GENERACIÓN DE UN SIMULADOR DE CARRERAS DE BAJO CONSUMO

MÁSTER CAD/CAM/CIM

16/09/2011

Autor: **FRANCISCO PALACIOS LÓPEZ**

Director: **VICENTE COLOMER ROMERO**

AGRADECIMIENTOS

A **Sergio García-Nieto Rodríguez**, mi profesor de Optimización, por su desinteresada ayuda en esta tesina en toda la parte relativa a la puesta a punto del modelo Simulink y del Algoritmo Genético.

A **Vicente Colomer Romero**, mi tutor, por su ayuda en toda la parte relativa a la técnica y el funcionamiento de un mundo totalmente desconocido para mí antes de la entrada a este Máster, como es el de las carreras de bajo consumo.

A todos los profesores de este Máster, por este maravilloso año en Valencia.

ÍNDICE

1.- INTRODUCCIÓN. LA COMPETICIÓN SHELL ECO-MARATHON.

Creada hace 27 años para promover la eficiencia energética, la Shell Eco-Marathon ha anticipado los desafíos a los que hoy en día se somete el sector del transporte. Estas competiciones son generalmente abiertas a los estudiantes de un amplio rango de niveles: desde estudiantes de secundaria, ciclos formativos, hasta universidades y escuelas de ingeniería. Sea cual sea su nivel, el reto para los estudiantes y sus profesores es siempre el mismo: diseñar un vehículo que cumpla con las normas de la carrera y que consiga el menor consumo de combustible. Dado que para el desarrollo de este tipo de proyectos es necesario un amplio conocimiento de diversas disciplinas como mecánica, electrónica, aerodinámica, física,... el interés de la participación en este evento por parte de los estudiantes y de sus profesores reside en la preparación que de una u otra forma deben recibir para afrontar esta competición, que resulta muy útil para sus futuras carreras profesionales como ingenieros, técnicos o científicos. Esta oportunidad para ganar experiencia, trabajando en un proyecto de naturaleza económica, social, medioambiental, es una invitación para contribuir a la sostenibilidad energética del transporte.

Figura 1.1.- Formación de los distintos equipos en la Shell Eco-Marathon.

El rendimiento del vehículo se evalúa sobre mangas de carrera de unos 20 km a una media igual o superior a los 30 km/h. En general los vehículos arrancan desde parados y completan los 20 km. La posición final del vehículo en el ranking se determina calculando el consumo equivalente en gasolina, en función del tipo de combustible utilizado.

El consumo en esta carrera se estima mediante los valores de poder calorífico inferior de ambos combustibles, y los resultados son expresados en kilómetros por litro. Esta forma de expresar los resultados puede llevar a confusión al público ajeno a la normativa de esta carrera, ya que la forma más natural de expresar el consumo es l/100km. La razón de expresar los resultados de esta manera no es otra que la de marcar diferencias apreciables entre los mejores vehículos. Por ejemplo 5385 km/l es más evocativo que 0.01857 l/100km. Hay que tener en cuenta que los consumos, al ser ínfimos, cualquier pequeña variación en ellos supone una gran diferencia en cuanto al kilometraje recorrido.

Estas competiciones son por tanto el lugar propicio para la innovación técnica. Uno de los avances técnicos más remarcable y que más ha perdurado en el tiempo ha sido probablemente la estrategia de "stop and go". Esta estrategia consiste en apagar el motor cuando no es necesario aplicar la fuerza motriz al vehículo, por ejemplo cuando va andando cuesta abajo, y encenderlo cuando su uso es más eficiente por ejemplo durante las pendientes de subida. Nuestro vehículo al poseer motor de combustión interna, utiliza la estrategia de carrera antes citada de “stop and go”. Cuando hay que utilizar esta estrategia en la competición, la primera pregunta que se nos viene a la cabeza es: ¿cuáles serán los puntos óptimos de arrancada y parada de motor?

Esta pregunta a priori tiene difícil respuesta, ya que son prácticamente infinitas las combinaciones que se pueden hacer de arrancada y parada del motor con la restricción de acabar la manga de la carrera a una velocidad superior a 30km/h: ¿sería mejor hacer un mayor número de arrancadas más cortas?, ¿será aconsejable hacer menos arrancadas pero más largas?, ¿hasta qué velocidad mínima será bueno dejar descender el vehículo?, ¿a partir de qué valor de velocidad máxima será aconsejable parar el motor para no disparar el consumo?

De todas estas preguntas y muchas más, nace la idea principal de esta tesina: desarrollar un simulador que incorpore el modelo matemático del vehículo y de distintos circuitos para poder realizar un gran número de pruebas, estudiando el impacto que tienen en el consumo, los distintos parámetros de diseño del vehículo y la estrategia de carrera elegida.

2.- OBJETIVOS.

El propósito general de esta tesis de máster, llevada a cabo en la Universidad Politécnica de Valencia, trata sobre la generación de un simulador en Matlab-Simulink para predecir y optimizar el comportamiento en carrera del vehículo del IDF para la Shell Eco-Marathon.

Este vehículo, equipado con un motor de combustión interna siendo su combustible etanol, utiliza la estrategia de carrera llamada como "stop and go". Esta estrategia de ahorro de combustible consiste en parar el motor cuando no es necesaria la fuerza motriz (pendientes descendentes o cuando el vehículo lleva velocidad suficientemente alta), y arrancarlo cuando es más eficiente el uso del mismo (pendientes ascendentes o cuando la velocidad cae por debajo de ciertos valores), con la única restricción en carrera de que la media sea superior o igual a 30 km/h.

El objetivo del simulador será determinar mediante un algoritmo genético, los puntos óptimos de arrancada y parada del motor, dado un circuito y condiciones climatológicas concretas, para realizar el máximo kilometraje posible. Como consecuencia de esto, también se podrán evaluar con el simulador el impacto en el consumo de parámetros como el peso del vehículo, su aerodinámica, el tipo de neumáticos utilizado… El simulador deberá ser fácil de utilizar por cualquier persona con conocimientos básicos de informática y será lo suficientemente flexible para admitir nuevas topografías de circuitos y configuraciones de los datos físicos del vehículo que no hayan sido incluidas en el momento de su creación. Se deberá poder modificar:

1) Circuito.
2) Curvas de potencia, par y consumo específico del motor.
3) Datos aerodinámicos, rozamientos, tablas de rozamientos en curvas…
4) Relación de transmisión y su rendimiento.
5) Velocidad y dirección del viento.
6) Elección del tipo de optimización a utilizar y variación de sus parámetros.

Este trabajo debido al tiempo y a los medios de que se dispone para realizarlo será estrictamente teórico. Quedarán por tanto para futuras mejoras el contrastar los valores que arroja el simulador frente a los medidos en la realidad cuando se disponga del sistema de telemetría necesario. En previsión de ello, se facilitará la futura modificación de coeficientes de rozamientos, tablas de datos, gráficas de potencias,... necesarios para un correcto ajuste entre la teoría y la realidad.

3.- TRAYECTORIA DE NUESTRO EQUIPO EN LA SHELL ECO-MARATHON

En el año 2006 se iniciaron las participaciones de nuestro equipo en la Shell Eco-Marathon con la construcción de un primer prototipo y la consecución de una marca de 250 Km con un litro de gasolina. En este proyecto participaron 20 alumnos y 3 profesores.

Figura 3.1.- Prototipo 2006 en la pista de Nogaro (Francia)

En el año 2007 se evoluciono el modelo anterior y se cambió de categoría, pasando a competir en la categoría de combustibles alternativos (Etanol E-85), tras el desarrollo del nuevo prototipo no fue posible homologar una marca ya que la organización suspendió la categoría de Etanol. En este proyecto participaron 12 alumnos y 3 profesores.

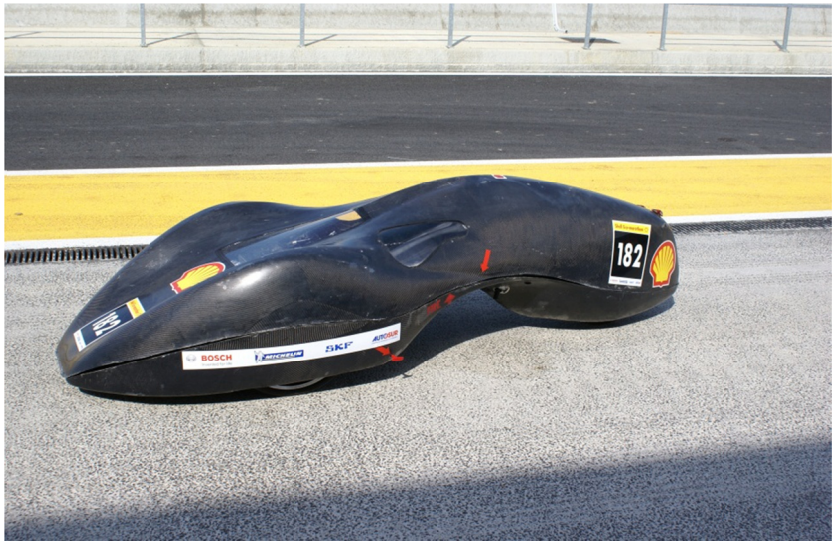


Figura 3.2- Prototipo 2007 en la pista de Nogaro (Francia)

En el año 2008 se participó en la Shell Eco-Marathon Youth Challenge UK consiguiendo un resultado de 1744'8 millas por galón equivalente a 633 Km por litro de gasolina. Siendo 2º clasificados en la categoría Etanol y 19 clasificado en la general de motores de combustión. En este proyecto participaron 8 alumnos y 2 profesores. Esta marca supone que nuestro prototipo es el primer vehículo español en combustibles alternativos y el cuarto a nivel europeo, siendo el segundo a nivel español en la temporada 2008 por resultado en km.

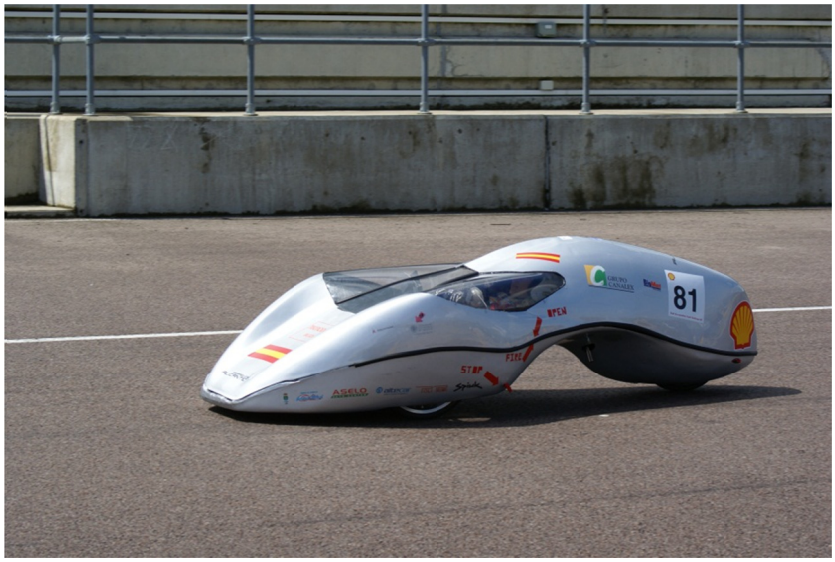


Figura 3.3.- Prototipo 2008 en la pista de Rockingham (Reino Unido)

En 2009 se obtuvo una marca de 548 km/l en la 25º Edición de la European Shell Eco-Marathon en Alemania. Obteniendo la posición 44 de la clasificación general, 2º en la categoría Etanol, 4 vehículo español y primer vehículo español en bio-combustibles. En este proyecto participaron 12 alumnos y 1 profesor.

Figura3.4.- Vehículo IDF-09 participante en la European Shell Eco-Marathon 2009

En la Murcia Solar Race 2009 edición inaugural del certamen español se obtuvo una marca de 492 km/l, 1ª posición en motores de combustión, 1ª posición en Bio-Combustibles y primer equipo español participante. En este proyecto participaron 4 alumnos y un profesor.

Figura 3.5.- Vehículo IDF-09 participando en la Murcia Solar Race 2009.

En 2010 se construyó un nuevo prototipo con mejoras en la aerodinámica y el peso del vehículo se participo en la European Shell Eco-Marathon en Alemania, sin poder obtener resultados por problemas en el inyector del motor. En este proyecto participaron 7 alumnos de la EPSA, un alumno Erasmus-Intership y un profesor. Este ha sido el proyecto más ambicioso que nuestro equipo ha llevado a cabo dado que en un tiempo de 7 meses fue desarrollado y construido el prototipo.

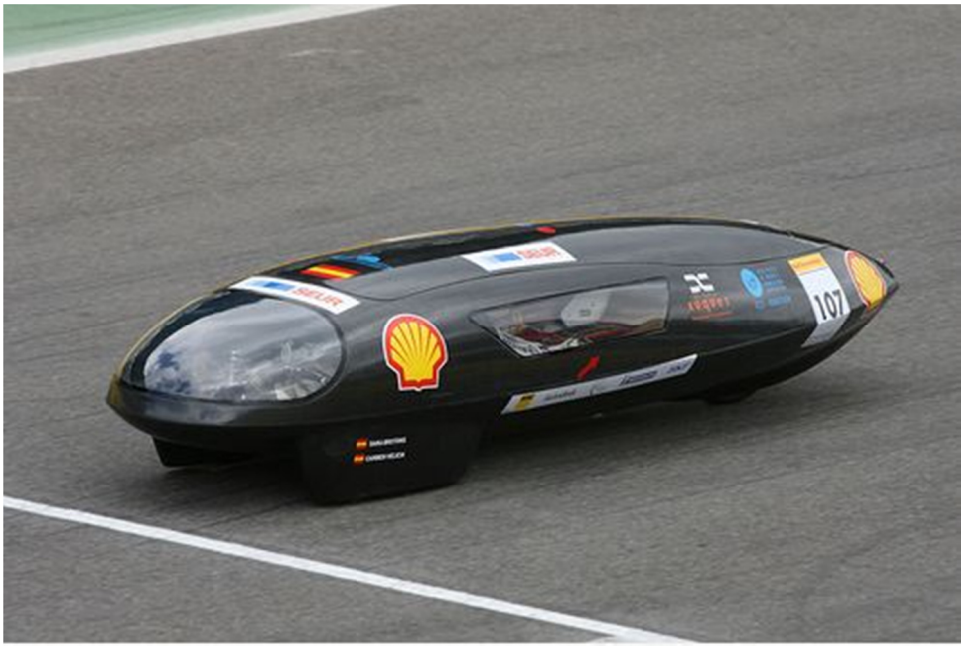


Figura 3.6.- Vehículo IDF-10 participante en la European Shell Eco-Marathon 2010.

En el mes de Octubre de 2010 participamos en colaboración con la Universidad Politécnica de Cartagena en la Murcia Solar Race 2011 con un prototipo impulsado con Energía Solar Fotovoltaica. En este proyecto participaron 3 alumnos y un profesor de la EPSA.

Figura 3.7.- Vehículo Aníbal participante en la Murcia Solar Race 2011.

3.2.- DESARROLLOS DE LA TEMPORADA 2011.

Después del diseño del nuevo vehículo en 2010 y su puesta en marcha en la Shell Eco-Marathon de ese año, se van a realizar las siguientes actividades en 2011:

- Construcción de una nueva carrocería en material compuesto.
- Realizar las pruebas necesarias en nuestro banco de pruebas y en el circuito Ricardo Tormo de Cheste.
- Puesta a punto de un sistema de telemetría.
- Puesta a punto de un simulador de carreras para mejorar las estrategias de conducción.
- Adiestrar a nuestros nuevos pilotos en las estrategias de conducción de esta competición.

4.- ¿QUÉ SON LOS ALGORITMOS GENÉTICOS?

Los Algoritmos Genéticos son una técnica de resolución de problemas de búsqueda y optimización inspirada en la teoría de la evolución de las especies y selección natural. Estos algoritmos reúnen características de búsqueda aleatoria con características de búsqueda dirigida que provienen del mecanismo de selección de los individuos más adaptados. La unión de ambas características les permite abordar los problemas de una forma muy particular, ya que tienen capacidad para acceder a cualquier región del espacio de búsqueda, capacidad de la que carecen otros métodos de búsqueda exhaustiva. A la vez que exploran el espacio de soluciones de una forma mucho más eficiente que los métodos puramente aleatorios. [Gil, 2006].

Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin (1859).

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros, tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagaran en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes "superindividuos", cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Por imitación de este proceso, los AG son capaces de ir creando soluciones para problemas del mundo real. Dado un problema específico a resolver, la entrada del AG es un conjunto de soluciones potenciales a ese problema, codificadas de alguna manera, y una métrica llamada función de aptitud que permite evaluar cuantitativamente a cada candidata. Estas candidatas pueden ser soluciones que ya se sabe que funcionan, con el objetivo de que el AG las mejore, pero se suelen generar aleatoriamente.

Luego el AG evalúa cada candidata de acuerdo con la función de aptitud. En un grupo de candidatas generadas aleatoriamente, por supuesto, la mayoría no funcionarán en absoluto, y serán eliminadas. Sin embargo, por puro azar, unas pocas pueden ser prometedoras -pueden mostrar actividad, aunque sólo sea actividad débil e imperfecta, hacia la solución del problema.

Estas candidatas prometedoras se conservan y se les permite reproducirse. Se realizan múltiples copias de ellas, pero las copias no son perfectas; se introducen cambios aleatorios durante el proceso de copia. Luego, esta descendencia digital prosigue con la siguiente generación, formando un nuevo acervo de soluciones candidatas, y son sometidas a una ronda

de evaluación de aptitud. Las candidatas que han empeorado o no han mejorado con los cambios en su código son eliminadas de nuevo; pero, de nuevo, por puro azar, las variaciones aleatorias introducidas en la población pueden haber mejorado a algunos individuos, convirtiéndolos en mejores soluciones del problema, más completas o más eficientes. De nuevo, se selecionan y copian estos individuos vencedores hacia la siguiente generación con cambios aleatorios, y el proceso se repite. Las expectativas son que la aptitud media de la población se incrementará en cada ronda y, por tanto, repitiendo este proceso cientos o miles de rondas, pueden descubrirse soluciones muy buenas del problema.

Aunque pueda parecer asombroso, los algoritmos genéticos han demostrado ser una estrategia enormemente poderosa y exitosa para resolver problemas, demostrando de manera espectacular el poder de los principios evolutivos. El poder de los AG proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el AG encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria.

En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al AG, tanto en rapidez como en eficacia. El gran campo de aplicación de los AG se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridándolas con los AG.

4.1.- ¿POR QUÉ UTILIZAR ALGORITMOS GENÉTICOS EN LA OPTIMIZACIÓN?

La razón del creciente interés por los AG es que estos son un método global y robusto de búsqueda de las soluciones de problemas. La principal ventaja de estas características es el equilibrio alcanzado entre la eficiencia y eficacia para resolver diferentes y muy complejos problemas de grandes dimensiones. Lo que aventaja a los AG frente a otros algoritmos tradicionales de búsqueda es que se diferencian de estos en los siguientes aspectos:

1) Trabajan con una codificación de un conjunto de parámetros, no con los parámetros mismos.

2) Trabajan con un conjunto de puntos, no con un único punto y su entorno (su técnica de búsqueda es global). Utilizan un subconjunto del espacio total, para obtener información sobre el universo de búsqueda, a través de las evaluaciones de la función a optimizar. Esas evaluaciones se emplean de forma eficiente para clasificar los subconjuntos de acuerdo con su idoneidad.

3) No necesitan conocimientos específicos sobre el problema a resolver; es decir, no están sujetos a restricciones. Por ejemplo, se pueden aplicar a funciones no continuas, lo cual les abre un amplio campo de aplicaciones que no podrían ser tratadas por los métodos tradicionales.

4) Utilizan operadores probabilísticos, en vez de los típicos operadores determinísticos de las técnicas tradicionales.

5) Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivas en paralelo.

6) Cuando se usan para problemas de optimización, resultan menos afectados por los máximos locales que las técnicas tradicionales.

4.2.- VENTAJAS

● El primer y más importante punto es que los AG son intrínsecamente paralelos. La mayoría de los otros algoritmos son en serie y sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al mismo tiempo, y si la solución que descubren resulta no óptima, no se puede hacer otra cosa que abandonar todo el trabajo hecho y empezar de nuevo. Sin embargo, ya que los AG tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez. Si un camino resulta ser un callejón sin salida, pueden eliminarlo fácilmente y continuar el trabajo en avenidas más prometedoras, dándoles una mayor probabilidad en cada ejecución de encontrar la solución.

● Sin embargo, la ventaja del paralelismo va más allá de esto. Consideramos lo siguiente: todas las cadenas binarias (cadenas de ceros y unos) de 8 dígitos forman un espacio de búsqueda, que puede representarse como ******** (donde * significa “o 0 o 1”). La cadena 01101010 es un miembro de este espacio. Sin embargo, también es un miembro del espacio 0*******, del espacio 01******, del espacio 0******0, del espacio 0*1*1*1*, del espacio 10*01**0, etc. Evaluando la aptitud de esta cadena particular, un AG estaría sondeando cada uno de los espacios a los que pertenece. Tras muchas evaluaciones, iría obteniendo un valor cada vez más preciso de la aptitud media de cada uno de estos espacios, cada uno de los cuales contiene muchos miembros. Por tanto, un AG que evalúe explícitamente un número pequeño de individuos está evaluando implícitamente un grupo de individuos mucho más grande. De la misma manera, el AG puede dirigirse hacia el espacio con los individuos más aptos y encontrar el mejor de ese grupo. **En el contexto de los algoritmos evolutivos, esto se conoce como teorema del esquema, y es la ventaja principal de los AG sobre otros métodos de resolución de problemas.**

Afortunadamente, el paralelismo implícito de los AG les permite superar incluso este enorme número de posibilidades, y encontrar con éxito resultados óptimos o muy buenos en un corto periodo de tiempo, tras muestrear directamente sólo regiones pequeñas del vasto paisaje adaptativo.

Al principio, el AG genera una población inicial diversa, lanzando una “red” sobre el paisaje adaptativo. [Koza, 2003] compara esto con un ejército de paracaidistas cayendo sobre el paisaje del espacio de búsqueda de un problema, cada uno de ellos con órdenes de buscar el pico más alto). Pequeñas mutaciones permiten a cada individuo explorar sus proximidades, mientras que la selección enfoca el progreso, guiando a la descendencia del algoritmo cuesta arriba hacia zonas más prometedoras del espacio de soluciones, permitiendo escapar de mínimos locales.

● Sin embargo, el cruce es el elemento clave que distingue a los AG de los otros métodos como el algoritmo de “ascenso a colina” y el recocido simulado. Sin el cruce, cada solución individual va por su cuenta, explorando el espacio de búsqueda en sus inmediaciones sin referencia de lo que el resto de individuos puedan haber descubierto. Sin embargo, con el cruce en juego, hay una transferencia de información entre los candidatos prósperos los individuos pueden beneficiarse de lo que otros han aprendido, y los esquemas pueden

mezclarse y combinarse, con el potencial de producir una descendencia que tenga las virtudes de sus dos padres y ninguna de sus debilidades.

