

Fclase

Clase Aleatorio

Metodos:

GetAleatorio(min:real, max:real) retorna real
// Devuelve un real aleatorio entre min y max
ModificaSemilla()
// Cambia la semilla de la función que calcula los aleatorios

Fclase

8.5.1.1 La clase Individuo

Definición de un individuo:

Representa las funciones de pertenencia de todas las variables del controlador difuso.
Tiene un cromosoma por cada variable del controlador. Representa una posible solución al problema: encontrar unas funciones de pertenencia que hagan llegar al robot al destino, evitando los obstáculos.

La clase individuo ha de permitir:

Crear un individuo.
Inicializar sus cromosomas de forma aleatoria.
Obtener sus cromosomas.
Verificar que es válido.
Mostrar el individuo.

Clase Individuo:

Usa clase Cromosoma

Datos:
aDesti: Cromosoma tipo A
dDesdi: Cromosoma tipo B
aObst: Cromosoma tipo C
dObst: Cromosoma tipo D
V: Cromosoma tipo E
W: Cromosoma tipo F

Métodos:
CreaIndividuo()
// crea un individuo con sus cromosomas vacíos.
Inicializa() retorna booleano
// inicializa aleatoriamente sus cromosomas.
EsValido() retorna booleano
// si cumple las condiciones de validez retorna TRUE
Mostrar()
// muestra los cromosomas del individuo
GetaDesti() retorna cromosoma
GetdDesti() retorna cromosoma
GetaObst() retorna cromosoma
GetdObst() retorna cromosoma
GetV() retorna cromosoma
GetW() retorna cromosoma
Fclase

8.5.1.2 La clase Simulación

Definición de una simulación:

Emula el comportamiento del robot, usando como parámetros de entrada, los valores de los cromosomas de un individuo. Obtiene los valores para evaluarlo, es decir la distancia recorrida y la distancia al destino, pero también indica si ha llegado o no.

La clase simulación ha de permitir:

Crear una simulación.
Crear un sistema difuso con la librería.
Realizar la simulación usando la librería del simulador. Obtener la distancia recorrida, la distancia al destino y si ha llegado.

ClaseSimulacion:

Usa librería CreaSistemaFuzzy.dll
Usa librería SimuladorRobot.dll
Usa clase Individuo

Datos:

DistRec: real
DistAlDest: real
Llegado: booleano

Métodos:

CreaSistemaFuzzy(indi:Individuo) retorna booleano
// Con los datos de los cromosomas del individuo, crea el
// archivo del sistema fuzzy para el controlador del robot.

Simula(indi: Individuo) retorna booleano
// Crea el sistema fuzzy, lo simula y llena los datos
// de la simulación.

GetDistRec() retorna real
GetDistDest() retorna real
HaLlegado() retorna booleano

Fclase

8.5.2 Fase 2: Análisis y diseño del algoritmo principal, la población y los métodos evolutivos.

Características del algoritmo principal:

El algoritmo tiene como datos una población de individuos.
Organización de la población:

Se introducen los individuos en una tabla ordenada y decreciente.
Cada elemento de la tabla tiene:
Un Individuo.
Evaluación del Individuo según los datos de la simulación.

Como interesa que se pueda parar en cualquier momento de forma manual, y reanudarlo en otro momento, ha de ser capaz de:

1) Al inicializar la población se ha de seleccionar una opción:
   a. Inicializar una población nueva.
   b. Cargar los datos de una población que ya ha estado evolucionada anteriormente.

2) Guardar la población cada X generaciones.

Para evolucionar la población tiene que usar los métodos de selección, cruce, mutación y reinserción adecuados, teniendo en cuenta que en sus cromosomas se usan valores reales.

El objetivo de la evolución, es encontrar como mínimo un individuo que en la simulación, el robot alcance el destino evitando los obstáculos.

Para decidir cuándo se para el algoritmo se revisa:

Un contador de evolución, con las generaciones que lleva evolucionando.
Un contador de estancamiento, con las generaciones que lleva el mismo individuo siendo el mejor de la población.
Si el mejor individuo de la población realiza una simulación que llega al destino.

Diseño básico del algoritmo principal:

Al iniciar mostrará la opción de cargar la población del archivo, o bien la de crear una población nueva aleatoria. Después pasará a evolucionar la población que se ha inicializado.

Si crea una nueva población:

- Crea los individuos con cromosomas aleatorios.
- Evalúa cada individuo de la población.
- Ordena los individuos según su evaluación.
- Pone el contador de evolución a 0.
- Pone el contador de estancamiento a 0.

Si carga una población desde un archivo:

- Carga el contador de evolución y de estancamiento.
- Lee cuantos individuos tiene la población.
- Carga los individuos y las evaluaciones en la tabla de población.

Mientras no se cumpla una condición de final de evolución:

- Selecciona individuos de la población.
- Genera nuevos individuos válidos por cruce y mutación.
- Evalúa los nuevos individuos de la población.
- Inserta/Sustituye los nuevos individuos en la población.
- Si hay mejora del más adaptado, pone a 0 el contador de estancamiento. Pero si no hay mejora del más adaptado, incrementa el contador de estancamiento.
- Incrementa el contador de evolución.
- Si el número de generaciones es múltiplo de X:
  - Muestra el número de generaciones.
  - Muestra la evaluación de la población.
  - Guarda en el archivo: el contador de evolución y el de estancamiento.
  - Guarda en el archivo los datos de la tabla de la población y los individuos.
- Si el contador de generaciones llega M, ha llegado al final.
- Si el contador de estancamiento llega a N, ha llegado al final.

Después de la evolución, guarda en el archivo de datos:

- El contador de evolución y de estancamiento.
- Los datos de la tabla de la población y los individuos.

Muestra el número de generaciones evolucionadas y estancadas.
Muestra el mejor individuo de la población y su evaluación.
Crea el archivo del sistema fuzzy del mejor individuo.

Métodos para la evolución de la población:

a) Método de evaluación de un individuo:

Creamos un sistema *fuzzy* con los parámetros de sus cromosomas.
Simulamos el robot y se obtiene: la distancia al destino y la recorrida.
La simulación tendrá un tiempo máximo de 30 segundos.

Si el robot acaba la simulación sin llegar al destino, la evaluación depende: de la distancia recorrida y la distancia al destino. Cuanto menor son las distancias, más bueno es el individuo.

Si el robot llega al destino, la evaluación depende de la distancia recorrida. Cuanto menor es la distancia más bueno es el individuo.

Ecuación del método de evaluación:

Como el escenario hace unos 6 m de largo por 2 m de ancho, la distancia máxima al destino es de unos $\sqrt{36+4}=6{,}32\approx 6{,}4$ metros.

Como en la puntuación es más importante que haya llegado al final, la distancia al destino ha de tener más peso que la distancia recorrida.

El peor caso, recorre todo lo que puede y queda en el punto más lejos del destino. Si la velocidad máxima del robot es de 1m/s:

Distancia al destino es 6,4 m
La distancia recorrida 1m/s * 30 sec = 30 metros.

El mejor caso, de forma ideal, es el que va recto y llega al destino:

La distancia al destino es 0.
La distancia recorrida como si no hubiera obstáculos, sería 6 m, es decir la distancia inicial al destino.

Para obtener más puntos por la distancia al destino, se le asigna un peso superior, a los que se pueden obtener por la distancia recorrida.
Si cada metro recorrido es un punto, la distancia recorrida aporta un máximo de 30 puntos. Se puede escoger un peso de 100 para cada metro que se esté del destino.

Así el número de puntos se calcula con la siguiente ecuación:

Puntuación=distAlDestino*100 + distRecorrida.

En el peor caso tenemos una puntuación de:

Puntuacion= 6,4·100 + 30 = 640 + 30 = 670.

En el mejor caso tenemos una puntuación de:

Puntuación=0·100 + 6 = 6

Si la evaluación tiene esos puntos directamente, estaría mejor adaptado el peor caso, así que se ha de hacer una normalización e invertir la puntuación.

Para obtener una evaluación entre 0 y 1000, que asigne más puntos al que está más cerca del destino, se puede usar la siguiente ecuación:

Evaluación =((Pmax-P(i))/(Pmax-Pmin))*1000.
El peor caso obtiene Evaluación=(670-670/664)*1000=0.
El mejor caso obtiene Evaluación=(670-6/664)*1000=1000

b) Método de selección de individuos:

Se usa la selección por ranking lineal y ruleta:

Clasificación por ranking lineal con un SP[1] de 1'2 o 1'3.
Selección por ruleta de X individuos.

Se escoge este método de selección ya que:

La pérdida de diversidad genética es baja.
La adaptación de la población no es rápida.
La varianza en la selección, reparte la adaptación en la población.

Para realizar este método necesitamos una tabla de dos filas:

Una para el valor de ranking.
Otra para el valor de la ruleta.

Aplicación el Ranking lineal:

Los individuos están ordenados en la tabla de la población, de forma decreciente según su evaluación.
La fórmula para calcular el valor del ranking ordenado decreciente es:

$$p(x) = \frac{1}{N}\left(SP + \frac{2(SP-1)(N-x)}{N-1}\right)$$

En donde:

x es la posición del individuo en la tabla de la población.
N es el número de individuos de la tabla de la población.
SP es el coeficiente de la presión de selección.

El valor de ranking del más adaptado, en la posición 1, es (3SP-2)/N.
El valor de ranking del menos adaptado, en la posición N, es (SP/N).

Calculamos el valor del ranking para cada posición y guardamos su valor en la primera fila de la tabla para la selección.

[1] SP es el coeficiente de presión de la selección. Su valor está entre 1 y 2. Cuanto mayor es, más probabilidades de selección tiene un individuo mejor adaptado.

Aplicación del método de la ruleta:

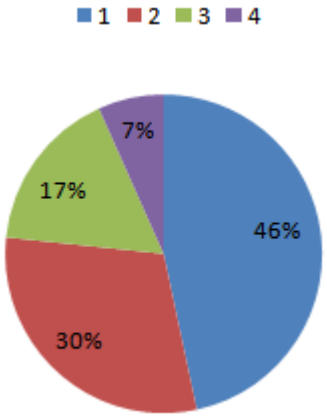
La porción de ruleta que le corresponde a cada individuo, es proporcional a su valor de ranking. Del último al primero, el valor de ruleta de una posición, es su valor de ranking más el valor de ruleta de la posición posterior:

Ruleta[i]=Ruleta[i+1]+Ranking[i];

Por ejemplo si tenemos el ranking:

Ranking	0,5	0,4	0,3	0,2
Ruleta	1,4	0,9	0,5	0,2

Gráfico del reparto de la ruleta:



Aplicación de la selección por ruleta:

Seleccionan los padres que por cruce y mutación generaran nuevos individuos.

Para seleccionar X individuos:

Se calculan X valores aleatorios diferentes entre 0 y el valor de ruleta máximo.
Se ordenan de menor a mayor los valores aleatorios.
Por cada valor aleatorio:

Se recorre la tabla de menor a mayor ranking.
Se selecciona el primero que tenga un valor de ruleta superior o igual al valor aleatorio.
Añadimos el individuo seleccionado a la tabla de padres, manteniendo un orden decreciente por la evaluación.

c) Método de cruce

Cruce entre cromosomas:

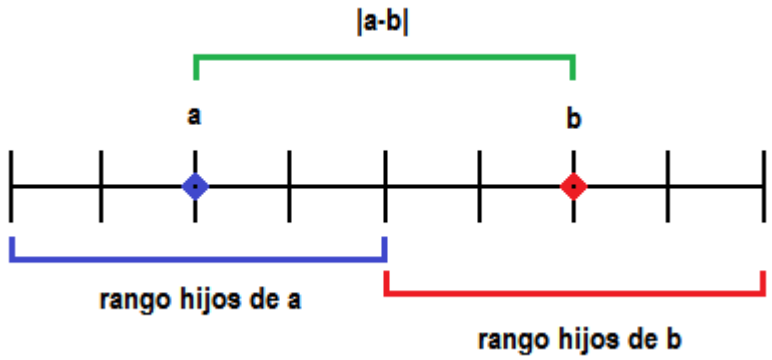
Como los genes son números reales, se usa la recombinación lineal, donde los genes de los hijos se crean sumando una distancia a los genes de los padres.
Se selecciona dos cromosomas padres, del mismo tipo.
Un cromosoma será la base y otro la influencia.
Por cada gen del cromosoma:

Se usa el gen del padre como base del salto.
Se obtiene de forma aleatoria el signo del salto, hacia delante, positivo, o hacia atrás negativo.
Se obtiene el grado o presión del salto, un valor entre 0 y 1, que puede ser fijo o aleatorio.
Se obtiene la distancia que hay entre los genes de los padres.
El salto es igual a signo por grado por distancia entre genes.
El gen del hijo tiene el valor del gen del padre más el valor del salto.

Por ejemplo si tenemos que crear en la posición x el gen del hijo, cruzando los padres, p1 y p2, los pasos son los siguientes:

```
s=aleatorioSigno()      // s=1 ó s=-1
p=gradoSalto()          // 0<=r<=1
d=|p1[x]-p2[x]|         // d>=0
h1[x]=p1[x]+s*p*d
```

Si p1[x]=a y p2[x]=b los hijos de cada padre pueden estar en estas zonas:



Adaptación del grado del salto:

Como inicialmente se usaba un valor fijo y los valores de los padres pueden estar a mucha distancia, los hijos acababan siendo inválidos en la mayoría de cruces.
Para solucionar el problema de invalidez en el cruce, se puede hacer que el grado del salto dependa de la distancia entre los genes de los padres.
La idea básica es, que cuando los padres sean muy diferentes en un gen, se salte poco, y cuando sean más parecidos, se pueda saltar más, por ejemplo hasta un máximo del 50%.

Así el grado del salto es:

El 25% si los padres son muy diferentes.
El 38% si los padres son diferentes.
El 50% si los padres son parecidos.

Luego el grado de parecido o diferencia entre padres se calcula así:

Cada cromosoma tiene un valor mínimo y máximo.
Cada cromosoma tiene un número de valores lingüísticos.
De cada cromosoma podemos calcular la distancia que hace que los valores lingüísticos estén en una distribución uniforme en el cromosoma. Esta será la distancia de cercanía entre dos genes.

Por ejemplo si un cromosoma tiene el rango de -90 a 90 y 5 valores lingüísticos, el valor de cercanía es 180/5=36.

Si el valor de cercanía es k, dos genes g1 y g2 serán:

Muy diferentes si $|g1-g2|>=2k$
Diferentes si $2k<|g1-g2|<=k$
Parecidos si $k<|g1-g2|<=0$

De esta forma salen menos hijos inválidos creados por cruce. Aunque aún hay parejas de cromosomas que son incapaces de crear un hijo válido por cruce.

Adaptación del signo del salto:

Para intentar aumentar la tasa de validez en el cruce de cromosomas, como los valores de los genes han de estar ordenados, si en un gen se salta hacia adelante y en otro hacia atrás, se puede crear un estado de invalidez.

Si el gen del hijo sale inválido, se intenta darle la vuelta al salto.
Es más, en algunos casos tanto el salto hacia delante como hacia atrás son inválidos, así que en ese caso se deja el valor del padre.

Así cada vez que se cruzan los genes se realizan estos pasos:

Se escoge saltar hacia delante o atrás, de forma aleatoria.
Si el primer salto es invalido, se salta en sentido contrario.
Si el segundo salto es inválido, se deja el valor del padre.

Ventajas e inconvenientes sobre la adaptación del grado del salto y del signo del salto:

La ventaja es, que se obtienen tasas de cromosomas inválidos más bajas, ya que hay más probabilidades de acertar un salto correcto.

El inconveniente es que hay que rediseñar el método que comprueba la validez de un cromosoma, ya que se necesita saber si un gen es válido teniendo sólo en cuenta sus genes anteriores, porque aún no se sabe que valores tendrán los genes siguientes, que aún están por cruzar.

Cruce entre individuos:

Para cruzar dos individuos sólo tenemos que usar el método de cruce entre sus cromosomas.

Para que el cruce sea correcto sólo podemos cruzar entre cromosomas del mismo tipo, es decir, A con A, B con B, etc.

Como el cruce entre cromosomas toma de base un padre, de una pareja de padres se crea un hijo con base de cada padre.

Por ejemplo de una pareja de padres p1 y p2:

```
h1=p1.cruza(p2)        // hijo con base del padre1
h2=p2.cruza(p1)        // hijo con base del padre2
```

d) Método de mutación

Mutación en los cromosomas:

El método es similar al de cruce, pero en este caso no se usa la influencia del gen de otro padre, si no que dependiendo del sentido del salto, la distancia está en función del gen anterior o posterior.

Se obtiene una posición x de forma aleatoria, que no sea ni la primera ni la última del cromosoma, ya que han de ser unos valores fijos.
Se obtiene un signo de salto aleatorio.
Se obtiene la distancia teniendo en cuenta que:

Si salta hacia delante, d=|c[x+1]-c[x]|
Si salta hacia atrás, d=|c[x]-c[x-1]|

El grado del salto es un valor fijo pequeño, por ejemplo del 5%, así la mutación es capaz de ir de un valor a otro en saltos pequeños y podrá explorar más variedad de genes.

Para aumentar la tasa de validez, se usa la técnica de girar el salto, así si el primer salto es inválido, se intenta en el sentido contrario, y hay más probabilidades de hacer una mutación válida.

Mutación en el individuo:

Para mutar el individuo, como tiene varios cromosomas, al principio se intentó mutar un gen de cada uno, pero las pruebas fueron pésimas, ya que cuando los individuos están muy adaptados, la mayoría de las mutaciones crea hijos que están menos adaptados. Esto se debe a que es difícil acertar la mutación de 6 genes a la vez para que creen un individuo mejor.

Así que al final se optó por mutar un sólo un gen de todo el individuo, ya que es más factible que mejore el individuo.

Para implementar esta mutación, sólo tenemos que seleccionar uno de los cromosomas del individuo y mutarlo en un gen.

El método realiza estos pasos:

Se obtiene un aleatorio entre 0 y 599.
Si 0<=x<=99 se muta aDesti
Si 100<=x<=199 se muta dDesti
Si 200<=x<=299 se muta dObst
Si 300<=x<=399 se muta aObst
Si 400<=x<=499 se muta V
Si 500<=x<=599 se muta W

e) Creación de una generación

En una selección hay 8 padres, diferentes o alguno repetido, pero están ordenados de forma creciente por su adaptación.
Al crear las parejas que generan hijos, se usa una ventana de 2 padres, que recorre la tabla de la selección como si fuera un círculo cerrado. Crea una pareja con un individuo y el siguiente, siempre que estos sean diferentes.

Por ejemplo si tenemos en la tabla de selección:

a	b	b	c	d	e	e	f

Las parejas serán: (a,b) (b,c) (c,d) (e,f) (f,a)

Se cruzan los mejores, los medios, los peores y el peor con el mejor. Esta secuencia de parejas se sigue hasta que se cierra el círculo o hasta que se consigue crear los hijos máximos de una generación.

Este método tendría que evitar a largo plazo problemas de parentesco y pérdida de variedad en los genes, como puede pasar en una unión elitista, como por ejemplo: (a,b) (a,c) … (a,f), (b,a) … (b,f), etc.

En principio se tendrían que crear el mismo número de hijos por cruce como por mutación, así se mantiene un equilibrio entre la explotación genética que proporciona el cruce con la exploración genética que proporciona la mutación.

La explotación se entiende como la capacidad de evolucionar al mezclar las características genéticas que tienen los individuos.

La exploración se entiende como la capacidad de evolucionar al generar nuevas características genéticas en los individuos.

Como los hijos generados pueden ser inválidos, se crean hijos en función del éxito que tiene cada método. Por ejemplo:

Si el cruce de (a,b) crea 2 hijos, se intentan crear 2 hijos por mutación, uno de a y otro de b.
Si el cruce de (a,b) crea un hijo, se intenta crear un hijo por mutación, o bien de a o bien de b.
Si el cruce de (a,b) no produce hijos, se intenta crear un hijo por mutación de a o de b.

De esta forma la relación entre métodos estará más igualada, aunque en el peor caso, los hijos de una pareja serán todos inválidos.

f) Método de inserción/sustitución de individuos en la población.

El método es una inserción hibrida entre inserción elitista e inserción basada en adaptación.

La inserción elitista, genera menos hijos que individuos en la población, y sustituye los peores individuos de la población por los hijos creados, ignorando si los hijos son mejores o peores a los que sustituyen.

La inserción basada en adaptación, genera más hijos que individuos hay en la población, y sustituye toda la población por los mejores hijos.

El método híbrido elitista-adaptación:

Generar menos hijos que individuos hay en la población.
Reemplazar los peores de la población por mejores hijos.

Ventajas del método:

Como sólo se reemplazan los peores padres, los padres bien adaptados sobreviven más de una generación, manteniendo sus características genéticas en la población.

Los individuos poco adaptados de la población son sustituidos por mejores individuos. Esto hace que la población tenga cada vez más adaptación, o se estanque, pero nunca puede empeorar su adaptación, así que también evita problemas de balanceo en la adaptación de la población, como le puede pasar a los métodos anteriores por separado.

Clases del Algoritmo Principal

La población es un conjunto de individuos que están asociados a una simulación y una evaluación. En vez de tener una tabla de 3 campos, se puede crear una agregación y juntar esos datos en un participante.
Un participante es la agregación de: un individuo, su simulación y su evaluación.

Así una población es una tabla ordenada de participantes. Está ordenada de forma decreciente según la evaluación de cada participante.

Los participantes se han de poder cruzar y mutar entre ellos para evolucionar la población.

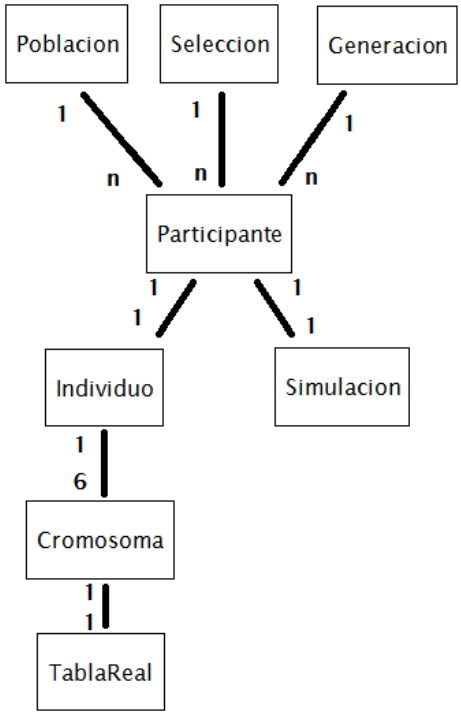
Una selección es la agrupación de x participantes de la población. Como la población es una tabla, la selección puede ser una agrupación de las posiciones de los participantes que forman esa selección.

Una generación es un conjunto de participantes que surgen del cruce y la mutación de los participantes de una selección.

Para evolucionar la población una generación, tenemos que realizar una selección, emparejar los padres seleccionados, generar los hijos e insertarlos en la población.

Después de este análisis el algoritmo genético necesita las clases: Población, Participante, Individuo, Simulación, Cromosoma, Selección, Generación.

Diagrama de clases:



La clase Participante

La clase Participante ha de permitir:

Crear un participante.
Simular el individuo del participante y asignarle una evaluación.
Cruzar el participante con otro y crear un hijo participante.
Mutar el participante y crear un hijo participante.
Saber si un participante es válido.
Obtener el individuo, la simulación, la evaluación.
Mostrar el participante.
Saber si su simulación ha llegado al destino.

Clase Participante:

Usa clase Individuo
Usa clase Simulacion

Datos:

Indi: Individuo
Sim: Simulacion
Eva: real

Metodos:

CreaParticipante()
// Crea un participante con sus datos vacios o a 0.
Inicializa() retorna booleano
// Inicializa el individuo de forma aleatoria y lo evalúa
Evalua() retorna booleano
// Simula el individuo y asigna una evaluación
Cruzar(par:Participante, hij:Participante) retorna booleano
// Cruza el individuo con el de la pareja y crea un hijo
Mutar(hij:Participante) retorna booleano
// Muta el individuo y crea un hijo.
EsValido() retorna booleano
// Comprueba que cumple las condiciones de validez
GetIndividuo(Indi:Individuo) retorna booleano
// Obtiene el individuo del participante
GetSimulacion(Sim:Simulacion) retorna booleano
// Obtiene la simulación del participante
GetEvaluacion() retorna real
// Obtiene la evaluación.
Mostrar()
// Muestra el individuo, la simulación y la evaluación
HaLlegado() retorna booleano
// Indica si en la simulación se ha llegado al destino

Fclase

Para cruzar y mutar un participante se necesita tener los métodos para cruzar y mutar, tanto a nivel de individuo como de cromosoma, así que hay que rediseñar las clases individuo y cromosoma con esos métodos.

La clase Individuo

La clase individuo ha de permitir:

Crear un individuo.
Inicializar sus cromosomas de forma aleatoria.
Obtener sus cromosomas.
Verificar que es válido.
Mostrar el individuo.
Cruzar el individuo con otro para crear un hijo.
Mutar el individuo, para crear un hijo.

Clase Individuo:

Usa clase Cromosoma

Datos:
aDesti: Cromosoma tipo A
dDesdi: Cromosoma tipo B
aObst: Cromosoma tipo C
dObst: Cromosoma tipo D
V: Cromosoma tipo E
W: Cromosoma tipo F

Métodos:
CreaIndividuo()
// crea un individuo con sus cromosomas vacíos.
Inicializa() retorna booleano
// inicializa aleatoriamente sus cromosomas.
EsValido() retorna booleano
// si cumple las condiciones de validez retorna TRUE
Mostrar()
// muestra los cromosomas del individuo
GetaDesti() retorna cromosoma
GetdDesti() retorna cromosoma
GetaObst() retorna cromosoma
GetdObst() retorna cromosoma
GetV() retorna cromosoma
GetW() retorna cromosoma
Cruzar(par:Individuo, hij:Individuo) retorna booleano
// Cruza el individuo con el de la pareja y crea un hijo
Mutar(hij:Individuo) retorna booleano
// Muta el individuo y crea un hijo.
Fclase

Clase Cromosoma

La clase cromosoma ha de permitir:

Crear un cromosoma.
Inicializarlo de forma aleatoria.
Verificar que es válido.
Mostrar el cromosoma
Cruzar el cromosoma con otro del mismo tipo.
Mutar el cromosoma en uno de sus genes.

Clase Cromosoma:

Usa clase TablaReal
Usa clase Aleatorio

Constantes:

LONCA:=19	MINA:=-180	MAXA:=180	[º]
LONCB:=12	MINB:=0	MAXB:=1000	[cm]
LONCC:=19	MINC:=-90	MAXC:=90	[º]
LONCD:=12	MIND:=0	MAXD:=300	[cm]
LONCE:=15	MINE:=0	MAXE:=100	[cm]
LONCF:=19	MINF:=-180	MAXF:=180	[º]

Datos:
Cadena : Tabla de real.
Tipo : caracter

Métodos:

CreaCromosoma(t : caracter)
// Crea un cromosoma de tipo t y la cadena a vacía.
Inicializa()
// Llena la cadena del cromosoma según su tipo.
LlenaCromosomaACF() retorna booleano
// Llena un cromosoma de tipo ACF aleatorio y válido.
LlenaCromosomaBD() retorna booleano
// Llena un cromosoma de tipo BD aleatorio y válido.
LlenaCromosomaE() retorna booleano
// Llena un cromosoma de tipo E aleatorio y válido.
EsValido() retorna booleano
// Comprueba según su tipo si el cromosoma es válido
GetCadena() retorna cromosoma
// Devuelve la cadena de reales del cromosoma
GetTipo() retorna caracter
// Devuelve el tipo del cromosoma

Mostrar()

// Muestra el tipo y la cadena del cromosoma
Cruzar(par: Cromosoma, hij:Cromosoma) retorna booleano
// cruza con la pareja y crea un cromosoma hijo
Mutar(pos:entero, sentSalt:entero) retorna booleano
// Muta aleatoriamente en una posicion, indicando la
// posicion y el sentido del salto
MutarPos(pos:entero,sentSalt:entero) retorna booleano
// Muta en la posición y en el sentido del salto especificados

Métodos privados:

EsValidoACF() retorna booleano
// Comprueba si el cromosoma de tipo A, C o F es válido
EsValidoBD() retorna booleano
// Comprueba si el cromosoma de tipo B o D es válido
EsValidoE() retorna booleano
// Comprueba si el cromosoma de tipo E es válido

Fclase

Se tiene que modificar la clase Aleatorio ya que:

Necesitamos obtener el salto de un cruce o de una mutación de forma aleatoria.