● Otra área en el que destacan los AG es su habilidad para manipular muchos parámetros simultáneamente. Muchos problemas de la vida real no pueden definirse en términos de un único valor que hay que minimizar o maximizar, sino que deben expresarse en términos de múltiples objetivos, a menudo involucrando contrapartidas: uno sólo puede mejorar a expensas de otro. Los AG son muy buenos resolviendo estos problemas: en particular, su uso del paralelismo les permite producir múltiples soluciones, igualmente buenas, al mismo problema, donde posiblemente una solución candidata optimiza un parámetro y otra candidata optimiza uno distinto y luego un supervisor humano puede seleccionar una de esas candidatas para su utilización. Si una solución particular a un problema con múltiples objetivos optimiza un parámetro hasta el punto en el que ese parámetro no puede mejorarse más sin causar una correspondiente pérdida de calidad en algún otro parámetro, esa solución se llama óptimo de Pareto.

● Finalmente, una de las cualidades de los AG que, a primera vista, puede parecer un desastre, resulta ser una de sus ventajas: a saber, los AG no saben nada de los problemas que deben resolver. En lugar de utilizar información específica conocida a priori para guiar cada paso y realizar cambios con un ojo puesto en el mejoramiento, como hacen los diseñadores humanos, son “relojeros ciegos”; realizan cambios aleatorios en sus soluciones candidatas y luego utilizan la función objetivo para determinar si esos cambios producen una mejora.

La virtud de esta técnica es que permite a los AG comenzar con una mente abierta, por así decirlo. Como sus decisiones están basadas en la aleatoriedad, todos los caminos de búsqueda posibles están abiertos teóricamente a un AG; en contraste, cualquier estrategia de resolución de problemas que dependa de un conocimiento previo, debe inevitablemente comenzar descartando muchos caminos a priori, perdiendo así cualquier solución novedosa que pueda existir. Los AG, al carecer de ideas preconcebidas basadas en creencias establecidas sobre “cómo deben hacerse las cosas” o sobre lo que “de ninguna manera podría funcionar”, los AG no tienen este problema. De manera similar, cualquier técnica que dependa de conocimiento previo fracasará cuando no esté disponible tal conocimiento, pero, de nuevo, los AG no se ven afectados negativamente por la ignorancia. Mediante sus componentes de paralelismo, cruce y mutación, pueden viajar extensamente por el paisaje adaptativo, explorando regiones que algoritmos producidos con inteligencia podrían no haber tenido en cuenta, y revelando potencialmente soluciones de asombrosa e inesperada creatividad que podrían no habérseles ocurrido nunca a los diseñadores humanos.

4.3.- INCONVENIENTES

Aunque los AG han demostrado su eficiencia y potencia como estrategia de resolución de problemas, no son la panacea. Los AG tienen ciertas limitaciones, sin embargo, se demostrará que todas ellas pueden superarse y que ninguna de ellas afecta a la validez de la evolución biológica.

● La primera y más importante consideración al crear un AG es definir una representación del problema. El lenguaje utilizado para especificar soluciones candidatas debe ser robusto, es decir, debe ser capaz de tolerar cambios aleatorios que no produzcan constantemente errores fatales o resultados sin sentido. Esta elección no es ni mucho menos trivial y de ella dependerá en gran medida la bondad de nuestro algoritmo y su capacidad para resolver el problema.

● El problema de cómo escribir la función objetivo debe considerarse cuidadosamente para que se pueda alcanzar una mayor aptitud y verdaderamente signifique una solución mejor para el problema dado. Si se elige mal una función objetivo o se define de manera inexacta, puede que el AG sea incapaz de encontrar una solución al problema, o puede acabar resolviendo el problema equivocado.

● Además de elegir bien la función objetivo, también deben elegirse cuidadosamente los otros parámetros de un AG: el tamaño de la población, las probabilidades de cruce y mutación, el tipo y fuerza de la selección. Si el tamaño de la población es demasiado pequeño, puede que el AG no explore suficientemente el espacio de soluciones para encontrar buenas soluciones consistentemente. Si el ritmo de cambio genético es demasiado alto o el sistema de selección se escoge inadecuadamente, puede alterarse el desarrollo de esquemas beneficiosos y la población puede entrar en catástrofe de errores, al cambiar demasiado rápido para que la selección llegue a producir convergencia.

● Un problema muy conocido que puede surgir con un AG se conoce como convergencia prematura. Si un individuo que es más apto que la mayoría de sus competidores su valor emerge rápidamente muy por encima del resto, lo que conlleva que se puede reproducir tan abundantemente que merme la diversidad de la población demasiado pronto, provocando que el algoritmo converja hacia el óptimo local que representa ese individuo, en lugar de rastrear el paisaje adaptativo lo suficientemente a fondo para encontrar el óptimo global. Esto es un problema especialmente común en las poblaciones pequeñas, donde incluso una variación aleatoria en el ritmo de reproducción puede provocar que un genotipo se haga dominante sobre los otros.

● Finalmente, varios investigadores aconsejan no utilizar AG en problemas resolubles de manera analítica. No es que los AG no puedan encontrar soluciones buenas para estos problemas; simplemente es que los métodos analíticos tradicionales consumen mucho menos tiempo y potencia computacional que los AG y, a diferencia de los AG, a menudo está demostrado matemáticamente que ofrecen la única solución exacta.

5.- ESTADO DEL ARTE: ALGORITMOS GENÉTICOS

Los algoritmos genéticos fueron desarrollados por John Holland, junto a su equipo de investigación, en la universidad de Michigan en la década de 1970 [Holland, 1975]. Éstos combinan las nociones de supervivencia del más apto con un intercambio estructurado y aleatorio de características entre individuos de una población de posibles soluciones, conformando un algoritmo de búsqueda que puede aplicarse para resolver problemas de optimización en diversos campos [Goldberg, 1989]. Imitando la mecánica de la evolución biológica en la naturaleza, los algoritmos genéticos operan sobre una población compuesta de posibles soluciones al problema. Cada elemento de la población se denomina "cromosoma". Un cromosoma es el representante, dentro del algoritmo genético, de una posible solución al problema. La forma en que los cromosomas codifican a la solución se denomina "Codificación" (ver figura 5.1).

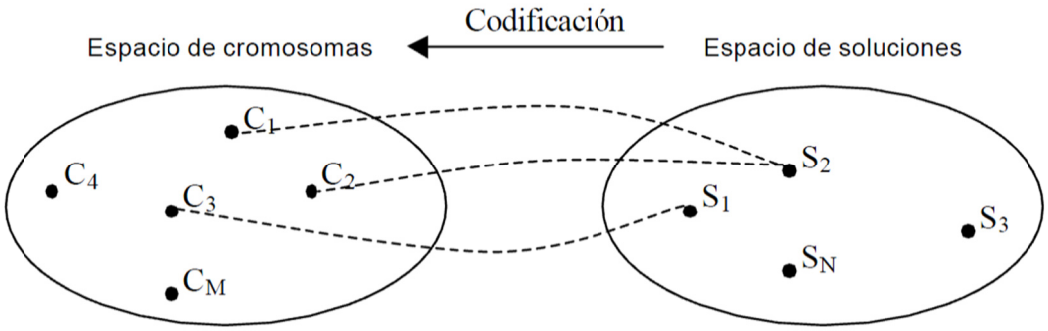


Figura 5.1.- Cada cromosoma representa una posible solución del problema

El algoritmo genético va creando nuevas "generaciones" de esta población, cuyos individuos son cada vez mejores soluciones al problema. La creación de una nueva generación de individuos se produce aplicando a la generación anterior operadores genéticos, adaptados de la genética natural.

La figura 5.2 representa el esquema de funcionamiento del algoritmo genético. El proceso comienza seleccionando un número de cromosomas para que conformen la población inicial. A continuación se evalúa la función de adaptación para estos individuos. La función de adaptación da una medida de la aptitud del cromosoma para sobrevivir en su entorno. Debe estar definida de tal forma que los cromosomas que representen mejores soluciones tengan valores más altos de adaptación.

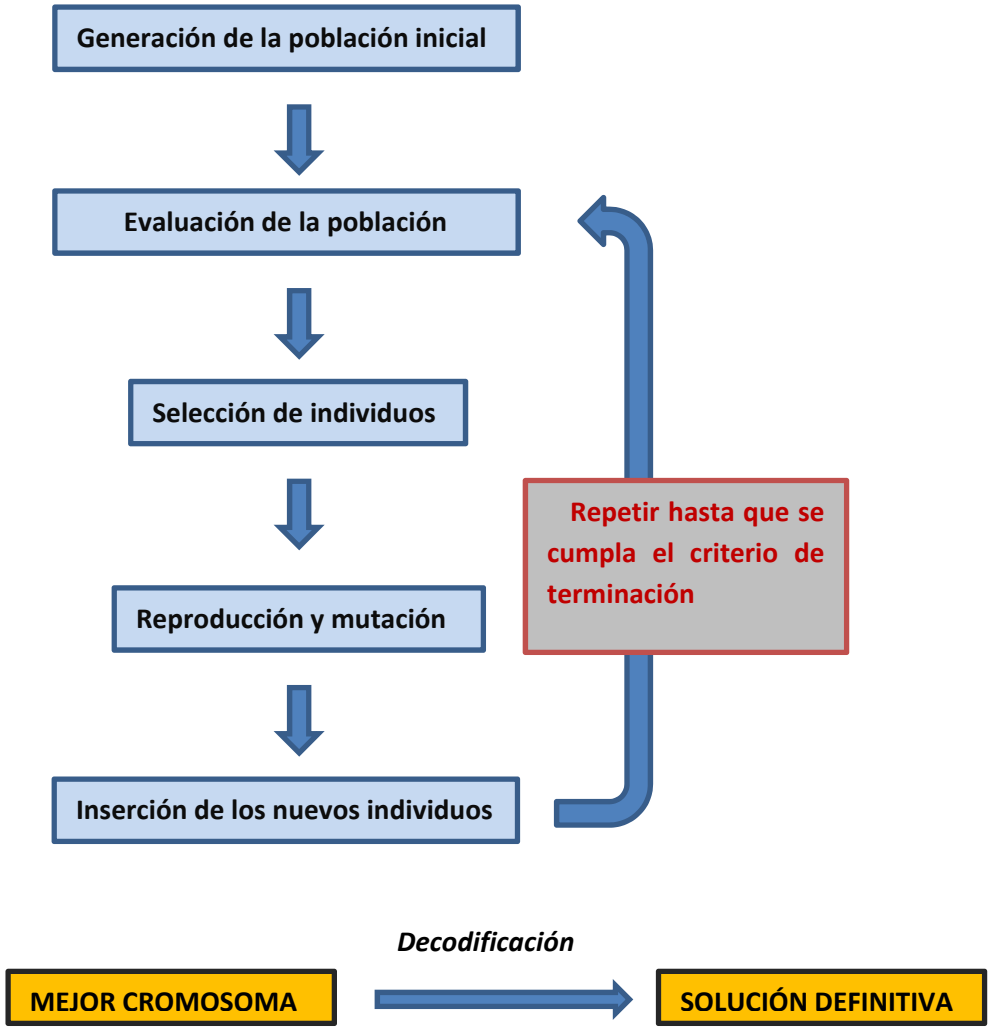


Figura 5.2.- Esquema general de funcionamiento de un algoritmo genético

Los individuos más aptos se seleccionan en parejas para reproducirse. La reproducción genera nuevos cromosomas que combinan características de ambos padres. Estos nuevos cromosomas reemplazan a los individuos con menores valores de adaptación. A continuación, algunos cromosomas son seleccionados al azar para ser mutados. La mutación consiste en aplicar un cambio aleatorio en su estructura. Luego, los nuevos cromosomas deben incorporarse a la población; estos cromosomas deben reemplazar a cromosomas ya existentes. Existen diferentes criterios que pueden utilizarse para elegir a los cromosomas que serán reemplazados. El ciclo de selección, reproducción y mutación se repite hasta que se cumple el criterio de terminación del algoritmo, momento en el cual el cromosoma mejor adaptado se devuelve como solución.

5.1.- CODIFICACIÓN

Se supone que los individuos (posibles soluciones del problema), pueden representarse como un conjunto de parámetros (que denominaremos genes), los cuales agrupados forman una ristra de valores o cromosoma.

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina fenotipo. El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como genotipo. Los mismos términos se utilizan en el campo de los Algoritmos Genéticos. Así se usa gen para referirse a la codificación de una determinada característica del individuo. En los AG se suele identificar un gen con cada posición de la cadena binaria, aunque esto no tiene por qué ser siempre así. Se usa alelo para los distintos valores que puede tomar un gen y locus para referirse a una determinada posición de la cadena binaria.

La figura 5.3 muestra un posible esquema de representación para un problema que tiene como soluciones a los polígonos regulares. Los parámetros que identifican a cada solución son 2 (cantidad de lados y longitud del lado), y estos se codifican en el cromosoma en forma binaria. El cromosoma se compone de una cadena de 10 bits, en los que los primeros 3 son la cantidad de lados, y los siguientes 7 bits representan la longitud de los lados en milímetros.

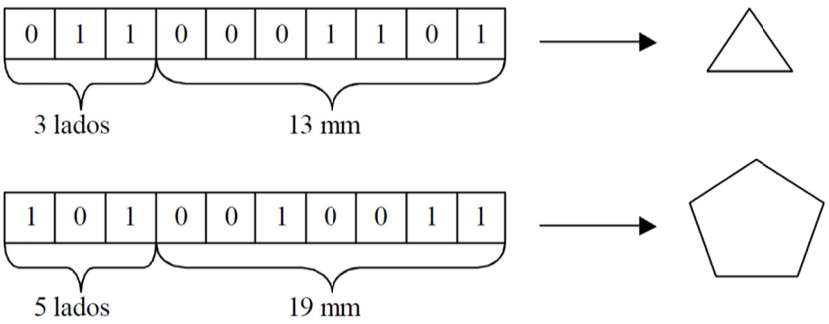


Figura 5.3.- Ejemplo de esquema de representación para polígonos regulares.

El esquema de representación debería ser tal que exista al menos una posible codificación para cada una de las soluciones posibles. Las soluciones que no estén dentro del espacio de cromosomas no serán exploradas por el algoritmo genético. En el ejemplo de la 5.3, el algoritmo genético no explorará soluciones que se compongan por polígonos de más de 7 lados ni longitudes mayores a 127 milímetros, ya que con 3 y 7 bits pueden codificarse solamente números del 0 al 7, y del 0 al 127, respectivamente. Por el mismo motivo (porque la búsqueda se hace sobre el espacio de cromosomas), es deseable que no haya redundancia en la representación: que cada solución sea representada por solamente un cromosoma.

Si existen *k* cromosomas por cada solución, el espacio de búsqueda sobre el que opera el algoritmo genético es *k* veces más grande que el espacio de soluciones, haciendo más lento el proceso de evolución. La representación ejemplificada en la figura 5.3 no tiene redundancia, cada polígono es representado sólo por un cromosoma. Otro problema que puede presentarse es que haya cromosomas que no representan ninguna solución. En el ejemplo de la figura 5.3, un cromosoma que tenga todos 0 en los primeros 3 ó en los últimos 7 bits no representa un polígono válido. En caso de que la representación lo permita, los operadores del algoritmo genético deben adaptarse para tratar con este tipo de cromosomas.

La codificación ejemplificada en la figura 5.3 se denomina "binaria", ya que cada posición del cromosoma contiene un bit. Esta es la representación clásica propuesta por los primeros autores y que todavía es utilizada ampliamente [Goldberg, 1989; Cole, 1998; Falkenauer, 1999]. Sin embargo, hay problemas para los cuales esta representación no es la más conveniente.

El funcionamiento de los algoritmos genéticos está basado en lo que se denomina la "hipótesis de los bloques constructores" [Goldberg, 1989]. Esta hipótesis requiere que los cromosomas se compongan por bloques significativos que codifiquen las características de la solución lo más independientemente posible.

El ejemplo de la figura 5.3 es un claro ejemplo de una representación que no cumple con esta premisa. Sería más apropiado un esquema en el cual el cromosoma se componga por 2 números enteros, uno de los cuales codifique el número de lados y el otro la longitud. La figura 5.4 muestra esta representación.

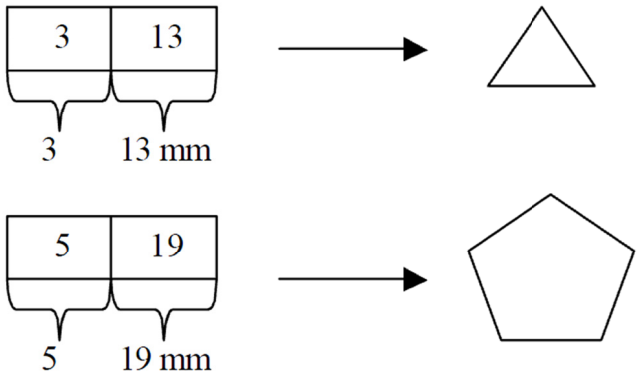


Figura 5.4.- Ejemplo de representación no binaria para el problema de los polígonos regulares.

5.2.- GENERACIÓN DE POBLACIÓN INICIAL

La población inicial es la principal fuente (luego se verá que el operador de mutación también trabaja sobre este punto) de material genético para el algoritmo genético. La población inicial debe contener cromosomas que estén bien dispersos por el espacio de soluciones. La manera más simple de cumplir con este objetivo es elegir cromosomas al azar.

El uso de una heurística [Bezdek *et.al.*, 1994] puede ayudar a generar una población inicial compuesta de soluciones de mediana calidad, ahorrando tiempo al proceso de evolución. Este punto puede ser ventajoso de cara a conseguir una rápida convergencia pero por el contrario puede hacer caer al algoritmo en un mínimo local. Por tanto, es imprescindible para el buen funcionamiento del algoritmo dotar a la población de suficiente variedad para poder explorar todas las zonas del espacio de búsqueda.

Una cuestión que uno puede plantearse es la relacionada con el tamaño idóneo de la población. Parece intuitivo que las poblaciones pequeñas corren el riesgo de no cubrir adecuadamente el espacio de búsqueda, mientras que el trabajar con poblaciones de gran tamaño puede acarrear problemas relacionados con el excesivo costo computacional.

[Goldberg, 1989] efectuó un estudio teórico, obteniendo como conclusión que el tamaño óptimo de la población para ristras de longitud *L*, con codificación binaria, crece exponencialmente con el tamaño de la ristra. Este resultado traería como consecuencia que la aplicabilidad de los Algoritmos Genéticos en problemas reales sería muy limitada, ya que resultarían no competitivos con otros métodos de optimización combinatoria. [Alander ,1992], basándose en evidencia empírica sugiere que un tamaño de población comprendida entre L y 2L es suficiente para atacar con éxito los problemas por él considerados.

5.3.- GRADO DE ADAPTACIÓN DE LOS INDIVIDUOS.

La evolución de la población depende en gran medida de la calidad de los individuos que la componen y compiten por aumentar la presencia de sus genes en generaciones futuras, mediante la participación en las operaciones de reproducción. Esta calidad se mide por la adecuación de cada individuo a ser la solución óptima del problema.

La única forma que tenemos de evaluar la calidad de los individuos es mediante la función de adaptación. La determinación de ésta es uno de los aspectos cruciales para el buen funcionamiento del algoritmo genético, ya que deberá garantizar la diversidad de los valores de adaptación de los individuos. Valores muy similares de adaptación entre los mejores individuos y el resto de la población, provocaría el incorrecto funcionamiento de los mecanismos de selección, haciendo los cruces aleatorios en lugar de guiados hacia los individuos más aptos. Con una problemática similar, grandes discrepancias entre valores de adaptación podrían provocar la rápida convergencia del algoritmo hacia un óptimo local, debido a que los mejores individuos obtendrían gran cantidad de copias de sus genes, llegando rápidamente a una situación de falta de diversidad.

Para detectar situaciones de falta de diversidad, en los algoritmos evolutivos se utilizan diferentes medidas [Goldberg y Deb, 1991]:

- **Presión selectiva:** nos indica el grado en el que se favorecen a individuos más adaptados. Valores pequeños de ella conllevan una convergencia muy lenta del algoritmo y valores muy elevados pueden dar lugar a una convergencia prematura en un mínimo local [Hancock, 94].

$$Presión\ selectiva = \frac{Adaptación\ máxima}{Adaptación\ media}$$

- **Tiempo de posesión:** mide el número de generaciones necesarias para que toda la población esté compuesta por copias del mismo individuo cuando el único operador que se aplica es la selección.

Diversos autores han implementado mecanismos de modificación de la función objetivo para obtener una función de adaptación que favorezca la diversidad en la población.

5.3.1.- Escalado de la función de adaptación

Esta técnica nos permite establecer una separación adecuada entre los valores de adaptación de los distintos individuos, para que el mecanismo de selección de nuestro algoritmo funcione correctamente. Puede consistir en una dilatación del rango cuando los valores están muy cercanos, o una contracción cuando los valores están demasiado alejados.

- ***Escalado lineal:*** Con este método la adaptación de un individuo x_i se obtiene a partir de la función objetivo $g(x_i)$ como:

$$f(x_i) = a \cdot g(x_i) + b$$

Donde a y b se eligen de forma que la adaptación media de la población corresponda con la media de los valores de la función objetivo y que la adaptación del mejor sea el producto por un cierto parámetro P por la adaptación media. P normalmente tiene valores entre 1.2 y 2.

$$b = (1 - a) \cdot \bar{g}$$

$$a = \frac{(P - 1) \cdot \bar{g}}{g_{máx} \cdot \bar{g}}$$

- ***Escalado Sigma:*** Esta técnica, propuesta inicialmente por [Forrest, 85] hace que el valor de la adaptación de un individuo dependa del valor de la función objetivo, del valor medio de dicha función para toda la población y de la dispersión σ de valores. Este mecanismo hace que el comportamiento de la selección varíe a lo largo de la evolución. En las primeras generaciones se tendrán valores altos de la desviación estándar, lo que hará que las variaciones entre las adaptaciones de los individuos sean más pequeñas y las oportunidades de supervivencia se repartan. Según avanza la evolución, la desviación se reduce y los individuos más adaptados tienen más oportunidades.

$$f(x_i) = \begin{cases} 1 + \dfrac{f(x_i) - \bar{f}}{2 \cdot \sigma} & si\ \sigma \neq 0 \\ 1 & si\ \sigma = 0 \end{cases}$$

Siendo σ la desviación estándar:

$$\sigma = \sqrt{\frac{n \cdot \sum f^2 - (\sum f)^2}{n^2}}$$

- ***Escalado basado en potencias:*** Con este método propuesto en [Macgilvary, 85], el valor de la adaptación se obtiene elevando el valor de la función objetivo a alguna potencia k.

$$f(x_i) = g(x_i)^k$$

Valores de K utilizados en la literatura han sido 1.005 [Mickalewicz, 94]. El método y los parámetros necesarios a utilizar en cada problema es una cuestión de práctica y experiencia.

5.4.- OPERADOR GENÉTICO DE SELECCIÓN

Los algoritmos de selección serán los encargados de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no. Puesto que se trata de imitar lo que ocurre en la naturaleza, se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto la selección de un individuo estará relacionada con el valor obtenido en la función de adaptación. No se debe sin embargo eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población se volvería homogénea. Una opción bastante común consiste en seleccionar el primero de los individuos participantes en el cruce mediante alguno de los métodos expuestos a continuación y el segundo de manera aleatoria.

5.4.1.- Selección por ruleta.

Propuesto por [De Jong, 75], es posiblemente el método más utilizado desde los orígenes de los algoritmos genéticos. A cada uno de los individuos de la población se le asigna una parte proporcional a su ajuste de una ruleta de tal forma que la suma de todos los porcentajes sea la unidad. Los mejores individuos recibirán una porción de la ruleta mayor que la recibida por los peores. Generalmente la población está ordenada en base al ajuste por lo que las porciones más grandes se encuentran al inicio de la ruleta. Para seleccionar un individuo basta con generar un número aleatorio del intervalo [0-1] y devolver el individuo situado en esa posición de la ruleta. Esta posición se suele obtener recorriendo los individuos de la población y acumulando sus proporciones de ruleta hasta que la suma exceda el valor obtenido.

Es un método muy sencillo pero ineficiente a medida que aumenta el tamaño de la población. Presenta además el inconveniente de que el peor individuo puede ser seleccionado más de una vez.

En la siguiente tabla podemos ver un ejemplo concreto de selección medio de ruleta para 5 individuos.

INDIVIDUO	ADAPTACIÓN	PROBABILIDAD
1	6.82	0.31
2	1.11	0.05
3	8.48	0.38
4	2.57	0.12
5	3.08	0.14
TOTAL	22.05	1

Haciendo un símil con una ruleta de casino es como si repartiéramos a los individuos a lo largo de la ruleta haciendo que los individuos con más adaptación tuvieran más posiciones de la ruleta de manera proporcional a éste. Al lanzar una bola en la ruleta caerá, en la casilla de un individuo, teniendo más probabilidad de salir los individuos con mejor adaptación. En la figura 5.5 se muestran en forma de ruleta los datos calculados en el ejemplo anterior:

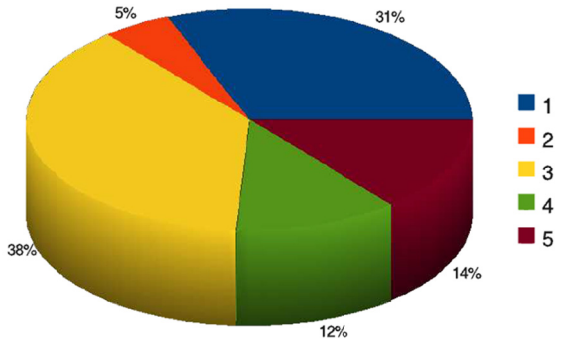


Figura 5.5.- Ruleta repartida según las probabilidades de los individuos

5.4.2.- Muestreo estocástico universal.

Es un procedimiento similar al de selección por ruleta, pero en este caso se genera un sólo número aleatorio y a partir de él se generan los K números que se necesitan, para generar K individuos espaciados de igual forma. Los números se calculan de la siguiente forma:

$$a_j := \frac{a + j - 1}{k} \quad (\forall j = 1, \dots, k)$$

Una vez generados estos números, el método funciona de la misma forma que la selección por ruleta. Este método es más eficiente que el de la ruleta.

5.4.3.- Muestreo por restos.

Este método realiza una selección proporcional a la adaptación de los individuos, garantizando copias de los mejores individuos sin intervención del azar, dejando que éste sólo intervenga en la parte de los individuos a seleccionar para completar la muestra. Concretamente, de cada individuo x_i se selecciona $p_i \cdot k$ copias para la muestra, siendo k el número de individuos a seleccionar, y p_i la probabilidad de selección del individuo i. Los individuos que quedan hasta completar el tamaño k de la muestra se seleccionan por alguno de los métodos anteriores.

5.4.4.- Selección por torneo.

Los tres operadores de selección descritos anteriormente no permiten regular la presión selectiva. El operador de selección basado en restos siempre seleccionará a los mejores individuos de la población. La selección por ruleta, si no se aplica ninguna función de escala, aplica una presión selectiva muy alta cuando las aptitudes de los individuos son variadas, y muy baja cuando las aptitudes son similares [Goldberg, 1989].

La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo:

- Determinística.
- Probabilística.

En la versión determinística se selecciona al azar un número T de individuos (generalmente T = 2). De entre los individuos seleccionados se escoge el más apto para pasarlo a la siguiente generación.

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio del intervalo [0-1], si es mayor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo más alto y en caso contrario el menos apto. Generalmente P toma valores en el rango $0.5 < p \leq 1$.

Los parámetros *T* y *p* permiten regular la presión selectiva. Cuanto más grandes son los valores de *T* y *p*, mayor es la presión selectiva. En el caso extremo de que *p* sea igual a 1 y *T* igual al tamaño de la población, el algoritmo genético solamente seleccionará al mejor individuo de la población. En el otro extremo, si *T* es igual a 1, se logra la presión selectiva más baja (los cromosomas se seleccionan al azar). Manteniendo estos parámetros constantes, se logra una presión selectiva que es independiente de los valores absolutos de aptitud de la población, y sin requerir la aplicación de funciones de escala sobre la función de adaptación.

5.4.5.- Elitismo.

El elitismo consiste en asegurar la supervivencia de los mejores individuos de la población. En su detallado estudio del comportamiento de los algoritmos genéticos en la optimización de una colección de funciones seleccionadas a tal fin, [De Jong, 75] descubrió que el elitismo acelera la convergencia de funciones unimodales, es decir, con único valor óptimo y por tanto relativamente sencillas. Sin embargo, en funciones multimodales, más complejas, el elitismo puede degradar el comportamiento del algoritmo.

Por lo tanto, debemos utilizar el elitismo cuidadosamente, teniendo en cuenta las características del problema. En general, el porcentaje de la población perteneciente a la élite no debe ser mayor del 1 ó 2% del total. Sin embargo, es una técnica muy útil que no sólo acelera la convergencia, sino que asegura que si en algún momento de la evolución del algoritmo hemos alcanzado una buena solución, ésta no se perderá en generaciones posteriores.

5.5.- OPERADOR GENÉTICO DE REPRODUCCIÓN

Una vez seleccionados los individuos, éstos son recombinados para producir algunos individuos que no estaban presentes en la generación anterior. De esta forma el algoritmo genético va accediendo a nuevas regiones del espacio de búsqueda. Su importancia para la transición entre generaciones es elevada, puesto que las tasas de cruce con las que se suele trabajar rondan el 80%. [Gestal, 04]

La fase de reproducción se implementa por medio de un operador de reproducción. El operador de reproducción es el encargado de transferir el material genético de una generación a la siguiente. Es este operador el que confiere a la búsqueda de soluciones mediante algoritmos genéticos su característica más distintiva [Falkenauer, 1999].

A diferencia de otros métodos de optimización, los algoritmos genéticos no solamente exploran el vecindario de las buenas soluciones, sino que recombinan sus partes para formar nuevas soluciones. Se ha hecho notar que el descubrimiento de nuevas teorías combinando nociones ya conocidas es un mecanismo que el hombre ha utilizado constantemente a lo largo de la evolución de la ciencia [Goldberg, 1989].

El objetivo de los operadores de reproducción es, partiendo de dos cromosomas padres, generar uno o más cromosomas hijos que hereden características de ambos padres, como se muestra en la figura 5.6. Se dice que en el hijo se “recombinan” las características de los padres. Si las características se traspasan en bloques significativos, se espera que un hijo que recombina características de buenas soluciones sea una buena solución, tal vez mejor que cualquiera de sus padres [Falkenauer, 1999].

Los diferentes métodos de cruce podrán operar de dos formas diferentes. Si se opta por una estrategia destructiva los descendientes se insertarán en la población temporal aunque sus padres tengan un mejor ajuste (trabajando con una población a esta comparación se realizará con los individuos a reemplazar). Por el contrario utilizando una estrategia no destructiva, la descendencia pasará a la siguiente generación únicamente si supera la bondad del ajuste de los padres o de los individuos a reemplazar. Al compartir las características buenas de dos individuos, la descendencia, o al menos parte de ella, debería tener una bondad mayor que cada uno de los padres por separado. Si el cruce no agrupa las mejores características en uno de los hijos y la descendencia tiene un peor ajuste que los padres no significa que se esté dando un paso atrás.

Optando por una estrategia de cruce no destructiva garantizamos que pasen a la siguiente generación los mejores individuos. Si aún con un ajuste peor se opta por insertar a la descendencia, y puesto que los genes de los padres continuarán en la población (aunque dispersos y posiblemente levemente modificados por la mutación), en posteriores cruces se podrán volver a obtener estos padres, recuperando así la bondad previamente perdida.

Los operadores de cruce más comunes tratados en la bibliografía son los siguientes:

- ***Operador de cruce monopunto:*** En esta modalidad de cruce, seleccionamos aleatoriamente un punto por el cual se cortarán los cromosomas del individuo padre y madre, de forma que cada uno de los dos hijos llevará una parte del cada individuo. Este operador produce de esta forma dos hijos que combinan propiedades de ambos padres, lo que puede llevar a una mejora de la adaptación de los hijos respecto a la de los padres.

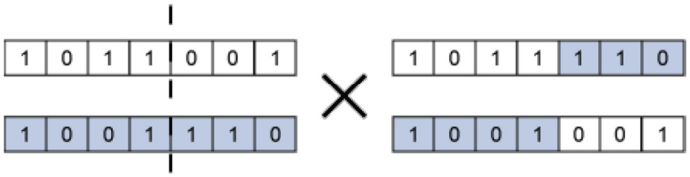


Figura 5.6.- Cruce monopunto.

- ***Operador de cruce de dos puntos:*** Se trata de la generalización del cruce de 1 punto. En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior se realizan dos cortes. Deberá tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se origen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre.

  Generalizando se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto. Sin embargo existen estudios que desaprueban esta técnica [De Jong, 75]. Aunque se admite que el cruce de 2 puntos aporta una sustancial mejora con respecto al cruce de un solo punto, el hecho de añadir un mayor número de puntos de cruce reduce el rendimiento del AE. El problema principal de añadir nuevos puntos de cruce radica en que es más fácil que los segmentos originados sean corrompibles, es decir, que por separado quizás pierdan las características de bondad que poseían conjuntamente. Sin embargo no todo son desventajas y añadiendo más puntos de cruce se consigue que el espacio de búsqueda del problema sea explorado más a fondo.

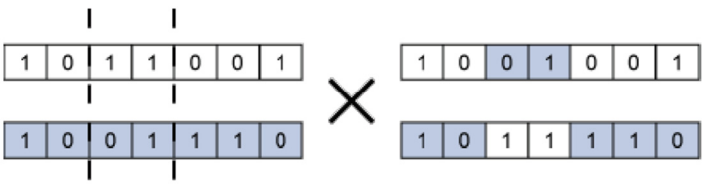


Figura 5.7.- Cruce de dos puntos.

- ***Operador de cruce uniforme:*** El cruce uniforme es una técnica completamente diferente de las vistas hasta el momento. Fue definido por [Syswerda, 91]. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro padre. Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si por el contrario hay un 0 el gen se copia del segundo padre. Para producir el segundo descendiente se intercambian los papeles de los padres o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce.

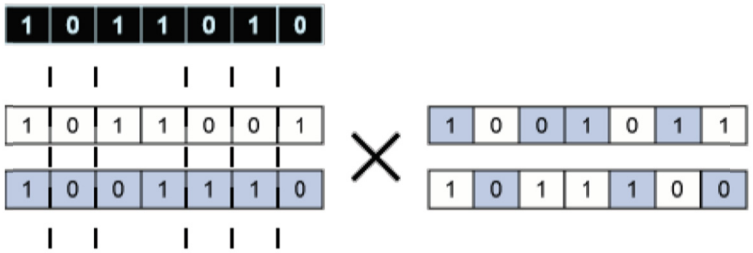


Figura 5.8.- Cruce uniforme

- ***Cruces específicos de codificaciones no binarias:*** Los tres tipos de cruce vistos hasta el momento son válidos para cualquier representación del genotipo. Si se emplean genotipos compuestos por valores enteros o reales pueden definirse otro tipo de operadores de cruce:

    - ***Media:*** el gen de la descendencia toma el valor medio de los genes de los padres. Tiene la desventaja de que únicamente se genera un descendiente en el cruce de dos padres.

    - ***Media geométrica:*** Cada gen de la descendencia toma como valor la raíz cuadrada del producto de los genes de los padres. Presenta el problema añadido de qué signo dar al resultado si los padres tienen signos diferentes.

    - ***Extensión:*** Se toma la diferencia existente entre los genes situados en las mismas posiciones de los padres y se suma al valor más alto o se resta del valor más bajo. Solventa el problema de generar un único descendiente.

5.6.- OPERADOR GENÉTICO DE MUTACIÓN.

El operador de mutación debe introducir pequeñas variaciones, casi siempre aleatorias, en la codificación de los genomas. En muchos casos la mutación produce individuos con peor adaptación que los individuos originales, ya que la mutación puede romper las posibles correlaciones entre genes que se hayan formado con la evolución de la población. Sin embargo, contribuyen a mantener la diversidad de la población, que es fundamental para el buen funcionamiento del algoritmo.

- ***Mutación por inversión:*** Es un operador de mutación clásico en las codificaciones binarias. Su funcionamiento consiste en invertir el bit, como mostramos en la siguiente figura:

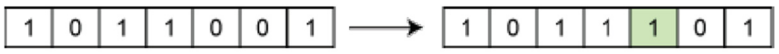


Figura 5.9.- Mutación por inversión.

- ***Mutación por intercambio repetido:*** Escoge dos genes del vector de elementos de forma arbitraria y los intercambia, provocando así una mutación en el genoma del individuo. Repite esta acción con cada gen de la tira. Las alteraciones que provoca no invalidan nunca el genoma. Podemos ver un ejemplo en la siguiente figura:

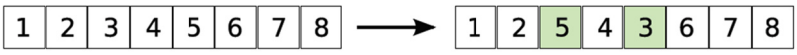


Figura 5.10.- Mutación por intercambio.

- ***Mutación uniforme:*** Se utiliza en genomas modelizados como vectores de números reales. Recorremos todo el vector de reales y para cada posición según cierta probabilidad lo cambiamos por otro aleatorio, siempre que esté dentro del rango adecuado a nuestro problema, como mostramos en la siguiente figura:

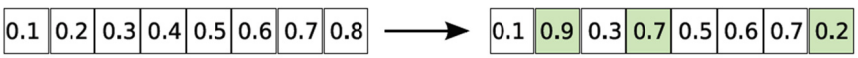


Figura 5.11.- Mutación uniforme

5.7.- PROCESO DE REEMPLAZO.

Habitualmente los algoritmos genéticos mantienen un tamaño de población constante, aunque existen otras posibilidades. Para mantener el tamaño de la población, los nuevos individuos creados mediante los operadores genéticos deben reemplazar a otros de la población anterior. En función de la cantidad de individuos reemplazados en la población anterior se consideran distintos tipos de algoritmos genéticos [Araujo Cervigón, 09]:

- **AG generacionales:** en estos algoritmos la población se renueva por completo de una generación a otra.

- **AG con estado estacionario:** la descendencia de los individuos seleccionados en cada generación se incluye en la población, reemplazando a algunos de los individuos de la población anterior. En este caso se conserva parte de la población de generación en generación.

Existen diversas posibilidades para establecer los criterios de reemplazo en los AG con estado estacionario. Los más usuales son los siguientes:

- ***Reemplazo de los padres:*** los hijos sustituyen a sus padres.

- ***Reemplazo aleatorio:*** los individuos a eliminar se eligen aleatoriamente. El número de individuos a eliminar viene dado por el tamaño de la descendencia, que a su vez queda definido por las tasas de cruces y el tamaño de la población.

- ***Reemplazo de los individuos peor adaptados:*** los individuos a eliminar se eligen aleatoriamente, pero sólo entre los que tiene el valor de adaptación más bajo. Valores bajos de adaptación se suelen considerar por debajo del 10% de la adaptación media.

- ***Reemplazo de individuos de adaptación similar:*** Cada nuevo individuo reemplaza a un individuo de la población anterior que tiene un valor de adaptación similar al suyo.

5.8.- TÉCNICAS PARA TRATAMIENTO DE PROBLEMAS CON RESTRICCIONES.

Cuando nos enfrentamos a la resolución mediante AG de un problema restringido, existe la probabilidad, debido a lo aleatorio de los mecanismos de creación y cruce de individuos, que un número de ellos, obtenga un alto índice al ser evaluado en la función de adaptación, pero viole las restricciones de nuestro problema.

Dada la importancia tecnológica de las aplicaciones que se corresponden con este tipo de problemas (de empaquetado, de distribuciones en planta, cálculo de trayectorias), se han desarrollado diversas técnicas de trabajo con este tipo de problemas [Tsang, 99], [Rossi, 06].

Las técnicas de tratamiento de restricciones en algoritmos evolutivos pueden agruparse en tres tipos básicos:

- ***Técnicas de penalización:*** son las más generales, ya que pueden aplicarse a cualquier problema con restricciones. Consisten en generar soluciones para el problema ignorando las restricciones y penalizar después en la evaluación a aquellas soluciones que no cumplan las restricciones del problema. A menudo la función de penalización depende del grado de la violación de la restricción, es decir, es alguna función (logaritmo, exponencial, etc.) del grado de la violación. En ocasiones también se hace que la penalización cambie a medida que avanza la evolución, de manera que al comienzo del proceso haya más permisividad de soluciones que violan las restricciones y a medida que se acerca el final de la evolución la penalización se incremente.

- ***Técnicas de reparación:*** Son aquellas en las que se busca algún mecanismo para corregir las soluciones que violan las restricciones del problema. Estas técnicas son específicas de cada problema y en general son difíciles de encontrar.

- ***Técnicas de codificación:*** Consisten en buscar una representación especial del problema que garantice que se cumplen las restricciones. Al igual que las técnicas de reparación, son específicas del problema y difíciles de encontrar en general.

5.9.- CRITERIOS DE TERMINACIÓN

Una vez definido por completo el AG, con sus mecanismos de codificación, selección, cruce, mutación,… necesitamos implementar algún criterio de terminación. Michalewicz [Michalewicz, 94] considera los siguientes criterios:

- Alcanzar un número máximo de generaciones.
- Alcanzar un número máximo de llamadas al cálculo de la adaptación.
- Llegar a una situación con escasas posibilidades de que se produzcan cambios significativos en la generación siguiente.

Las dos primeras opciones tienen el problema de que necesitamos tener una idea de la complejidad del problema y necesitaríamos una etapa de pruebas previa para estimar este parámetro.

El tercer criterio de terminación se basa en el avance que ha conseguido el algoritmo en un cierto número de generaciones. Este número de generaciones también puede ser un parámetro. Las posibilidades de progreso del algoritmo pueden estimarse en función de dos aspectos:

- El genotipo de los individuos: Los algoritmos cuya terminación dependen de este aspecto comprueban el número de genes que han convergido. Se considera que un gen ha convergido si un porcentaje predeterminado de la población tiene el mismo valor para ese gen. Si el número de genes que han convergido a un mismo valor supera cierto porcentaje, entonces el algoritmo termina.

- El valor de adaptación de los individuos: El algoritmo terminará si la mejora de la función de adaptación (la media de la población o el máximo) en un número predefinido de generaciones está por debajo de un cierto valor umbral dado por otro parámetro.

Dependiendo de nuestro conocimiento del problema, podemos utilizar otras condiciones de terminación. Por ejemplo, si se trata de un problema de búsqueda pura, sin optimización, el algoritmo puede terminar al encontrar la primera solución al problema, si es que sólo se necesita una de ellas. También en los problemas de optimización, podemos utilizar criterios basados en las necesidades particulares del caso.

6.- ESTADO DEL ARTE: SIMULACIÓN EN LA SHELL ECO-MARATHON.

6.1.- THE WORLD´S MOST FUEL EFFICIENT VEHICLE [Santin et al., 2007].

Este libro, publicado por el Instituto Federal Suizo de Tecnología (ETH) de Zurich, representa uno de los trabajos más completos acerca de la construcción y simulación de un vehículo de bajo consumo **propulsado por motor eléctrico**, para la carrera Shell Eco-Marathon. En él se hace un extenso repaso de los factores a tener en cuenta en el diseño de cada uno de los bloques constructivos de un vehículo de carreras de bajo consumo: la dirección, carrocería, frenos, ruedas, transmisión,…

La particularidad de este vehículo es que el combustible utilizado es hidrógeno, y mediante un generador eléctrico alimenta al sistema motriz formado por dos motores. En la figura 6.1 podemos ver el esquema del compacto sistema motriz y transmisión. Consta de un soporte que sirve de chasis para los motores, pinzas de freno y eje de la rueda y la transmisión de los motores hacia la rueda se realiza mediante una corona dentada unida solidariamente a ella. El acople/desacople de los motores a esta corona se realiza mediante dos servos, uno para cada motor.

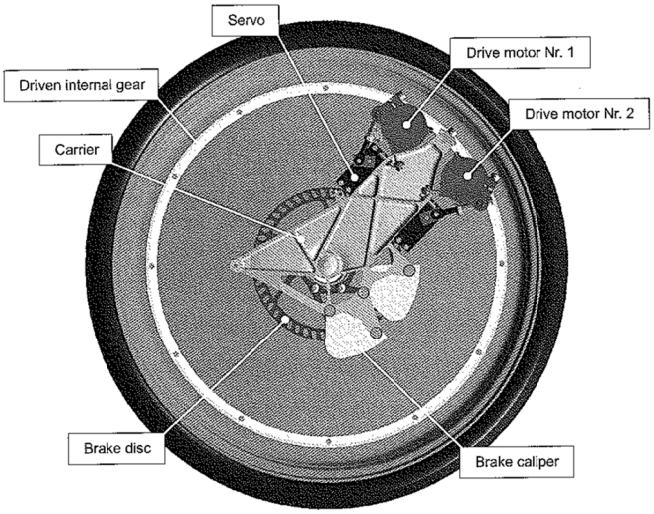


Figura 6.1.- Esquema del sistema motriz del PAC-Car II

Este sistema consta de 3 sensores de velocidad, uno para cada engranaje, de modo que en todo momento se va muestreando estas velocidades para que en el momento de acople, el deslizamiento relativo entre engranajes sea nulo. De esta forma se evita la posible rotura de algún diente del engranaje.

En un principio, durante la fase de diseño del sistema motriz y transmisión barajan como la estrategia más adecuada la de "stop and go", aunque no descartan la posibilidad de utilizar permanentemente un motor (siempre y cuando la pendiente no sea descendente) para mantener una velocidad de crucero alrededor de los 30 km/h, y dejar el otro motor sólo para las situaciones de demanda excepcional de par en la prueba, como aceleraciones o subida por pendientes ascendentes.

Por ello los dos motores utilizados serán diferentes:

- ***"Precious motor":*** Este motor, cuya eficiencia se muestra en la figura 6.2, utiliza escobillas con alto contenido en metales preciosos, lo que mejora notablemente su eficiencia, aunque el inconveniente es que la intensidad máxima queda limitada a 2 A, suficiente para conseguir el par necesario a velocidad crucero pero no para lanzamientos o pendientes de subida. El valor del par máximo queda por tanto limitado a 0.06 N·m.