Necesitamos mutar de forma aleatoria un individuo en un cromosoma y un cromosoma en una posición o gen.

Clase Aleatorio

Metodos:

GetAleatorio(min:real, max:real) retorna real
// Devuelve un real aleatorio entre min y max
ModificaSemilla()
// Cambia la semilla de la función que calcula los aleatorio
GetPosAleatoria(posMax:entero) retorna entero
// devuelve un entero entre 0 y posMax
GetSignoAleatorio() retorna entero
// devuelve -1 o 1

Fclase

La clase Población

La clase población ha de permitir:

Crear una población de participantes.
Inicializar los participantes de forma aleatoria.
Agregar los participantes que se leen de un archivo.
Insertar participantes de forma ordenada.
Para evolucionar la población:
    Seleccionar unos padres.
    Crear los hijos de esos padres.
    Insertar los hijos en la población.
Mostrar información de la Población.

Clase Poblacion:

Usa clase Participante
Usa clase Seleccion
Usa clase Generacion

Datos:
    Pueblo: Tabla de Participantes.
    numParticipantes: entero

Métodos:
    CreaPoblacion()
    // crea una población vacía con sitio para X participantes
    InicializaPoblacion(n:entero) retorna booleano
    // inicializa la población con n participantes aleatorios
    AgregaParticipante(par:Participante) retorna booleano
    // agrega un participante en la última posición de la tabla.
    InsertaOrdenado(par:Participante) retorna booleano
    // inserta el participante en la tabla conservando el orden.
    SeleccionaPadres(n:entero, sel:Seleccion) retorna booleano
    // hace una selección de n participantes
    GeneraHijos(sel:Seleccion, gen:Generacion) retorna
    booleano
    // con la seleccion de padres crea hijos por cruce y
    // mutación, y los guarda en la generacion.
    InsertaHijos(gen:Generacion) retorna booleano
    // inserta los hijos de la generación en la población
    GetParticipante(pos:entero, par:Participante) retorna
    booleano
    // si la posicion es correcta, par es el participante
    Mostrar()
    // muestra los participantes de la población.
    MuestraEval()
    // muestra sólo las evaluaciones de los participantes.
Fclase

La clase Seleccion

La clase Seleccion ha de permitir:

Crear una selección de X participantes de una población de N.
Llenar los datos del ranking.
Llenar los datos de la ruleta.
Seleccionar los participantes por el método de la ruleta.
Mostrar información de la generación.
Para no tener una tabla con participantes, en su lugar se usa una tabla de índices de los participantes de la población.

Clase Seleccion:

Usa clase Participante

Datos:
Ranking: Tabla de real
Ruleta: Tabla de real
Selectados: Tabla de entero
numElemPob: entero
numElemSel: entero
numRank: entero
numRul: entero
numSel: entero

Metodos:
CreaSeleccion()
// crea una selección vacía con 0 pariticpantes.
InicializaSelección(n:entero, m:entero)
// crea una selección de m elementos de una población de
// n participantes
LlenaRanking() retorna booleano
// llena el ranking para hacer la selección
LlenaRuleta() retorna booleano
// llena la ruleta para hacer la selección
Selecciona() retorna booleano
// selecciona los participantes por el método de la ruleta
GetNumSel() retorna entero
// devuelve el número de participantes seleccionados
GetIndexPadre(pos:entero) retorna entero
// devuelve el índice del padre que en la selección ocupa la
// posición pos.
Mostar()
// muestra la tabla de la selección
MostrarDatos()
// muestra todos los datos de la selección
Fclase

La clase Generación

La clase Generación ha de permitir:

Crear una generacion de un máximo de X participantes hijos.
Crear hijos por cruce de dos participantes padres y añadirlos.
Crear hijos por mutación de un participante padre y añadirlos.
Mostrar información de la generación.

Clase Generacion:

Datos:

tablaHijos: Tabla de Participante
lonTabla: entero
numHijos: entero

Metodos:

CreaGeneracion()
// crea una generación con la tabla vacía y 0 participantes.
Inicializa(nElem)
// crea la tabla para un máximo de nElem participantes
CreaHijosCruce(p1:Participante, p2:Participante) retorna booleano
// crea máximo 2 hijos por cruce y los añade a la tabla
CreaHijosMutacion(p1:Participante) retorna booleano
// crea máximo 2 hijos por mutación y los añade a la tabla
GetNumHijos() retorna entero
// devuelve el numero de hijos que tiene la generación
GetHijo(pos:entero, hij:Participante) retorna booleano
// si pos es correcta, devuelve el participante de la posición
Mostrar()
// Muestra la tabla, la longitud y el número de hijos
MuestraEval()
// Muestra la adaptación de los hijos de la generacion

Fclase

La clase DocDatos

La clase DocDatos ha de permitir:

1) Crear, abrir y cerrar el archivo de datos.
2) Leer y escribir los datos del estado del algoritmo genético: los contadores de generaciones, de estancamiento, la población, los participantes, las simulaciones, los individuos y los cromosomas.

Clase DocDatos:

```
    Usa clase Poblacion, Participante, Individuo, Simulacion
    Usa clase Cromosoma

    Datos:
        archivo: FlujoDatos

    Metodos:
        creaDocDatos()
        // crea el flujo de datos para leer y escribir en el archivo
        abre() retorna booleano
        cierra() retorna booleano
        guardaContadores( nGen:entero, nEstan:entero) retorna
        booleano
        // guarda los contadores de generaciones y estancamiento
        guardaPoblacion(pob:Poblacion) retorna booleano
        escribeParticipante(par:Participante) retorna booleano
        escribeInidividuo(ind Individuo) retorna booleano
        escribeSimulacion(sim:Simulacion) retorna booleano
        escribeCromosoma(crom:Cromosoma) retorna booleano
        cargaContadores(nGen:entero, nEstan:entero) retorna
        booleano
        // lee los contadores generaciones y estancamiento
        cargaPoblacion(pob:Poblacion) retorna booleano
        leeParticipante(par:Participante) retorna booleano
        leeInidividuo(ind Individuo) retorna booleano
        leeSimulacion(sim:Simulacion) retorna booleano
        leeCromosoma(crom:Cromosoma) retorna booleano
Fclase
```

Fichas de las clases:

Poblacion
+ pueblo : tabla de Participantes + nElem : entero
+ CreaPoblacion() + InicializaPoblacion(n:entero) retorna booleano + InsertaOrdenado(par:Participante) retorna booleano + AgregaParticipante(par:Participante) retorna booleano + SeleccionaPadres(n:entero, sel:Seleccion) retorna booleano + GeneraHijos(sel:Seleccion, gen:Generacion) retorna booleano + InsertaHijos(gen:Generacion) retorna booleano + GetParticipante(pos:entero, par:Participante) retorna booleano + Mostrar() + MuestraEval()

Participante
+ indi : Individuo + sim : Simulacion + eval : real
+ CreaParticipante() + Inicializa() retorna booleano + Evalua() retorna booleano + Cruzar(par:Participante, hij:Participante) retorna booleano + Mutar(hij:Participante) retorna booleano + EsValido() retorna booleano + GetIndividuo(Indi:Individuo) + GetSimulacion(Sim:Simulacion) + GetEvaluacion() retorna real + Mostrar() + HaLlegado() retorna booleano

Individuo
+ aDesti : Cromosoma + dDesti : Cromosoma + aObst : Cromosoma + dObst : Cromosoma + V : Cromosoma + W : Cromosoma
+ CreaIndividuo() + Inicializa() retorna booleano + EsValido() retorna booleano + Mostrar() + GetaDesti() retorna cromosoma + GetdDesti() retorna cromosoma + GetaObst() retorna cromosoma + GetdObst() retorna cromosoma + GetV() retorna cromosoma + GetW() retorna cromosoma + Cruzar(par:Individuo, hij:Individuo) retorna booleano + Mutar(hij:Individuo) retorna booleano

Cromosoma
+ cadena : tabla de real + tipo : caracter + longitud : entero
+ CreaCromosoma(t:caracter) + Inicializa() + LlenaCromosomaACF() retorna booleano + LlenaCromosomaBD() retorna booleano + LlenaCromosomaE() retorna booleano + EsValido() retorna booleano + GetCadena() retorna cromosoma + GetTipo() retorna carácter + Mostrar() + Cruzar(par: Cromosoma, hij:Cromosoma) retorna booleano + Mutar(pos:entero, sentSalt:entero) retorna booleano + MutarPos(pos:entero,sentSalt:entero) retorna booleano + EsValidoACF() retorna booleano + EsValidoBD() retorna booleano + EsValidoE() retorna booleano

Simulacion
+ distAlDest : real + distRec : real + llegado :booleano
+ CreaSistemaFuzzy(indi:Individuo) retorna booleano + Simula(indi: Individuo) retorna booleano

TablaReal
+ tabla : tabla de real + longitud : entero + numElem : entero
+ CreaTablaReal(nElem:entero) + InsertaOrdenado(n:real) retorna booleano + Mostrar()

Aleatorio
+ GetAleatorio(min:real, max:real) retorna real + ModificaSemilla() + GetPosAleatoria(posMax:entero) retorna entero + GetSignoAleatorio() retorna entero

Generacion
+ tablaHijos : tabla Participante + lonTabla : entero + numHijos : entero
+ CreaGeneracion() + Inicializa(nElem) + CreaHijosCruce(p1:Participante, p2:Participante) ret booleano + CreaHijosMutacion(p1:Participante) retorna booleano + GetNumHijos() retorna entero + GetHijo(pos:entero, hij:Participante) retorna booleano + Mostrar() + MuestraEval()

Seleccion
+ ranking : tabla real + ruleta : tabla real + selectados : tabla entero + numElemPob : entero + numElemSel : entero + numRank : entero + numRul : entero + numSel : entero
+ CreaSeleccion() + InicializaSelección(n:entero, m:entero) + LlenaRanking()retorna booleano + LlenaRuleta() retorna booleano + Selecciona() retorna booleano + GetNumSel() retorna entero + GetIndexPadre(pos:entero) retorna entero + Mostar() + MostrarDatos()

DocDatos
+ archivo: FlujoDatos
+ creaDocDatos() + abre() retorna booleano + cierra() retorna booleano + guardaContadores(nGen:entero, nEstan:entero) retorna booleano + guardaPoblacion(pob:Poblacion) retorna booleano + escribeParticipante(par:Participante) retorna booleano + escribeInidividuo(ind Individuo) retorna booleano + escribeSimulacion(sim:Simulacion) retorna booleano + escribeCromosoma(crom:Cromosoma) retorna booleano + cargaContadores(nGen:entero, nEstan:entero) retorna booleano + cargaPoblacion(pob:Poblacion) retorna booleano + leeParticipante(par:Participante) retorna booleano + leeInidividuo(ind Individuo) retorna booleano + leeSimulacion(sim:Simulacion) retorna booleano + leeCromosoma(crom:Cromosoma) retorna booleano

Capítulo 9

Implementación y pruebas

En este capítulo se explica cómo se han desarrollado el sistema y sus componentes, siguiendo los pasos del proceso del análisis.

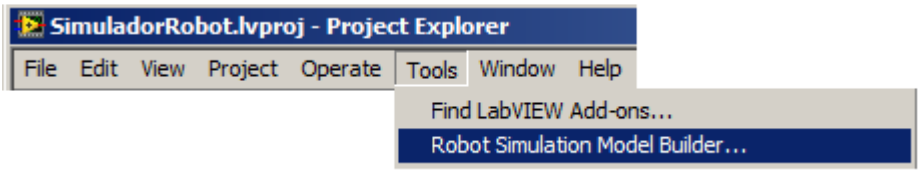
Para el sistema se han creado y desarrollado:

Un robot para Labview con las piezas diseñadas en Solidworks.
Un entorno para Labview previamente diseñado en Solidworks.
Un sistema VI de Labview que permite crear controladores fuzzy.
Un simulador en Labview, que contiene el robot, el entorno, el modelo cinemático y el controlador fuzzy.
Un Algoritmo Genético que aprende y optimiza una parte del controlador del robot.

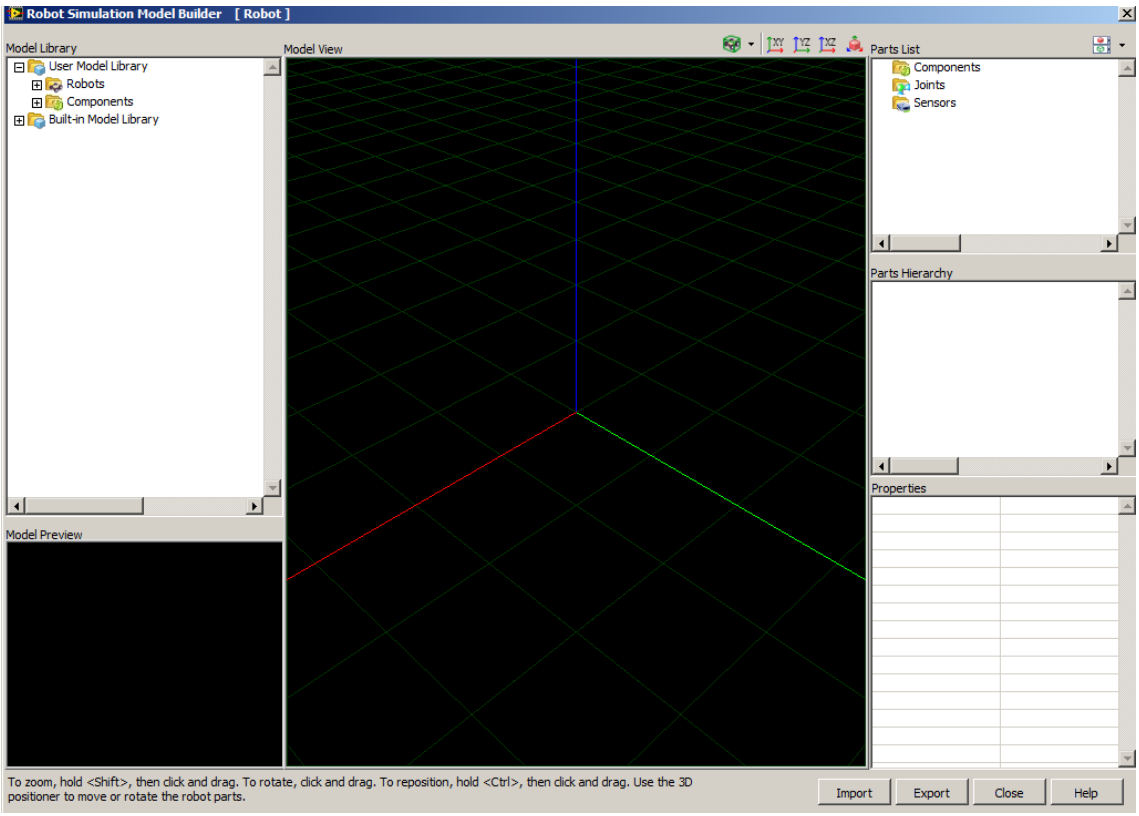
9.1 Creación del modelo del robot en el Labview

Primero se crean las piezas del robot en el Solidworks, se les da la forma geométrica necesaria, se extruyen para darles grosor, y se guardan cada una en archivo 3D. Luego se insertan las piezas en el Labview.

Se abre el Labview y se activa el creador de modelos robóticos, clicando en Tools->Robot Simulation Robot Builder.



Sale una interfaz que nos permite: importar las piezas del robot, ensamblarlas, y asignarle sensores.



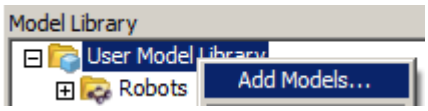
Arriba a la izquierda, hay la librería de robots y componentes robóticos, que ya tiene el Labview, con varios modelos robóticos y varias piezas para montar nuestros robots.

En la derecha hay 3 secciones:

Arriba está la lista de piezas que forman el robot.
En medio hay el árbol de las uniones entre piezas.
Abajo hay las propiedades de la pieza o la unión seleccionada.

Abajo tenemos los botones para importar y exportar robots.

Primero importamos las piezas del robot, que se han hecho con el Solidworks. Se va a la librería, colocamos el ratón encima de User Model Library, clicamos con el botón derecho y seleccionamos Add Models.



Sale un menú para que escojamos el archivo que contiene la pieza. El archivo puede estar en varios formatos, como ive, dae, wrl. Aun estando en esos formatos puede ser que el Labview no sea capaz de leer el archivo.

Para solucionar este error, hay que modificar las opciones con las que se guardan los archivos de las piezas en el Solidworks, o se puede usar un programa de edición 3D, que haga de puente entre el Solidworks y el Labview.

Como el Labview usa la Librería libre Open Scene Grafics, reconoce el formato “osg”, así que se puede usar el SimLab para que haga de puente entre el Solidworks y el Labview, pasando los archivos de las piezas del Solidworks a formato “osg”.

Antes de importarlas al Labview, se ha de cambiar la extensión de osg a lvsg. Luego se usa el sistema ”DeLVSGaIVE.vi” con el Labview, para cambiar de formato “lvsg” a formato “ive”.
Las piezas en formato “ive” ya se pueden importar con el Robot Simulation Robot Builder del Labview.

Esquema del sistema VI para convertir de lvsg a ive:

Panel Frontal:

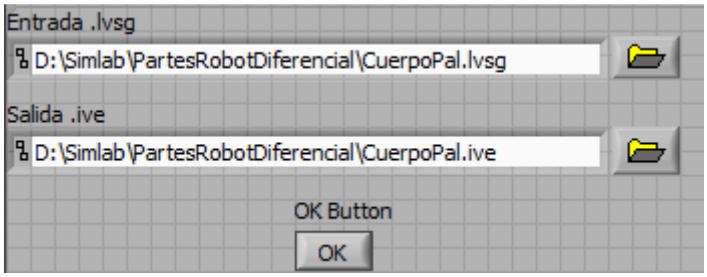
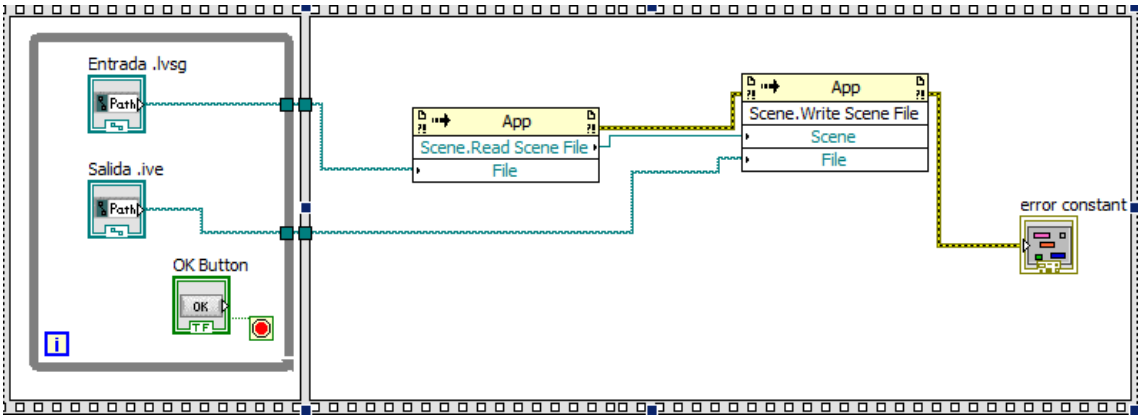
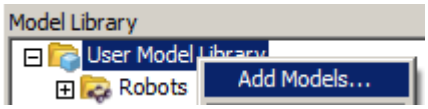
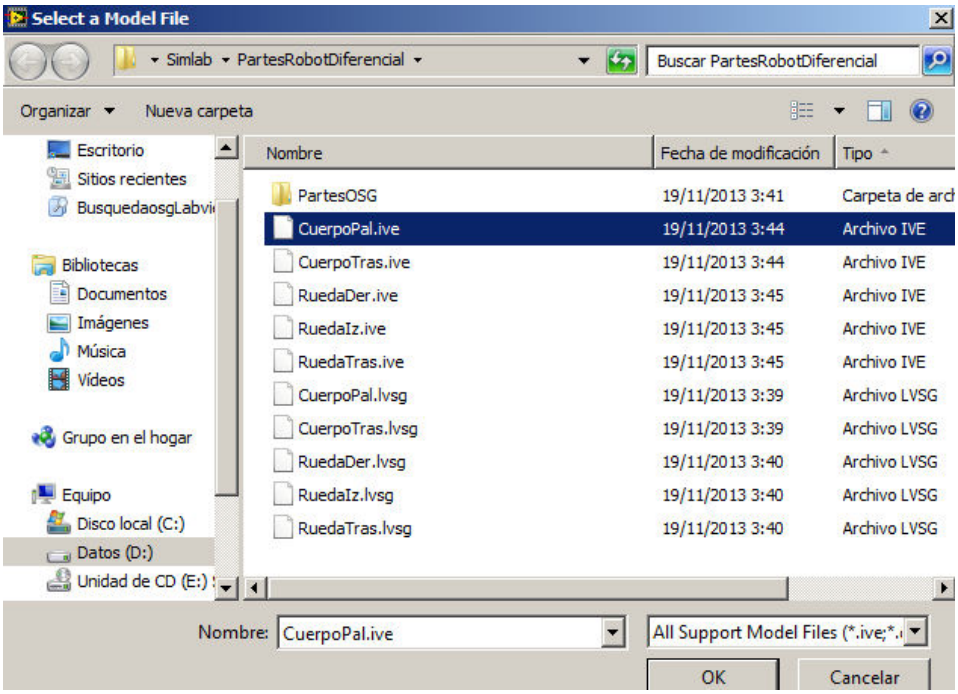


Diagrama de Bloques:

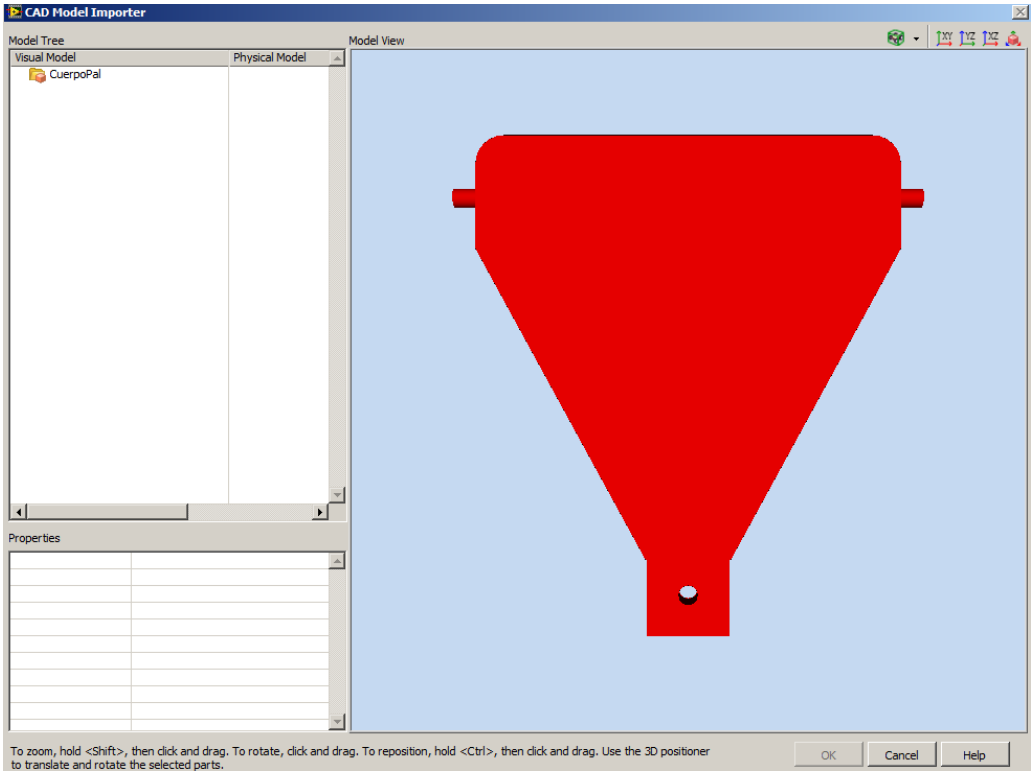


Se importa la pieza “ive” en el Robot Simulation Robot Builder:

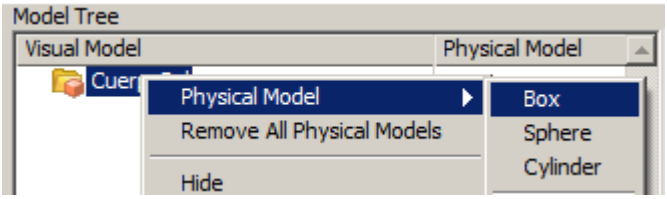




Sale una ventana con la pieza que diseñamos en el Solidworks:



En la parte izquierda de la ventana hay el árbol del modelo que contiene los componentes de la pieza. Aquí tenemos que seleccionar la pieza y asignarle un modelo físico de tipo Box.



Ahora se comprueban las propiedades y se cambian las que sean necesarias, como la posición, rotación, material, color, etc.

Properties

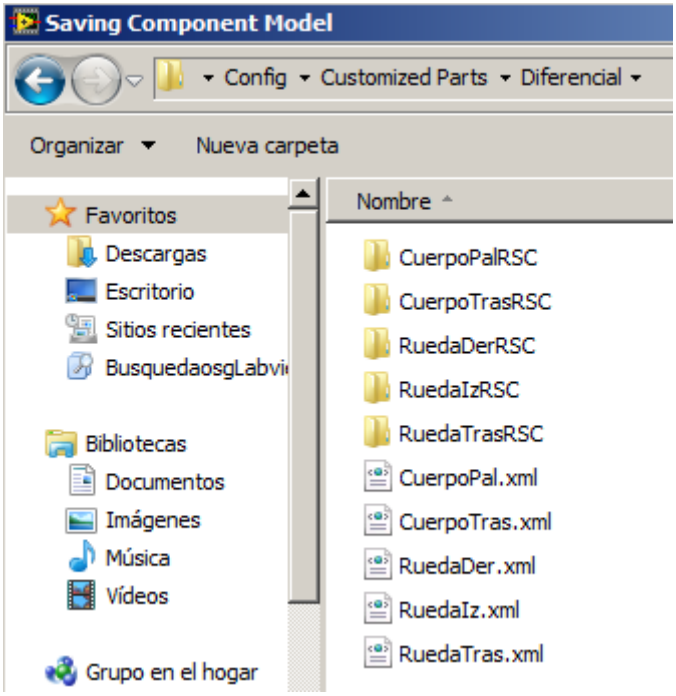
Material Type	Steel
Position X (m)	0
Position Y (m)	0,01
Position Z (m)	-0,015
Rotation X (deg)	0
Rotation Y (deg)	0
Rotation Z (deg)	0
Visible	True
Subspace Name	
Color	R:0 G:255 B:0

Una vez se tiene la pieza con su modelo físico, se guarda dando al botón OK.