- ***"Improved motor":*** Este segundo motor, utiliza escobillas de grafito, con eficiencia más baja que el "precious motor", pero permiten un empuje adicional en situaciones de alta demanda de par como lanzamientos desde parado o pendientes ascendentes. Podemos ver la gráfica de su eficiencia en la figura 6.2.

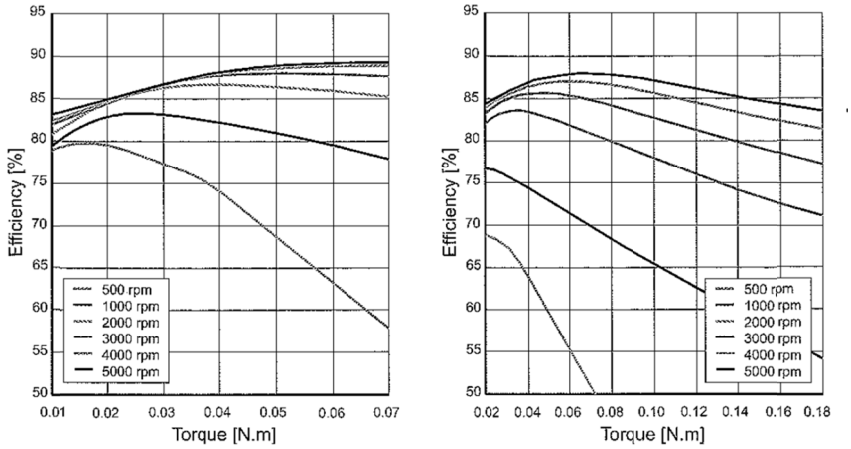


Figura 6.2.- Eficiencias medidas en ambos motores.

Para el control de ambos motores se diseñó en Matlab el diagrama de bloques representado en la figura 6.3. Las entradas al diagrama son el modo de conducción ("boost" o "cruise") y la velocidad del vehículo. Mediante el análisis de las mismas en el intérprete se genera una salida que permite permutar entre 3 modos de alimentación a los motores:

- ***Motores apagados.***
- ***Ambos motores encendidos.***
- ***Motor en modo crucero.***

Las salidas generadas por el diagrama son las intensidades correspondientes a cada motor.

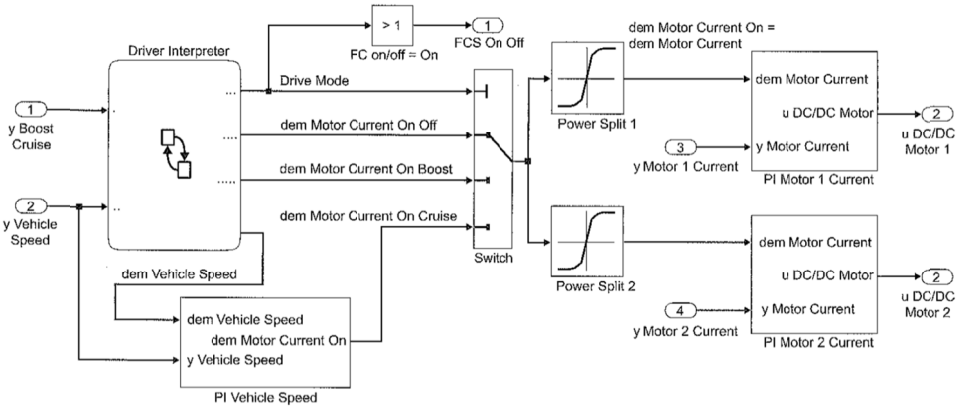


Figura 6.3.- Diagrama de bloques del control de corriente del motor.

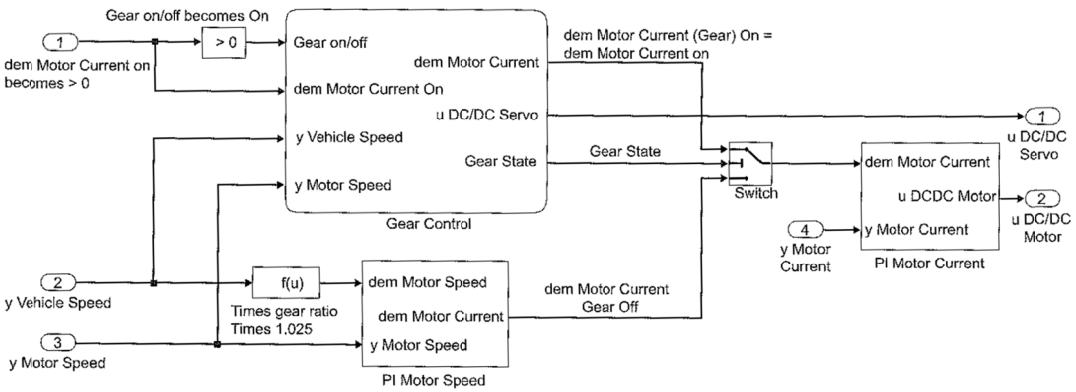


Figura 6.4.- Diagrama de control de velocidad del vehículo.

6.1.1.- Toma de datos en la pista de Ladoux.

Para el correcto ajuste de las simulaciones, se realizó una toma de datos en la pista de Ladoux, durante la consecución del récord mundial. Teóricamente, durante cada vuelta en este circuito se requieren dos fases de empuje extra ("boost"), como se puede ver en la figura 6.5 donde se muestra el esquema del circuito con las indicaciones acerca de la estrategia de carrera.

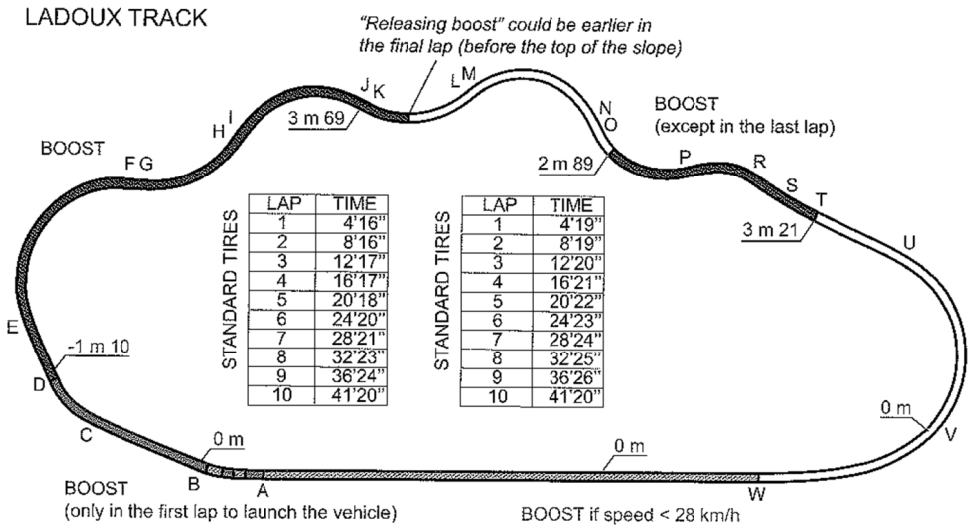


Figura 6.5.- Esquema del circuito de Ladoux con las indicaciones acerca de la estrategia de carrera para el piloto.

Las mediciones mostraron que la elección de dos motores independientes es bastante apropiada para la topografía de esta pista, ya que en algunos puntos, la demanda de par es bastante baja como para mantenerlos encendidos los dos, lo que provoca el desacople automático de uno de ellos, reduciendo bastante el consumo de potencia, como podemos ver en la figura 6.6.

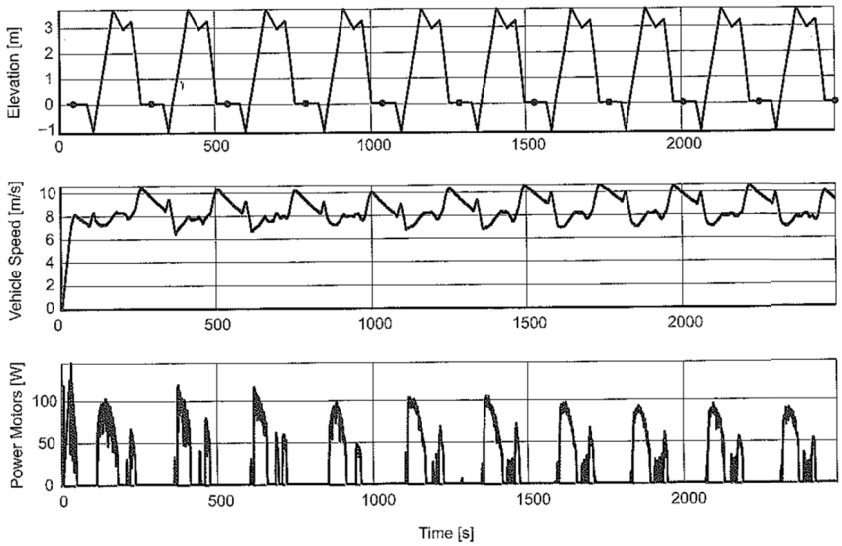


Figura 6.6.- Mediciones de velocidad del vehículo, potencia consumida por los motores durante la carrera.

6.1.2.- Optimización de la estrategia de carrera.

Al comienzo de este capítulo, el vehículo había sido optimizado en todos sus aspectos físicos: la aerodinámica era perfecta, los mecanismos funcionaban como un reloj suizo, el peso del vehículo había sido minimizado y la eficiencia del sistema motriz maximizada. Con todo esto el vehículo está preparado para mejorar las marcas de consumo de combustible, pero falta lo más importante: de nada sirve un vehículo excelente conducido por el mejor piloto sin una buena estrategia de carrera.

El término "estrategia de carrera" se refiere a la manera en que es utilizado el sistema motriz. Muchos de los equipos en esta competición utilizan la llamada estrategia de "stop and go". En este capítulo se debatirá la comparativa entre este tipo de estrategia y la de llevar el motor encendido durante toda la prueba. Para ello utilizan la optimización matemática, concluyendo que la estrategia de "stop and go" penaliza con un 7% más de consumo, en su caso, de un vehículo propulsado por pila de hidrógeno.

Para llevar a cabo esta optimización construyen un modelo basado en diagramas de bloques en Matlab, el cual incluirá los distintos sistemas del vehículo y permitirá realizar simulaciones, para posteriormente validar las hipótesis mediante la toma de datos durante una carrera. La optimización se llevará a cabo mediante un algoritmo basado en el principio de optimalidad de Bellman. En la figura 6.7 se puede ver el modelo Simulink global del PAC-Car II.

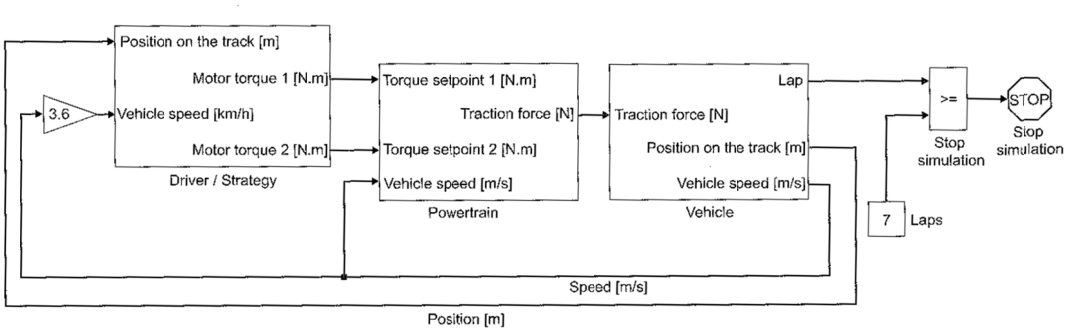


Figura 6.7.- Modelo general del vehículo PAC-Car II

Está formado por 3 grandes bloques: por una parte el controlador de los motores encargado del acople/desacople de los mismos a la transmisión, por otra todo el sistema de transmisión y motor y por último el modelo del vehículo que incluye el cálculo de todas las pérdidas por rozamiento y el esfuerzo en desniveles. La simulación se detiene cuando el coche completa 7 vueltas al circuito.

El autor de este capítulo Jerôme Bernard, enuncia que la ecuación que rige el comportamiento del vehículo es:

$$M_{car} \cdot \frac{dV_{car}(t)}{dt} = \sum_{i=1}^{5} F_i(t)$$

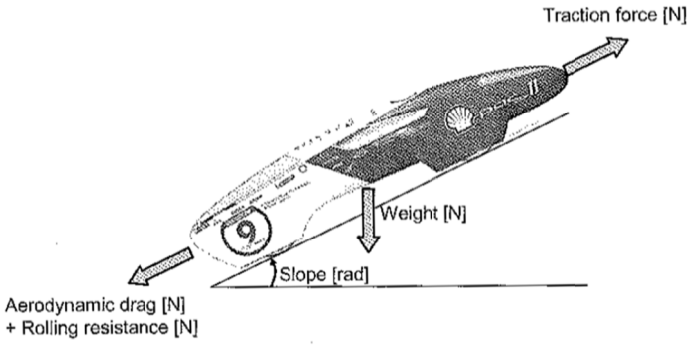


Figura 6.8.- Fuerzas que actúan sobre el vehículo.

Las fuerzas F_i que actúan sobre el vehículo, mostradas en la figura 6.9 pueden ser expresadas como:

$F_1 = F_{traction}(t)$ Representado la fuerza de tracción (N)

$F_2 = -\frac{1}{2} \cdot \rho_{air} \cdot A_f \cdot C_x \cdot V_{car}{}^2(t)$ Representando el arrastre aerodinámico (N).

$F_3 = -M_{car} \cdot g \cdot \cos(\alpha_{track}(t)) \cdot C_r$ Representando la resistencia a rodadura (N).

$F_4 = F_{bearings}$ Representando la resistencia a rodadura de los rodamientos (N).

$F_5 = -M_{car} \cdot g \cdot \cos(\alpha_{track}(t))$ Representando la resistencia por desniveles (N).

Donde:

M_{car} es la masa del vehículo en Kg.

ρ_{air} es la densidad del aire en kg/m^3.

A_f es el área frontal en m^2.

C_x es el coeficiente de arrastre aerodinámico.

g es la constante gravitacional en m/s^2.

C_r es el coeficiente de resistencia a rodadura.

α_{track} es el ángulo de desnivel de la pista en radianes.

En la figura 6.10 se muestra el modelo Simulink teniendo en cuenta todas las fuerzas que hemos citado anteriormente. En el modelo podemos distinguir los integradores necesarios para el cálculo de velocidad y espacio, las funciones de cálculo de fuerzas y la "lookup table" con la topografía correspondiente al circuito de Nogaro, pista donde tras realizar las simulaciones se verificará la bondad de la misma mediante la toma de datos durante una carrera.

Cabe citar que en este modelo se han despreciado tanto la influencia de la penalización por resistencia a rodadura en curvas, así como el efecto del viento, que se supondrá 0 y por tanto la única componente de la velocidad para el cálculo del arrastre aerodinámico será la propia del vehículo.

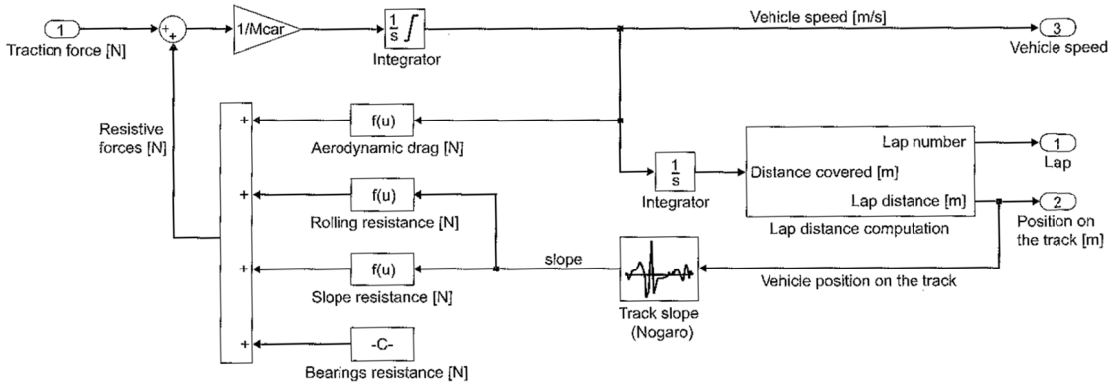


Figura 6.9.- Modelo de las fuerzas de arrastre.

El diagrama Simulink que representa al motor, tiene como entradas el par solicitado por el vehículo en cada momento sobre la pista y las revoluciones a las que gira. Mediante estas entradas y dos tablas con la eficiencia y potencia del motor, se calcula la potencia demandada por el mismo.

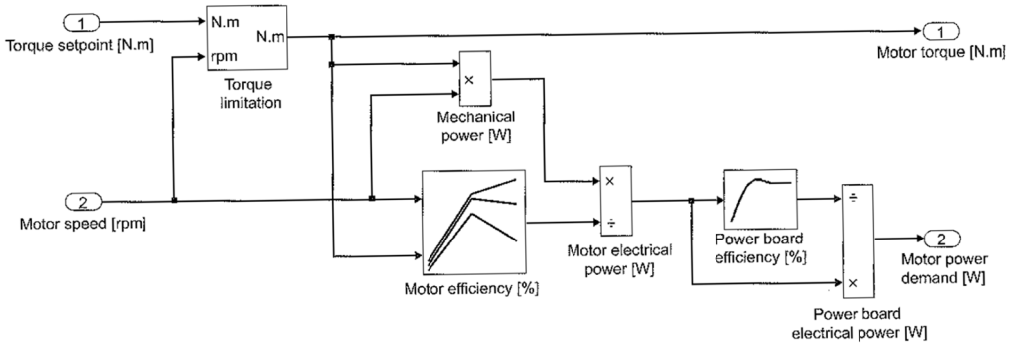


Figura 6.10.- Modelo del motor.

Esta potencia se irá acumulando para calcular el consumo en H_2 del vehículo durante una simulación de 7 vueltas al circuito. Se han obviado los diagramas referentes a las pilas de hidrógeno por no tener mucha relación con la tesina actual, así como los diagramas referentes a transmisión y ruedas por su simplicidad.

Para el cálculo de la optimización, Jerôme Bernard utiliza un algoritmo basado en el principio de optimalidad de Bellman, que enuncia que la solución óptima está compuesta de una serie de sub-soluciones óptimas.

La distancia recorrida por el vehículo está limitada por un valor mínimo y un máximo. En otras palabras, una vez que la restricción del tiempo necesario para acabar la prueba a una media de 30 km/h se cumple, el vehículo no puede ir a ningún lado. Un vehículo que comienza la prueba en t=0 y utiliza toda su potencia disponible, con toda seguridad finalizará la carrera antes de tiempo (línea gris en la figura 6.11). Supongamos ahora que el vehículo comienza tarde la prueba y una vez más utiliza toda su potencia disponible para cruzar la línea de meta justo en el tiempo de finalización de la prueba, utilizando por tanto la restricción del tiempo máximo para acabar con la velocidad media mínima establecida (línea negra en la figura 6.11). Esta figura representa las dos posiciones límite y entre ellas, el espacio de los infinitos estados intermedios.

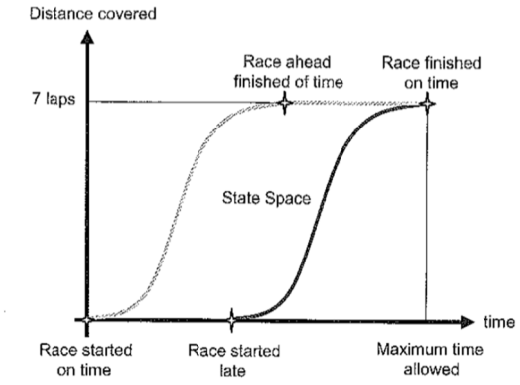


Figura 6.11.- Posiciones límite del vehículo.

En la figura 6.12 se muestra este espacio de estados muestreados mediante una rejilla de puntos. Cada uno de ellos representa un estado intermedio entre los límites.

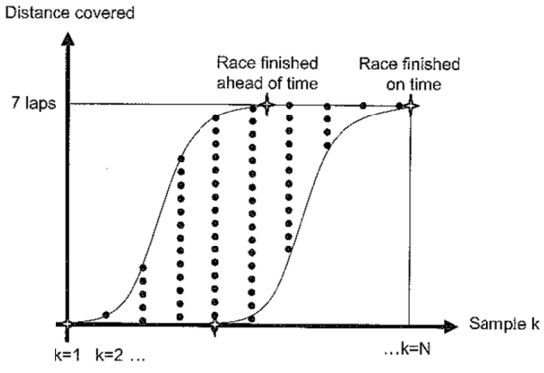


Figura 6.12.- Espacio de estados muestreado.

El próximo paso es determinar los puntos óptimos (llamados estados) que deben ser cruzados. Este trabajo es realizado por el algoritmo de programación dinámica. La meta es ir desde el estado en ***k* = N** al estado en ***k* = 1** consumiendo la mínima cantidad posible de hidrógeno. En la figura 6.13 se muestra el principio de la programación dinámica donde cada vector se caracteriza por la velocidad, fuerza de tracción y consumo de hidrógeno para ir de un estado a otro. Cada estado es caracterizado por la mínima cantidad de hidrógeno consumida.

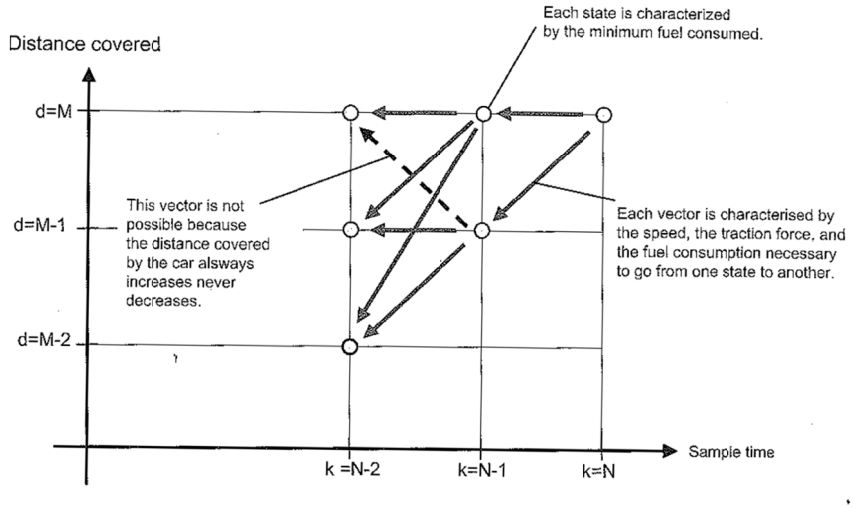


Figura 6.13.- Principio de programación dinámica.

La ecuación recurrente para evaluar el mínimo consumo de hidrógeno es:

$$m_{H_2}^{min}(k,d) = \min_{i \in [0,m]} (m_{H_2}(k-1,i-1) + m_{H_2}\{(k-1,i-1) \rightarrow (k,i)\})$$

La programación de este algoritmo es fácil en Matlab. El principal problema es el tiempo necesario para calcular la solución. La generación de muchos estados implicaría mayor precisión, pero el costo computacional crecería exponencialmente

6.1.3.- Validación del modelo y resultados en la carrera.

Con las simulaciones realizadas y una vez que se obtuvieron las soluciones óptimas, pasaron a validar los resultados mediante la toma de datos en el circuito de Nogaro en Francia. En la figura

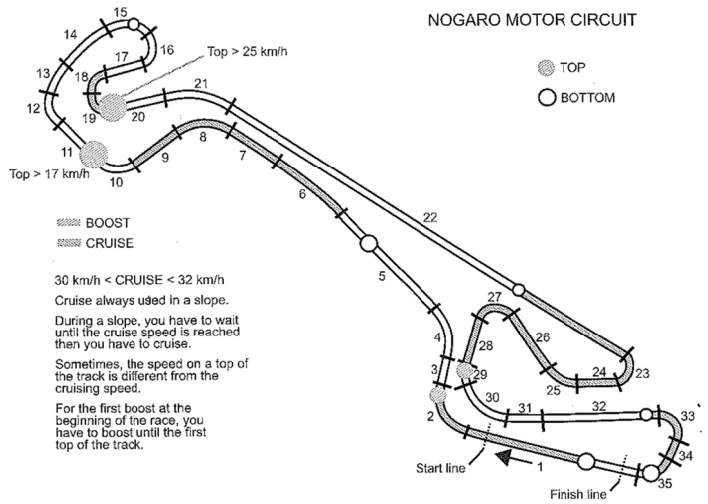


Figura 6.14.- Esquema del circuito de Nogaro con las indicaciones para el piloto.

Durante la carrera de Nogaro que tuvo lugar los días 21-22 de Mayo del 2005, el piloto siguió casi al pie de la letra las indicaciones acerca de la estrategia de carrera que arrojaron las simulaciones, a excepción de una reducción en la velocidad (figura 6.15) aconsejable para evitar el vuelco en los tramos 15-16 del circuito (figura 6.14). Situación esta que evidentemente no fue contemplada por el algoritmo.

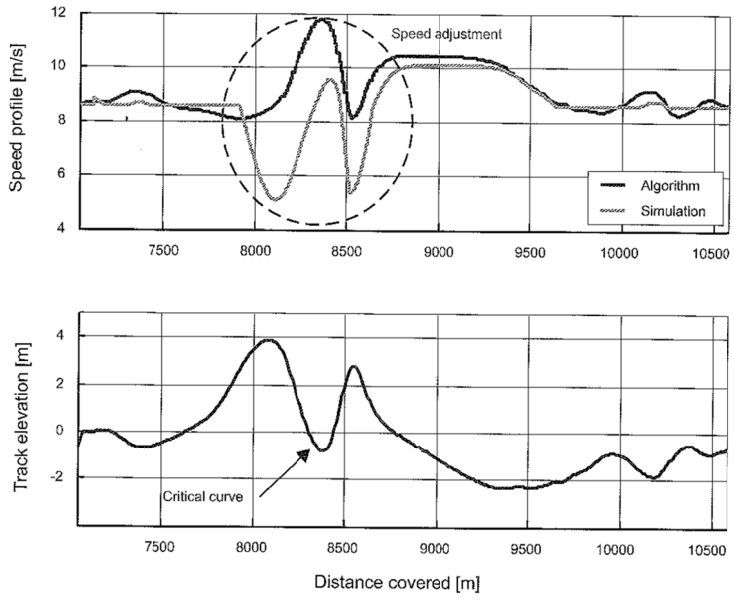


Figura 6.15.- Modificación de la estrategia para evitar el vuelco.

En la figura 6.16 se muestran los datos procedentes del sistema de telemetría, correspondientes a la grabación de velocidades en las distintas vueltas de la prueba, y su comparación con las velocidades simuladas, donde se puede apreciar la estrecha relación entre el modelo y la realidad.

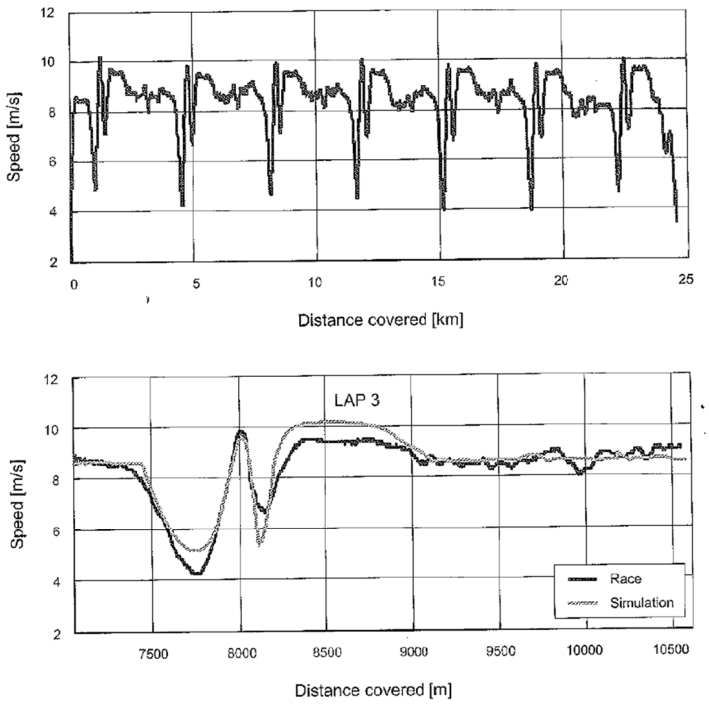


Figura 6.16.- Comparativa entre velocidades reales y simuladas en una vuelta al circuito de Nogaro.

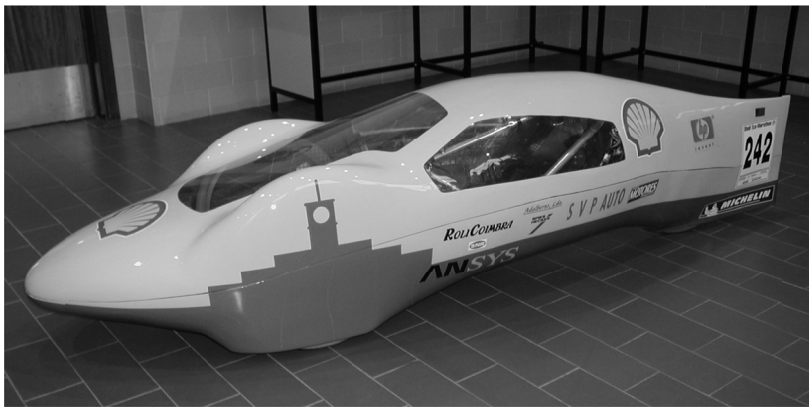
En conclusión, con esta estrategia de carrera obtenida mediante optimización, lograron reducir el consumo, obteniendo una marca aún mejor que la prevista en las simulaciones, y reduciendo en un 7% el consumo con respecto al medido mediante la estrategia de "stop and go" (figura 6.17).

	H_2 consumption [g]	Equivalent consumption [km/liter of gasoline]
Simulation	1.79	3,768
Real race	1.75	3,836
Stop-and-go strategy	1.92	3,514

Figura 6.17.- Comparativa entre consumo simulado, real y mediante estrategia "stop and go" en el circuito de Nogaro.

6.2.-* SIMULATION OF THE PERFORMANCE OF AN EXTRA-LOW FUEL CONSUMPTION VEHICLE *[Figueiredo, 2002].

Pedro de Figueiredo, profesor de la Universidad de Coímbra, propone un método para predecir el consumo muy diferente del tratado en el anterior apartado. El algoritmo que desarrolla está basado en un método integral energético. Su primera versión data de 1998, antes incluso de la construcción del vehículo con el que participarían por primera vez en la Shell Eco-Marathon de 1999, en el circuito de Paul Ricard (Francia). En un principio se desarrolló con la idea de que sirviera de guía de diseño, simulando el comportamiento del vehículo de bajo consumo, identificando de esta forma los parámetros físicos críticos para conseguir una buena marca. En la figura 6.18 se muestra el "Eco Vehículo", perteneciente al equipo formado por profesores del Departamento de Ingeniería Mecánica de esta universidad.



Como citábamos en el párrafo anterior, el algoritmo está basado en cálculo integral energético. Tiene en cuenta dentro de cada punto de la topografía del circuito:

- Energía cinética del vehículo.
- Energía potencial del vehículo.
- Energía disipada por rozamiento a rodadura.
- Energía disipada por la resistencia a rodadura en curvas.
- Energía disipada por el rozamiento aerodinámico.
- Energía disipada en la frenada.
- Trabajo realizado por la fuerza motriz generada en el motor.

Para el modelado de las curvas de par y consumo específico del motor utiliza funciones ajustadas mediante polinomios.

6.2.1.- Algoritmo.

Cuando el vehículo se desplaza desde un punto ***i*** del circuito a un punto ***i+1*** hay un cambio en la energía total del vehículo que es igual al trabajo de las fuerzas no conservativas actuando sobre el mismo. Este cambio se expresa analíticamente por la ecuación (1):

$$E_{i+1} - E_i = \sum_{j=1}^{n} \vec{F}_j \times (\vec{s}_{i+1} - \vec{s}_i) \qquad (1)$$

En la energía total del vehículo se consideran la energía potencial, cinética debida al movimiento lineal del vehículo y cinética debida a la rotación de las ruedas:

$$E_i = M_t g h_i + \frac{1}{2} M_t v_i^{\,2} + \frac{1}{2}\left(\sum_{k=1}^{n} I_{w,k} \frac{4}{D_{w,k}^{\,2}} \right) v_i^{\,2} \qquad (2)$$

Donde:

g es la aceleración de la gravedad (9.8 m/s^2).
h_i es la altitud del punto ***i*** del circuito en m.
I_w es el momento de inercia de cada rueda respecto a su centro de rotación en kg · m^2.
D_w es el diámetro de cada rueda en m.
M_t es la masa total del vehículo más la del piloto en kg.
v_i es la velocidad del vehículo en el punto ***i*** en m/s.

Las fuerzas no conservativas consideradas en la ecuación (1) son la resistencia a rodadura, la componente de la fuerza de resistencia aerodinámica paralela al eje longitudinal del vehículo, la fuerza de resistencia a rodadura debida a las curvas, la fuerza de frenado y la fuerza motriz generada por el motor y transmisión. Todas estas fuerzas son paralelas al desplazamiento del vehículo.

El módulo de la resistencia a rodadura en un punto ***i*** es dado por ecuación (3):

$$\left|\boldsymbol{F}_{R,i}\right| = C_{r,i} M_t g \left[\cos\alpha_i (\cos\beta_i + \frac{v_i^{\,2}}{R_i\, g} \sin\beta_i) \right] \qquad (3)$$

Donde:

$\boldsymbol{C_r}$ es el coeficiente global de resistencia a rodadura en el punto i.
$\boldsymbol{R_i}$ es el radio de la curva en el punto i del circuito expresado en m.
$\boldsymbol{\alpha_i}$ es el ángulo de inclinación longitudinal de la pista expresado en radianes
$\boldsymbol{\beta_i}$ es el ángulo de peralte de la pista en radianes.

En el cálculo del coeficiente de resistencia a rodadura global en la ecuación (4), son considerados 4 términos correspondientes a:

- Resistencia a rodadura de las ruedas.
- Resistencia a rodadura debido a los momentos que actúan sobre cada eje, generados por la resistencia aerodinámica de cada rueda.
- Resistencia a rodadura de los rodamientos de cada rueda.
- Resistencia a rodadura del sistema de embrague.

$$C_{r,i} = C_{rT} + \sum_{j=1}^{3} C_{r,eq,wj,i} \qquad (4)$$

Todos los coeficientes correspondientes a esas resistencias fueron medidos experimentalmente. El coeficiente del momento de fricción aerodinámico depende evidentemente de la velocidad tangencial de las ruedas y fue determinado mediante una regresión por el método de los mínimos cuadrados, de los datos tomados experimentalmente. El polinomio de 2º grado obtenido corresponde a la ecuación (5), donde los valores de los coeficientes $\boldsymbol{a_i}$ se pueden ver en la figura 6.19.

$$T_{f,wj,i} = a_{2,wj} v_i^{\,2} + a_{1,wj} v_i + a_{0,wj} \qquad (5)$$

$$C_{r,eq,wj,i} = \frac{2T_{f,wj,i}}{D_w M_t g} \qquad (6)$$

El módulo de la componente del arrastre aerodinámico paralela al desplazamiento del vehículo en un punto ***i,*** $F_{Ax,i}$ se determina mediante la ecuación (7), donde se admite la inexistencia de viento, lo que indica que el aire no tiene movimiento relativo con respecto al suelo.

$$\left|\vec{F}_{Ax,i}\right| = C_{x,i} A_f \frac{1}{2} \rho_a v_i^{\,2} \qquad (7)$$

Siendo:

C_x el coeficiente de arrastre aerodinámico.
A_f el área frontal del vehículo en m^2.
ρ_a la densidad del aire en kg/m^3.

El módulo de la fuerza radial que afecta a las ruedas en el plano de la pista, cuando el vehículo está tomando una curva en un punto ***i*** , $F_{Y,i}$ se calcula mediante la ecuación (8) donde β_i es el ángulo de peralte de la curva en un punto ***i*** y R_i es el radio de la curva en ese mismo punto.

$$\left|\boldsymbol{F}_{Y,i}\right| = \left|M_t g \left(\frac{v_i^{\,2} \cos\beta_i}{R_i g} - \sin\beta_i\right)\right| \qquad (8)$$

El módulo de la resistencia a rodadura durante una curva en las ruedas en un punto ***i,*** $F_{TCD,i}$, se calcula mediante la ecuación 9 donde C_α es la compresibilidad de la rueda en curvas expresada en (N/rad).

$$\left|\vec{F}_{TCD,i}\right| = \left|\vec{F}_{Y,i}\right| \sin\left(\frac{\left|\vec{F}_{Y,i}\right|}{C_\alpha}\right) \qquad (9)$$

La fuerza de frenado debido a la acción de los frenos en un punto i se designa por $F_{b,i}$, y el módulo de la fuerza motriz en un punto **i**, $F_{P,i}$ es calculada mediante la ecuación (10).

$$\left|\vec{F}_{P,i}\right| = \frac{2T_{b,i}i_{tr}\eta_{tr}}{D_w} \qquad (10)$$

Donde:

T_b es el par medido al freno del motor en N·m.
i_{tr} es la relación de transmisión.
η_{tr} es el rendimiento de la transmisión.

Sustituyendo las expresiones anteriores en la ecuación (1), queda escrita como la ecuación (11):

$$E_{i+1} - E_i = \left(-\left|\vec{F}_{R,i}\right| - \left|\vec{F}_{Ax,i}\right| - \left|\vec{F}_{TCD,i}\right| - \left|\vec{F}_{b,i}\right| + \left|\vec{F}_{P,i}\right|\right)\left(s_{i+1} - s_i\right) \qquad (11)$$

La energía total gastada en una vuelta al circuito considerando que la velocidad del vehículo al finalizar la vuelta es la misma que al principio es igual a la suma de los trabajos de las fuerzas no conservativas y se calcula mediante la ecuación (12).

$$E_{lap} = \sum_{i=0}^{n-1}\left(F_{R,i} + F_{Ax,i} + F_{TCD,i} + F_{b,i}\right)\left(s_{i+1} - s_i\right) \qquad (12)$$

Esta energía consumida por los trabajos de las fuerzas no conservativas debe ser reintroducida por la acción del motor y la transmisión.

La energía aportada por el sistema motriz en un desplazamiento $(S_{i+1} - S_i)$ se calcula mediante la ecuación (13):

$$E_{p,i} = \frac{2T_{b,i} i_{tr} \eta_{tr} (s_{i+1} - s_i)}{D_w} \qquad (13)$$

El trabajo producido por esta fuerza motriz en un desplazamiento $(S_{i+1} - S_i)$ del vehículo puede ser determinado mediante la ecuación (14):

$$W_{b,i} = \frac{2T_{b,i} i_{tr} (s_{i+1} - s_i)}{D_w} \qquad (14)$$

El término de esta ecuación $T_{b,i}$, correspondiente al par del motor fue medido experimentalmente como función de la velocidad de giro del cigüeñal y fue ajustado mediante una curva polinómica donde T_b está expresado en (N·m) y n en r.p.m:

$$T_{b,i} = a_{3T} n_i^{\,3} + a_{2T} n_i^{\,2} + a_{1T} n_i + a_{0T}$$

La masa de combustible consumida en un desplazamiento $(S_{i+1} - S_i)$ del vehículo cuando el motor está en marcha se calcula mediante la ecuación (15):

$$m_{f,i}[\mathrm{kg}] = \frac{W_{b,i}[\mathrm{J}] \times bsfc_i[\mathrm{g/kW.h}]}{3.6 \times 10^9} \qquad (15)$$

De una forma similar a la función que dictamina el par motor, el término de esta ecuación $bsfc_i$, correspondiente al consumo específico fue medido experimentalmente como función de la velocidad de giro del cigüeñal y fue ajustado mediante una curva polinómica donde $bsfc$ está expresado en gr/Kw·h y n en r.p.m:

$$bsfc_i = a_{3bsfc} n_i^{\,3} + a_{2bsfc} n_i^{\,2} + a_{1bsfc} n_i + a_{0bsfc}$$

6.2.2.- Resultados y conclusiones.

La tabla representada en la figura 6.19 tiene los valores de los datos correspondientes a las constantes y coeficientes utilizados como entradas a las ecuaciones expuestas en el apartado anterior. Representan las condiciones más favorables que se presentaron en 18ª edición de la Shell Eco–Marathon en el año 2002.

M_v /kg	43.3		h_{max} /m	98.757
M_p /kg	45.0		p_a /Pa	1.0024E+5
M_t /kg	88.3		ρ_f /(kg/m^3)	750
A_f /m^2	0.359		$a_{2,w1}$, $a_{2,w2}$	2.2532E-4
c_x	198		$a_{1,w1}$, $a_{1,w2}$	7.1364E-4
b	-0.50		$a_{0,w1}$, $a_{0,w2}$	-1.5720E-4
L /m	2.650		$a_{2,w3}$	2.4807E-4
T_c /K	301.15		$a_{1,w3}$	1.1523E-3
T_t /K	308.65		$a_{0,w3}$	6.3394E-3
C_{rT}	0.00161		a_{3T}	0
D_w /m	0.498		a_{2T}	-5.857E-8
I_w /(kg.m^2)	0.046		a_{1T}	6.329E-4
i_{tr}	15.81		a_{0T}	-4.976E-1
η_{tr}	0.97		a_{3bsfc}	6.262E-9
C_α /(N/rad)	22000		a_{2bsfc}	-8.276E-5
$\bar{v}$ /(km/h)	27.29		a_{1bsfc}	3.149E-1
h_{min} /m	93.097		a_{0bsfc}	2.430E+1

Figura 6.19.- Tabla con los datos de entrada para simulaciones

En la figura 6.20 se puede apreciar la evolución de la altitud topográfica medida desde la línea de meta en el circuito de Nogaro. De acuerdo a ella la máxima pendiente que se presenta está entorno al 3% de bajada en el tramo que va de los 1100m a los 1230m aprox.

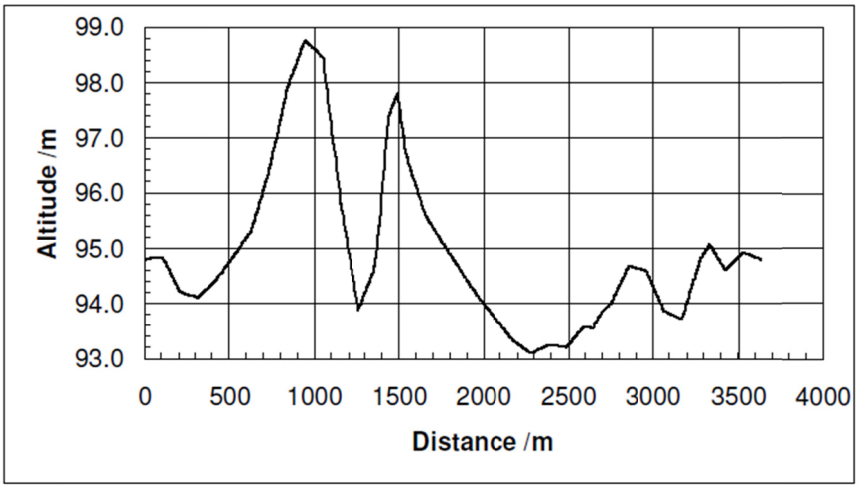


Figura 6.20.- Evolución de la altitud en el circuito de Nogaro.

En la figura 6.21 se detalla la información relativa a los radios de curvatura y ángulos de peralte de las mismas correspondientes al circuito de Nogaro.

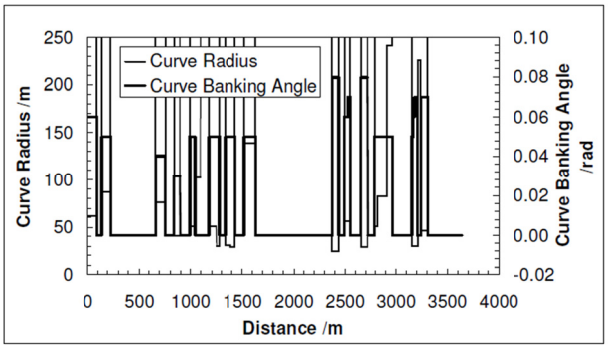


Figura 6.21.- Información topográfica relativa a los radios de curvas y ángulos de peralte del circuito de Nogaro

La figura 6.22 representa los datos que arrojaron las simulaciones, donde se puede ver claramente los puntos donde el motor realiza las arrancadas, con el par proporcionado en cada momento y la velocidad instantánea del vehículo para todos los puntos de la vuelta al circuito.

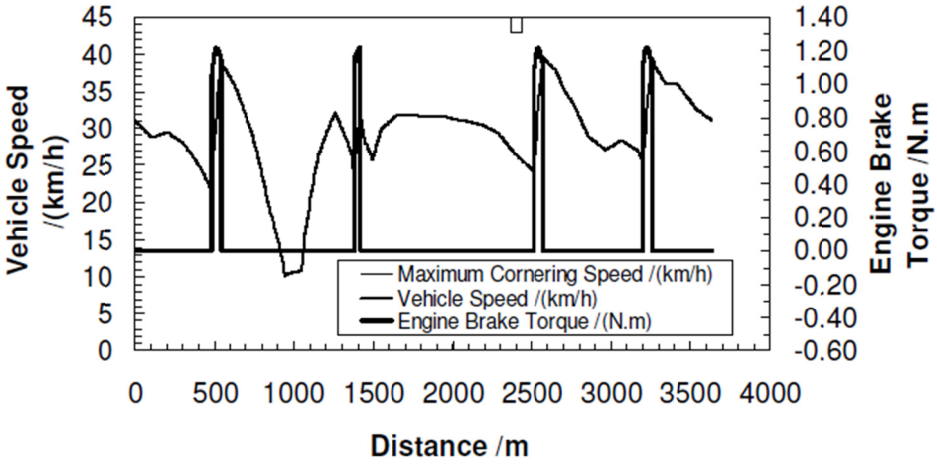


Figura 6.22.- Predicción de la velocidad y par suministrado durante una vuelta al circuito.