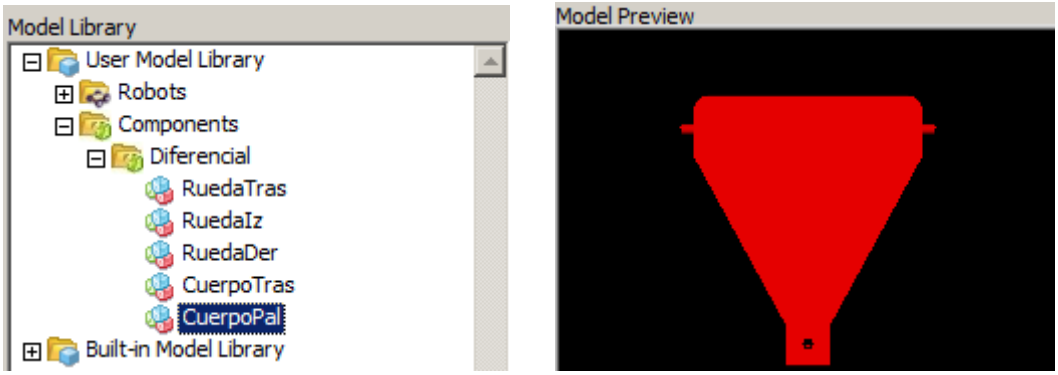
Sale una ventana, se pone el nombre y se guarda.
El Labview solo puede guardar las piezas en esta ruta:

C:\Users\NomUser\Documents\LabVIEW Data\Robotics\Simulator\Model\Config\Customized Parts\

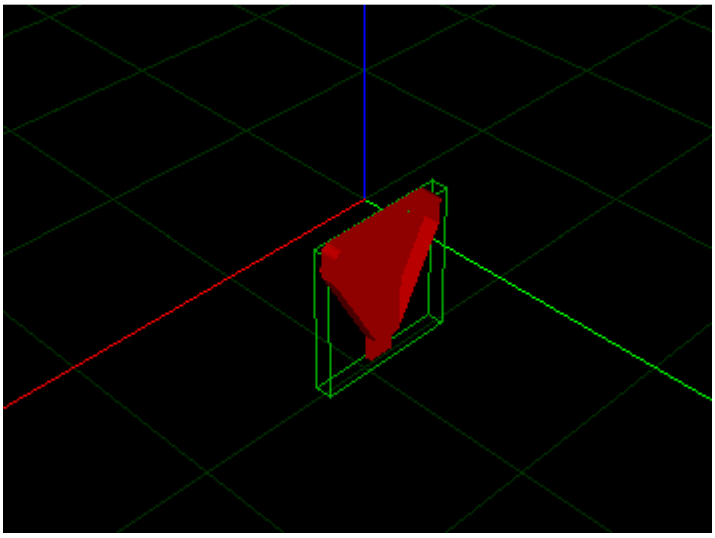
Aquí se puede crear una carpeta para cada robot que se construye, así sus piezas estarán agrupadas, por ejemplo la carpeta Diferencial. Al guardar la pieza, el Labview nos crea un archivo "xml" y una carpeta que contiene el archivo 3D en formato "ive" :



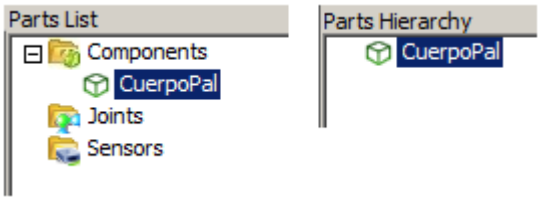
Después de importar las piezas, se pueden seleccionar, en la librería del Robot Simulation Robot Builder, para crear el robot:



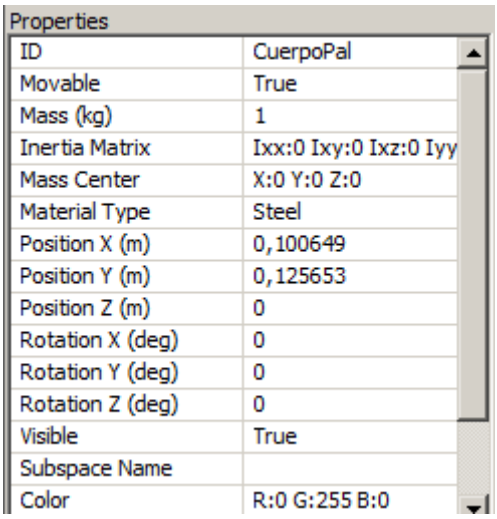
En el árbol de la librería, se clica la pieza y se arrastra al entorno de montaje en la ventana central:



En la derecha, la pieza se ha añadido a la lista y al árbol de piezas.



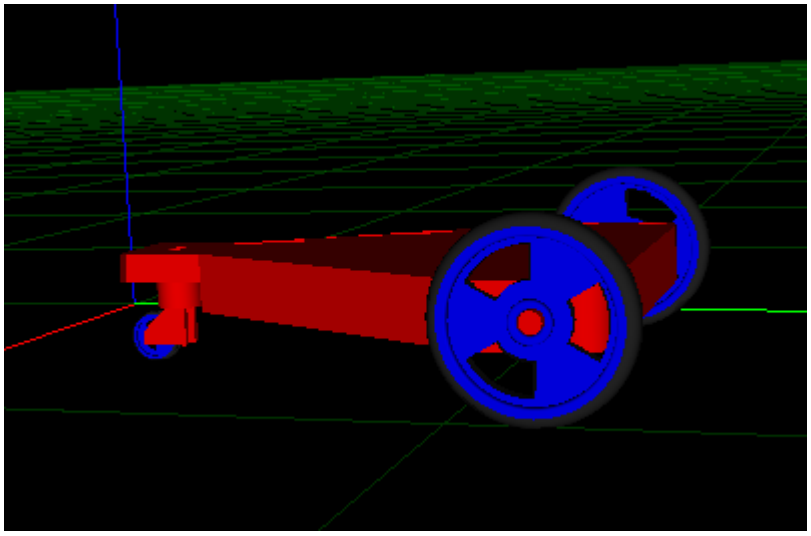
Debajo aparecen las propiedades de cada pieza, como la posición, la orientación, la masa, etc.



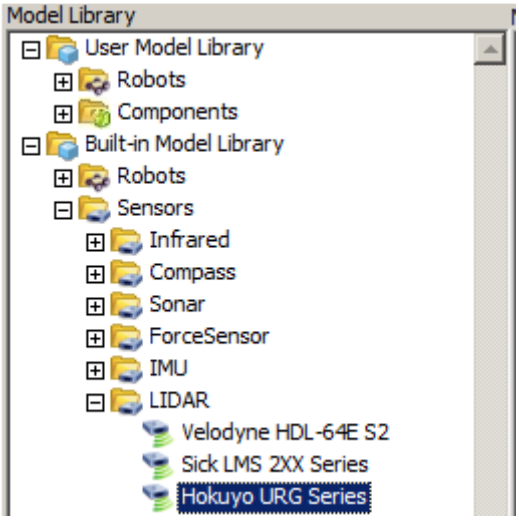
Properties

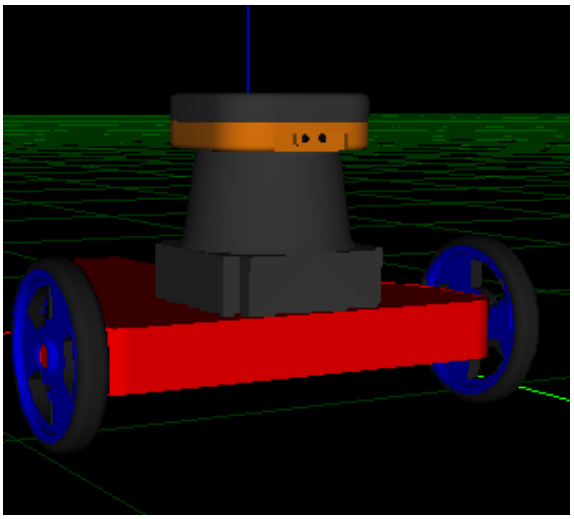
ID	CuerpoPal
Movable	True
Mass (kg)	1
Inertia Matrix	Ixx:0 Ixy:0 Ixz:0 Iyy
Mass Center	X:0 Y:0 Z:0
Material Type	Steel
Position X (m)	0,100649
Position Y (m)	0,125653
Position Z (m)	0
Rotation X (deg)	0
Rotation Y (deg)	0
Rotation Z (deg)	0
Visible	True
Subspace Name	
Color	R:0 G:255 B:0

Se añaden todas las piezas, se colocan en la posición y orientación correcta, y se les asigna una masa adecuada para que el robot sea equilibrado.



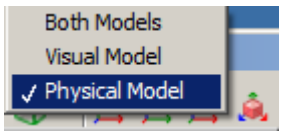
El robot está montado pero no tiene sensores, así que se le añaden. Se escoge un sensor en la librería, en este caso un sensor LIDAR, que detecta los obstáculos que tenga alrededor:



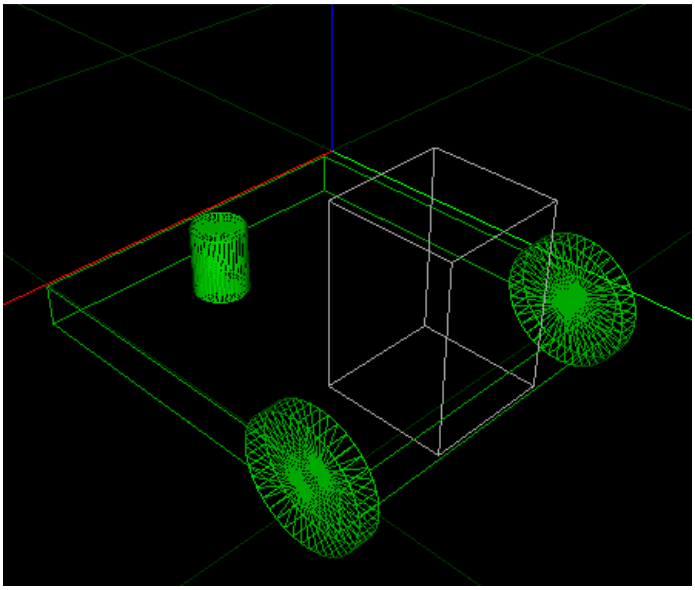


Ahora el robot ya está ensamblado, pero carece de movimiento entre sus piezas, así que se asignan unas articulaciones, una por par de piezas moviles. Hay diferentes tipos de articulación: motorizadas, de movimiento libre, o de unión fija.

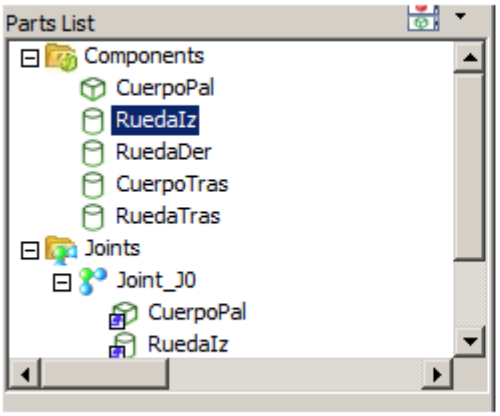
Primero cambiamos la vista a modo físico:

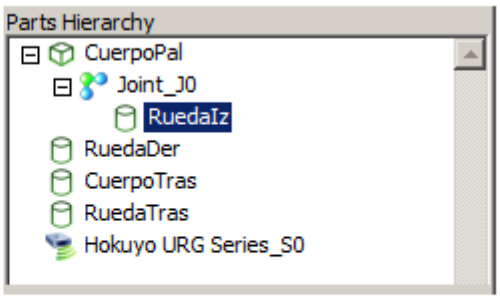


Ahora en el entorno se ven las cajas y cilindros que forman el modelo físico del robot:

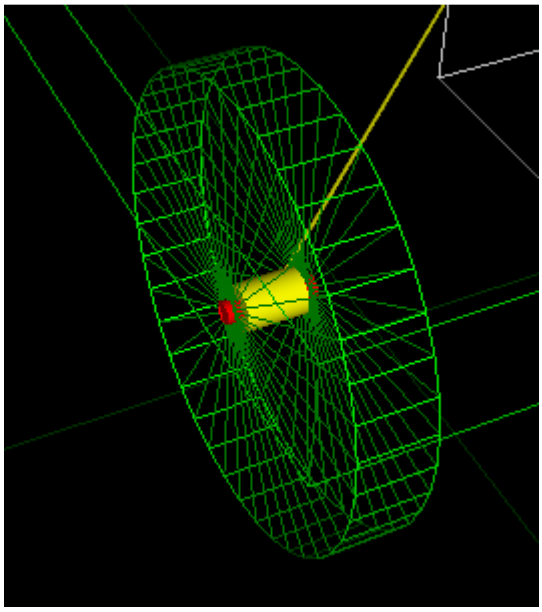


Para añadir una articulación con motor, entre el cuerpo principal y la rueda izquierda, se va al árbol de piezas, se selecciona la pieza de la rueda izquierda y se arrastra a la del cuerpo principal.
Al hacerlo se crea una unión entre las dos piezas:





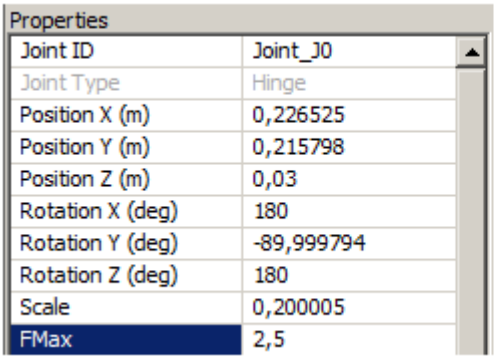
Entre las dos piezas se crea un cilindro o bisagra con un eje de color rojo y un rotor de color amarillo:

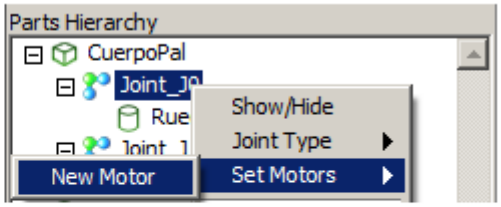


Para escoger una unión motorizada hay que hacer dos pasos:

Primero, se va a las propiedades de la unión, y en el campo FMax, le asignamos una fuerza máxima de torque, que será la fuerza con la que alcanza la velocidad de rotación del motor.

Segundo, se clica encima la unión con el botón secundario, y le asignamos un motor.

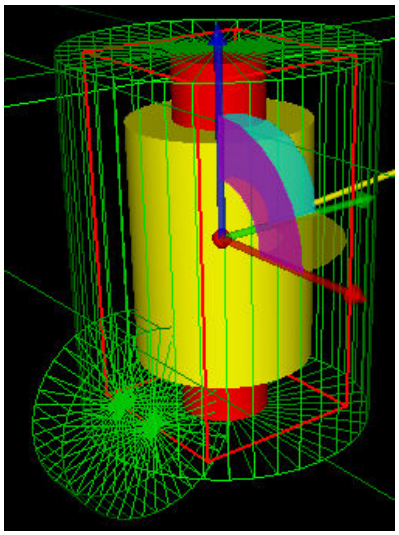




En las propiedades, aparece el nombre del motor, que se usará en el simulador para inicializarlo y asignarle la velocidad.

Hay otras propiedades que afectan a la unión, que están descritas en el apéndice, que controlan la elasticidad, los rebotes, factores de corrección, etc.

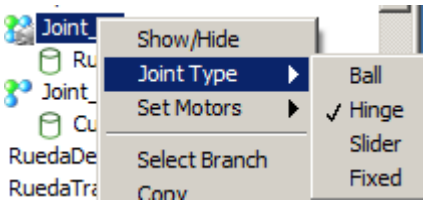
Para insertar una unión de giro libre, como en los ejes de la rueda trasera del robot, hay que poner la propiedad FMax a 0 y no asignarle ningún motor.



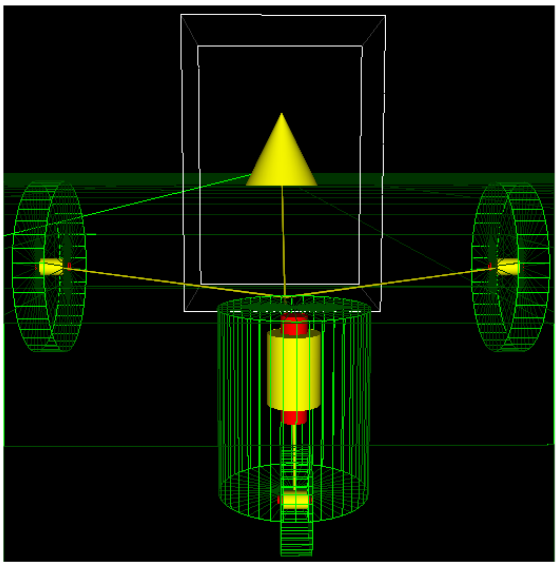
Properties

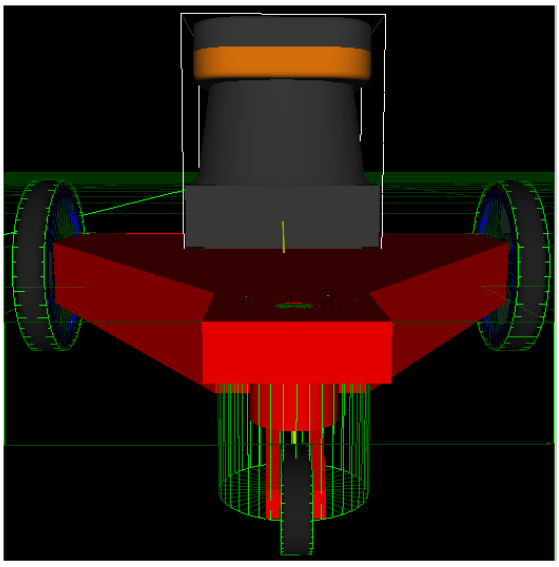
Joint ID	Joint_J1
Joint Type	Hinge
Position X (m)	0,146025
Position Y (m)	0,070798
Position Z (m)	0,02318
Rotation X (deg)	0
Rotation Y (deg)	0
Rotation Z (deg)	0
Scale	0,54
FMax	0
Fudge Factor	0,1
Bounce	0
CFM	0,001
Lo Stop	-Inf
Hi Stop	Inf

El Labview tiene más tipos de uniones, podemos cambiar el tipo de unión, si clicamos con el botón derecho encima de ella, sale un menú y se puede seleccionar: unión esférica, bisagra, cilíndrica y fija.

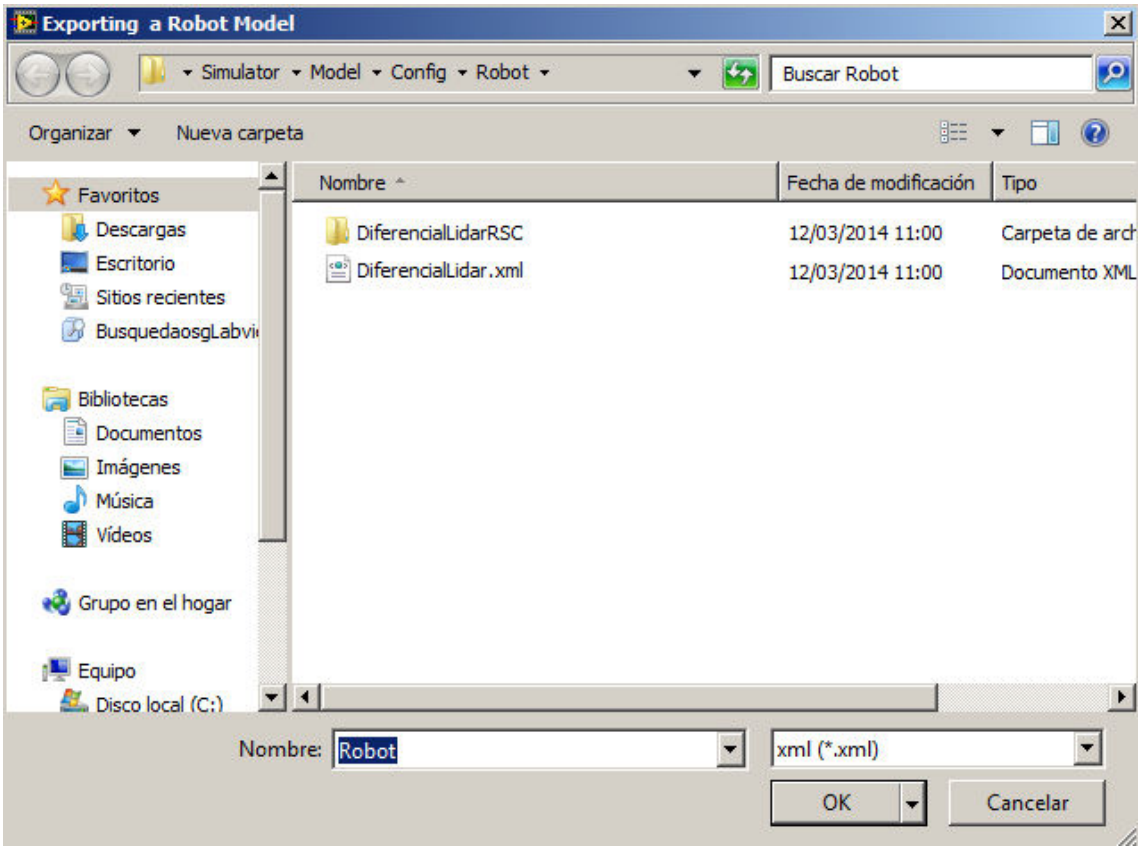


Se crean todas las uniones que necesite el robot, y se guarda para usarlo en los simuladores.





Para guardar el robot, se clica en el boton de exportar, sale una ventana, se pone el nombre del robot y se clica en OK.



El Labview crea un archivo “XML” y la carpeta con los archivos “ive” de cada pieza que lo compone.

El Labview solo puede guardar los robots en esta ruta:

C:\Users\nomUser\Documents\LabVIEW Data\Robotics\Simulator\ Model\ Config\Robot

Ahora ya se puede usar el robot, y añadirlo a la creación de un simulador.

9.2 Creación del entorno de simulación en el Labview

Primero se crea el entorno con el Solidworks, dándole formas geométricas y extruyéndolas, y se guardan en un archivo 3D.

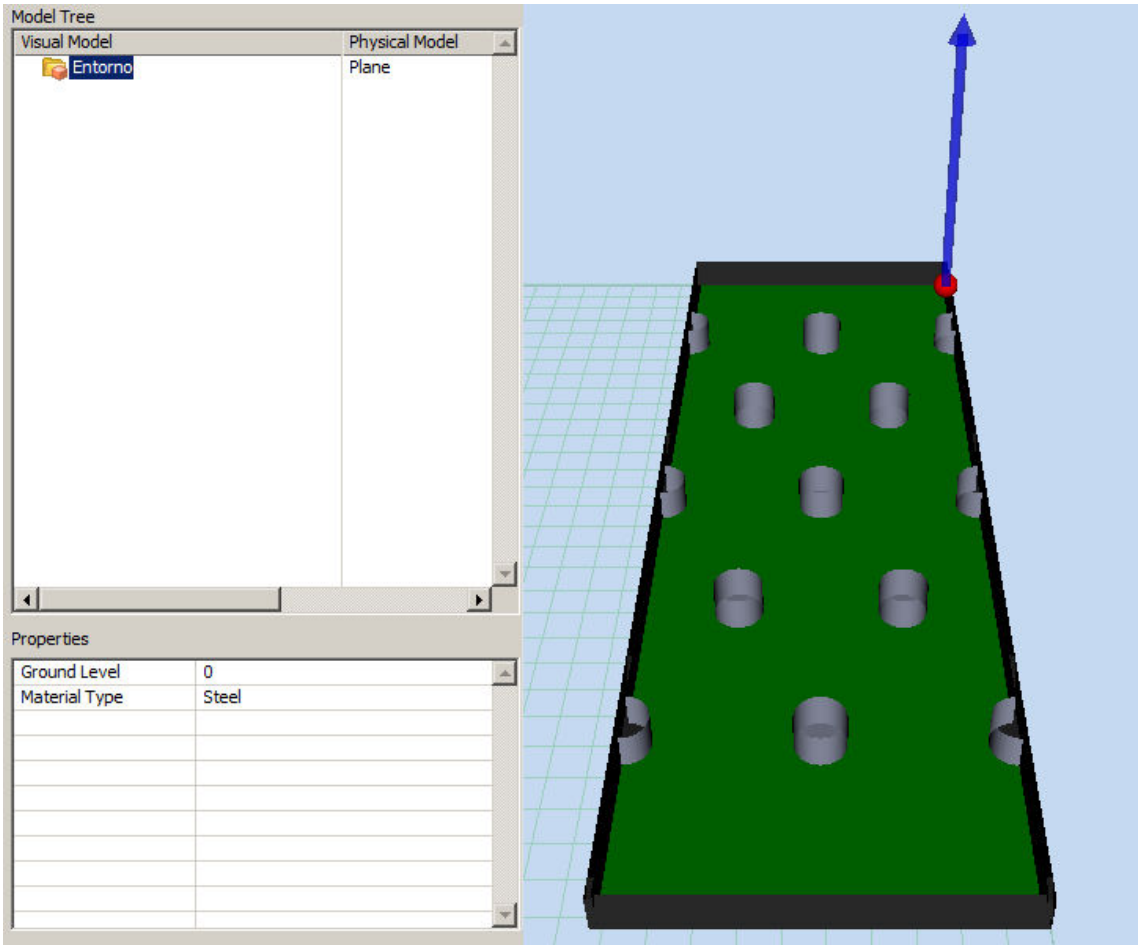
Al igual que pasaba con las piezas del robot, se ha de importar el archivo del entorno al Labview.

Para importar el entorno al Labview, se siguen los mismos pasos, usando el SimLab como puente, guardando el archivo en formato "osg", cambiando la extensión a "lvsg", usando el sistema "DeLVSGaIVE.vi" para cambiar de formato "lvsg" a "ive".

Cuando se tiene el entorno en formato "ive", se usa el sistema "ImportarEntornoIVE.vi" para crear un entorno en el Labview.

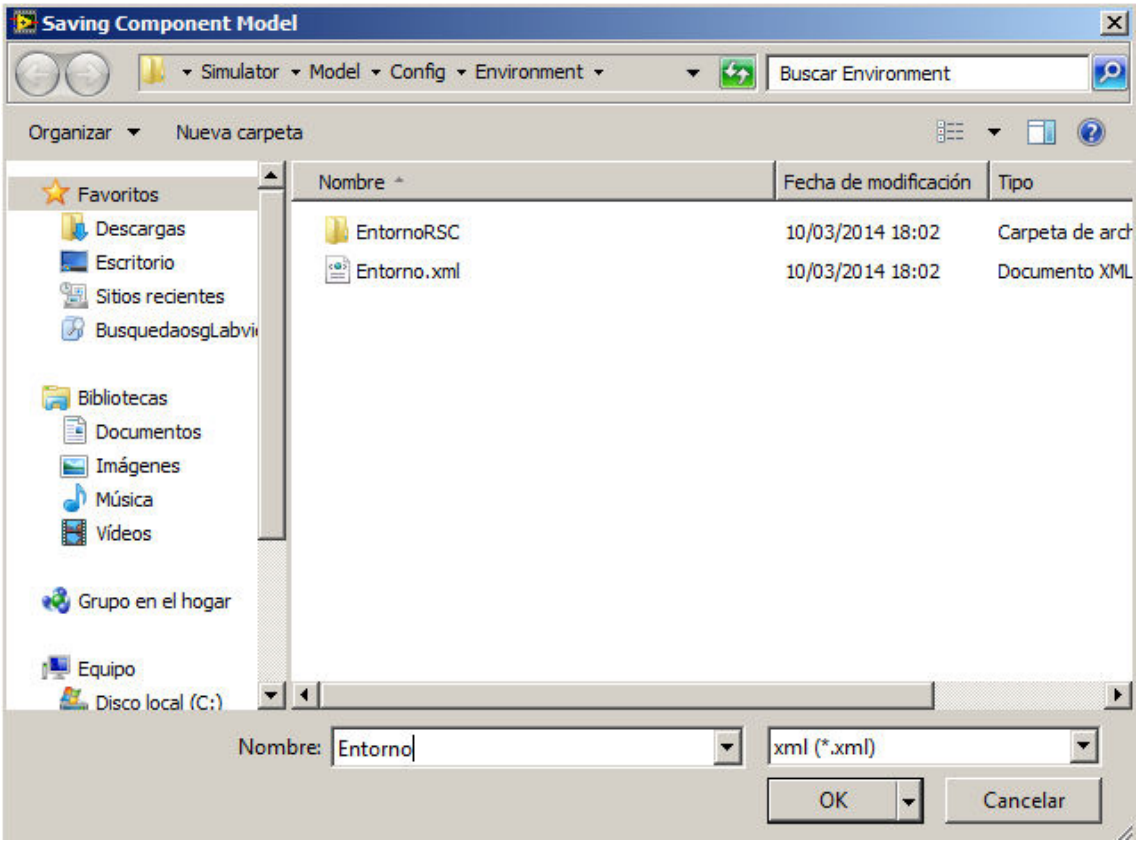
Funcionamiento del sistema ImportarEntornoIVE.vi:

Se selecciona el archivo "ive" del entorno y ejecutamos el sistema. Sale una ventana mostrando el entorno, arriba a la izquierda está el árbol de componentes con su modelo físico, en este caso un plano, abajo del árbol están las propiedades, donde se escoge el material.