Se aprecia la tendencia en estas simulaciones a **entrar a las pendientes descendentes a una velocidad mínima** para aprovechar posteriormente la pendiente para acelerar. Esto parece ser una constante dentro de las estrategias de "stop and go". Reforzando esta hipótesis vemos que se hace una arrancada muy pequeña antes del punto 1500m donde comienza otra pendiente descendente. Una arrancada muy larga en ese punto hubiera desperdiciado gran cantidad de energía.

La figura 6.23 representa los datos que arrojaron las simulaciones acerca de la evolución de las fuerzas no conservativas que actúan sobre el vehículo. Podemos destacar que la resistencia a rodadura en curvas puede llegar a ser bastante importante, igualando en algunos casos al arrastre aerodinámico, como por ejemplo alrededor de la distancia 2700m. Por tanto, **sería un error despreciarla** como en el simulador del ETH. De acuerdo a estos datos, implementaremos en nuestro simulador un módulo para el cálculo de esta resistencia de cara a hacerlo más preciso.

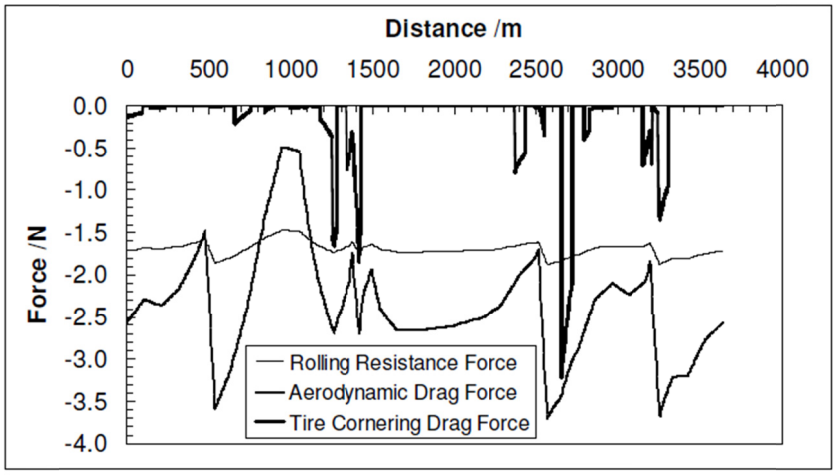


Figura 6.23.- Predicción de la evolución de las fuerzas no conservativas que actúan sobre el vehículo.

Por último la tabla mostrada en la figura 6.24 muestra la comparativa entre la predicción realizada por las simulaciones y los datos obtenidos durante la prueba.

Actual Performance Results /(km/l)	1734
Simulation Performance Results /(km/l)	1982
Actual/Simulation Performance Ratio	0.8748

Figura 6.24.- Comparativa entre kilometraje simulado y real

Según los datos expuestos en la tabla 6.24, la diferencia entre la calidad de la predicción se puede tomar en torno al 87%. Como se puede apreciar, las simulaciones son ligeramente optimistas El autor del artículo atribuye esta diferencia a varios factores:

- La no consideración del efecto del viento en las simulaciones.
- La diferencia de temperaturas entre las pruebas del motor y las condiciones durante la prueba, que en el caso de la prueba son más bajas aumentando el consumo específico.
- La viscosidad del aceite que al ser la temperatura de funcionamiento menor crece aumentando las pérdidas por rozamiento viscoso del mismo.
- La inevitable discrepancia entre la estrategia prevista y la que realiza el piloto.
- Pequeños deslizamientos de los neumáticos por el comportamiento dinámico de la transmisión.

7.- MODELO MATLAB DEL VEHÍCULO.

En este capítulo se hará una breve descripción de los bloques en los cuales se compone el modelo del vehículo. Posteriormente en el capítulo 8, se hará una descripción más detallada de la dinámica del vehículo y de los factores tenidos en cuenta a la hora de la creación de los distintos bloques.

La razón del modelado del vehículo no es otra que la de simular el comportamiento dinámico del mismo para proporcionar una estimación del consumo de combustible. Por definición, un modelo es falso, ya que no representa correctamente la realidad, sino sólo una parte de ella. Este hecho nos plantea una cuestión: ¿qué precisión debemos alcanzar en el modelo? Por una parte, si el modelo es muy preciso será muy complicado su uso y requerirá mucho tiempo para simular la prueba. En oposición con esto, si es muy simple, la diferencia entre los datos que nos arroja y la realidad será excesiva, llevándonos a conclusiones erróneas. Han sido varias las simplificaciones hechas en este modelo (ver apartado 7.1), aunque todas ellas han sido escogidas con cuidado para no reducir sustancialmente su robustez.

Una de las ventajas de la utilización de Matlab/Simulink, es la facilidad con la que se puede crear un modelo relativamente complejo como el que nos ocupa de una forma muy intuitiva. Con sólo arrastrar y conectar bloques podemos hacer en minutos un trabajo que mediante programación escrita nos llevaría horas o incluso días.

Si bien en cualquier lenguaje de programación necesitaríamos crearnos (o buscar en alguna librería) la mayoría de funciones matemáticas y añadirlas en nuestro código, en Matlab, al ser un software creado específicamente para el mundo científico, se dispone de vastas librerías de funciones preparadas para ser llamadas dentro del código de nuestro programa. Sólo hay que hacer una breve búsqueda por la ayuda para determinar los parámetros de entrada de las mismas y utilizarlas. La utilización de este software nos ahorra gran cantidad de trabajo, ya que nos permite centrarnos únicamente en la resolución de nuestro problema, sin perder tiempo intentando “reinventar la rueda”.

Podría parecer que la ejecución de un programa creado en Matlab sería más lento que el mismo creado en otro lenguaje habitual como C++, C#, VB,…, ya que en el 2º caso, sólo tendríamos nuestro programa cargado en memoria, mientras que el 1º además de nuestra creación, estará cargado Matlab consumiendo recursos. Dependiendo del caso esta afirmación podría ser cierta, pero en casos en los que abunde el trabajo con matrices, Matlab es mucho más rápido ya que es lenguaje muy optimizado para este tipo de cálculos, con unos algoritmos muy eficientes. Nuestro caso no es una excepción y como veremos en el caso de la optimización con algoritmo genético, las matrices de la población de individuos que manejará el programa pueden llegar a ser inmensas (del orden de 500 x 400 o superiores).

De cualquier modo, este modelo es susceptible de ser ampliado y mejorado en futuras versiones del mismo, debido a la modularidad con la cual será construido mediante diagramas de bloques en Matlab/Simulink.

7.1.- SIMPLIFICACIONES REALIZADAS EN EL MODELO.

Las simplificaciones que hemos realizado a la hora del modelado matemático han sido las siguientes:

1) Supondremos que el vehículo gira por el circuito en todo momento por el centro de la pista. De este modo no tendremos en cuenta la trazada que el piloto en la realidad haga en las curvas. Esta simplificación es necesaria para que el camino que realice el vehículo coincida con la trayectoria del circuito introducido. De otra forma sería muy complejo el cálculo de los radios de curvatura de la trayectoria en los diferentes puntos.

2) El centro de gravedad en nuestro modelo estará situado a ras de suelo, y de esta forma evitaremos independizar el cálculo de rozamientos distintos dependiendo de si la rueda se encuentra en el interior o exterior de la curva.

3) A efectos de cálculo aerodinámico sólo contemplaremos el efecto frontal y posterior de empuje del viento. Los efectos laterales serán despreciados, ya que sólo contribuirían a modificar ligeramente la carga de los neumáticos contra el suelo.

4) Debido a los desniveles tan pequeños con que cuentan la mayoría de circuitos, mantendremos constante el valor de las normales únicamente para el cálculo de rozamientos de neumáticos y rodamientos de la rueda. A este respecto se realizaron pruebas con el simulador que indicaron la aptitud de esta modificación. Si bien la diferencia de valores de kilometraje obtenidos entre ambos casos era despreciable, el ahorro en tiempo de cálculo fue sustancial. **Sólo tendremos en cuenta este desnivel para el cálculo de la fuerza de subida o bajada por desnivel del vehículo**.

5) No incluiremos la influencia de las condiciones meteorológicas en el rendimiento del motor, al no tener pruebas realizadas en banco de potencia para distinta temperatura ambiente, humedad y presión atmosférica. Tampoco se ha tenido en cuenta la pequeña caída de prestaciones que sufre el motor cuando se calienta en exceso durante la prueba.

6) Se utilizará un rendimiento fijo del sistema de transmisión independiente del régimen de giro del mismo, al no disponer en el momento de realización de la tesina de las curvas de rendimiento de la misma.

7.2.- BLOQUES PRINCIPALES DEL MODELO.

Los bloques principales de los que consta el modelo son los siguientes:

1) **Modelo general:** El modelo general Simulink es el bloque que engloba a todos los demás modelos que describiremos a continuación. Hay dos variantes en función de si va a ser utilizado por el algoritmo genético básico o por el avanzado que se describirán en el apartado 9. Sus salidas son el tiempo utilizado para la simulación, el nº de arrancadas efectuadas por el vehículo, el consumo del motor durante la simulación, la velocidad y espacio recorrido.

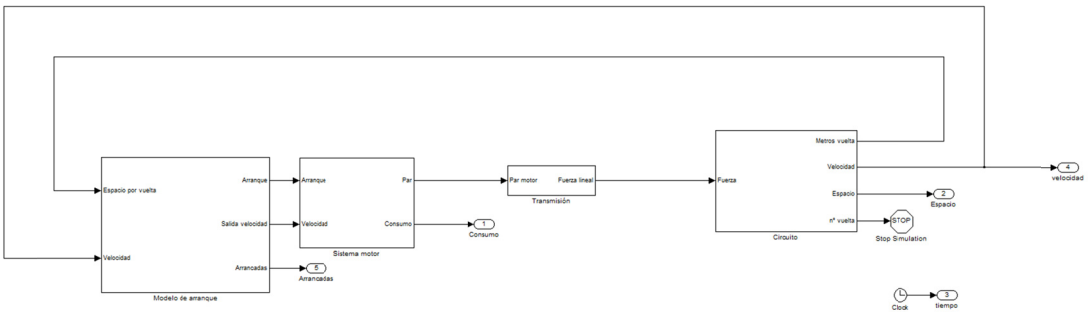


Figura 7.1.- Modelo general Simulink.

2) **Modelo de arranque:** Podemos definir este bloque como el "cerebro" de nuestro sistema. Es el que define mediante los parámetros establecidos y sus entradas, los puntos de arrancada, así como su duración o criterio de parada del motor. Existen varios bloques de modelo de arranque para las diferentes estrategias de optimización escogidas: estrategia con un único rango óptimo de velocidades, estrategia con velocidades máximas y mínimas dependientes del punto del circuito, estrategia de arranque y parada por punto kilométrico en lugar de la velocidad… Genéricamente sus entradas serán la velocidad y el kilometraje recorrido en el circuito y su salida será una señal binaria que indicará la parada o arranque del motor. En la siguiente imagen mostramos uno de estos bloques:

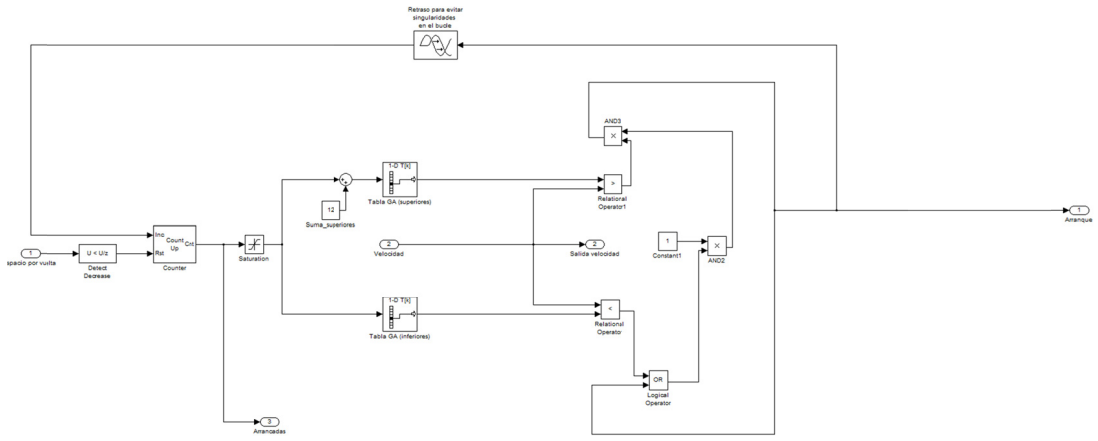


Figura 7.2.- Modelo de control de arranque del motor del vehículo.

3) **Modelo del motor:** En este diagrama quedarán definidas las curvas de par y consumo específico del motor. Este bloque toma como entradas la velocidad actual del vehículo y la señal binaria de arranque procedente del sistema de arranque. Su utilidad reside por un lado en contabilizar el consumo durante la simulación y generar el valor del par de torsión dependiendo de la velocidad con el cual alimentaremos el siguiente diagrama de la transmisión.

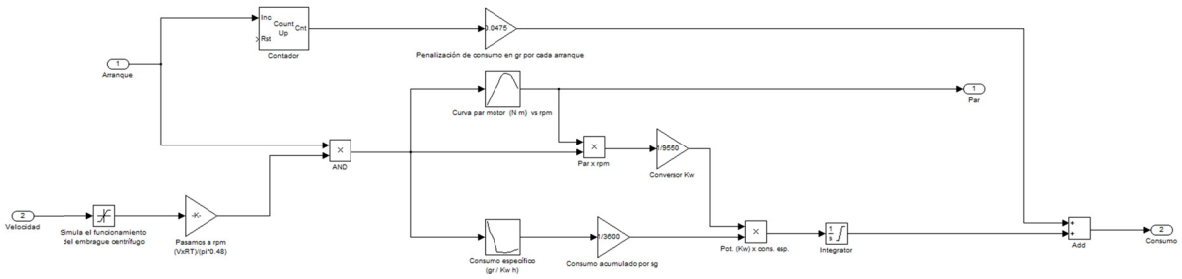


Figura 7.3.- Modelo del motor del vehículo.

4) **Modelo de la transmisión:** Este diagrama es muy simple debido a que nuestro vehículo cuenta con solo una sola velocidad cuya relación de transmisión es fijada previamente a la prueba. Toma como entrada el valor del par generado en el sistema del motor y da como salida la fuerza de impulso lineal del vehículo.

Figura 7.4.- Modelo de la transmisión del vehículo.

5) **Modelo general del circuito:** Este bloque toma como entrada la fuerza lineal proveniente del sistema de la transmisión. Mediante esta fuerza lineal se puede calcular la aceleración y velocidad del vehículo en cada momento, teniendo en cuenta el punto del circuito en el que nos encontramos. Es dentro de este diagrama donde se incluyen las pérdidas por rozamiento a rodadura, fricción aerodinámica, y el esfuerzo o impulso que afecta al vehículo dependiendo del desnivel de la pista.

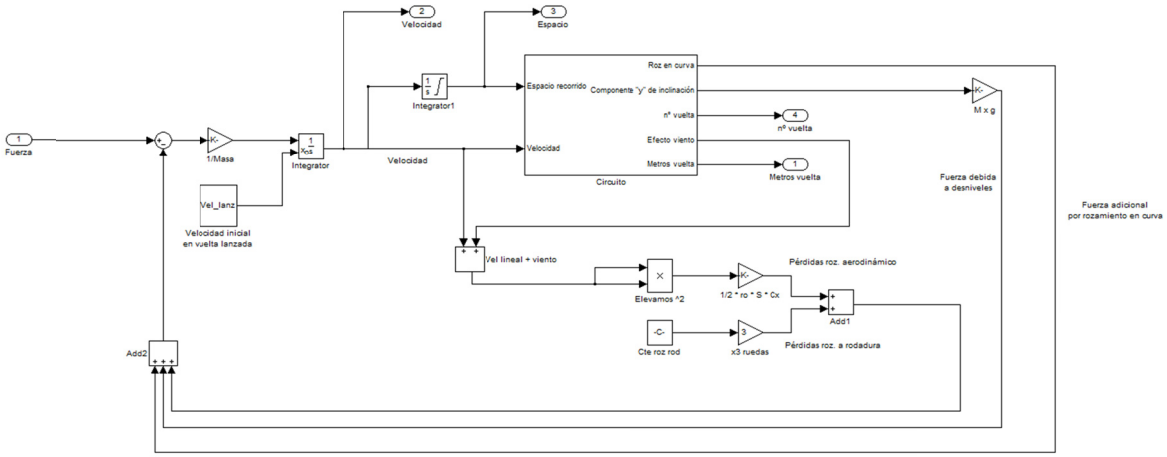


Figura 7.5.- Modelo general del circuito con las distintas fuerzas de arrastre que se presentan.

6) **Modelo de topografía del circuito:** Este bloque está anidado dentro del anterior para una mayor claridad y modularidad. En él se establecen las características topográficas del circuito: curvas, desniveles y la dirección y velocidad del viento reinante durante la prueba.

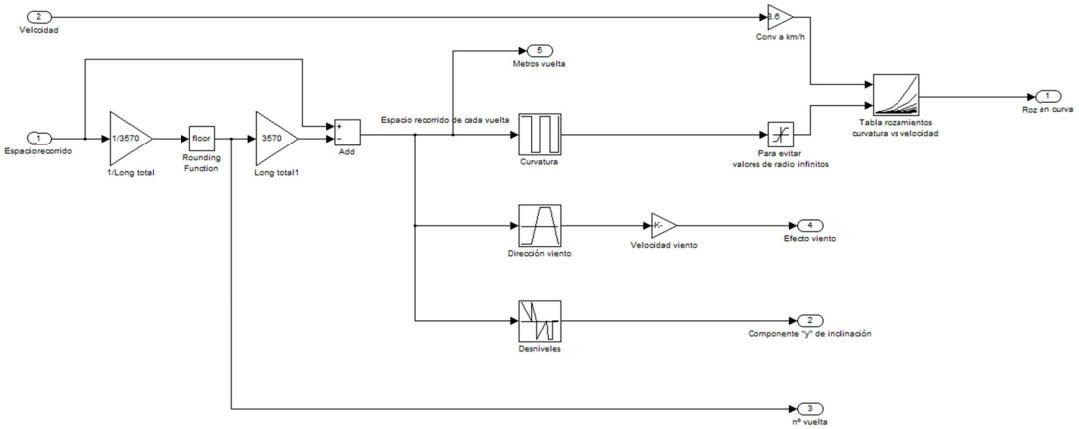


Figura 7.6.- Modelo de topografía del circuito, con desniveles, curvas y acción del viento.

8.- DINÁMICA DEL VEHÍCULO

En este apartado estudiaremos las distintas fuerzas que actúan sobre nuestro vehículo para poder realizar un correcto modelado de su dinámica. Hay 4 fuerzas principales que actúan sobre un vehículo en movimiento: la fuerza de aceleración y otras 3 que se oponen al movimiento del vehículo debido a:

- Arrastre aerodinámico.
- Resistencia a rodadura.
- Resistencia debida a desniveles.

Las fuerzas debidas al arrastre aerodinámico y la resistencia a rodadura son pérdidas puras, mientras que las fuerzas debidas a resistencia por desniveles y por aceleración son conservativas con la posibilidad, al menos en parte de ser recuperadas. La regla general sostiene que cuanto menores sean las fuerzas de arrastre, menor será el consumo de combustible. Llamamos arrastre a la fuerza longitudinal de retención generada por la resistencia a la rodadura, aerodinámica, desnivel del circuito o aceleración del vehículo.

El origen físico de estas fuerzas y las hipótesis acerca de su modelado han sido ampliamente descritas en la literatura y por tanto no profundizaremos en ello. Sin embargo, sí realizaremos un pequeño resumen de cada una de ellas, para dar una idea general del problema que nos ayudará a comprenderlo mejor para su modelado en Matlab.

8.1.- RESISTENCIA A RODADURA DE LOS NEUMÁTICOS

La fuerza a resistencia a rodadura de un neumático, cuando rueda sobre superficies duras proviene principalmente de la deformación de éste sufre y del comportamiento del material. Esta deformación produce una disipación de energía que representa el 90% del total de la resistencia a rodadura. El 10% restante se debe principalmente al arrastre aerodinámico producido en la rueda por el "efecto ventilador". Cuando se habla de vehículos de bajo consumo, este porcentaje es prácticamente despreciable debido a las bajas velocidades a las que se mueven estos vehículos.

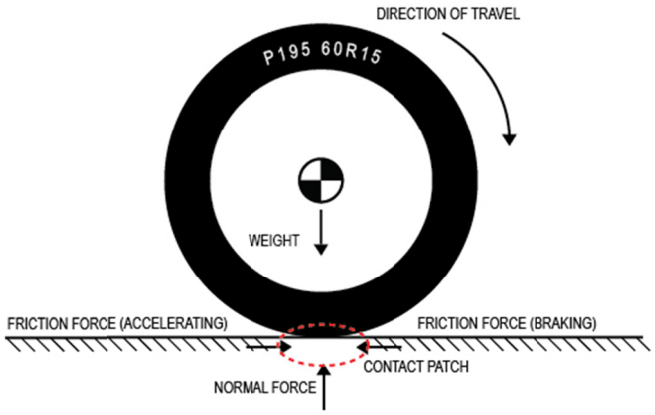


Figura 8.1.- Origen de la resistencia a rodadura de un neumático.

La histéresis del material del neumático representa la mayor contribución a esta resistencia a rodadura. Esta histéresis surge cuando la carcasa del neumático se deforma, causando una distribución asimétrica de las reacciones en el suelo (figura 8.1.). Debido a la naturaleza visco-elástica del material del neumático, esta distribución de presiones en el suelo es siempre superior en la parte frontal del área o huella del neumático en el suelo. La energía necesaria para la compresión de la parte frontal de la huella del neumático, no es recuperada completamente en la expansión de ésta, una vez que va desapareciendo el contacto en la parte posterior.

Los principales parámetros que afectan al coeficiente de resistencia a rodadura son:

1) **Construcción del neumático:** Los neumáticos de estructura radial, ya sea para vehículos de pasajeros o para vehículos pesados, tienen un coeficiente de resistencia a rodadura más bajo que los de estructura diagonal debido a la orientación de la carcasa, que hace trabajar de diferente forma al material. De igual forma, los neumáticos "tubeless" son más eficientes que los que incorporan recámara, debido a que eliminan las pérdidas de energía por deformación de la misma y por los microdeslizamientos que aparecen durante el movimiento del vehículo entre la carcasa y ella.

2) **Materiales:** La goma natural tiene un mejor comportamiento mecánico a la histéresis que la goma sintética y por tanto proporciona un coeficiente de resistencia a rodadura más bajo. Los refuerzos interiores de la goma, que ayudan a resistir mejor el desgaste y hacerla más rígida pueden ser de carbono o de sílice, esta última con menor coeficiente a rodadura.

3) **Presión de inflado:** En general, excepto en suelos muy blandos como arena, cuanto mayor es la presión de inflado, menor es la resistencia a rodadura. A altas presiones la deformación de la rueda es menor, aunque no hay que olvidar que el riesgo de reventar o pinchar el neumático es siempre mayor y ello puede acarrear el abandono de la prueba.