Ahora se clica en el botón OK y muestra una ventana para guardarlo. No deja cambiar la carpeta, sale un error, y solo se puede guardar en:

C:\Users\nomUser\Documents\LabVIEW Data\Robotics\Simulator\Model\Config\Environment



Se le pone nombre y se clica en el botón OK, el Labview crea un archivo "XML" y las carpetas con los archivos "Entorno.ive" y "Entorno.stl".

Ahora ya se puede usar el entorno y añadirlo en la creación de un simulador.

9.3 Desarrollo del sistema que crea sistemas difusos.

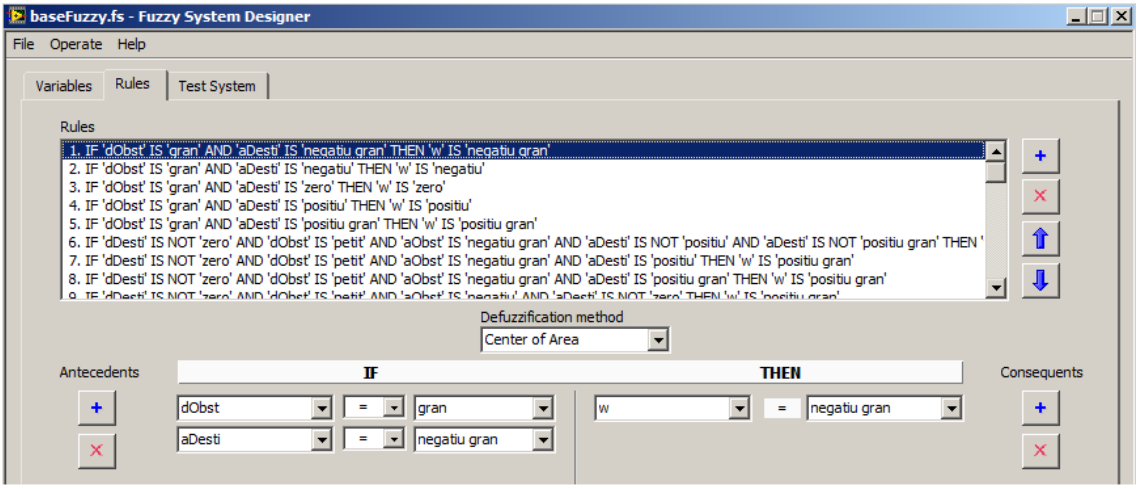
Como se ha comentado antes, se necesita un sistema que genere un sistema *fuzzy*, para luego usarlo en el controlador del simulador. El generador ha de tener de entrada los valores de las funciones de pertenencia y ha de contener las reglas diseñadas en el apartado 8.3.
Como este generador lo usa la clase Simulación, diseñada en la sección 8.5.1.2, se ha de poder usar como una librería dll.

Pasos del desarrollo:

<u>Creación de las base de reglas para los sistemas fuzzy:</u>

Con el asistente para diseñar sistemas *fuzzy* se crea un archivo que contenga las reglas del sistema, llamado por ejemplo baseFuzzy.fs

Se crean las variables con los valores lingüísticos, de manera uniforme, ya que esos valores sólo se usan para poder añadir las reglas.
Se añaden las reglas de la sección 8.3.

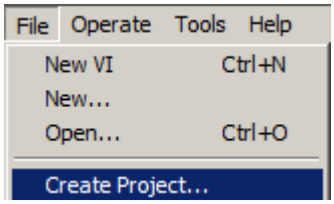


Cuando tenemos las reglas introducidas, se guarda con el nombre “baseFuzzy.fs”.

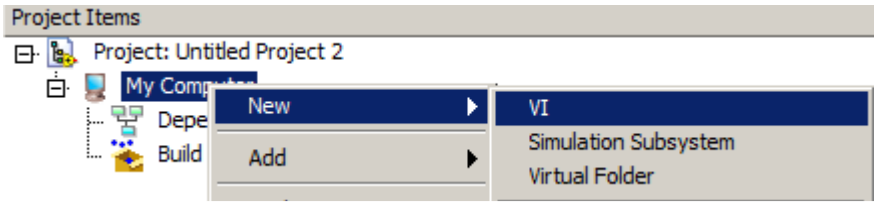
Ahora las reglas están en un archivo ”fs”, del que luego recuperaremos en el generador para introducirlas en los sistemas fuzzy.

Creación del generador de sistemas fuzzy:

Se crea un proyecto nuevo:



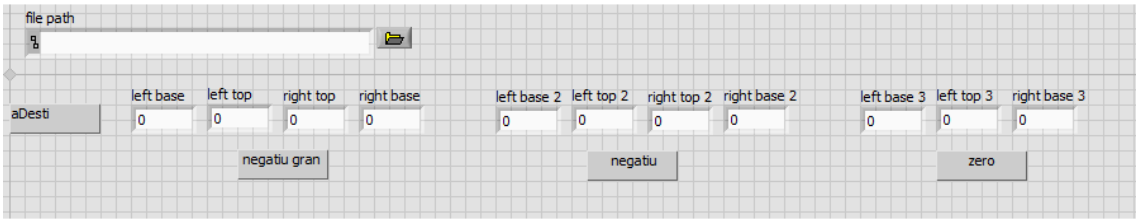
Se selecciona Blank Project y se clica en Finish.
Se va al árbol de archivos del proyecto y se añade un VI.



En el nuevo sistema se añaden:

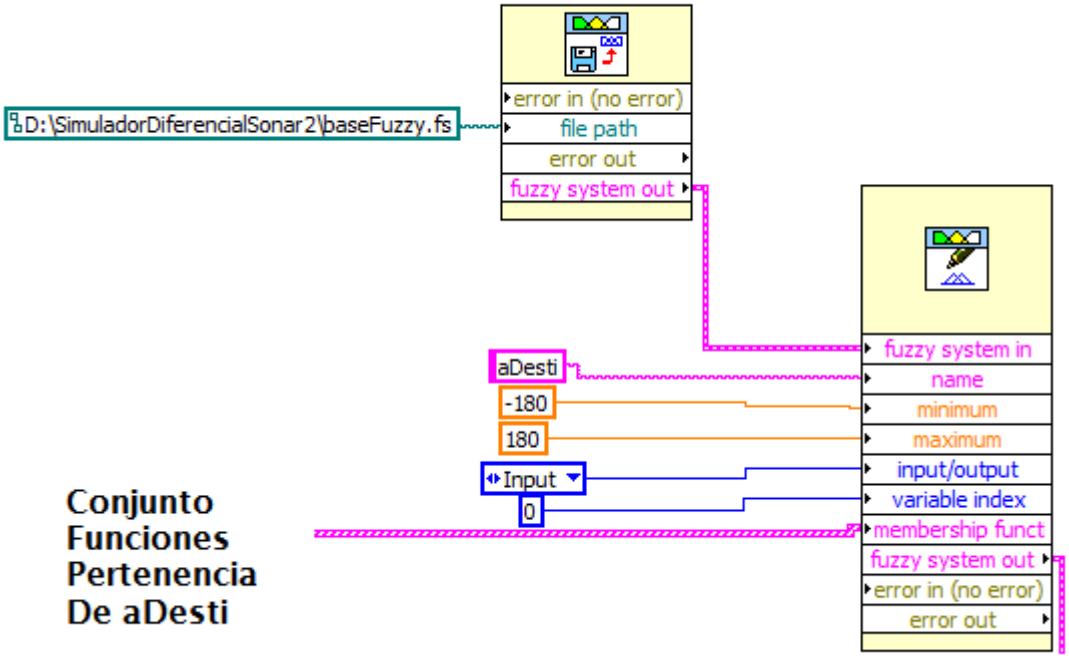
Controles numéricos, para los valores de cada función de pertenencia.
Unos módulos, para importar la reglas lógicas del archivo baseFuzzy.fs.
Unos módulos, para crear y guardar el nuevo sistema fuzzy.

Muestra del esquema del panel frontal con los controles numéricos y la ruta para guardar el nuevo sistema fuzzy en un archivo:



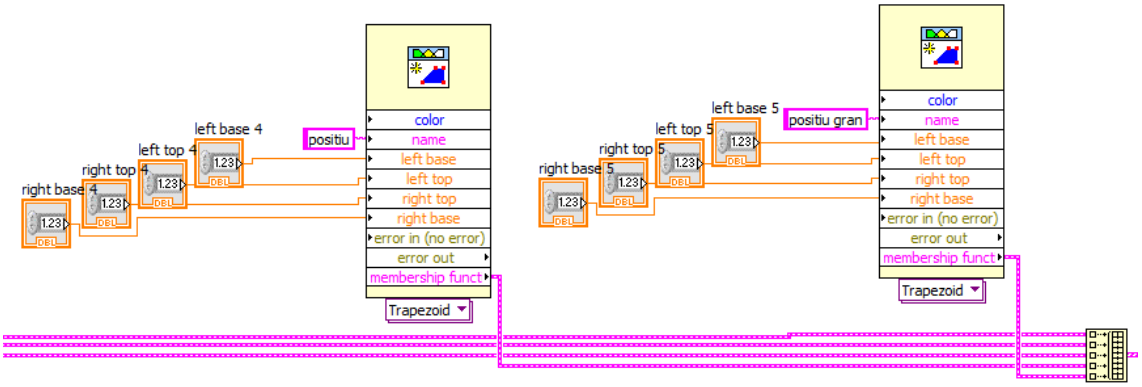
Partes del diagrama de bloques del sistema:

Bloques que importan el archivo baseFuzzy.fs y establece el conjunto de funciones de pertenencia para la variable aDesti:



Cada variable se crea con la unión de bloques que definen funciones de pertenencia, un bloque por valor lingüístico.

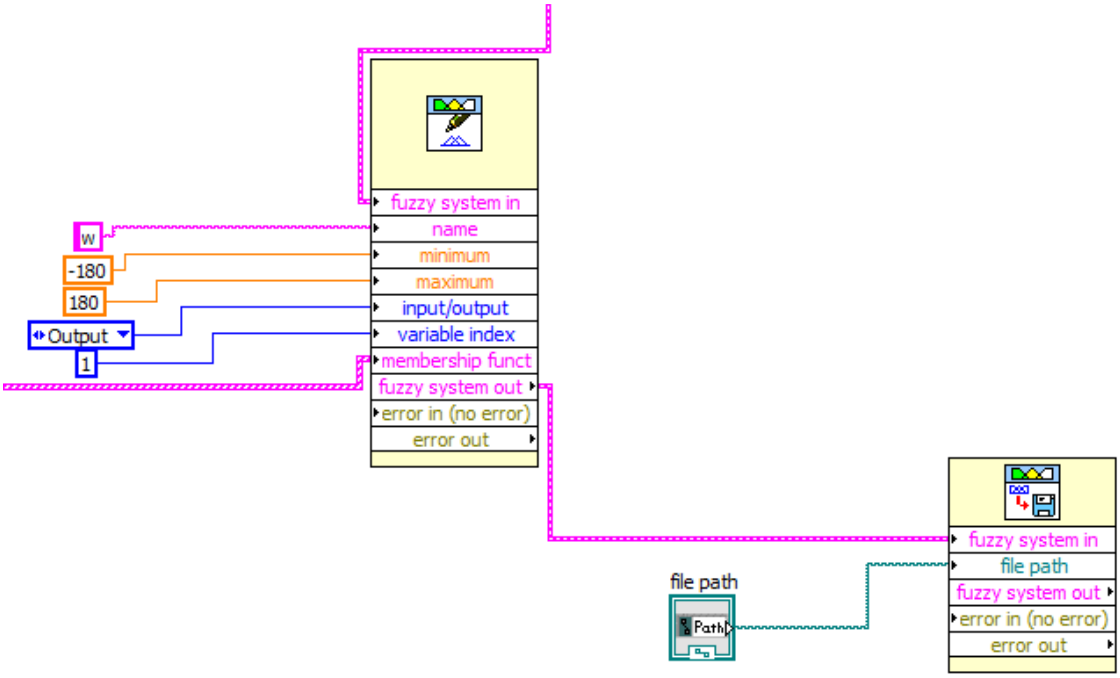
Muestra de una parte de los bloques para la variable aDesti y su union:



Se hace lo mismo para el resto de variables y funciones de pertenencia.

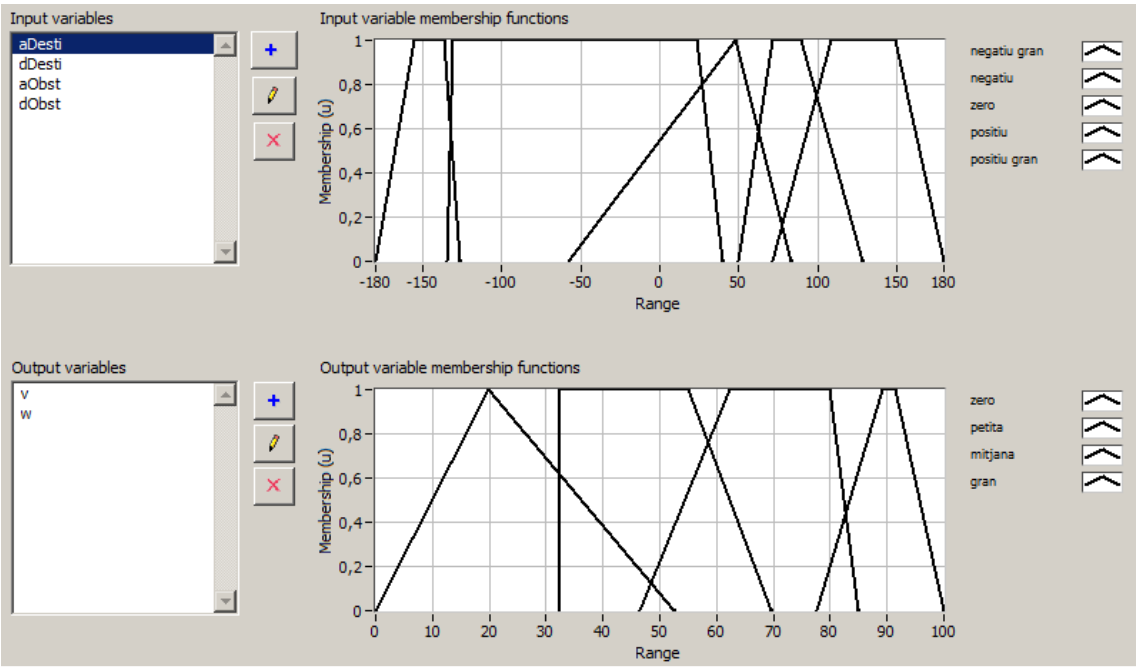
Cuando se tienen todas las variables con las funciones de pertenencia y las reglas lógicas, hay que crear el módulo que lo guarda.

Muestra del diagrama de bloques que añade al sistema la ultima variable del sistema, la velocidad angular “W”, y usa la ruta del archivo donde guardara el sistema difuso:



En el panel frontal, se introducen unos valores en los controles numéricos, el nombre del sistema que contiene las normas y probamos a crear un sistema fuzzy.

Se abre el archivo del sistema con el asistente para ver como ha quedado:



Antes de pasar a crear la librería, se tiene que especificar que variables de entrada y salida tiene el sistema.

En este caso las variables de entrada son:

a) Un array de valores númericos que defininen las funciones de pertenencia, así que se cambian los controles numéricos por un array numérico para cada variable:

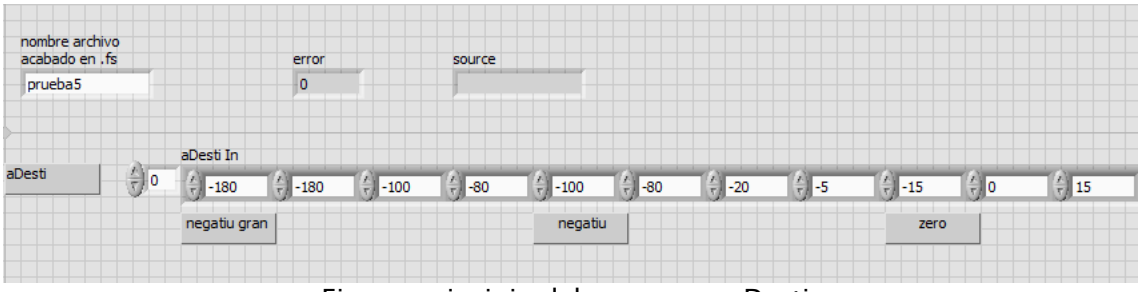


Figura: principio del array para aDesti

b) El nombre o ruta del archivo fuzzy con el que se creará el archivo del sistema difuso:

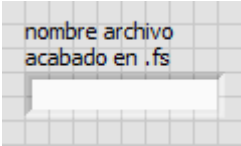


Figura: casilla para el nombre o ruta

Como salida del sistema, necesitamos retornar si ha habido error. Se usa el código de error y el texto del error:

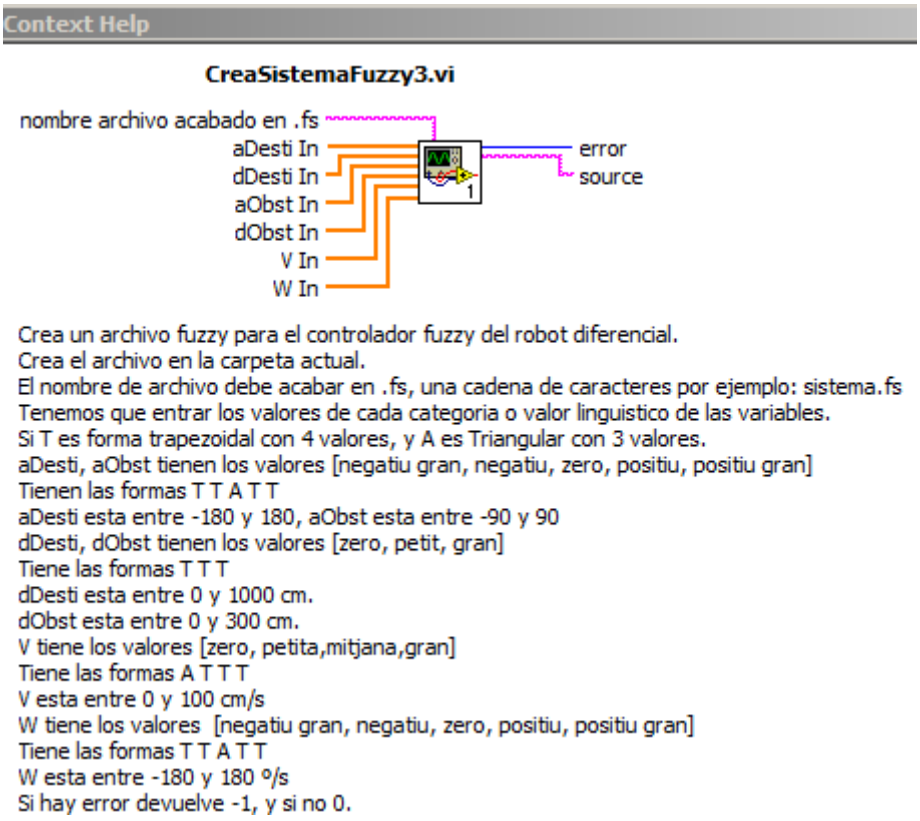
Figura: indicadores del error

Para asignar las entradas y salidas en un sistema del Labview, en el panel frontal, hay un cuadrado en la parte superior izquierda:

Las casillas de la izquierda se usan para las entradas y las de la derecha para las salidas.
Para asignar una casilla, clicamos encima de ella, el puntero cambia a una figura con forma de hilo o cuerda, y se clica encima del control o elemento del sistema, y entonces la casilla cambia de color, indicando que se ha asignado una entrada o salida del sistema.

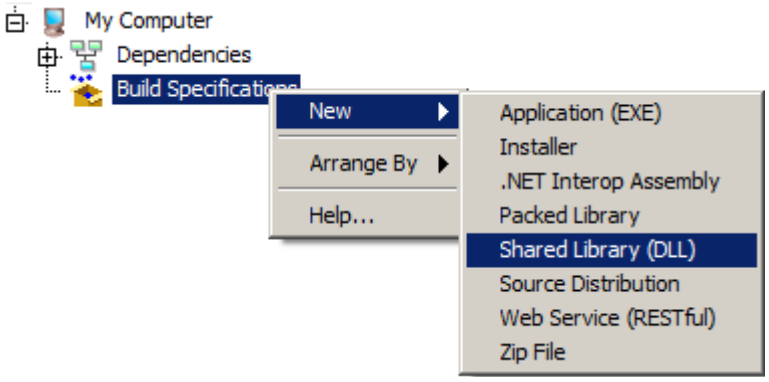
Se configuran todas las entradas y salidas del sistema y se deja preparado para crear una librería dll.

Si activamos la ventana del Context Help y nos colocamos encima del cuadrado de las entradas y salidas, muestra como ha quedado el sistema:



Para crear la librería tipo dll, se siguen estos pasos:

En el explorador del proyecto, se va al árbol de archivos y se selecciona la opción de crear la librería dll:



Sale un asistente en el que hay que añadir información y seleccionar unas opciones.

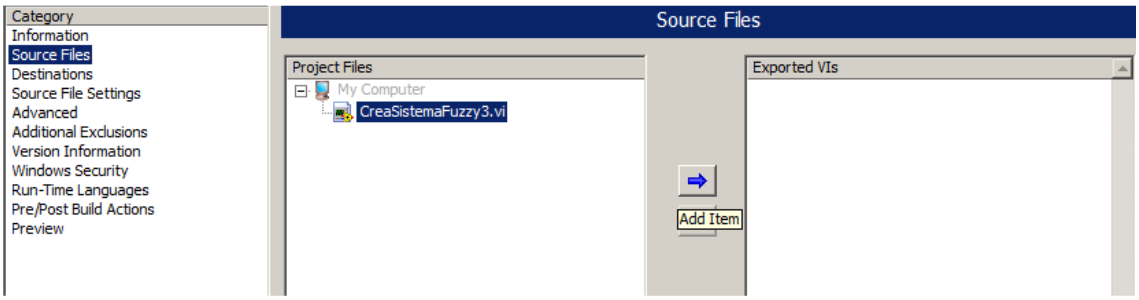
Los pasos principales son:

En Information:

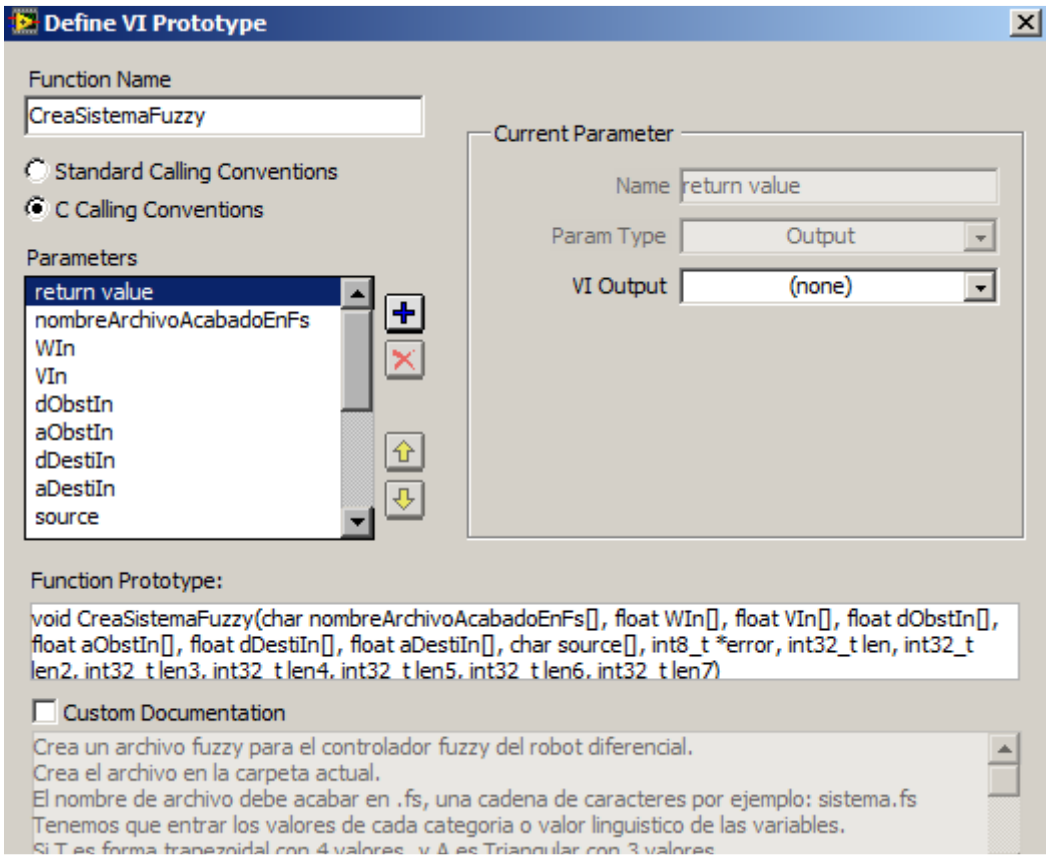
Se introduce el nombre de la creación (Build … name)
Se introduce el nombre de la librería (Traget filename)
Se escoge la carpeta donde guardará la librería (Destination …).

En Source Files:

Añadimos a exported VIs los archivos del proyecto.
Por cada VI añadido crea una función en la librería.

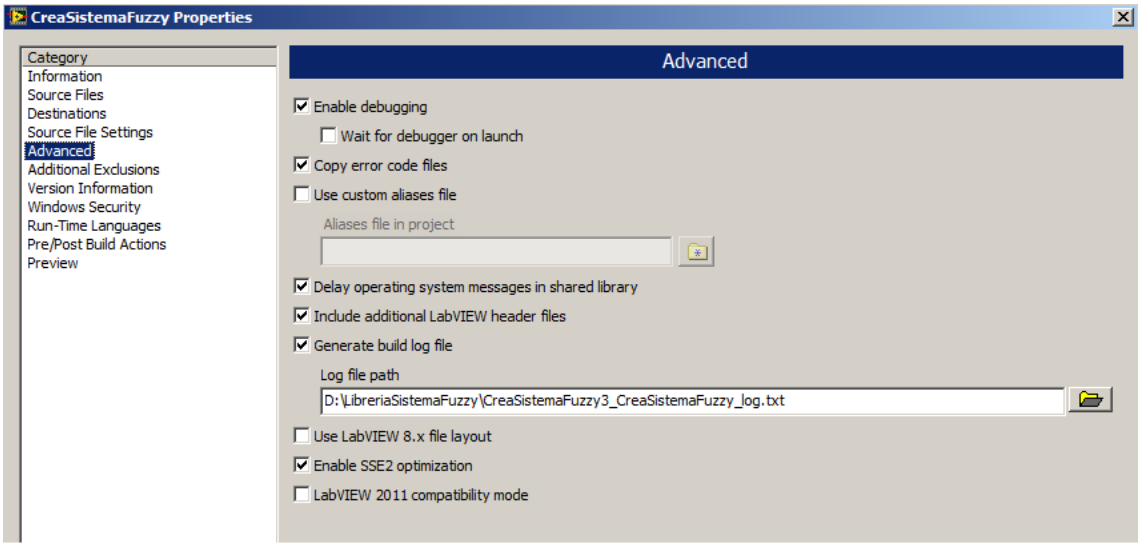


Muestra la ventana para configurar la función de la librería.



Seleccionamos las entradas, las salidas, y su orden en la función. Creamos una documentación para usar la función en nuestros programas en C o C++.

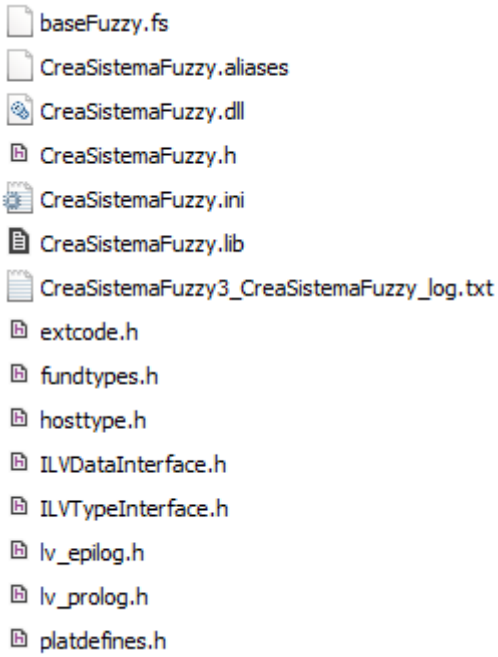
En advanced:



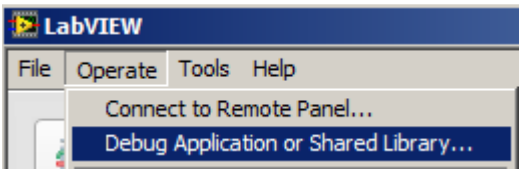
Marcamos la opción de incluir additional Labview header files, para usar la librería en programas en C.
Activamos la optimización SSE2, siempre que la CPU las soporte.

Después de esta configuración, creamos la librería.
En la carpeta tiene que haber creado: el archivo dll, una serie de cabeceras de C, y los archivos de datos, que necesita la librería para funcionar, como por ejemplo el archivo "baseFuzzy.fs"

Contenido de la carpeta de la librería:

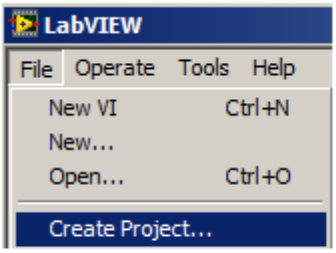


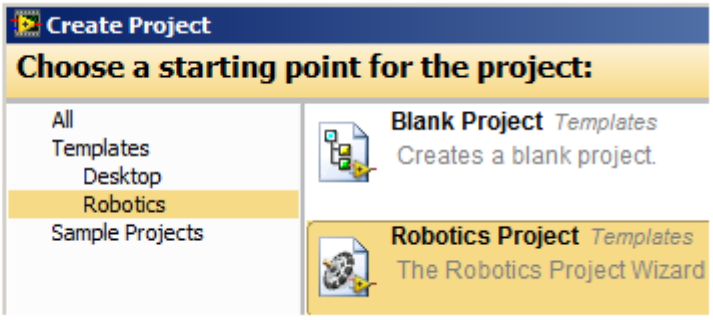
Si se realiza una prueba de la librería, puede que no funcione y de un error. El Labview permite monitorear o depurar la ejecución de una librería, y así solucionar los problemas que haya.
Para esto, tenemos que crear la librería con la opción "Wait for debugger on launch" que está en la pestaña Advanced de la configuración.
Ahora se puede ejecutar nuestro programa, ir al Debugger del Labview y ver que le ocurre, ya que no empieza la ejecución, hasta que no se sincroniza con el Debugger.



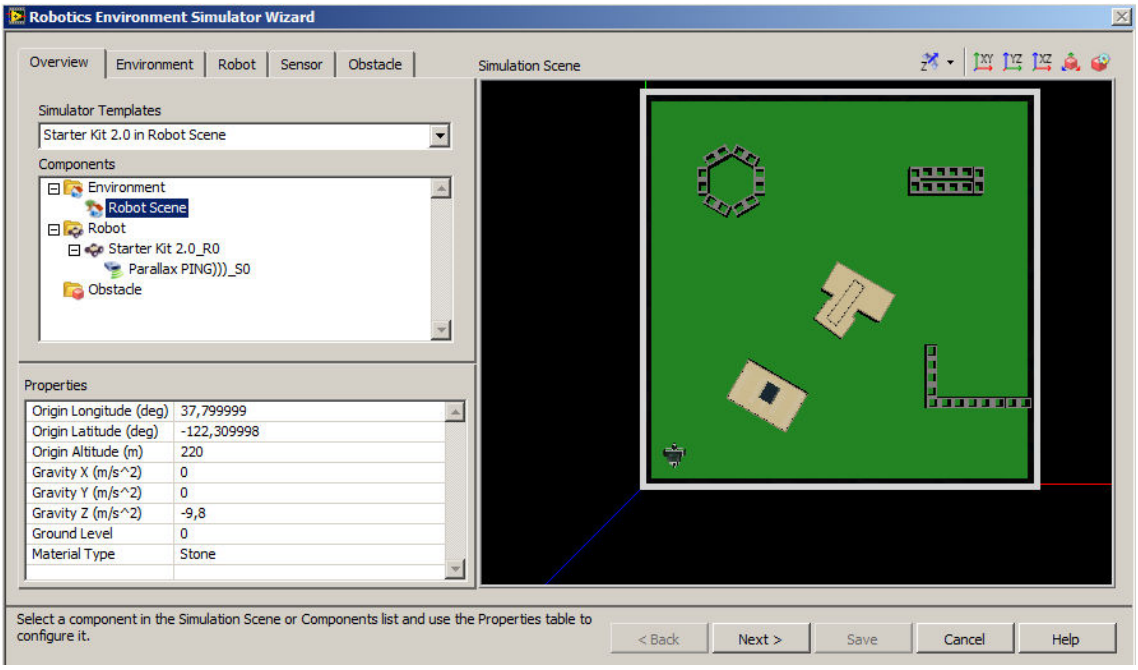
9.4 Creación del simulador cinemático en el Labview

Se crea un nuevo proyecto en el Labview y se escoge la plantilla de un proyecto robótico.

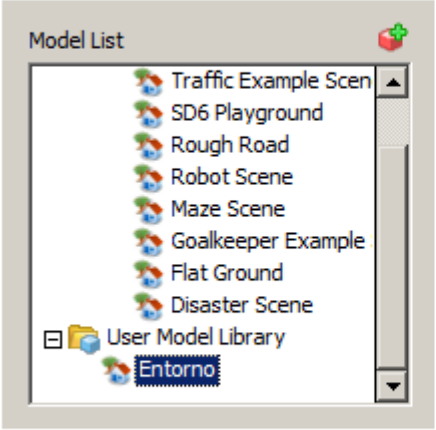


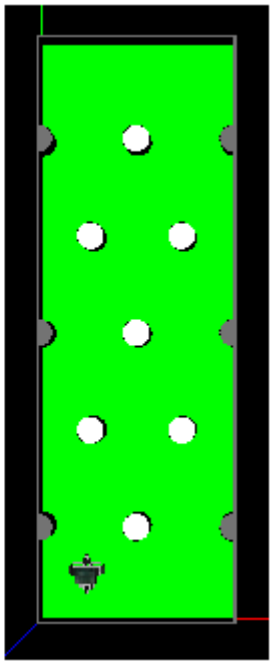


El Labview muestra una ventana para escoger entre: crear un robot con plataforma Windows o crearlo en un simulador de entornos.
Se escoge crear un robot en un simulador, y clicamos en Next.
Muestra la ventana del asistente que nos ayuda a crear el simulador.



Se selecciona la pestaña de Enviroment. En el árbol de la derecha, bajo el nodo de User Model Library, ha de aparecer el entorno que se ha importado en el apartado anterior. Para escogerlo hacemos doble clic.





Aquí también se pueden modificar algunas de las propiedades del entorno, como la posición, el material, la gravedad, etc.

Properties

Origin Longitude (deg)	37,799999
Origin Latitude (deg)	-122,309998
Origin Altitude (m)	220
Gravity X (m/s^2)	0
Gravity Y (m/s^2)	0
Gravity Z (m/s^2)	-9,8
Ground Level	0
Material Type	Steel

Seleccionamos la pestaña Robot y para cambiar el robot:

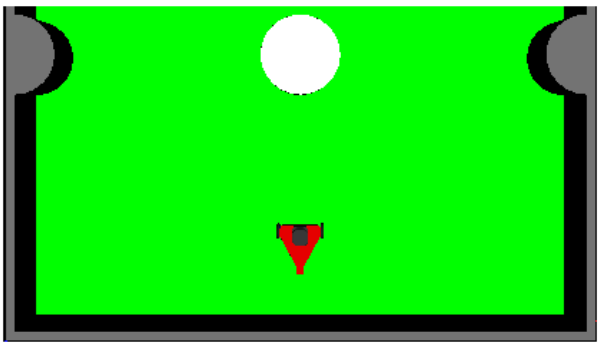
Primero, se clica encima del robot que hay en el entorno y se aprieta la tecla suprimir, esto quitará este robot.

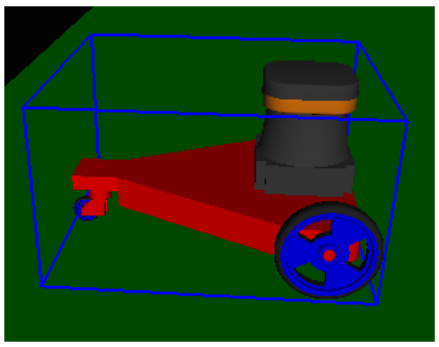
Segundo, se va al árbol de la librería, y bajo el nodo de User Model Library, ha de aparecer el robot que se ha importado en el apartado anterior. Se clica encima y se arrastra al entorno. Para colocarlo en la posición adecuada, editamos las propiedades de la posición y orientación.

Cálculo de la posición del robot:

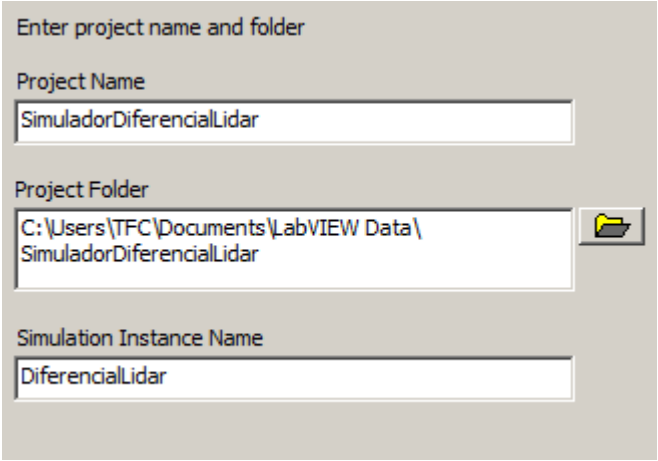
El entorno mide 2 metros de ancho, así que para que el robot esté centrado hay que ponerlo a x=1 m.
El robot mide 6cm de altura, para que esté sobre la superficie del entorno que hay que ponerlo a z=0'06 m.
Hay que ponerlo delante de la primera columna, así que y=0'2 m.

La posición inicial del robot es: (1, 0'2, 0'06)

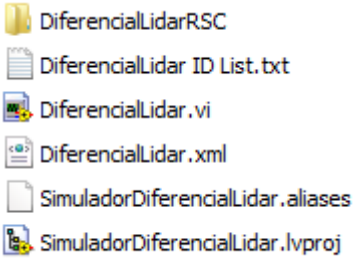




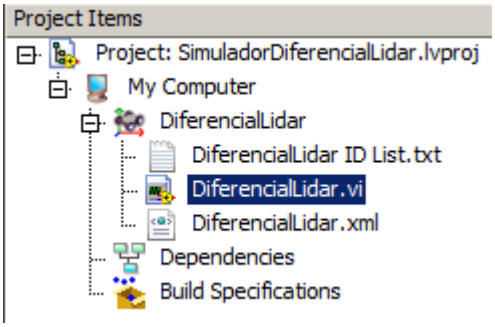
Ahora ya tenemos el robot y el entorno de simulación, se clica en Next. Muestra una ventana, se introduce: el nombre del proyecto, la ruta del proyecto y el nombre del simulador.



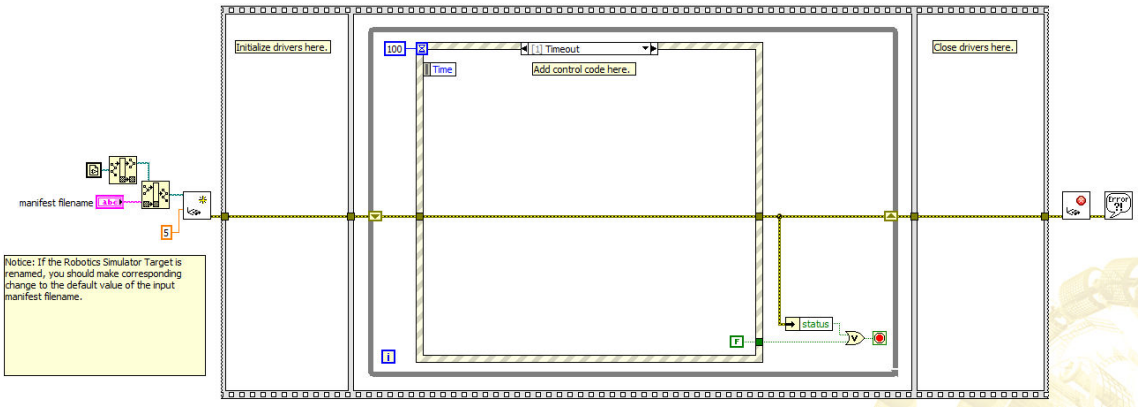
Se clica en Save, y el Labview nos crea el proyecto del simulador y crea los siguientes archivos:



En el explorador de proyectos muestra el siguiente árbol:



El archivo "DiferencialLidar.vi" es el sistema para simular el robot, es una plantilla, en la que tenemos que añadir, los sistemas que controlaran la simulación. La plantilla contiene 3 grandes bloques: inicialización, bucle de control y cierre.



En el primer bloque se han de inicializar los sensores y motores del robot.

En el bloque del medio, se han de añadir los sistemas que controlan el robot, los sensores, los motores, etc. Aquí es donde, en cada iteración del bucle se tiene que:

Usar el modelo cinemático y aplicar las velocidades a cada motor.
Obtener la distancia y ángulo al obstáculo más cercano.
Obtener la distancia y ángulo al destino.
Introducir los valores de entrada en el controlador difuso y obtener los valores de salida: la velocidad lineal y angular.
Comprobar las condiciones de final del bucle.

En el último bloque se tienen que cerrar los sensores, motores y otros dispositivos que se han inicializado en el primer bloque.

Pasos que realiza el simulador:

Inicializa variables, distancias, ángulos, velocidades, etc.
Comprueba que los archivos necesarios para el simulador existen, el archivo "xml" del simulador y el archivo *fuzzy* del controlador.
Configura el servicio de simulación y lo inicia.
Comprueba que no hay error al iniciar el servicio de simulación.
Inicializa los motores y sensores del robot.
Obtiene la posición inicial del robot.
Hasta final de simulación o error:

Con el modelo cinemático aplica la velocidad a cada motor.
Obtiene la posición y orientación del robot.
Obtiene la distancia respecto a la posición anterior, y la suma a la distancia recorrida.
Obtiene la distancia del robot al destino.
Calcula el ángulo relativo entre el destino y el robot.
Obtiene la distancia y el ángulo del obstáculo más cercano:

El sensor LIDAR detecta los obstáculos de un barrido. Se obtiene la distancia y el ángulo del más cercano al robot.

Si el obstáculo más cercano está por detrás del destino, asigna 3 metros a la distancia al obstáculo.
Introduce las variables de entrada en el sistema difuso:

Distancia y ángulo al destino.
Distancia y ángulo al obstáculo.

Obtiene las variables de salida del sistema difuso:

Velocidad lineal y angular del robot.

Comprueba las condiciones de final de simulación:

Distancia sea menor a 5 cm.
Tiempo de simulación mayor o igual a 30 segundos.

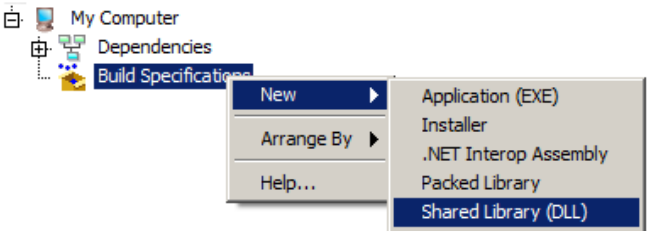
Finaliza la simulación.
Cierra los motores y sensores.
Cierra el servicio de simulación.
Actualiza los valores de salida del simulador: si ha llegado, si el tiempo se ha agotado, la distancia al destino, la distancia recorrida, si hay error al simular.

En el apéndice están los diagramas de bloques de los pasos anteriores.

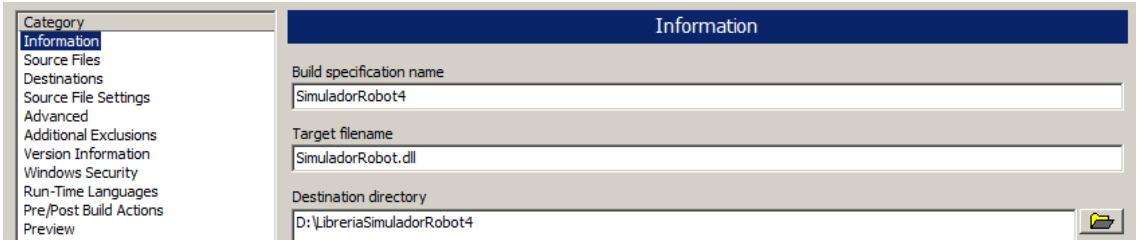
Creación de la librería tipo dll del simulador

Para que sea más fácil la modificación del simulador, se crea un sistema VI, que hará de puente entre el simulador y las llamadas a éste desde C. Así se puede modificar el simulador, y no se pierde la configuración de la función para crear la librería.

En el explorador del proyecto, vamos al árbol de archivos y seleccionamos la opción de crear la librería dll:



Sale un asistente que ayuda a crear la librería.
Hay que añadir información y seleccionar unas opciones.



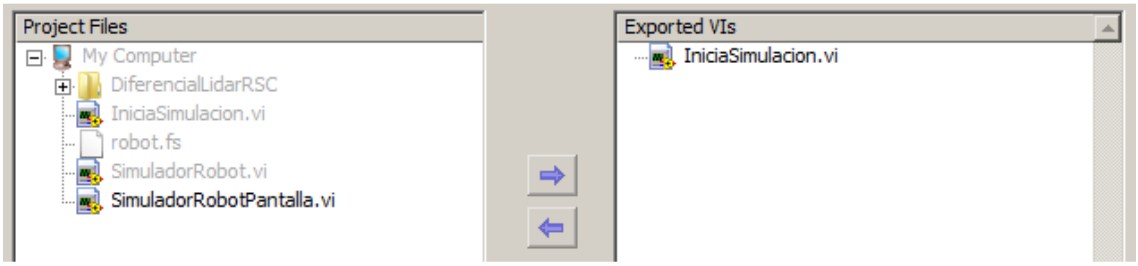
Los pasos principales para crear la librería son:

En Information:

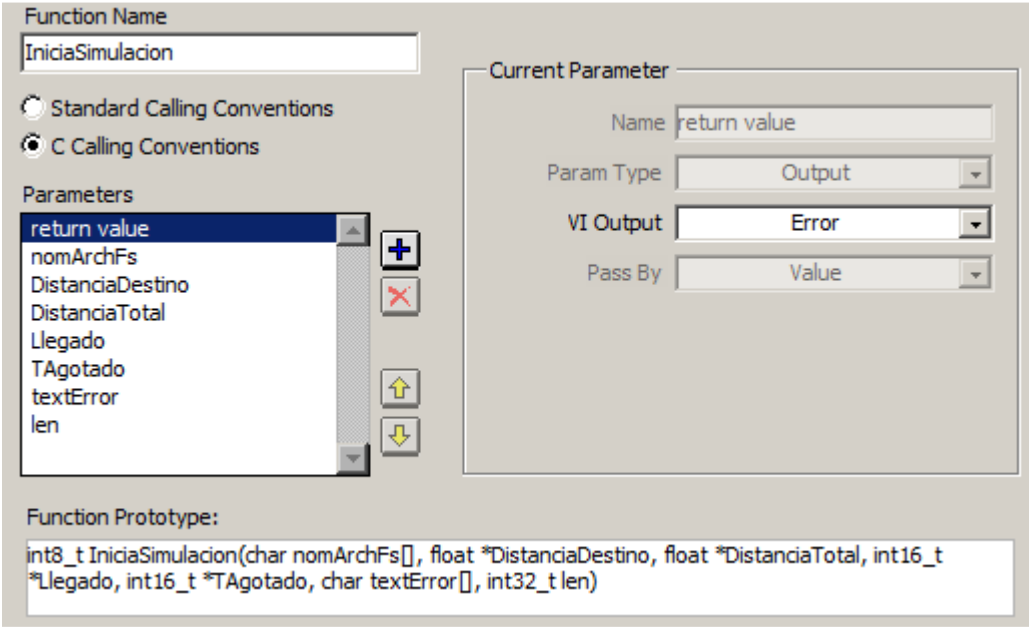
Se introduce el nombre de la creación (Build … name)
Se introduce el nombre de la librería (Traget filename)
Se escoge la carpeta donde guardará la librería (Destination …).

En Source Files:

Se añade a Exported VIs los archivos del proyecto.
Por cada VI añadido, crea una función en la librería.
En este caso añadimos el archivo "IniciaSimulación.vi"

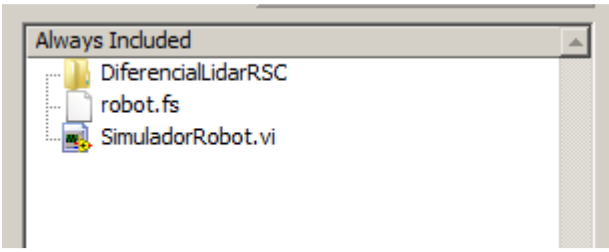


Muestra la ventana para configurar la función de la librería.



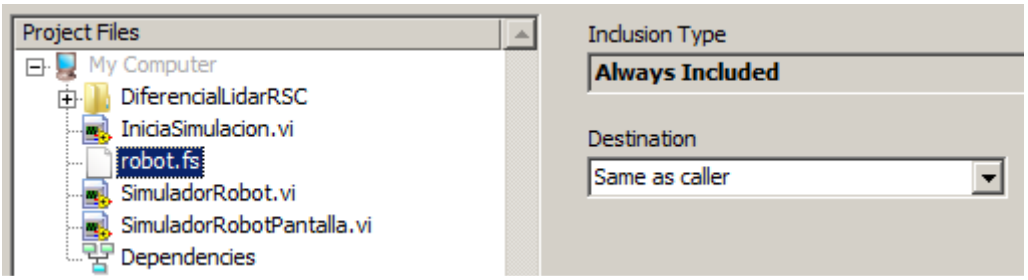
Seleccionamos las entradas, las salidas y su orden en la función.

En los archivos incluidos, añadimos: el sistema del simulador, la carpeta con los archivos "ive" y el archivo del controlador *fuzzy*.

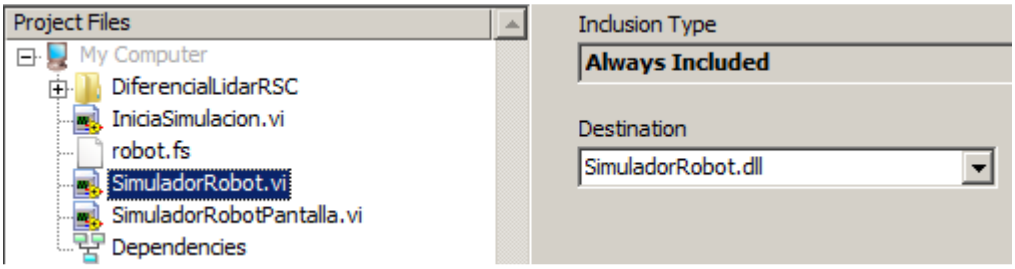


En Source File Settings:

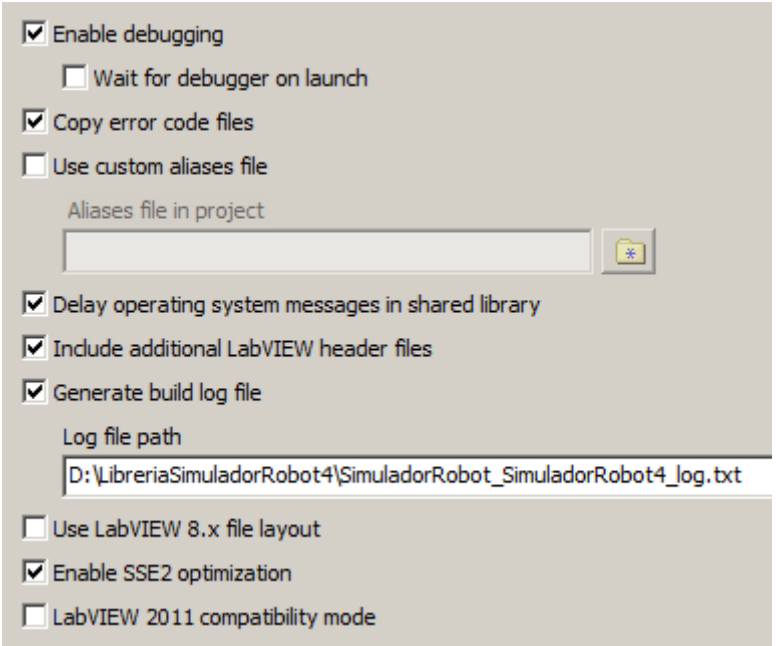
Se escoge donde se ha de guardar cada archivo en la librería.
Si no se tiene claro donde, es mejor dejar la opción de "same as caller", así los archivos que necesite la librería, habrá que colocarlos en la misma carpeta que el archivo dll.



En cambio, los sistemas VI que use la librería y los archivos de las dependencias, es mejor ponerlos dentro del mismo archivo dll.



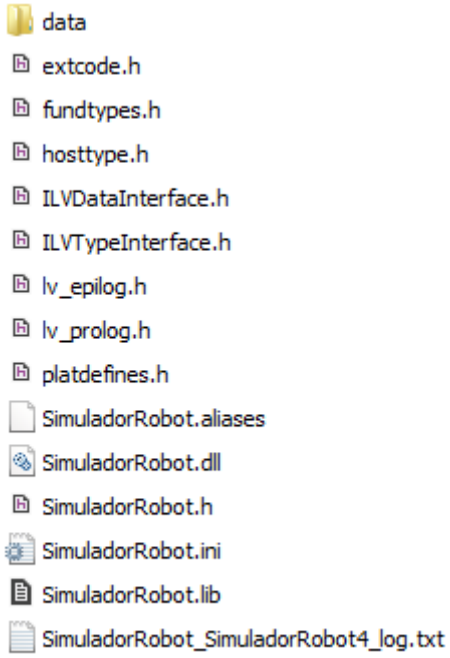
En advanced:



Se marca la opción de incluir additional Labview header files, para usar la librería en programas en C.
Se activa la optimización SSE2, siempre que la CPU las soporte.

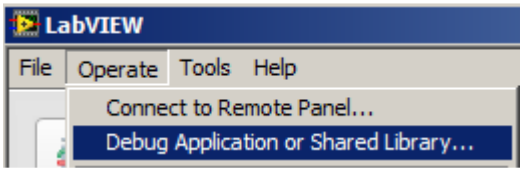
Después de esta configuración, creamos la librería.
En la carpeta especificada tiene que haber creado el archivo dll, una serie de cabeceras de C y los archivos de datos que necesita la librería para funcionar.

Contenido de la carpeta de la librería:



Si se intenta ejecutar una prueba de la librería, no funciona, ya que necesita otros archivos del Programa Labview. El listado de archivos adicionales se puede encontrar en el Capítulo 15.

Después de añadir los archivos del Labview, se realiza una depuración para solucionar los problemas que haya.
Creamos la librería con la opción "Wait for debugger on launch" que está en la pestaña Advanced de la configuración.
Se ejecuta nuestro programa, se va al Debugger del Labview y se observa que le ocurre.