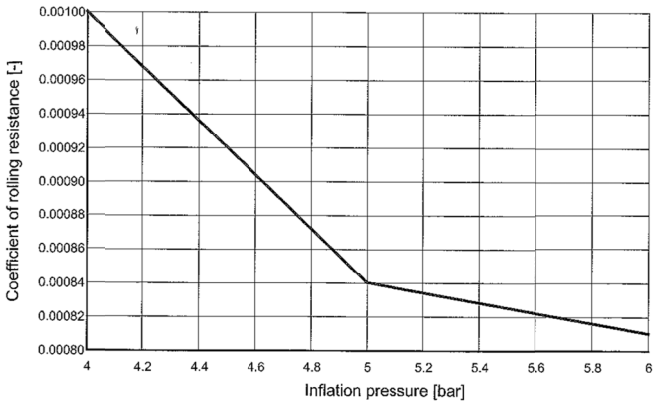


Figura 8.2.- Coeficiente de resistencia a rodadura del neumático Michelín 45-75R16.

4) **Temperatura:** Afecta de dos formas al comportamiento del neumático. Altas temperaturas favorecen una elevación de la presión interior, disminuyendo por tanto la resistencia a rodadura, pero por el contrario también actúa sobre el material que se vuelve más blando y deformable. Por tanto existe una temperatura óptima de funcionamiento del neumático y tanto si nos encontramos por debajo, como por encima de ella, la resistencia a rodadura aumenta.

5) **Condiciones del asfalto:** Un asfalto demasiado áspero, provocará una mayor deformación en la zona de la huella del neumático aumentando su resistencia a rodadura. En este aspecto un asfalto más liso es preferible. Se estima que el coeficiente de resistencia a rodadura en asfalto nuevo es un 33% mayor, mientras que en asfalto viejo pulido puede llegar a ser un 12% más bajo.

La fuerza de arrastre debida a la resistencia a la rodadura se puede obtener utilizando la siguiente ecuación [Santin et al., 2007]:

$$F_R = f_r \cdot M \cdot g$$

Donde f_r representa el coeficiente de resistencia a rodadura, M la masa del vehículo en kg y g la aceleración de la gravedad en m/s^2.

En la siguiente tabla podemos apreciar distintos valores del coeficiente de resistencia rodadura para varios neumáticos:

TIPO DE NEUMÁTICO	f_r
Neumático de vehículo sobre asfalto	0.013
Neumático de bicicleta	0.006
Michelín 44-406 de estructura diagonal para vehículos de bajo consumo	0.0024
Michelín 45-75R16 de estructura radial para vehículos de bajo consumo	0.00081

8.2.- RESISTENCIA A RODADURA DE RODAMIENTOS DE RUEDA.

La resistencia a rodadura generada por un rodamiento puede ser obtenida mediante la siguiente ecuación:

$$F_B = \mu \cdot M \cdot g \cdot \frac{d}{D}$$

Donde:

μ es el coeficiente de fricción del rodamiento (adimensional). En nuestro caso tiene un orden de magnitud de 0.0015 [Santin et al., 2007].

M es la masa del vehículo incluyendo el piloto (kg).

d es el diámetro interior del rodamiento (m).

D es el diámetro de la rueda (m).

g es la aceleración de la gravedad (m/s^2).

El coeficiente de fricción del rodamiento depende del tipo utilizado. No llega a representar más del 1% del arrastre total (incluidos todos los tipos). El valor que se ha indicado más arriba es válido para un rodamiento correctamente montado, propiamente lubricado y para rodamientos no sellados.

8.2.1.- Implementación en Matlab de las pérdidas por rodadura.

La parte de nuestro modelo matemático donde se evalúa la resistencia a rodadura de neumáticos y rodamientos de los mismos es dentro del diagrama de bloques "Circuito". En la siguiente figura podemos apreciar resaltados los bloques correspondientes a la implementación.

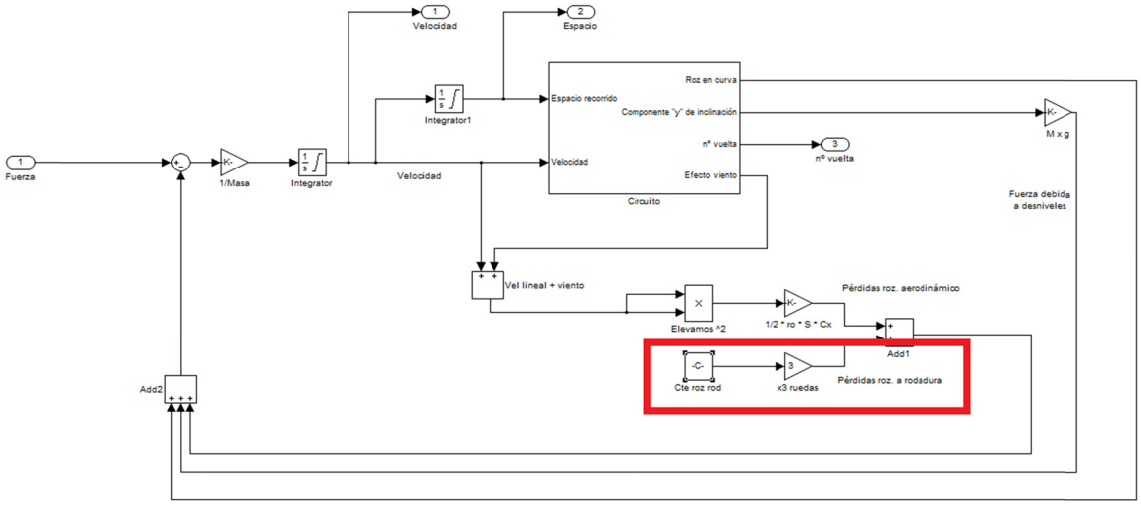


Figura 8.3.- Pérdidas por rozamiento a rodadura en nuestro modelo Matlab.

Se ha hecho uso de un bloque "const" donde introducimos las constantes correspondientes a masa del vehículo, coeficientes de fricción de neumáticos y rodamientos,… así como todos los datos necesarios para el cálculo. Posteriormente se añade una ganancia de valor 3 para evaluar las pérdidas en las 3 ruedas.

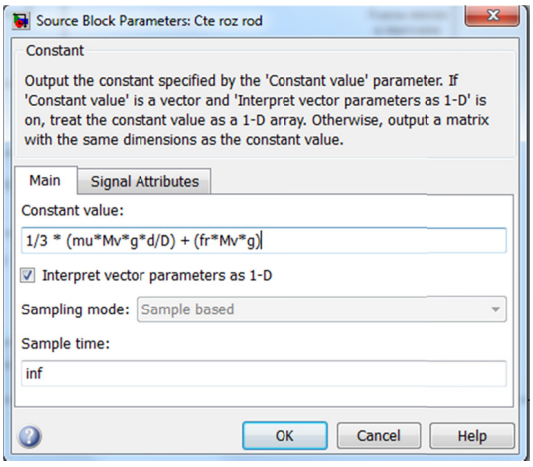


Figura 8.4.- Parámetros del bloque "const" que representa el valor del arrastre por rodadura en cada neumático.

8.3.- RESISTENCIA A RODADURA DEL NEUMÁTICO EN LAS CURVAS

En el apartado 8.1 se introdujo la noción de resistencia a rodadura en los neumáticos. Esa hipótesis sólo es cierta cuando el vehículo se desplaza por una línea recta, que aunque depende de los circuitos, en la mayoría de los casos representa menos de los 2/3 del total de la manga. Durante el otro tercio de la manga, el vehículo se encuentra trazando curvas. Este cambio en la dirección del vehículo tiene también un coste energético.

Estas pérdidas por rozamiento tienen su origen las fuerzas en dirección axial que ejerce el neumático sobre el asfalto para hacer posible el giro. Estas fuerzas, añaden una deformación extra el neumático, aumentando por tanto la energía disipada en él. Se ha determinado de manera experimental las fuerzas de arrastre en curva para un vehículo de bajo consumo de tres ruedas de características similares al nuestro. Podemos ver los resultados en la siguiente gráfica (figura 8.5) [Santin et al., 2007]:

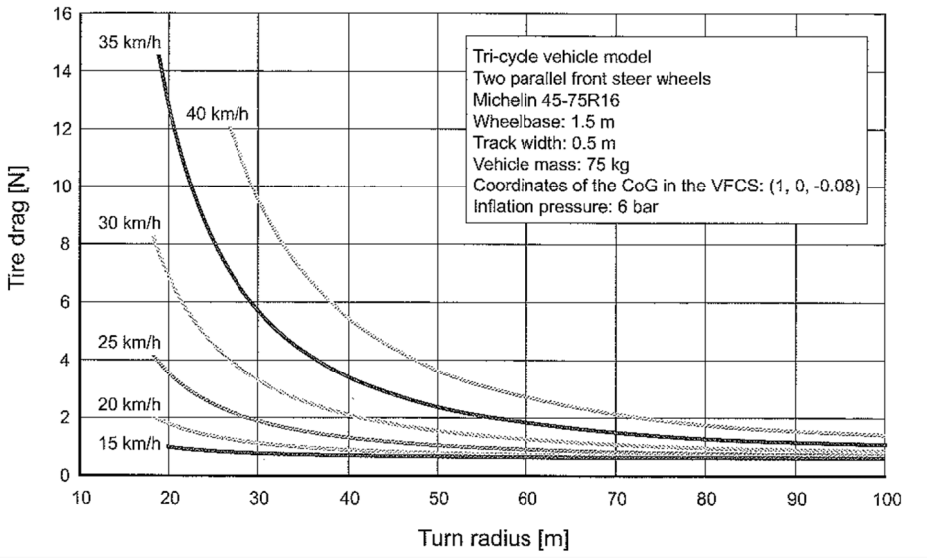


Figura 8.5.- Tabla de fuerzas de arrastre para diferentes velocidades y radios de curvatura del circuito.

Del estudio de esta gráfica podemos ver que a velocidades de 15 km/h esa fuerza es casi despreciable, mientras que el crecimiento exponencial de la misma hace que para velocidades superiores a 30 km/h y radios de curvatura por debajo de los 40m, los valores son ya importantes, llegando a superar con creces el valor de arrastre aerodinámico.

8.3.1.- Implementación en Matlab de las pérdidas por rodadura en curvas.

La penalización en cuanto a resistencia a rodadura se refiere cuando nos desplazamos por una curva a una cierta velocidad, queda evaluada en dentro del diagrama de bloques "Circuito/Circuito" del modelo general. En la figura 8.6 hemos resaltado los bloques correspondientes a este cálculo:

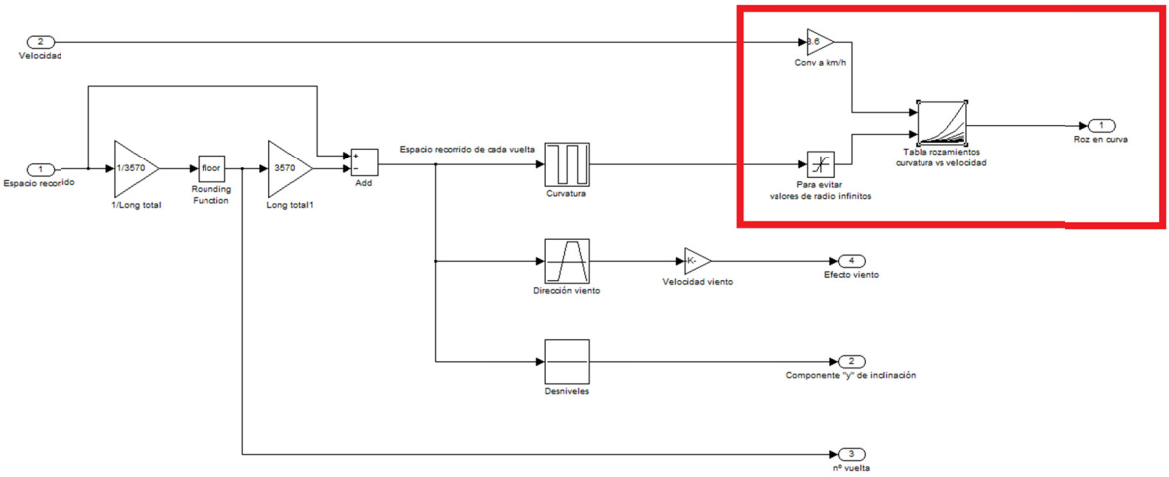


Figura 8.6.- Implementación en nuestro modelo Matlab del rozamiento debido a la curvatura del circuito.

Para la implementación se han hecho uso de los siguientes bloques:

- ***Ganancia:*** Para convertir la velocidad a km/h que es la entrada de la tabla representada en la figura 8.5.

- ***Saturación:*** Para evitar entradas en la tabla con valores de radio infinitos al desplazarnos en recta.

- ***"Lookup table 2D":*** Cuyas propiedades son representadas en la figura 8.7, donde hemos representado en forma matricial los valores principales de las curvas en la tabla de la figura 8.5. Matlab, tratará estos datos internamente, interpolando entre ellos dependiendo de la entrada que reciba este bloque.

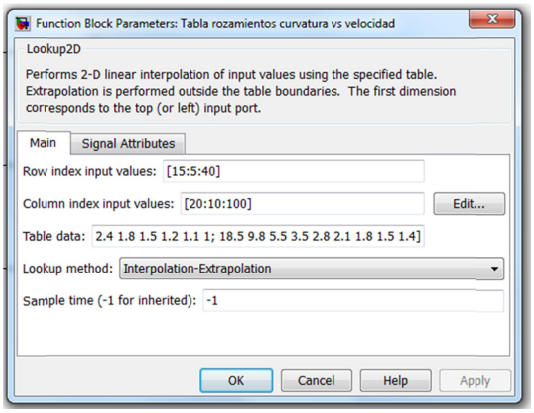


Figura 8.7.- "Lookup table" del modelo con los datos referentes a las curvas del circuito.

8.4.- AERODINÁMICA

Se estima que la fuerza de arrastre aerodinámica es la responsable de la mitad de la demanda de potencia en un vehículo de bajo consumo, cuando se desplaza sobre asfalto en llano [Santin et al., 2007]. Basándonos en esta realidad, una aerodinámica de alta calidad en el vehículo es esencial para cosechar éxitos.

Básicamente hay dos tipos de arrastre aerodinámico: arrastre aerodinámico por fricción y por presión. Un vehículo en movimiento dentro de un fluido genera ambos arrastres aerodinámicos. Sus proporciones relativas dependen de la forma del cuerpo y se muestran en la siguiente fórmula:

$$F_A = F_{fricción} + F_{presión}$$

- ***Resistencia aerodinámica por fricción:*** Cuando el aire fluye sobre un vehículo, la fricción se genera entre el aire y la superficie del mismo. Se puede afirmar que cuando el régimen del fluido es laminar, hay una capa de fluido con velocidad relativa 0 con respecto al cuerpo, es decir está adherida a su superficie (figura 8.8). A medida que nos alejamos en la dirección a la normal de la superficie, la velocidad del fluido va aumentando, produciéndose un esfuerzo de cortadura entre las distintas capas del fluido a diferente velocidad, que es la causa del arrastre aerodinámico por fricción.

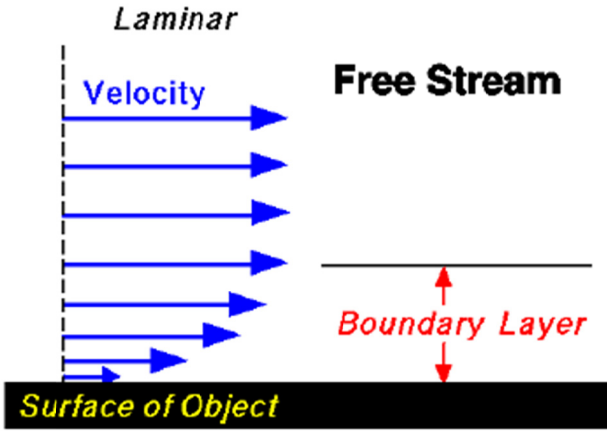


Figura 8.8.- Gradiente de velocidades en flujo laminar.

- ***Resistencia aerodinámica debida a la presión:*** La distribución de presiones sobre el cuerpo del vehículo produce una fuerza, donde la componente en la dirección del movimiento es la que genera el arrastre aerodinámico por presión. Existen otras dos componentes que no consideraremos, que son la lateral y la vertical. El principal origen de esta fuerza en nuestro caso es provocada por la separación del fluido (figura 8.9). Cuando el gradiente de presiones a lo largo de la línea de corriente del vehículo se vuelve muy alto, el fluido no puede seguir la superficie y se separa de la misma. En estas zonas de separación, la presión es más o menos igual a la del ambiente, mientras que en la parte frontal del vehículo existe una zona de sobrepresiones. Esta diferencia de presiones entre la parte frontal y la trasera es la que genera la fuerza de arrastre aerodinámica por presión.

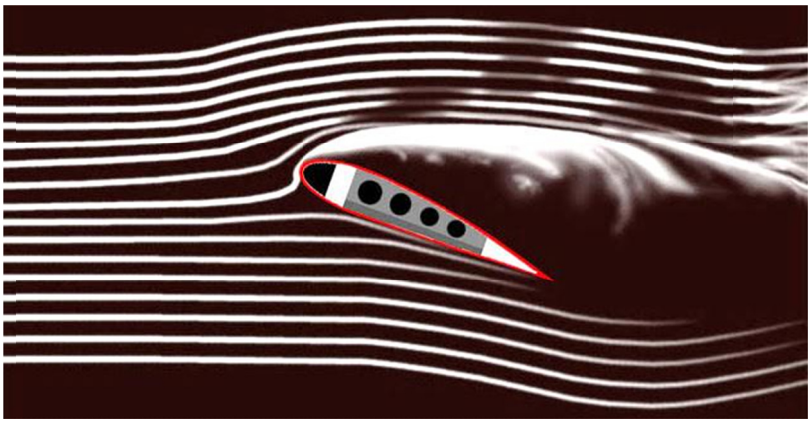


Figura 8.9.- Desprendimiento de la capa límite en un perfil de ala con excesivo ángulo de ataque.

Debido a complejidad de los efectos del aire sobre el vehículo y para hacer posible su estudio, principalmente antes de la aparición de los ordenadores potentes de la actualidad, se hace depender estas relaciones de fuerzas en una única variable (coeficiente) que permita, de una forma simple, conocer los efectos que resulten presentes. El valor de estos coeficientes es determinado de forma experimental en un entorno controlado (por ejemplo, túnel de viento), en el que se puede conocer la velocidad, la densidad del aire, el área de referencia del automóvil (factor de forma y superficie frontal) y el arrastre y la sustentación producida sobre un cuerpo conocido (modelo del sistema).

También se pueden determinar de forma teórica, haciendo uso de ordenadores para resolver las ecuaciones de la mecánica de fluidos. En la figura 8.10 podemos ver el estudio de [Jacob, 2010], ex-alumno de este máster, acerca de la determinación por medio de cálculo CFD de los diversos valores que toma el coeficiente C_x en nuestro vehículo para diversos ángulos de ataque del viento (figura 8.10 y 8.11).

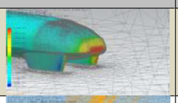
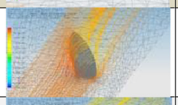
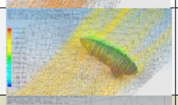
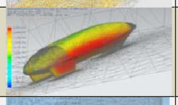
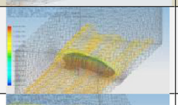
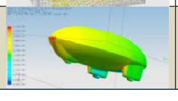
	Imagen del análisis	Ángulo	Área m² a considerar (F=frontal, L=lateral)	F_x o F_y (N)	C_x o C_y	SC_x o SC_y
1		0°	F 0,304	3,319 N-F_x	0,25 C_x	**0,07 SC_x**
2		30°	F 0,304	16,280 N-F_x	1,23 C_x	**0,37 SC_x**
3		60°	L 1,284	46,485 N -F_y	0,83C_y	**1,07 SC_y**
4		90°	L 1,284	66,450 N-F_y	1,19 C_y	**1,53 SC_y**
5		120°	L 1,284	47,82 N-F_x	0,85 C_y	**1,10 SC_y**
6		150°	F 0,304	16,750 N-F_x	1,27 C_x	**0,38 SC_x**
7		180°	F 0,304	3,305 N-F_x	0,25 C_x	**0,07 SC_x**

Figura 8.10.- Estudio de los diferentes valores del coeficiente C_x para diversos ángulos de ataque del viento.

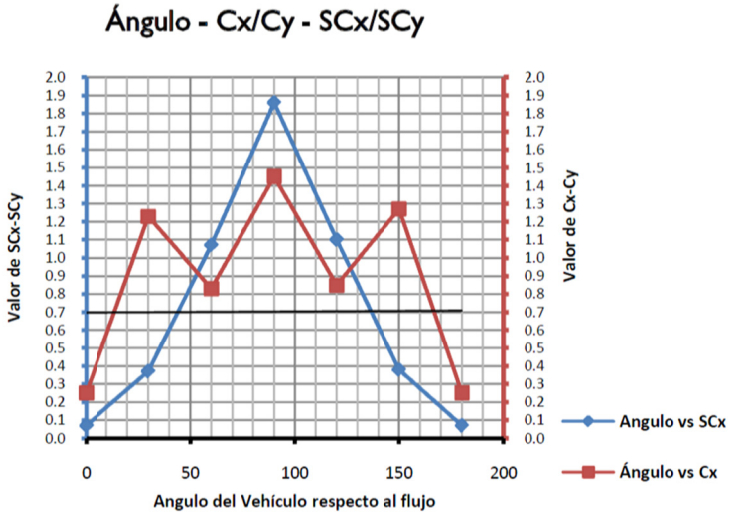


Figura 8.11.- Relación entre valores C_x-C_y para distintos ángulos del flujo respecto al vehículo.

La fuerza de arrastre aerodinámico generada por un vehículo que se mueve a través del aire se obtiene utilizando la siguiente ecuación [Jacob, 2010]:

$$F_A = \frac{1}{2} \cdot \rho \cdot A \cdot C_X \cdot v^2 \qquad (1)$$

Donde:

ρ es la densidad del aire (≈ 1.2 kg/m^3).

v es la velocidad relativa entre el vehículo y el aire (m/s).

C_x es el coeficiente adimensional de arrastre aerodinámico, y en nuestro vehículo tiene **un valor de 0.25** [Jacob, 2010].

A es el área frontal del vehículo, **en nuestro caso 0.304 m^2**.

El coeficiente C_x es sólo valor referencial para conocer el grado de eficiencia de un perfil aerodinámico determinado, para comparar la eficacia aerodinámica de distintos automóviles lo más apropiado es comparar el valor conocido como AC_x , que resulta de multiplicar C_x nuevamente por la superficie frontal de referencia, y esperándose obtener los valores más bajos posibles. Así no sólo se considerará el perfil del vehículo, sino también su tamaño, dado que si bien la eficiencia aerodinámica es independiente del tamaño del móvil y tiene una mayor relación con la forma y la suavidad de las superficies y transiciones del vehículo, el rendimiento real si estará en cierta medida condicionado por el tamaño [Briet, 2009].

Lo anterior significa que el C_x de dos vehículos de muy distinto tamaño podrá ser similar, pues sólo dependerá de la eficiencia aerodinámica de su forma, pero el valor del AC_x estará ligado también al tamaño de dicho móvil. Todo ello nos dará una referencia que permitirá, en concreto, definir las cualidades aerodinámicas que un vehículo posee en relación a otro.