Un error que impide el funcionamiento, es que al cambiar de carpeta la librería, y añadirla al directorio del programa desarrollado en C++, las rutas de los archivos de las piezas también cambian. Se tiene que retocar el archivo "XML" del simulador, que contiene las opciones de configuración del simulador.
En el "XML" se usan rutas relativas al directorio donde reside el dll, ya que en la configuración se escogió la opción de "Same as caller".

9.5 Descripción de métodos del Algoritmo Genético

En esta sección se describen:

Por una banda, los métodos del algoritmo genético que enlazan con las librerías, tanto para crear sistemas *fuzzy* con los datos de los cromosomas de un individuo, como para simular el robot con un sistema *fuzzy*.

Por la otra, los métodos de cruce y mutación que se usan, tanto a nivel de cromosomas como a nivel de individuos.

Apartados:

9.5.1 Explicación de los métodos que llaman a las librerías.

a) Método para crear un controlador difuso en un archivo
b) Método para realizar una simulación

9.5.2 Explicación de los métodos de cruce y mutación

a) Cruce a nivel de cromosoma
b) Cruce a nivel de individuo
c) Mutación a nivel de cromosoma
d) Mutación a nivel de Individuo

9.5.1 Explicación de los métodos que llaman a las librerías.

a) Método para crear un controlador difuso en un archivo:

La entrada es: un individuo, y el nombre del archivo del sistema fuzzy. Primero se obtienen las cadenas de los cromosomas del inidividuo, y se asignan, a cada variable de entrada de la funcion de la libreria.

```
bool Simulacion::creaControlRobot(Individuo &indi, char *nomfs){
    // crea un sistema fuzzy con los datos del individuo
    // guarda el sistema fuzzy en el archivo .fs
    bool ret=false;
    int8_t *resp=new int8_t();
    char *error=new char[500];
    int *lError=new int();
    float *aObst, *dObst, *aDesti, *dDesti, *velLin, *velAng;
    *lError=500;
    *resp=0;
    Cromosoma crom;
    crom=indi.getAObst();
    aObst=crom.getCadena();
    crom=indi.getDObst();
    dObst=crom.getCadena();
    crom=indi.getADesti();
    aDesti=crom.getCadena();
    crom=indi.getDDesti();
    dDesti=crom.getCadena();
    crom=indi.getVelLin();
    velLin=crom.getCadena();
    crom=indi.getVelAng();
    velAng=crom.getCadena();
```

Llama a la funcion de la librería con los parametros de entrada:

```
CreaSistemaFuzzy(nomfs, aDesti, LONCA, dDesti, LONCB, aObst, LONCC,
    dObst, LONCD, velLin, LONCE, velAng, LONCF, error, lError, resp);
```

Si hay error, **resp** será menor a 0, así que muestra el texto del error y devuelve “true”:

```
if (*resp<0){
    for (int i=0; i<*lError; i++){
        cout<<error[i];
    }
    cout<<endl;
    ret=true;
}
else{
    ret=false;
}
return ret;
```

b) Método para realizar una simulación:

Antes de usar este método, se ha de crear el archivo que contiene el sistema *fuzzy*, con el método anterior, si no la llamada a la librería del simulador dará error.

Se crean las variables para obtener los valores devueltos por el simulador, y se llama a la función de la librería "IniciaSimulacion()".

```
bool Simulacion::simula(char *nomfs){
    // simula el comprotamiento del robot con el sistema fuzzy
    // el simulador le devuelve los valores de
    // distancia al destino, distancia recorrida, si ha llegado, si ha habido algun error.
    bool ret=false;
    int8_t resp=0;
    float *daldest=new float();
    float *drec=new float();
    int16_t *llega=new int16_t();
    int16_t *agota=new int16_t();
    char *error=new char[500];
    int lError=500;
    *daldest=DISTMAX;
    *drec=0.0f;
    *llega=0;
    *agota=0;
    // ya tenemos creado el control fuzzy para el robot
    // llamamos a la libreria del simulador para que simule el robot
    resp=IniciaSimulacion(nomfs, daldest, drec, llega, agota, error, lError);
    if (resp<0){    // si hay error durante la simulacion
        // intentamos simularlo una segunda vez, ya que a veces sale un error del proxycaller
        resp=IniciaSimulacion(nomfs, daldest, drec, llega, agota, error, lError);
        if (resp<0){
            ret=true;
        }
    }
    if (ret==false){ // si no hay error la simulacion se ha realizado
        distAlDest=*daldest;
        distRec=*drec;
        if (*llega==1){ // si el robot llega a su destino
            llegado=true;
        }
    }
    return ret;
}
```

En algunos casos, el Labview no es capaz de inicializar el servicio de simulación y genera un erro del "proxycaller". Por eso, si se da ese error, se intenta llamar otra vez al servicio de simulación.

Si el simulador ha terminado bien, actualiza los datos de la Simulacion y devuelve False. En caso de error devuelve True.

9.5.2 Explicación de los métodos de cruce y mutación

a) Cruce a nivel de cromosoma:

Primero mira si el cromosoma de la pareja y del hijo es del mismo tipo:

```
bool Cromosoma::cruzar(Cromosoma &pareja, Cromosoma &hijo){
    // cruza con un cromosoma del mismo tipo y crea un cromosoma hijo valido
    // el cromosoma actual esta inicializado, la pareja tambien
    // la pareja y el hijo han de ser del mismo tipo
    bool error=false;
    int sig, i;
    float r,pres,valmod;
    // miramos si son del mismo tipo
    if((pareja.tipo==tipo) && (hijo.tipo==tipo)){
```

Según el tipo, se procede al cruce de cada gen del cromosoma:

Selecciona un signo aleatorio.
Calcula la diferencia entre genes.
Obtiene la presión del salto, en función de la cercanía entre genes.
Calcula el valor del gen del hijo.

```
if (tipo=='A'){
    i=0; error=false;
    while((i<lon) && !error){
        // seleccionamos el signo del salto
        sig=getSignoAleatorio();
        // calculamos el salto maximo
        r=cadena[i]-pareja.cadena[i];
        if (r<0.0f) r=r*-1;
        // obtenemos la presion
        if (r>=2.0f*CERA){   // si los valores de los padres estan muy lejos
            pres=0.025f;            // el salto es un 2.5 %
        }
        else if(r<2.0f*CERA && r>=CERA){ // si los valores de los padres estan lejos
            pres=0.05f;                  // el salto es 5 %
        }
        else{ // si los valores de los padres estan cerca
            pres=0.10f;                     // el salto es 10 %
        }
        // ahora que ya tenemos el padre, el signo, el salto maximo y la presion
        // calculamos el valor del gen del hijo
        valmod=sig*r*pres;
        hijo.cadena[i]=cadena[i]+valmod;
```

Aquí se observa, como al final para obtener una tasa de invalidez menor el cruce de cromosomas, se han reducido la presión de salto propuesta en el análisis (50%, 38%, 25 %) por los valores (10%, 5%, 2,5%).

Si el gen del hijo que se ha calculado, invalida el cromosoma, se intenta cruzar cambiando de sentido el salto:

```
if (! hijo.esValidaPosACF(i)){ // si el hijo no es valido en esa posicion
    hijo.cadena[i]=cadena[i]-valmod;  // probamos a cambiar el signo
}
if (! hijo.esValidaPosACF(i)){  // si el hijo no es valido en esa posicion
    hijo.cadena[i]=cadena[i];   // dejamos el valor que tenia el padre
}
if(hijo.esValidaPosACF(i)){  // si es valido para esa posicion
     i++;                    // continuamos al siguiente
}
else{ // si el valor no es valido paramos, el hijo es invalido
    error=true;
}
```

El código anterior sólo muestra el caso de los cromosomas tipo A. Para el resto de cromosomas, de otros tipos, se realiza de forma similar, usando las funciones y las constantes adecuadas a su tipo.

Al final, se trata el caso donde el tipo del cromosoma actual es erróneo, y el caso en que la pareja o el hijo no son del mismo tipo:

```
        }
        else{ // si el tipo no es A,B,C,D,E,F el cromosoma es erroneo
            error=true;
        }
    }
    else{ // si la pareja y el hijo no son del mismo tipo hay error
        error=true;
    }
    return error;
}
```

Si hay error al realizar el cruce devuelve true, en caso contrario devuelve false.

b) Cruce a nivel de individuo:

Cada cromosoma del hijo, surge del cruce de los cromosomas del individuo actual con los cromosomas de la pareja.

```
bool Individuo::cruzar(Individuo &pind, Individuo &hind){
    // cruzamos con pind y creamos el hijo hind
    bool error,resp1,resp2,resp3,resp4,resp5,resp6;
    error=false;
    resp1=aDesti->cruzar(*pind.aDesti, *hind.aDesti);
    resp2=dDesti->cruzar(*pind.dDesti, *hind.dDesti);
    resp3=aObst->cruzar(*pind.aObst, *hind.aObst);
    resp4=dObst->cruzar(*pind.dObst, *hind.dObst);
    resp5=velLin->cruzar(*pind.velLin, *hind.velLin);
    resp6=velAng->cruzar(*pind.velAng, *hind.velAng);
    error=(resp1 || resp2 || resp3 || resp4 || resp5 || resp6);
    return error;
}
```

Si hay un error al cruzar uno de los cromosomas, devuelve true, en caso contrario devuelve false.

c) Mutación a nivel de cromosoma:

Primero intenta una mutación aleatoria, tanto en la posición como en el sentido del salto, sumando o restando una cantidad, al valor de un gen. Si la mutación crea un cromosoma inválido, intenta mutarlo en sentido contrario.

```
bool Cromosoma::mutar(){
    // muta el cromosoma en una posicion que no sea la primera o la ultima
    // si el cromosoma no ha sido modificado devuelve true indicando error
    // si ha sido modificado y es valido devuelve false, si no devuelve true
    bool resp=true;
    int pos, senSalt;
    // mutamos de forma laeatori y obtenemos la posicion y sentido
    resp=mutar(pos,senSalt);
    if(resp){ // si es invalida intentamos mutar en sentido contrario
        resp=mutarPos(pos,-senSalt);
    }
    return resp;
}
```

Mutación Aleatoria:

```
bool Cromosoma::mutar(int &pos1, int &senSalt1){
    // muta el cromosoma en una posicion que no sea la primera o la ultima
    // si el cromosoma no ha sido modificado devuelve true indicando error
    // si ha sido modificado y es valido devuelve false, si no devuelve true
    // devuelve la posicion y el sentido del salto aplicado en la mutacion
    bool error=false;
    bool resp=true;
    int pos=0;
    int sig=1;
    float r,p,salto;
    r=0.0f; p=0.0f; salto=0.0f;
    pos=0;
    // obtenemos una posicion diferente a los extremos
    while(pos==0 || pos==(lon-1)){
        pos=getPosAleatoria(lon-1);
    }
    // pos no es ni la primera ni la ultima de la cadena,
    // ni igual a la ultimna posicion mutada
    sig=getSignoAleatorio(); // obtenemos el signo aleatorio
    if (sig==-1){ // si es negativo mutamos con el gen anterior
        r=(cadena[pos]-cadena[pos-1]);
    }
    else if(sig==1){ // si es positivo mutamos con el gen posterior
        r=(cadena[pos]-cadena[pos+1]);
    }
    if (r<0){ // si la diferencia es negativa, hacemos el valor absoluto
        r=(r*(-1));
    }
    p=0.05f; // el salto es del 5% de la diferencia entre valores
    salto=sig*r*p;
    cadena[pos]=cadena[pos] + salto;
```

Actualiza los valores de salida:

```
    // devolvemos los valores de la posicion y del signo del salto
    pos1=pos;
    senSalt1=sig;
```

Comprueba a partir de la posición mutada si sigue siendo un cromosoma válido.

```
    if (tipo=='A' || tipo=='C' || tipo=='F'){
        resp=true; error=false;
        while(resp==true && pos<lon){
            resp=esValidaPosACF(pos);
            if (resp==true){
                pos++;
            }
        }
        if (resp==false){
            error=true;
        }
    }
    else if(tipo=='B' || tipo=='D'){
        resp=true; error=false;
        while(resp==true && pos<lon){
            resp=esValidaPosBD(pos);
            if (resp==true){
                pos++;
            }
        }
        if (resp==false){
            error=true;
        }
    }
    else if(tipo=='E'){
        resp=true; error=false;
        while(resp==true && pos<lon){
            resp=esValidaPosE(pos);
            if (resp==true){
                pos++;
            }
        }
        if (resp==false){
            error=true;
        }
    }
    else{ // tipo incorrecto
        error=true;
    }
    return error;
}
```

Mutación específica:

```
bool Cromosoma::mutarPos(int pos1, int senSalt1){
    // muta el cromosoma en la posicion y el sentido indicado.
    // siempre que la posicion no sea la primera o la ultima
    // si ha sido modificado y es valido devuelve false, si no devuelve true
    bool error=false;
    bool resp=true;
    int pos=0;
    int sig=senSalt1;
    float r,p,salto;
    r=0.0f; p=0.0f; salto=0.0f;
    pos=pos1;
    if((pos>0) && (pos<lon-1)){
        // si la posicion esta entre los extremos
        if (sig==-1){ // si es negativo mutamos con el gen anterior
            r=(cadena[pos]-cadena[pos-1]);
        }
        else if(sig==1){ // si es positivo mutamos con el gen posterior
            r=(cadena[pos]-cadena[pos+1]);
        }
        if (r<0){ // si la diferencia es negativa, hacemos el valor absoluto
            r=(r*(-1));
        }
        p=0.05f; // el salto es del 5% de la diferencia entre valores
        salto=sig*r*p;
        cadena[pos]=cadena[pos] + salto;
```

Al igual que la anterior, comprueba a partir de la posición mutada si sigue siendo un cromosoma válido.

```
if (tipo=='A' || tipo=='C' || tipo=='F'){
    resp=true; error=false;
    while(resp==true && pos<lon){
        resp=esValidaPosACF(pos);
        if (resp==true){
            pos++;
        }
    }
    if (resp==false){
        error=true;
    }
}
else if(tipo=='B' || tipo=='D'){
    resp=true; error=false;
    while(resp==true && pos<lon){
        resp=esValidaPosBD(pos);
        if (resp==true){
            pos++;
        }
    }
    if (resp==false){
        error=true;
    }
}
```

```
        else if(tipo=='E'){
            resp=true; error=false;
            while(resp==true && pos<lon){
                resp=esValidaPosE(pos);
                if (resp==true){
                    pos++;
                }
            }
            if (resp==false){
                error=true;
            }
        }
        else{ // tipo incorrecto
            error=true;
        }
```

Trata el caso que la posición que se quiere mutar no es correcta.

```
    }
    else{ // la poscion no esta entre los extremos excluidos
        error=true;
        cout<<"Las posicion de mutacion no es correcta"<<endl;
    }
    return error;
}
```

d) Mutación a nivel de Individuo:

Copia los cromosomas del padre, selecciona un cromosoma y lo muta.

```
bool Individuo::mutar(Individuo &hij1){
    // intenta crear un individuo por mutacion partiendo como base el individuo actua
    // si la mutacion no ha ido bien devuelve true
    bool resp=true;
    // inicializamos los cromosomas del hijo igual a los del padre
    *hij1.aDesti=*aDesti;
    *hij1.dDesti=*dDesti;
    *hij1.aObst=*aObst;
    *hij1.dObst=*dObst;
    *hij1.velLin=*velLin;
    *hij1.velAng=*velAng;
    // obtenemos un valor aleatorio entre 0 y 599
    int num;
    num=getPosAleatoria(599);
    // seleccionamos un cromosoma
    if (num>=0 && num<=99){
        //*****************************************************//
        // mutacion del primer hijo en el primer cromosoma
        *hij1.aDesti=*aDesti;
        resp=hij1.aDesti->mutar(); // mutamos el primer cromosoma
    }
    else if(num>=100 && num<=199){
        //*****************************************************//
        // mutacion del primer hijo en el segundo cromosoma
        *hij1.dDesti=*dDesti;
        resp=hij1.dDesti->mutar();
    }
    else if(num>=200 && num<=299){
        //*****************************************************//
        // mutacion del primer hijo en el tercer cromosoma
        *hij1.aObst=*aObst;
        resp=hij1.aObst->mutar();
    }
    else if(num>=300 && num<=399){
        //*****************************************************//
        // mutacion del primer hijo en el cuarto cromosoma
        *hij1.dObst=*dObst;
        resp=hij1.dObst->mutar();
    }
    else if(num>=400 && num<=499){
        //*****************************************************//
        // mutacion del primer hijo en el quinto cromosoma
        *hij1.velLin=*velLin;
        resp=hij1.velLin->mutar();
    }
    else if(num>=500 && num<=599){
        //*****************************************************//
        // mutacion del primer hijo en el sexto cromosoma
        *hij1.velAng=*velAng;
        resp=hij1.velAng->mutar();
    }
    // si todo va bien hemos mutado solo un cromosoma en un gen, así que resp=false
    // si no hemos podido mutar resp=true
    return resp;
}
```

9.6 Descripción de las pruebas:

9.6.1 Pruebas parciales

Se realizado pruebas parciales de cada clase, siguiendo una estrategia down-top, para asegurar que los métodos funcionan correctamente.

Las pruebas más importantes, han sido las realizadas a los métodos de cruce y mutación, tanto de cromosomas, como de individuos. Éstas han permitido:

Por un lado, observar si las estrategias de adaptación, de la presión y signo del salto, conseguían reducir la invalidez de los hijos generados de forma aleatoria, sin esas técnicas.

Por el otro, reducir la presión del salto en el cruce y la mutación, para conseguir unos niveles aceptables de individuos inválidos, por debajo del 50%.

9.6.2 Pruebas del algoritmo principal

Para comprobar el funcionamiento del algoritmo principal, y corregir sus errores, se usó una población de 20 individuos con un máximo de 16 generaciones de evolución y de estancamiento.

Luego se intentó obtener una solución con una población de 96 participantes y un máximo de 96 generaciones, pero el algoritmo terminó a causa del estancamiento sin obtener una solución.

Para conseguir una solución, cada vez que el algoritmo termina y no encuentra una solución, se sigue esta estrategia:

Se incrementa la población en 96 participantes.
Se usa como límite de estancamiento el valor devuelto por la siguiente ecuación:

$$\mathrm{LEstan} = \left(\frac{\frac{N}{L}}{\frac{N}{L} + 1} \right) N = \frac{N^2}{N + L}$$

En donde:

N es el número de individuos de la población.
L es el número de genes de un individuo.

Se usa como límite de generaciones el doble del límite de estancamiento.

La tabla de generaciones totales y estancadas con l=96 es:

N	Generaciones Estancadas	Generaciones Totales
96	48	96
192	128	256
288	216	432
384	307	614
…	…	…

Se ha escogido un incremento de 96, por ser el número de valores que se necesita encontrar para las funciones de pertenencia.

Se ha escogido esa función para el límite de estancamiento, ya que:

> Para la población inicial de 96 tenemos un límite de estancamiento de la mitad, 48.
>
> Para un población muy grande de N individuos, tendríamos un límite de estancamiento que tiende a N generaciones.

Se escoge el total de generaciones como el doble de estancamiento, para que en poblaciones grandes como mucho se hagan 2N generaciones.

Como al crear una población inicial se tarda bastante, se ha creado un programa que permite generarlas en un archivo. De esta forma, mientras en un ordenador estamos evolucionando la población actual, en otro se puede crear la siguiente población a usar.

Capítulo 10

Resultados

10.1 Resultados de la adaptación de la presión y el signo del salto.

10.1.1 Pruebas a nivel de cromosomas:

a) Cruce de cromosomas:

La prueba de cruce, por cada tipo de cromosoma, realiza un bucle de 1000 iteraciones que:

Inicializa aleatoriamente dos cromosomas.
Los cruza y genera un cromosoma hijo.
Si el cromosoma hijo es inválido lo añade a la cuenta.

Finaliza el bucle y calcula el % de cromosomas inválidos.

Sin la técnica de adaptación, con una presión de salto aleatoria entre el 0% y el 50%, y un signo aleatorio, se obtienen estos resultados:

Tipo	Invalidos (%)
A	98.6
B	83.8
C	97.2
D	82.8
E	93.3
F	98.4

Si bajamos la presión de salto aleatoria entre 0 y el 10%, los resultados de dos pruebas son:

Tipo	Invalidos (%)	Invalidos (%)
A	72	73.2
B	39.2	35.9
C	73.4	72.7
D	39.2	36
E	55.9	55.6
F	73.2	72.6

Estos resultados indican que hay una probabilidad muy alta de tener un cruce inválido en alguno de los cromosomas, y en consecuencia, que el cruce de individuos generará hijos inválidos de forma frecuente.

Con la técnica de adaptación del signo y de una presión de salto del 2.5% para los genes que están muy lejos, 5% para los genes que están lejos y del 10% para los genes que están cerca, los resultados de dos pruebas son:

Tipo	Invalidos (%)	Invalidos (%)
A	22.3	23.4
B	10.5	10.6
C	24.5	24.9
D	9.9	9.8
E	13.8	17.3
F	24.1	21.4

En cada tipo se ha conseguido reducir el porcentaje de inválidos.
La reducción de la tasa de inválidos ha sido de:

Tipo	Reducción (%)	Reducción (%)
A	69	68
B	73.2	70.4
C	66.6	65.7
D	74.7	72.7
E	75.3	68.8
F	67	70.5
Media	70.9	69.35

Los resultados muestran una reducción media alrededor del 70%, y en consecuencia, el cruce de individuos podrá crear más cantidad de hijos válidos.

b) Mutación de cromosomas:

La prueba de mutación por cada tipo de cromosoma realiza un bucle de 1000 iteraciones que:

Inicializa aleatoriamente un cromosoma.
Muta el cromosoma.
Si el cromosoma es inválido lo añade a la cuenta.

Finaliza el bucle y calcula el % de cromosomas inválidos.

Sin la técnica de adaptación, con una presión de salto fija del 5% y un signo aleatorio, los resultados de dos pruebas son:

Tipo	Invalidos (%)	Invalidos (%)
A	3.9	3.3
B	2.4	2.2
C	3.1	2.6
D	1.6	1.9
E	2.9	1.9
F	3.1	3.9

Con la adaptación del signo, los resultados de dos pruebas son:

Tipo	Invalidos (%)	Invalidos (%)
A	1	1.8
B	1.2	1.5
C	1.5	1.2
D	0.6	0.1
E	1.4	1.8
F	1.5	1.6

En cada tipo se ha conseguido reducir el porcentaje de inválidos. La reducción de la tasa de inválidos ha sido de:

Tipo	Reducción (%)	Reducción (%)
A	74.3	45.4
B	50	31.8
C	51.6	53.8
D	62.5	94.7
E	51.7	5.2
F	51.6	58.9
Media	56.95	48.3

Los resultados muestran una reducción media alrededor del 50%, y en consecuencia la mutación de individuos podrá crear más cantidad de hijos válidos.

10.1.2 Pruebas a nivel de Individuos:

a) Cruce de Individuos:

La prueba de cruce realiza un bucle de 50000 iteraciones que:

Inicializa aleatoriamente dos individuos.
Los cruza y genera un hijo con base a cada padre.
Si los hijos son inválidos los añade a la cuenta.

Finaliza el bucle y calcula el % de individuos inválidos.

Sin la técnica de adaptación, con una presión de salto aleatoria entre 0 y el 50% y un signo aleatorio, se obtiene una invalidez del 100%.

Si se baja la presión del salto a un valor entre 0 y 10%, entonces se obtiene una invalidez alrededor del 99,6%.

Estos resultados son debidos a que cada cromosoma tiene una probabilidad muy alta de ser inválido.

Con la técnica de adaptación del signo y la presión del salto, se obtienen mejores resultados.

En dos pruebas se obtienen las tasas del 44,1% y del 44,2%. La reducción en la invalidez es del 55%.
Así la técnica permite que un poco más de la mitad de los individuos creados por cruce sean válidos.

b) Mutación de Individuos:

La prueba de mutación realiza un bucle de 50000 iteraciones que:

Inicializa aleatoriamente un individuo.
Crea un hijo por mutación del individuo.
Si el individuo es inválido lo añade a la cuenta.

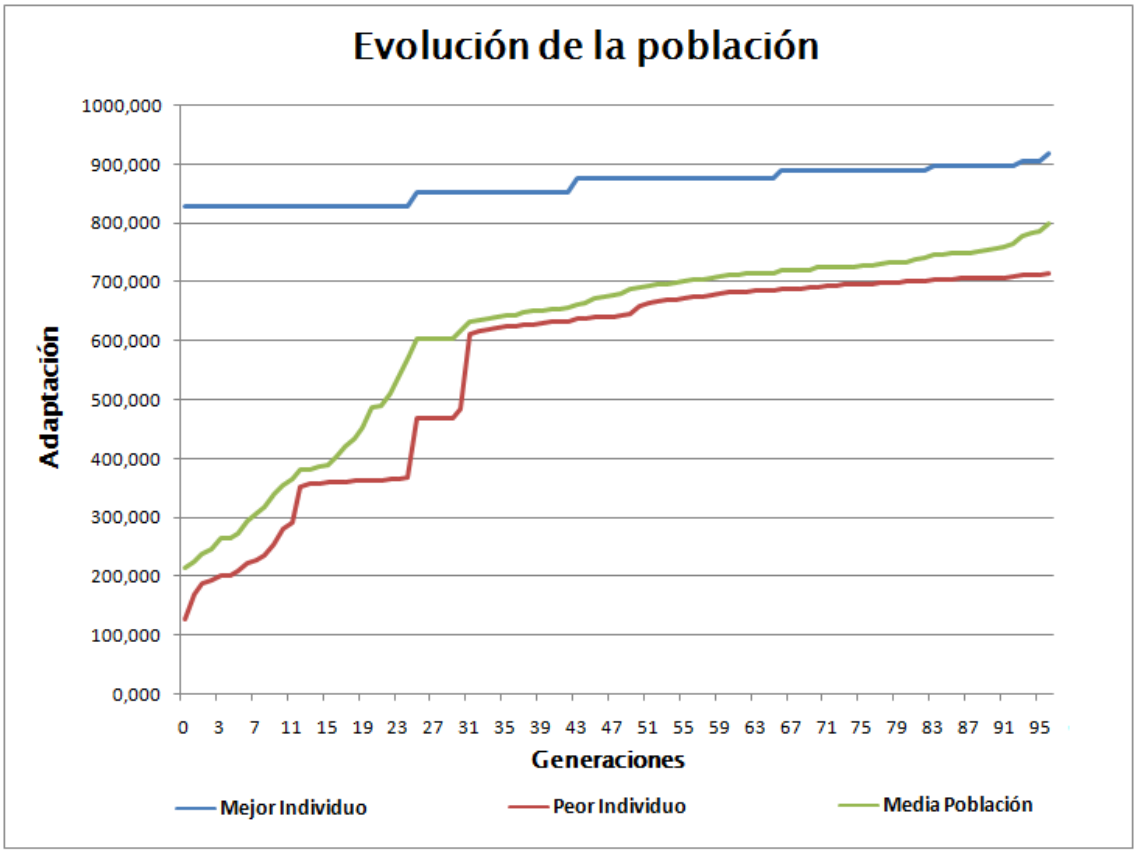
Finaliza el bucle y calcula el % de individuos inválidos.

Sin la adaptación del signo y una presión de salto fija del 5%, en tres pruebas se obtiene una invalidez del 3%, 3,02% y 2,98%. Así la mayoría de individuos creados por mutación serán válidos.

Con la adaptación del signo y una presión de salto fija del 5%, en dos pruebas se obtiene una invalidez 1.35%, 1.28%. Así casi todos los individuos creados por mutación serán válidos.

10.2 Resultados del Algoritmo Genético

Resultados de la evolución de una población de 96 participantes:



Interpretación de los resultados:

Aunque el algoritmo no ha sido capaz de encontrar un controlador, que lleve el robot al destino, se observa cómo va evolucionando la población, y los controladores son cada vez mejores.

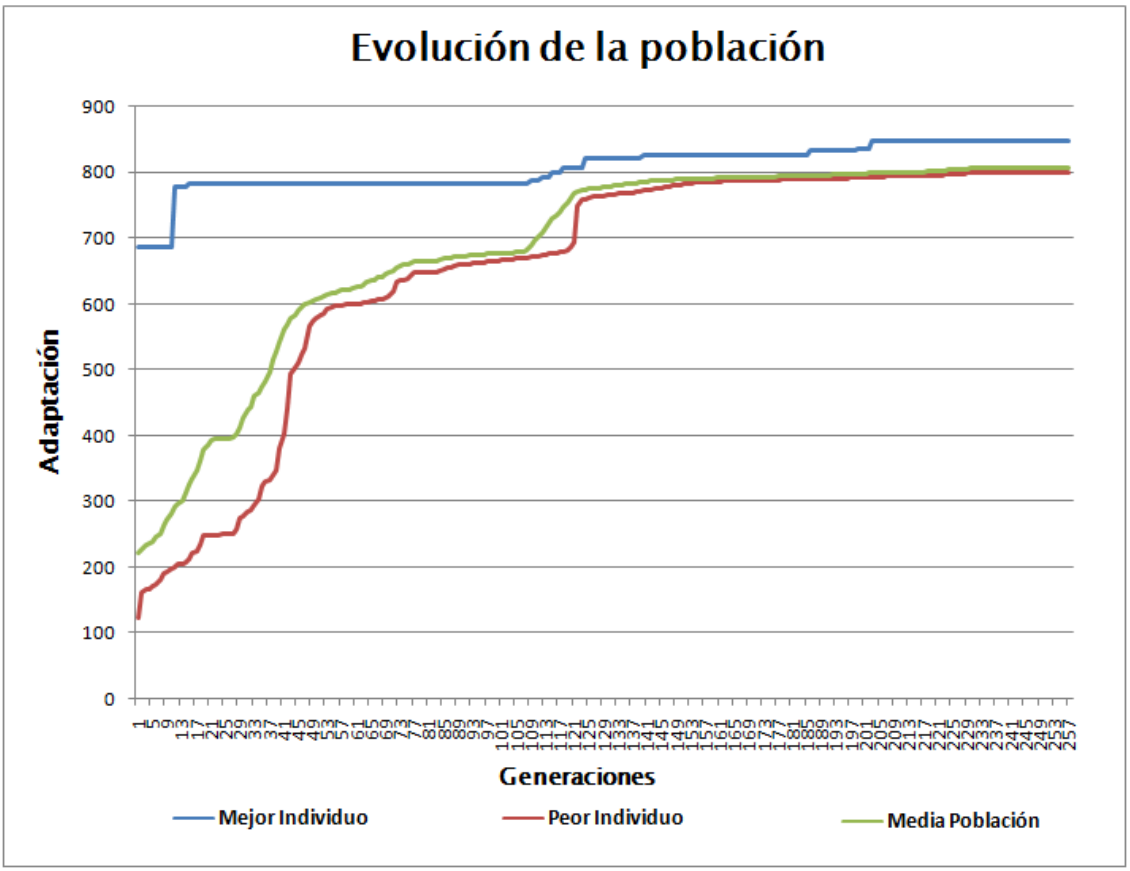
La ejecución ha finalizado por llegar al máximo de generaciones, pero sin llegar a estancarse:

El contador de generaciones es: 96
El contador de estancamiento es: 0
Los datos del mejor individuo son:

DistAlDest: 0.532697 metros.
DistRec: 7.34369 metros.
No ha llegado.
Evaluacion: 917.751

Aunque en las últimas generaciones, evoluciona más lentamente, parece que podría evolucionar un poco más, así que lo mejor sería parar el algoritmo cuando se estanca.

Resultados de la evolución de una población de 192 participantes:



Interpretación de los resultados:

Aunque el algoritmo no ha sido capaz de encontrar un controlador, que lleve el robot al destino, se observa cómo va evolucionando la población, y los controladores son cada vez mejores.

Su evolución ha sido menos rápida que la obtenida con 96 participantes, así que puede que se tenga que descartar la población, ya que su ejecución es peor. Aunque en este caso también se podría esperar a que la evolución se estancara para parar el algoritmo, no está tan claro que pueda evolucionar mucho más la población, como en el caso anterior.

La población ha acabado por estancarse:

El contador de generaciones es: 256
El contador de estancamiento es: 54
Los datos del mejor individuo son:

DistAlDest: 1.00636 metros.
DistRec: 6.0711 metros.
No ha llegado.
Evaluacion: 848.333

Capítulo 11

Conclusiones

Por un lado, se ha logrado crear un sistema difuso genético capaz de: aprender los valores de las funciones de pertenencia y obtener controladores cada vez mejores.

Por el otro, no se ha conseguido que el sistema obtenga un controlador que haga llegar el robot al destino. Las causas de esto pueden ser las siguientes:

Se ha dado poco tiempo de ejecución.
Se necesitan afinar los operadores de evolución, por ejemplo una tasa de invalidez menor en el cruce de individuos.
Se necesita crear más hijos en cada generación.
Se necesita una mayor población.
Se necesita realizar las simulaciones de forma más precisa.

Si se compara con otros sistemas difusos genéticos:

En vez de usar un simulador, usan un conjunto de datos de entreno, en dónde para unas entradas del sistema difuso, se sabe la salida adecuada.

En vez del error al destino, la función de evaluación de individuos, usa el error medio cuadrático de las salidas actuales, respecto a las de entreno.

Si se usa la técnica anterior con datos de entreno, se conseguiría evaluar mejor el controlador, ya que su evaluación no dependería del tiempo de la simulación, ni de su precisión. Pero, por el contrario, se tendrían que calcular esos valores de entreno, usando las trayectorias óptimas, y para cada posición en el plano, calcular las salidas que hacen avanzar el robot hacia el destino, esquivando los obstáculos de su alrededor.

Capítulo 12

Trabajo futuro

Desde el punto de vista económico, se podrían diseñar las piezas del robot y el entorno, con aplicaciones de código abierto como el Blender.
Se podría implementar el simulador, usando librerías de código abierto, como la Open Scene Grafics y la Open Dinamics Engine, que permiten configurar escenas 3D y simular el movimiento dinámico de elementos 3D.

Respecto a la implementación, se podría mejorar su rendimiento, si se programa para ejecutarse de forma paralela, como en el caso de un Paralel Virtual Machine, en donde cada PC pudiera simular y evaluar un individuo de la nueva generación. Así en cada generación se podrían crear tantos hijos como PCs tuviera el sistema PVM, y una generación solo tardaría en crearse el tiempo máximo de simulación, es decir 30 segundos.

Respecto a la adaptación del sistema a la evolución:

Por un lado, se podría estudiar la media de adaptación por generación, para tener en cuenta si el algoritmo evoluciona rápidamente o lentamente.
Por el otro se podría estudiar en que % son parecidos los individuos y detectar síntomas de parentesco.
Así con esta información, se podría crear un sistema difuso, que controlase y adaptara las capacidades de explotación y exploración del algoritmo genético, por ejemplo:

> Creando más hijos por cruce, si la población se adapta muy rápidamente y los individuos son muy diferentes.
>
> Creando más hijos por mutación si la población se adapta muy lentamente y los individuos son muy parecidos.

Capítulo 13

Bibliografía

[1] Siegwart,N., Nourbakhsh, R. (2004) Introduction to Autonomous Mobile Robots. Cambridge: The MIT Press

[2] Bräunl, T. (2006) EMBEDDED ROBOTICS: Mobile Robot Design and Applications with Embedded Systems. Berlin: Springer

[3] Siler, W., J.Buckley, J. (2005) Fuzzy Expert Systems and Fuzzy Reasoning. New Jersey: John Wiley & Sons, Inc.

[4] Mitchell. M. (1996) An Introduction to Genetic Algorithms. Massachusetts: The MIT Press

[5] Cordón, O., Herera, F. HoffMann, F., Magdalena, Luis. (2001) Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases. Singapore : World Scientific Publishing Co.

Webs:

Genetic and Evolutionary Algorithm Toolbox for use with Matlab
http://www.geatbx.com/docu/docutoc.html#TopOfPage

Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms
http://www.pg.gda.pl/~mkwies/dyd/geadocu/algindex.html

Manuals Labview
http://www.ni.com/manuals/esa/

National Instruments: How to Import a CAD Model from Solidworks to the Robotics Simulation Model Builder
http://www.ni.com/tutorial/14780/en/

Manual: Joint Types and Functions
http://ode-wiki.org/wiki/index.php?title=Manual:_Joint_Types_and_Functions

Introducción a Visual Studio
http://msdn.microsoft.com/es-es/library/ms165079.aspx

Videos Youtube:

Aibo simulation in LabVIEW
https://www.youtube.com/watch?v=c6Pvg59uryw

SolidWorks Aprendiendo SolidWorks 2011 desde cero
https://www.youtube.com/watch?v=9DN3wTZNLOg

Modelado de llanta para robot movil en Solidworks
https://www.youtube.com/watch?v=9xfgsufD_Tw

Design a Robot in SolidWorks
https://www.youtube.com/watch?v=PWsW00KMVFo

Comunicación Solidworks con Labview
https://www.youtube.com/watch?v=vEvE8Xigw-Y

LabVIEW Tutorial 42 - Building Shared Libraries (Enable Integration)
https://www.youtube.com/watch?v=70THpoCHFgo

Capítulo 14

Apéndice

14.1 Propiedades dinámicas de las uniones:

LoStop:

Posición mínima de la unión. Puede ser longitud o ángulo.
Si se establece en -infinito no hay posición mínima.
Para las uniones de revolución ha de ser superior a -pi.

HiStop:

Posición máxima de la unión. Puede ser longitud o ángulo.
Si se establece en infinito, no hay posición máxima.
Para las uniones de revolución, ha de ser inferior a pi.

Si LoStop es superior a HiStop, no son efectivas.

Fmax:

La fuerza máxima o torque máximo, que va usar el motor para alcanzar la velocidad deseada.
Tiene que ser superior o igual a 0. Si es 0 desactiva el motor.

FudgeFactor:

Cuando la unión está en una posición de final de recorrido, o de Stop, y el motor está intentando moverse pasando este límite, si se aplica una fuerza muy elevada causará un salto.

Este factor ajusta la propiedad de salto. Esta entre 0 y 1. Si se mueve dando muchos saltos, el factor es muy alto y hay que reducirlo para que no salte. Se pone a 0 si no queremos que sobrepase los finales de recorrido.

Bounce:

Indica si los finales de recorrido producen rebote o no.
0 no produce rebote y 1 rebotan el máximo.

CFM:

Rigidez o elasticidad de la unión, cuando no está en las posiciones de Stop o finales de recorrido, es decir, mientras recorre.

Stop ERP:

Parámetro de reducción de errores, usado en las posiciones de stop.

Stop CFM:

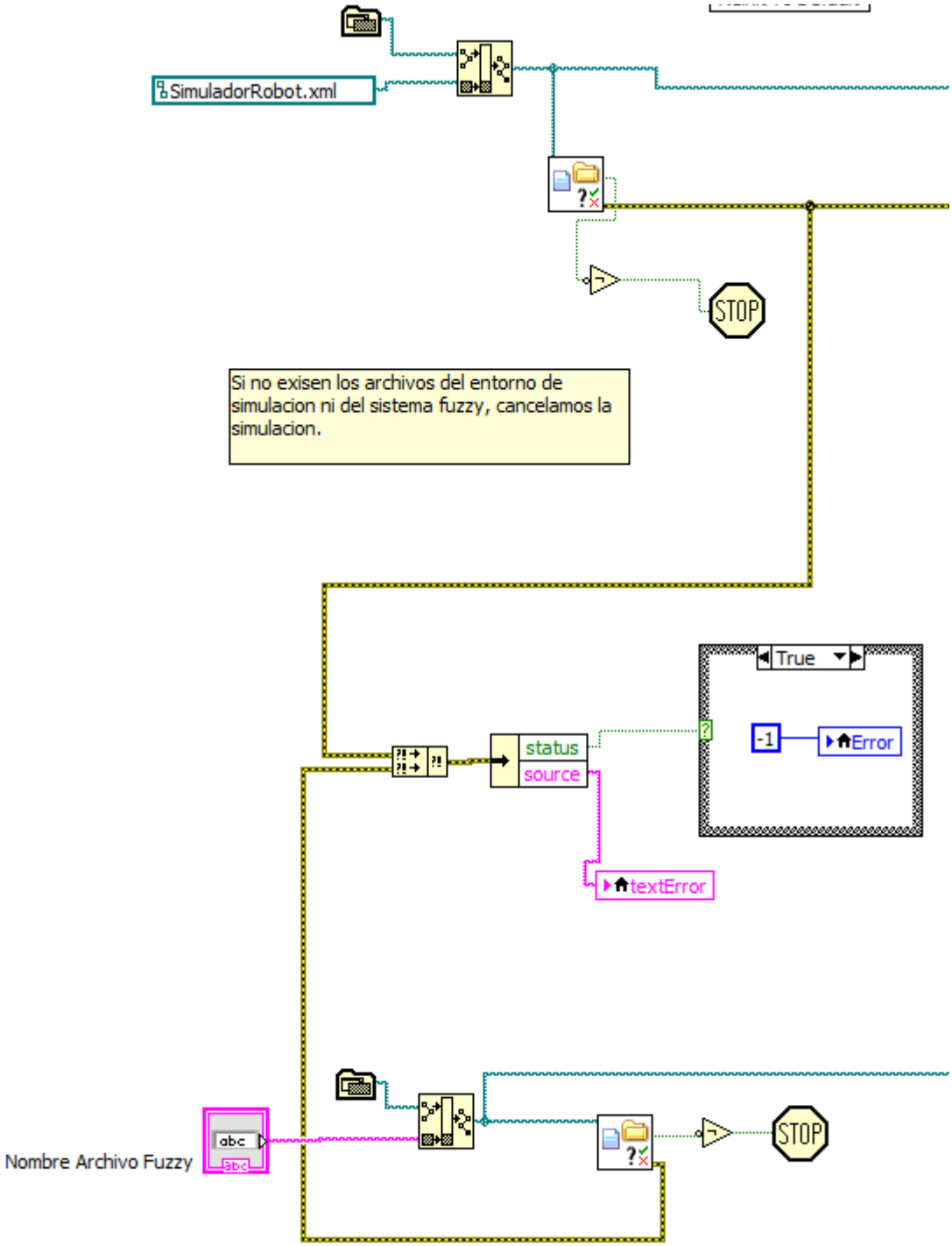
Rigidez o elasticidad de la posición de stop.
Se usa en las uniones no motorizadas, en las motorizadas no tiene el efecto deseado, ya que el motor llega a un final de recorrido o posición de stop.

Referéncias:

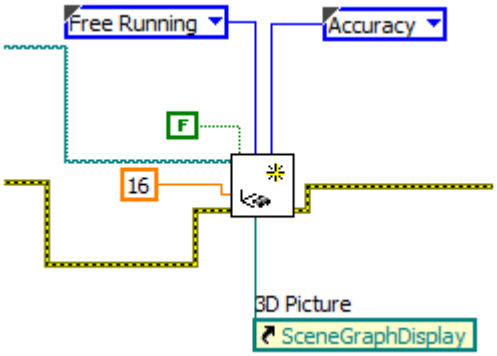
http://ode-wiki.org/wiki/index.php?title=Manual:_Concepts
http://ode-wiki.org/wiki/index.php?title=Manual:_Joint_Types_and_Functions

14.2 Capturas del diagrama del simulador

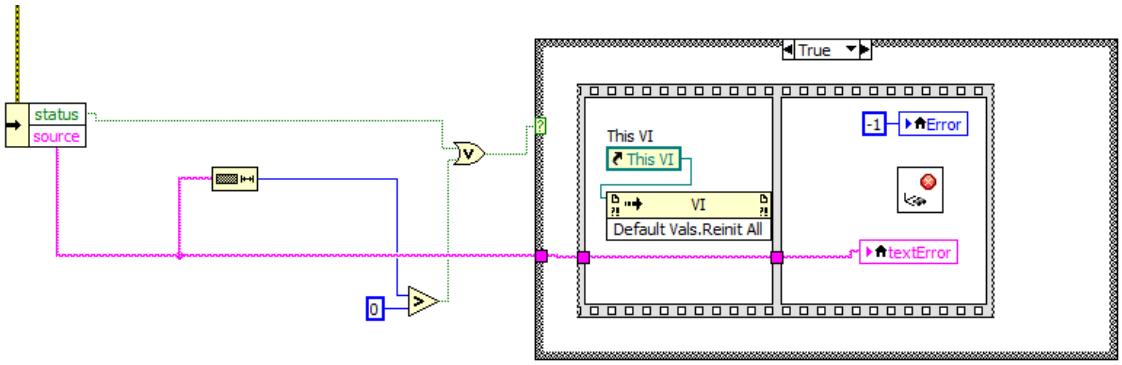
Comprobación de la existencia de los archivos XML y Fuzzy:



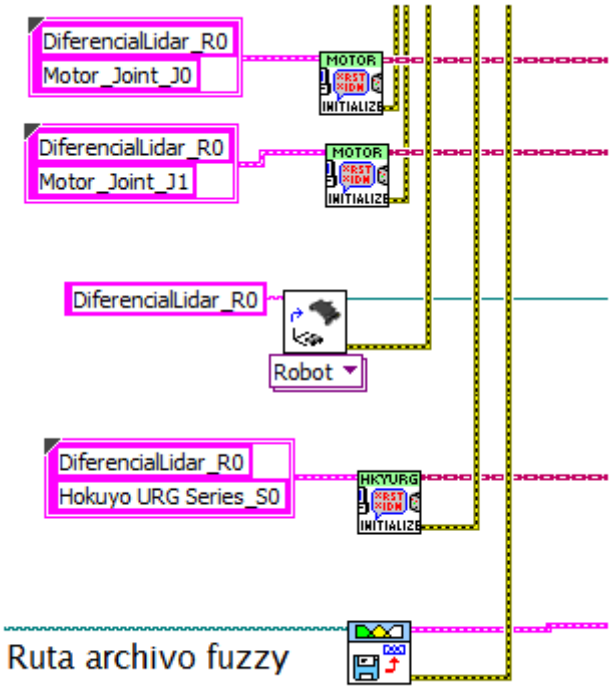
Configuración del servicio de simulación:



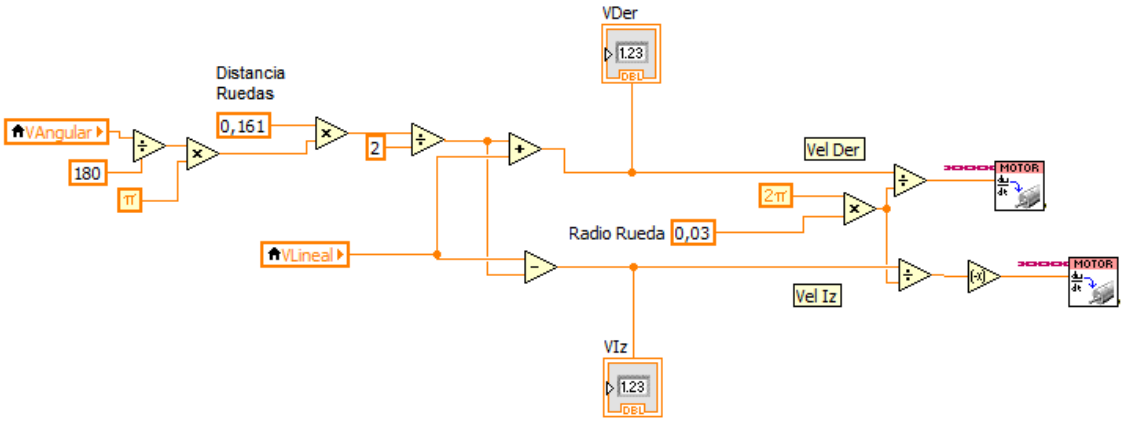
Comprobación del estado del servicio de simulación:



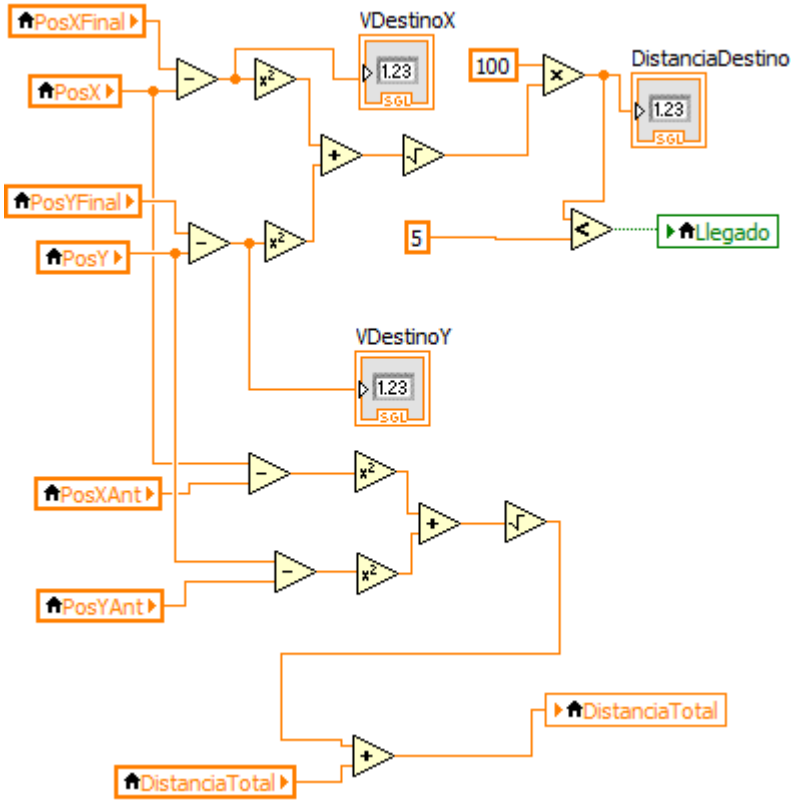
Inicialización del robot, motores, sensores y carga del sistema fuzzy:



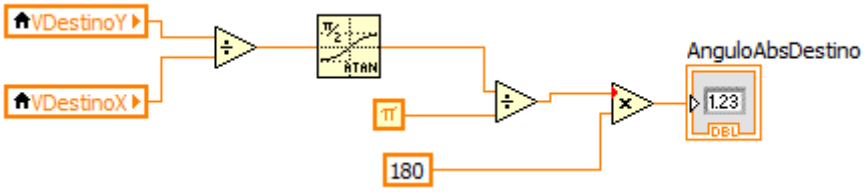
Modelo cinemático del robot y entrada de la velocidad en los motores:



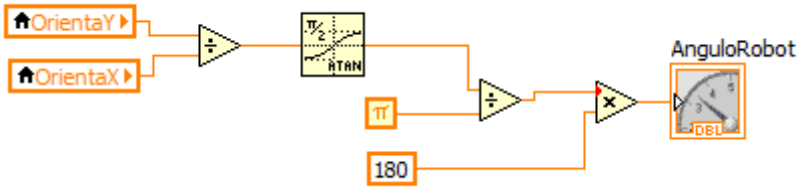
Cálculo del vector al destino y obtención de la distancia a través del módulo:



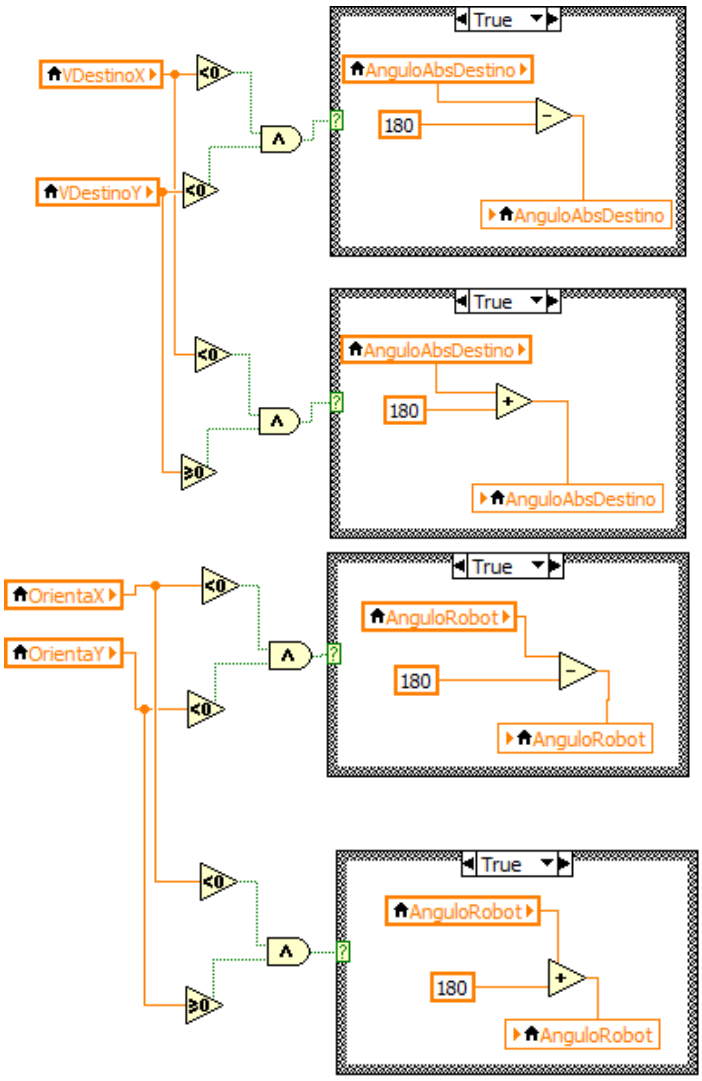
Cálculo del ángulo absoluto del vector al destino:



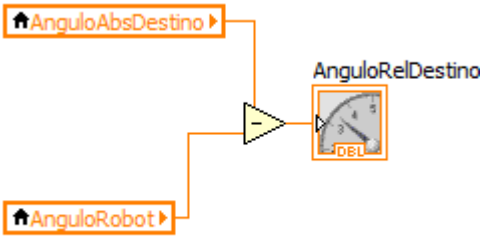
Cálculo del ángulo absoluto de la orientación del robot:



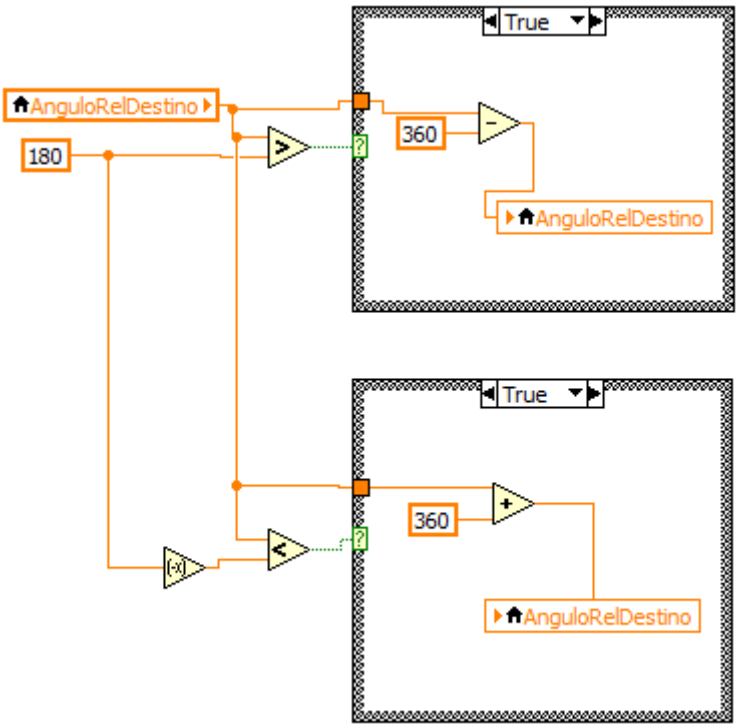
Rectificaciones de cuadrantes de la función atan:



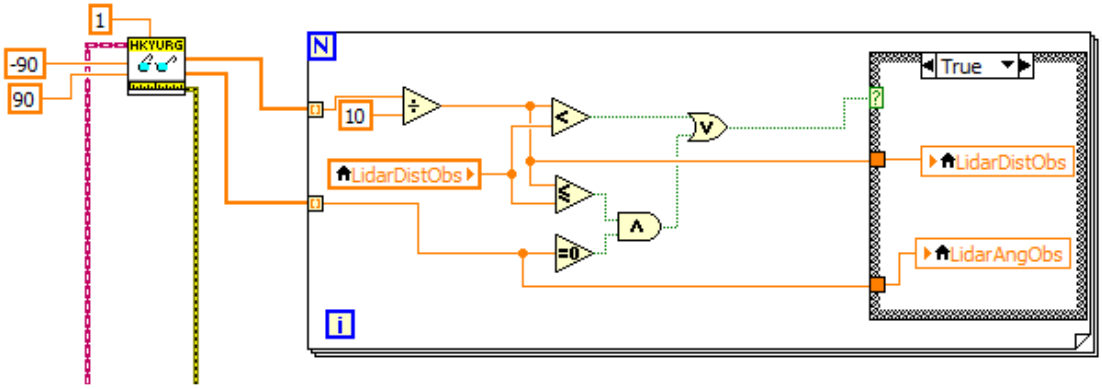
Cálculo del ángulo relativo al destino:



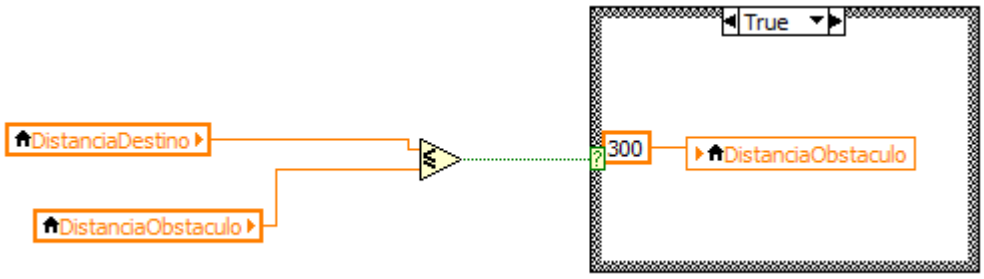
Corrección del ángulo dentro del rango [-180,+180]:



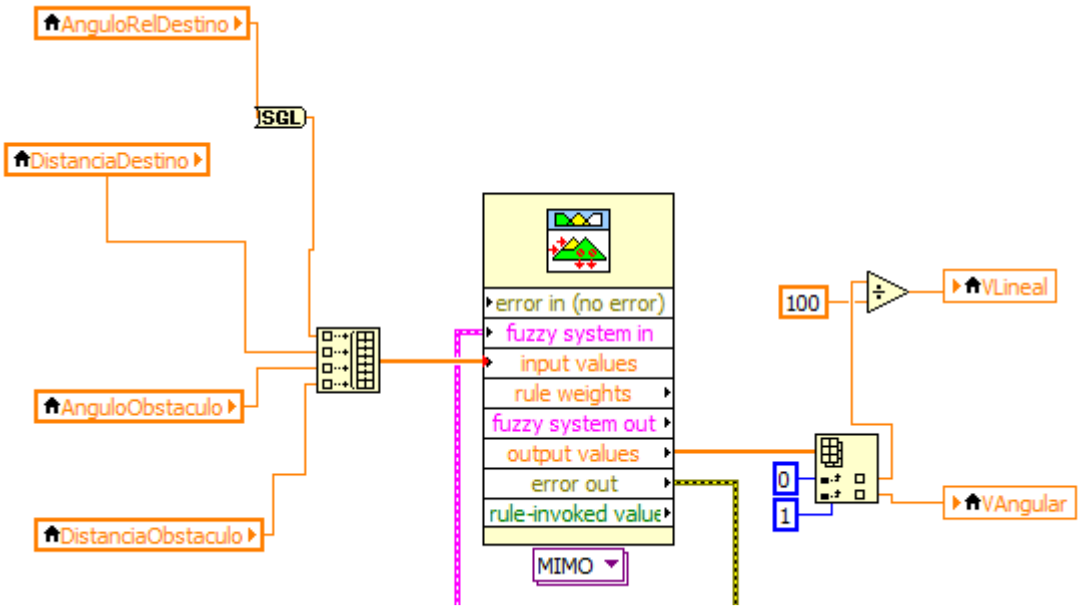
Barrido del sensor LIDAR y selección del obstáculo más cercano:



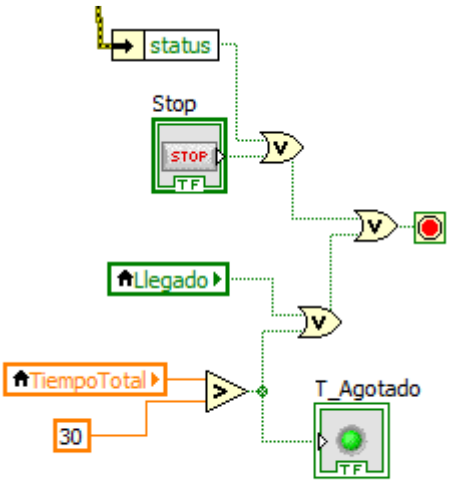
Rectificación de la distancia de un obstáculo por detrás del destino:



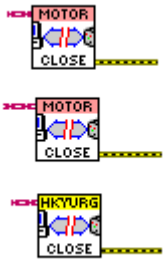
Aplicación del controlador fuzzy:



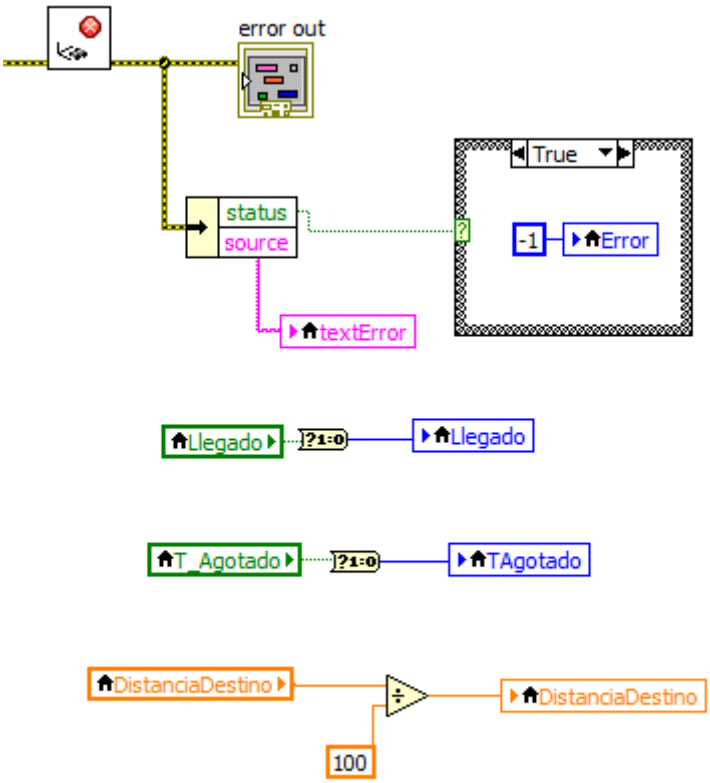
Comprobación de las condiciones de final de simulación:



Cierre de los motores y del sensor:



Cierre del servicio de simulación, y adaptación de algunos valores de salida del simulador:



Capítulo 15

Manual Instalación

En el CD se han incluido:

El proyecto del Labview para el generador de sistemas difusos.
El proyecto del Labview para el simulador del robot.
El proyecto del Visual Estudio con el código.
Los ejecutables y las librerías creados en el proyecto.

15.1 Instalación del algoritmo genético

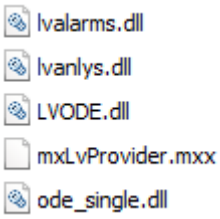
Para instalar el programa del algoritmo genético, se copia en el PC la carpeta TFC_MarioPelegrin\ProyectoC++\Ejecutables\AlgGenFuzzy

La carpeta contiene las librerías dll y los archivos ejecutables para la población de 96 y para la población de 192 individuos.
Antes de ejecutar cualquier archivo se tienen que añadir unos archivos del Labview que necesitan las librerías dll.

15.2 Archivos adicionales para la ejecución de las librerías:

Las librerías necesitan de algunos archivos del Labview para funcionar, que no se han añadido al proyecto, ya que tienen CopyRight de National Instruments. Además, también hay que tener instalado en el PC el Labview, ya que usan el Labview Runtime Engine del 2012.

En la carpeta donde tenemos los ejecutables y las librerías se tienen que añadir estos archivos:

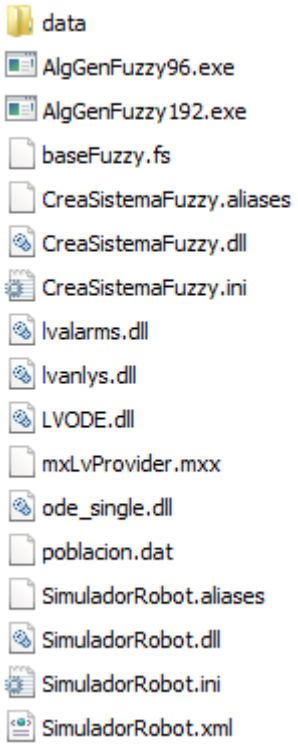


Las librerías lvalarms.dll y lvanlys.dll se pueden encontrar en:
C:\Program Files\National Instruments\Shared\LabVIEW Run-Time\2012

Las librerías LVODE.dll y ode_single.dll se pueden encontrar en:
C:\Program Files\National Instruments\LabVIEW 2012\resource

El archivo mxLvProvider.mxx se pude encontrar en:
C:\Program Files\National Instruments\LabVIEW 2012\resource\Framework\Providers

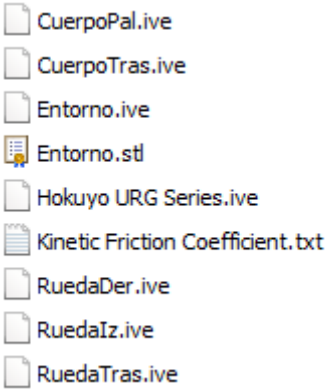
Se copian esos archivos en la carpeta del ejecutable, que ha de quedar así:



Ahora tenemos que añadir en la carpeta data el archivo Kinetic Friction Coefficient.txt que se puede encontrar en:

C:\Program Files\National Instruments\LabVIEW 2012\vi.lib\robotics\Simulator\Interface\Utility

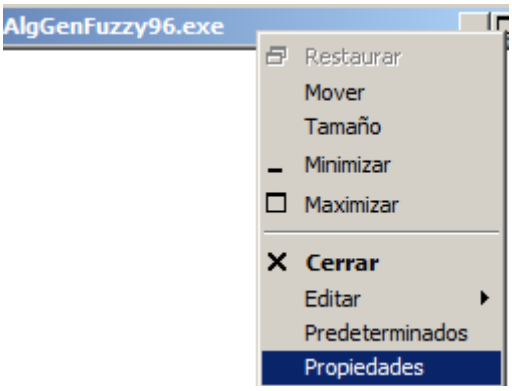
La carpeta data ha de contener:

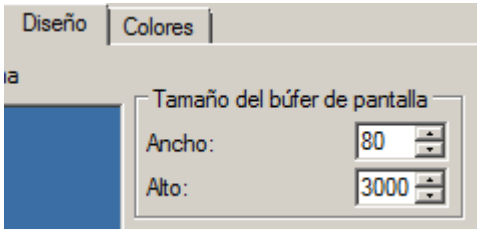


15.3 Manual de ejecución del algoritmo genético

Una vez tenemos instalado el programa, podemos ejecutar el archivo AlgGen96.exe, que evoluciona una población de 96 participantes.

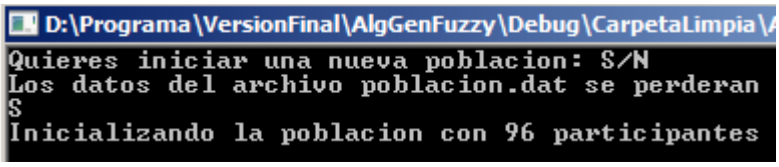
Nos pide una opción, pero antes se puede ir a las propiedades del ejecutable, clicando con el botón derecho en la barra azul, para aumentar el número de líneas de la consola, por ejemplo unas 3000.



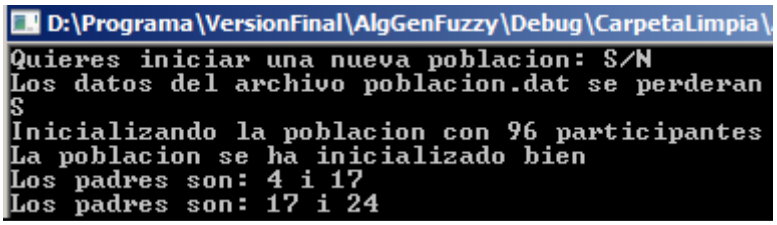


Ahora ya podemos indicarle una de las dos opciones: que inicie una nueva población o cargue la que se ha dejado en la carpeta, "poblacion.dat" con 96 individuos.

Si la opción es iniciar una población, muestra esta ventana:



Cuando acabe de inicializarla, mostrara esta pantalla y empezara a evolucionar la población:



Si por el contrario se pretende cargar la población de un archivo, entonces, mostrara esta pantalla:

```
D:\Programa\VersionFinal\AlgGenFuzzy\Debug\CarpetaLimpia\AlgGenFuzzy96.exe
Quieres iniciar una nueva poblacion: S/N
Los datos del archivo poblacion.dat se perderan
N
Cargando la poblacion del archivo poblacion.dat
Se han de cargar 96 participantes
Los contadores y la poblacion se han cargado
==============================================
El contador de generaciones es: 0
El contador de estancamiento es: 0
==============================================
Las evaluaciones son:
=====================
828.448 | 613.871 | 519.76 | 349.921 | 319.024 | 308.866 | 270.069 | 269.897 | 2
60.632 | 259.453 | 256.256 | 255.317 | 245.289 | 243.259 | 241.036 | 235.627 | 2
35.622 | 235.518 | 235.237 | 235.076 | 232.493 | 227.539 | 225.386 | 224.396 | 2
22.925 | 222.403 | 221.496 | 220.39 | 211.308 | 210.43 | 210.413 | 209.894 | 208
.873 | 208.758 | 207.998 | 207.637 | 207.483 | 202.664 | 202.228 | 201.775 | 200
.684 | 200.34 | 200.222 | 199.963 | 199.602 | 198.726 | 197.915 | 197.302 | 196.
75 | 196.587 | 194.931 | 194.517 | 194.082 | 193.375 | 193.005 | 192.976 | 191.8
94 | 191.877 | 190.277 | 189.884 | 189.544 | 189.265 | 189.001 | 188.384 | 188.1
41 | 187.875 | 187.541 | 185.612 | 185.541 | 185.047 | 180.449 | 177.855 | 177.4
22 | 176.48 | 175.268 | 173.819 | 173.428 | 172.568 | 171.442 | 170.02 | 168.833
 | 168.198 | 168.162 | 167.986 | 167.769 | 164.939 | 162.916 | 161.528 | 160.652
 | 159.246 | 158.707 | 157.363 | 141.001 | 131.505 | 129.202 | 126.304
==============================================
Los padres son: 32 i 36
```

En el proyecto se pueden encontrar las poblaciones usadas en las carpetas:

D:\TFC_MarioPelegrin\ProyectoC++\Ejecutables\Poblacion96
D:\TFC_MarioPelegrin\ProyectoC++\Ejecutables\Poblacion192

Se coge la población inicial, se copia en la carpeta del ejecutable y se renombra a “poblacion.dat”.

Lo ejecutamos y estará un máximo de generaciones o hasta que encuentra un robot que llega al destino.
Cuando acaba, muestra los contadores de generaciones y de estancamiento, las evaluaciones de la población final y el mejor individuo final.

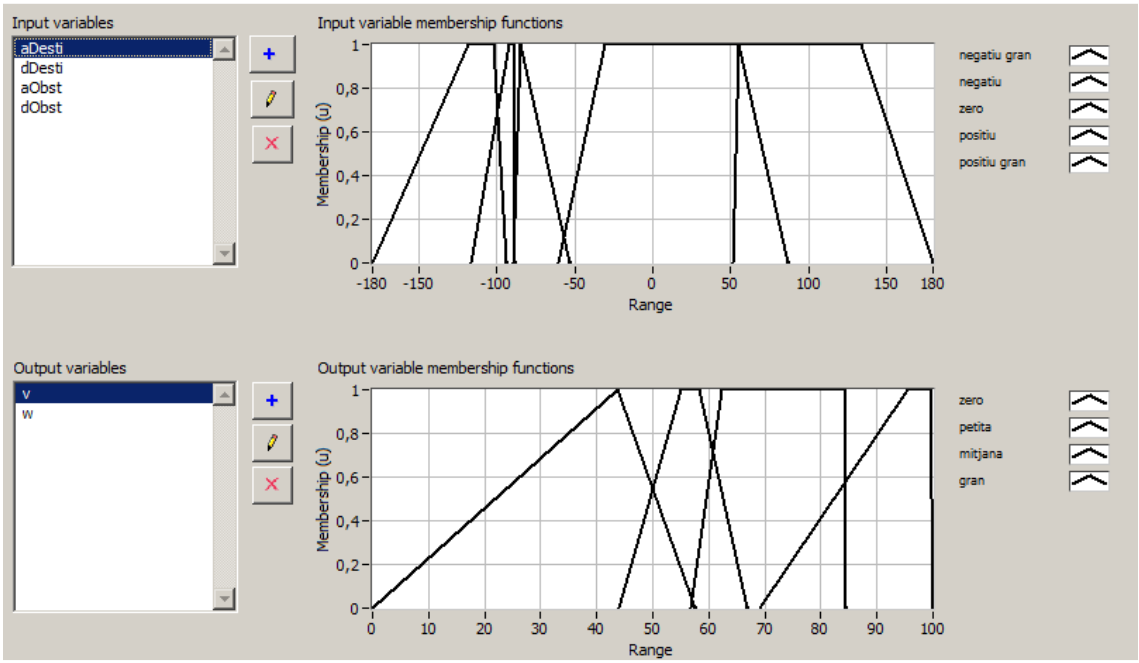
```
D:\Programa\VersionFinal\AlgGenFuzzy\Debug\CarpetaLimpia\AlgGenFuzzy96.exe
==============================================
La evolucion ha terminado con exito
==============================================
El contador de generaciones es: 96
El contador de estancamiento es: 0
==============================================
Las evaluaciones son:
=====================
917.751 | 905.749 | 901.174 | 896.21 | 895.11 | 894.931 | 890.719 | 889.313 | 88
7.808 | 885.911 | 885.903 | 884.276 | 882.781 | 879.544 | 878.517 | 877.473 | 87
5.174 | 875.056 | 874.859 | 873.465 | 872.369 | 870.649 | 866.603 | 863.591 | 86
2.632 | 857.395 | 856.337 | 855.012 | 854.494 | 853.043 | 851.332 | 851.277 | 84
9.989 | 848.934 | 845.625 | 838.238 | 837.544 | 835.202 | 830.654 | 828.448 | 82
4.04 | 819.813 | 811.131 | 801.698 | 801.634 | 795.754 | 792.471 | 790.996 | 790
.977 | 790.958 | 790.956 | 790.934 | 790.919 | 790.879 | 790.814 | 790.757 | 790
.586 | 790.291 | 788.923 | 788.407 | 786.741 | 734.763 | 731.811 | 730.147 | 729
.771 | 728.094 | 726.709 | 726.633 | 726.402 | 726.287 | 725.35 | 725.311 | 723.
543 | 722.059 | 721.126 | 720.995 | 720.29 | 720.208 | 720.132 | 719.815 | 719.6
18 | 718.956 | 718.412 | 717.751 | 717.675 | 717.325 | 717.215 | 716.815 | 716.6
59 | 716.571 | 716.424 | 716.28 | 716.137 | 716.015 | 715.961 | 715.856
==============================
```

```
El mejor individuo es:
==================================
aDesti->Cadena = -180 ¦ -114.411 ¦ -101.043 ¦ -96.5722 ¦ -114.123 ¦ -91.5083 ¦ -
89.6127 ¦ -81.7426 ¦ -87.253 ¦ -80.2679 ¦ -49.8187 ¦ -55.8357 ¦ -27.3859 ¦ 59.94
37 ¦ 89.8824 ¦ 49.5003 ¦ 54.6691 ¦ 131.209 ¦ 180 ¦
dDesti->Cadena = 0 ¦ 27.0177 ¦ 535.915 ¦ 723.631 ¦ 528.261 ¦ 561.38 ¦ 735.976 ¦7
82.585 ¦ 581.121 ¦ 781.213 ¦ 934.342 ¦ 1000 ¦
aObst->Cadena = -90 ¦ -78.8881 ¦ -76.519 ¦ -52.9764 ¦ -74.3915 ¦ -62.3204 ¦ -24.
7343 ¦ -20.5924 ¦ -46.2391 ¦ -11.0248 ¦ 9.05784 ¦ -6.92403 ¦ -4.92142 ¦ 11.836 ¦
 36.9037 ¦ 0.0559325 ¦ 35.888 ¦ 63.6578 ¦ 90 ¦
dObst->Cadena = 0 ¦ 26.1654 ¦ 119.727 ¦ 167.719 ¦ 76.0543 ¦ 142.17 ¦ 185.205 ¦ 2
28.94 ¦ 149.865 ¦ 191.587 ¦ 232.842 ¦ 300 ¦
velLin->Cadena = 0 ¦ 45.5546 ¦ 59.1481 ¦ 46.3595 ¦ 57.138 ¦ 59.6029 ¦ 68.0204 ¦5
8.7398 ¦ 62.5422 ¦ 82.571 ¦ 84.4969 ¦ 69.4396 ¦ 96.9709 ¦ 99.2403 ¦ 100 ¦
velAng->Cadena = -180 ¦ -160.33 ¦ -158.073 ¦ 11.4203 ¦ -141.417 ¦ -66.6419 ¦ 52.
0465 ¦ 55.3306 ¦ 47.8375 ¦ 57.4931 ¦ 82.2006 ¦ 73.3273 ¦ 138.973 ¦ 157.538 ¦ 161
.648 ¦ 149.822 ¦ 171.608 ¦ 174.167 ¦ 180 ¦
----------------------------------
DistAlDest: 0.532697 metros.
DistRec: 7.34369 metros.
No ha llegado.
----------------------------------
Evaluacion: 917.751
==================================
=============================================
Quieres salir: S/N
```

Le decimos que queremos salir con “S”.
El programa guarda la población en el archivo “poblacion.dat” y crea el sistema difuso del mejor individuo, “mejorRobot.fs”, además en la última versión se ha añadido que guarde en archivos, la evaluación del mejor, la evaluación media de la población y la evaluación del peor, en cada generación. Los archivos creados son: “mejor.txt”, “media.txt”, peor.txt”

Ahora con el archivo difuso podemos usar el sistema del Simulador del Labview, abriendo el archivo “SimuladorRobotPantalla.vi”, para ver cómo se comporta el robot con ese controlador. También se puede abrir con el asistente de sistemas difusos del Labview para ver como ha quedado.

Por ejemplo el sistema creado de la evolución de la población de 96 es:



Si se quiere ejecutar alguna de las pruebas, se copia el archivo ".exe" en la carpeta de los ejecutables y ya se pueden activar.

Para ejecutarlo y que la salida de la consola vaya directamente a un archivo se pueden usar comandos.

Abrimos la consola y nos situamos en la carpeta de los ejecutables.

Para iniciar y que se cree el archivo de salida, perdiendo lo que tenía:

```
echo N | AlgGenFuzzy96.exe>SalidaConsola.txt
```

Si queremos que lo haga iniciando una población nueva:

```
echo S | AlgGenFuzzy96.exe>SalidaConsola.txt
```

Para iniciar y se mantenga el archivo de salida, añadiendo el texto al final:

```
echo N | AlgGenFuzzy96.exe>>SalidaConsola.txt
```

Si queremos que lo haga iniciando una población nueva:

```
echo S | AlgGenFuzzy96.exe>>SalidaConsola.txt
```

Los mismos comandos se pueden usar con el ejecutable AlgGenFuzzy192.exe.

15.4 Manual de ejecución del simulador en el Labview

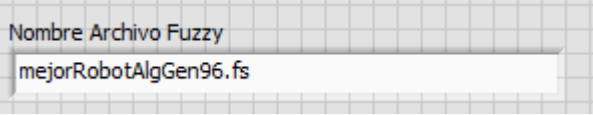
Para ejecutar el simulador, copiamos al PC la carpeta:

D:\TFC_MarioPelegrin\ProyectoLabview\SimuladorRobotFinal

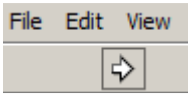
Abrimos el archivo del proyecto "SimuladorRobot.lvproj"
Abrimos el sistema "SimuladorRobotPantalla.VI"
Copiamos en la carpeta del proyecto un sistema fuzzy, por ejemplo el creado por la evolución de la población de 96 individuos:

D:\TFC_MarioPelegrin\ProyectoC++\Ejecutables\ResultadosPoblacion96

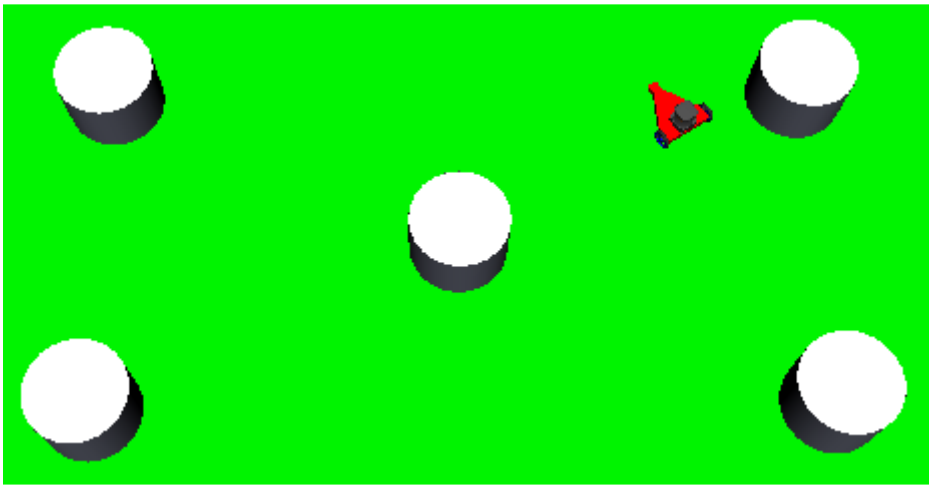
Introducimos el nombre del sistema fuzzy:



Ejecutamos el sistema dando al icono de la flecha:



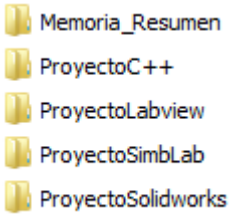
En la parte superior hay varios controles y display que muestran la información que usa el simulador.
En la parte inferior ha de aparecer una pantalla con el robot moviéndose dentro del entorno.



En este simulador se le da al robot el doble de tiempo que en la librería, ya que tiene la sobrecarga de la simulación gráfica, y si usara 30 segundos apenas mostraría su comportamiento.

15.5 Archivos incluidos en el CD:

Contiene las siguientes carpetas:



En Memoria_Resumen hay: la Memoria y el Resumen en PDF.

En ProyectoC++ hay:

Archivos del programa, tanto los ejecutables como el proyecto del Visual Studio que contiene los archivos del código de C++.
Archivos de los resultados de las ejecuciones.
Estadísticas de las evoluciones de las poblaciones de 96 y 192 individuos.

En ProyectoLabview hay:

Archivos usados para crear la librería que genera sistemas fuzzy.
Archivos para cambiar de formato 3D las piezas.
Archivos para crear el entorno 3D.
Proyecto que contiene los archivos del simulador del robot y que permite crear la librería dll.

En ProyectoSimbLab hay:

Archivos de las piezas en formato OSG, LVSG y IVE.
Archivos del Simlab que contienen las piezas.

En ProyectoSolidworks hay:

Archivos de las piezas en formato SLDPRT.

Se ha añadido una carpeta adicional con videos con parte de la ejecución por consola de AlgGen96.exe y de una simulación del robot con el mejor controlador encontrado en la pruebas.