Como referencia se adjunta la siguiente tabla con valores de AC_x para algunas formas y vehículos conocidos, algunos suministrados por los fabricantes otros calculados previamente. [Jacob, 2010]:

Tipo de vehículo	AC_x
Bicicleta	0.56
Bicicleta de carretera y ciclista en posición aerodinámica	0.30
Audi A3 (2003)	068
BMW Serie I (2004)	0.65
Vehículo Eco-Shell Marathon IDF	0.076

8.4.1.- Implementación en Matlab de la fuerza de arrastre aerodinámica.

La parte en la cual hemos implementado en Matlab estas pérdidas por rozamiento aerodinámico, es dentro del diagrama de bloques "Circuito", en la parte resaltada con el recuadro rojo. La entrada que recibe el sistema (figura 8.12) es la velocidad lineal del vehículo más la componente frontal del viento reinante en la pista. Esta entrada hay que elevarla al cuadrado y multiplicada por los distintos términos de la ecuación vista en el apartado anterior. Como salida obtenemos la fuerza de arrastre aerodinámica que será restada a la fuerza total que mueve el vehículo.

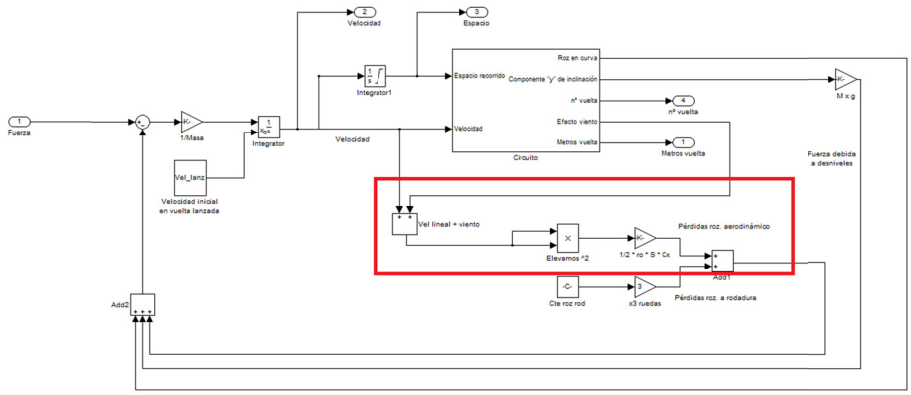


Figura 8.12.- Fuerza de arrastre aerodinámica debida a velocidad propia y viento en nuestro modelo Matlab.

Para la implementación se han hecho uso de los siguientes bloques:

- ***Bloque sumador:*** La utilidad de este bloque es sumar la velocidad lineal del vehículo con la componente del viento que incide paralelamente al desplazamiento del vehículo.
- ***Bloque multiplicador:*** De la salida del bloque anterior obtenemos el valor de la velocidad de incidencia relativa del viento con respecto al vehículo. Tal y como vimos en la expresión (1), este módulo necesita ser elevado al cuadrado. Mediante este bloque, conseguimos multiplicar la entrada por ella misma obteniendo como salida el cuadrado de ella.
- ***Ganancia:*** La función de este bloque es la de multiplicar la ganancia por el valor de una constante establecida en su interior. Dentro de las propiedades de este bloque establecemos el valor de las constantes de la ecuación (1) de la siguiente forma.

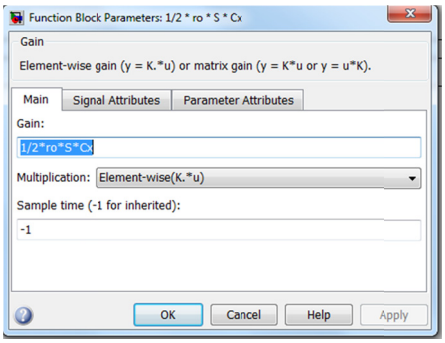


Figura 8.13.- Propiedades de la ganancia presente en la figura 8.12.

8.5.- ESFUERZO DEBIDO A DESNIVELES

La fuerza inducida por la gravedad cuando conducimos un vehículo sobre un suelo no horizontal es conversativa, lo que significa que se opone al movimiento cuando vamos en dirección de subida, pero lo ayuda cuando bajamos la pendiente. Podemos ver el esquema en la figura 8.13:

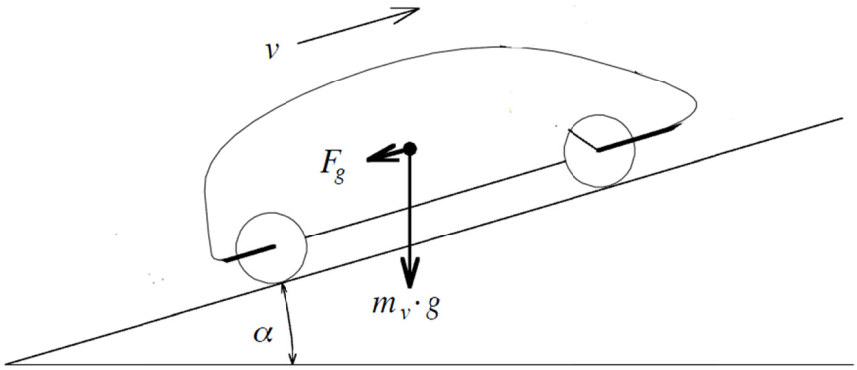


Figura 8.13.- Esquema de fuerzas cuando el vehículo viaja sobre una pendiente.

Viene dada por la componente axial de la fuerza peso del vehículo que se calcula por la siguiente fórmula:

$$F_g = M_V \cdot g \cdot \sin \alpha$$

Donde:

M_v es la masa del vehículo incluyendo el piloto (kg).

g es la aceleración de la gravedad (m/s^2).

$sin(\alpha)$ es el ángulo de desnivel (rad).

8.5.1.- Implementación en Matlab del esfuerzo debido a desniveles.

Para evaluar este esfuerzo, tomamos como entrada al sistema el punto kilométrico del circuito donde nos encontramos. Con este entrada y mediante una "Lookup table" en la que previamente hemos introducido los desniveles del circuito en sus distintos puntos, nos dará como salida la componente axial ***sin(α)***, que posteriormente multiplicaremos por la masa total del vehículo más el piloto y por la aceleración de la gravedad. En los siguientes esquemas se resalta los bloques correspondientes a este módulo:

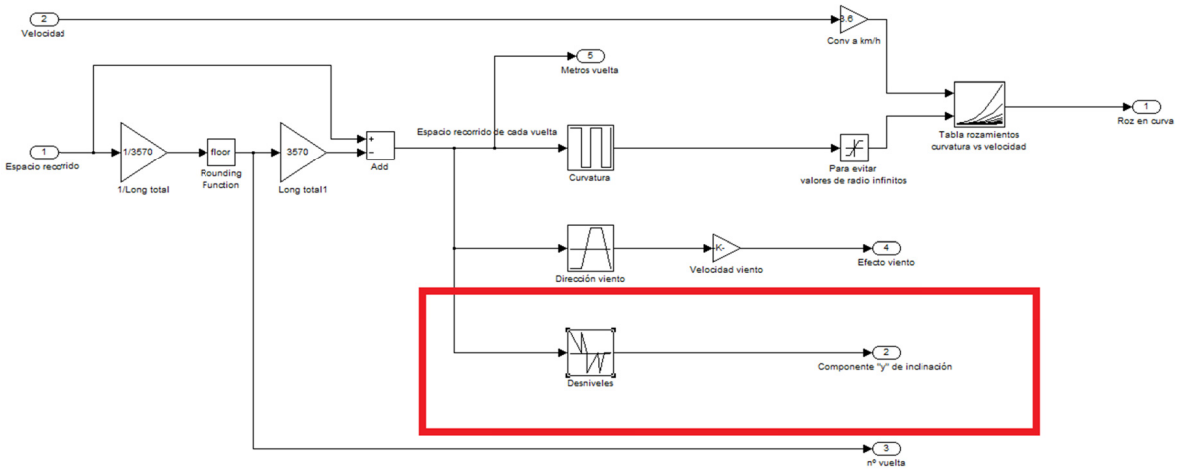


Figura 8.14.- Implementación del esfuerzo en desniveles en el modelo.

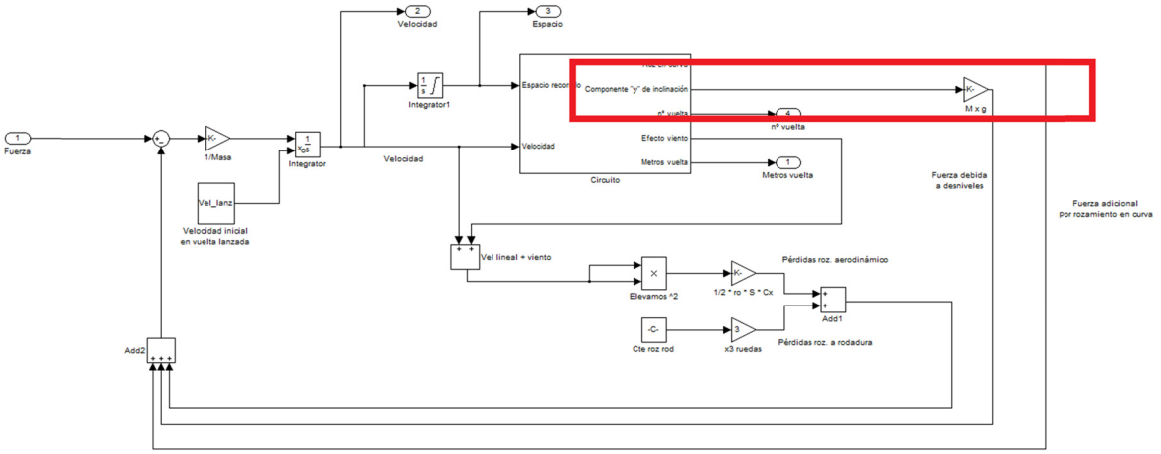


Figura 8.15.- Salida del submodelo en figura 8.14 con la componente "y" del peso debida a inclinación.

8.6.- ACELERACIÓN EQUIVALENTE.

La aceleración es el esfuerzo de tracción equivalente del vehículo, menos la sumatoria de las fuerzas que se oponen a su movimiento. La aceleración se puede describir, para movimiento lineal como:

$$a = \frac{F_{eq}}{M_V}$$

Dentro de Matlab se ha implementado en el diagrama "Circuito" (figura 8.16). Recordamos que la única entrada de este gran bloque era la fuerza lineal proveniente del sistema de transmisión del vehículo. A esta entrada y como podemos apreciar en la siguiente figura, sumamos todas las fuerzas que se oponen o ayudan al movimiento del vehículo, para posteriormente dividirlas entre la masa total del sistema vehículo-piloto y de esta forma calcular la fuerza lineal equivalente, ya sea positiva o negativa. Esta fuerza sólo tomará valores negativos cuando el vehículo se mueva con el motor parado y la pendiente no sea lo suficientemente descendente como para conseguir aumentar la velocidad del vehículo.

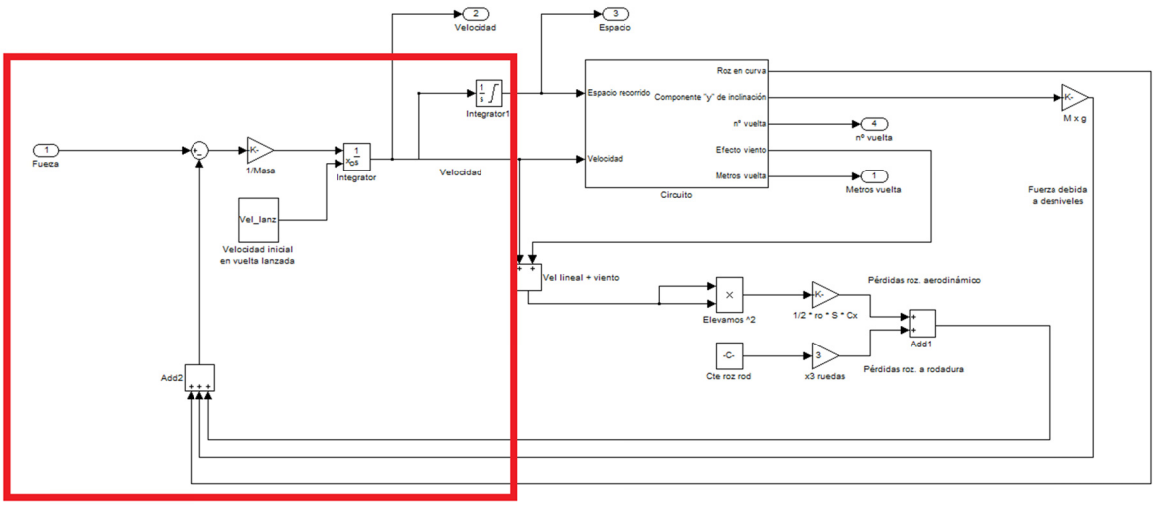


Figura 8.16.- Cálculo de la aceleración equivalente del vehículo en el modelo.

Apreciando la parte resaltada en la figura 8.16 podemos distinguir las siguientes partes principales:

- ***Bloque sumador de 3 entradas:*** Se encuentra situado en la parte inferior del recuadro resaltado. Su función es la de sumar los esfuerzos aerodinámicos, a rodadura y por desniveles.

- ***Bloque sumador de 2 entradas:*** La función de este bloque es restar la equivalente de las fuerzas anteriores, a la fuerza lineal cuando el vehículo se encuentra con el motor encendido.

- ***Ganancia:*** Sirve para dividir la fuerza equivalente entre la masa para calcular la aceleración equivalente del sistema. Está ubicada a la salida del bloque sumador anterior.

- ***Primer integrador:*** Integrando la aceleración, podemos calcular la velocidad de nuestro vehículo. Este primer integrador tiene como peculiaridad el tener como entrada una constante que le indica cuál es la velocidad de lanzamiento a la cual el vehículo comienza la vuelta al circuito.

- ***Constante de velocidad de lanzamiento:*** Como hemos indicado en el párrafo anterior, el primer integrador, para realizar correctamente su función necesita dos entradas, por un lado la aceleración equivalente y por otro la velocidad de lanzamiento que supone la condición inicial de comienzo de la simulación de la vuelta al circuito.

- ***Segundo integrador:*** Toma como entrada la salida del primer integrador y por tanto, integrando la velocidad, consigue contabilizar el espacio que lleva recorrido nuestro vehículo

En la figura 8.17 podemos ver la proporción relativa de las fuerzas de arrastre principales que frenan el vehículo:

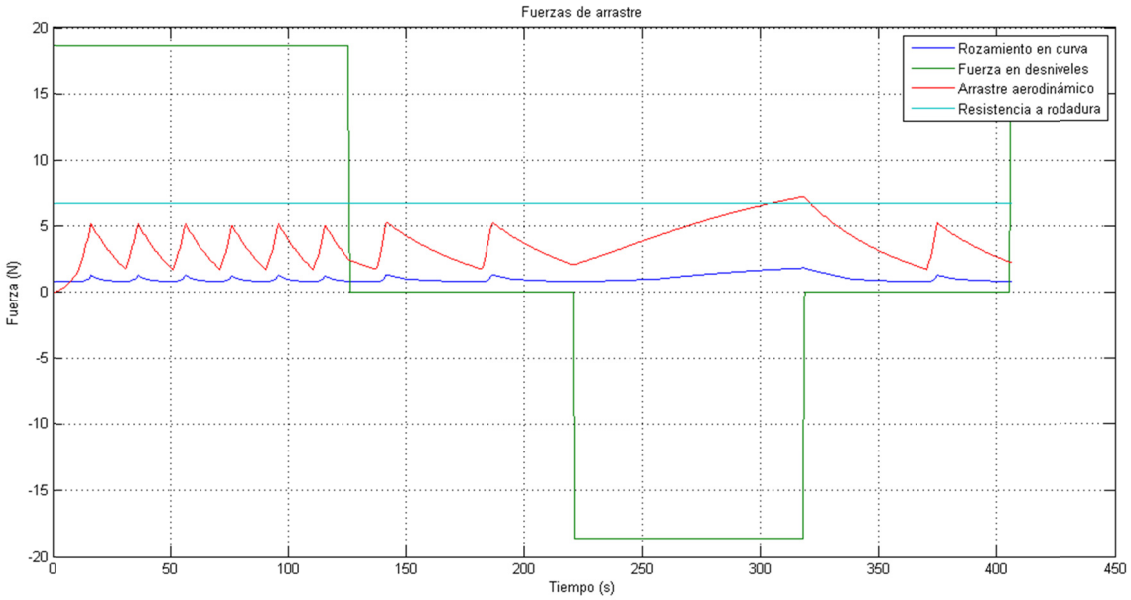


Figura 8.17.- Gráfica con las distintas fuerzas de arrastre que afectan a nuestro vehículo.

8.7.- MOTOR.

El motor será el responsable de generar la fuerza de empuje capaz de superar todas las fuerzas de arrastre enunciadas anteriormente. El motor que propulsa nuestro vehículo es de combustión interna, siendo su combustible etanol. Este tipo de motores alcanzan su máximo rendimiento cuando la apertura del acelerador es máxima, debido a que la mariposa situada en el conducto de admisión, a aperturas menores, actúa de estrangulador del flujo, haciendo que la depresión creada en el interior del cilindro no desaparezca en el momento de cierre de las válvulas de admisión. Esta situación provoca, que al comenzar a comprimir la mezcla, ya partimos de un handicap que es la depresión a la cual se encuentra el cilindro. Necesitamos por tanto, hacer desaparecer esta presión y comprimir la mezcla para que posteriormente sea inflamada por la chispa de la bujía.

Como el rendimiento de estos motores está directamente ligado a la relación de compresión, en casos de aperturas del acelerador parciales, la presión en el cilindro no llega a ser la adecuada, provocando un quemado menos eficiente de la misma. Esta es la razón principal de que los vehículos impulsados por motores de combustión interna utilicen la estrategia “stop and go”, haciendo funcionar el motor durante breves espacios de tiempo, a máxima apertura del acelerador.

Figura 8.18.- Motor Honda GX25 en su estado original.

El motor que equipa nuestro vehículo es el Honda GX25 (figura 8.18), el cual ha sido modificado en aspectos como un aumento de la relación de compresión para aprovechar mejor el poder calorífico del combustible, y la sustitución del sistema de alimentación original por carburador, por otro de inyección controlada electrónicamente.Los datos que nos interesan para su modelado son básicamente sus curvas de par, potencia y consumo específico. Estos datos han sido obtenidos mediante pruebas del vehículo en banco de potencia y los mostramos en las siguientes gráficas:

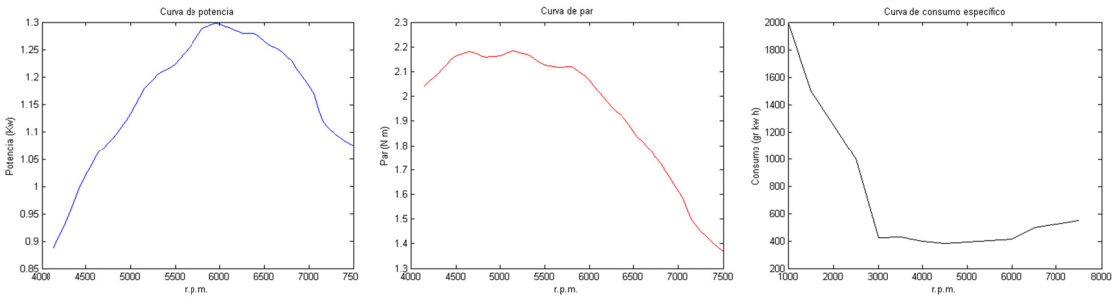


Figura 8.19.- Curvas de potencia, par y consumo específico del motor de nuestro vehículo.

8.7.1.- Implementación en Matlab de las curvas del motor.

Toda la parte correspondiente al motor, sus curvas de par y consumo específico, se encuentran dentro del modelo Simulink llamado "***Motor***", presentado en la figura 8.20. El bloque recibe como entradas una señal binaria (0-1) que indica si debe arrancar o parar el motor y la velocidad, que es utilizada para conocer en todo momento las revoluciones a las que girará el motor. Genera como salida el consumo expresado en gr de combustible. Esta unidad será convertida dentro del código del programa a volumen de combustible.

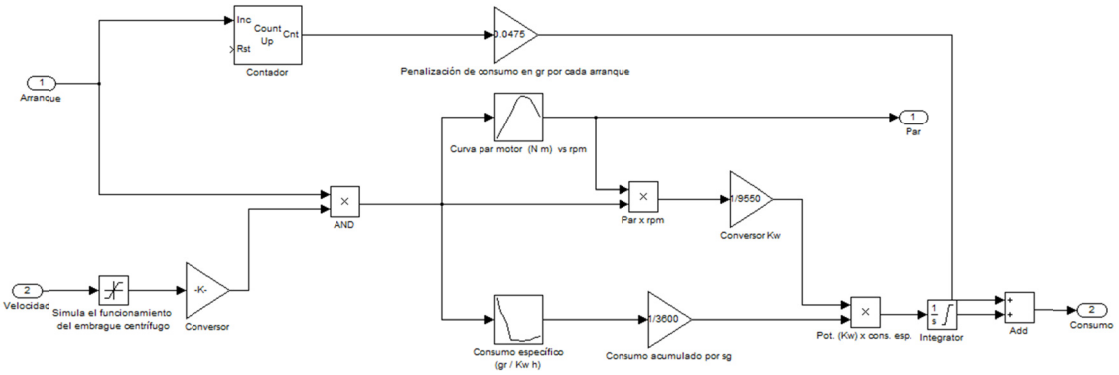


Figura 8.20.- Modelo Simulink del motor.

Apreciando la parte resaltada en la figura 8.20 podemos distinguir los siguientes bloques principales:

- ***Contador:*** Su función es la de contar el número de arranques que se han producido para penalizar el consumo global con un valor de 0.0475 gr por cada arranque.

- ***Curva de par motor:*** La curva de par motor ha sido implementada mediante una "Lookup-table" que contiene los vectores de par y de revoluciones del motor.

- ***Curva de consumo específico:*** Al igual que en el caso anterior la curva de consumo específico ha sido implementada mediante una "Lookup-table" que contiene los vectores de consumo específico y de revoluciones del motor.

- ***Saturación para simular embrague centrífugo:*** Por debajo de cierta velocidad, el embrague centrífugo actúa, manteniendo el nº de r.p.m. constantes, hasta que una vez rebasada, deja de actuar y el coche puede subir de revoluciones a medida que la velocidad incrementa.

- ***Bloque integrador:*** Es el encargado de sumar todos los consumos específicos instantáneos para obtener el total.

- ***Ganancia:*** Se han incluido varios de estos bloques cuya función en este bloque es la de convertir unidades.

9.- ALGORITMOS GENÉTICOS UTILIZADOS.

Con el modelo ya verificado en las pruebas del apartado anterior, nos dispusimos a realizar nuestras primeras pruebas de optimización mediante algoritmos genéticos, para las cuales utilizamos el GAToolbox desarrollado en esta universidad por el grupo de Control Predictivo y Optimización Heurística (CPOH).

Esta toolbox para Matlab está compuesta por un algoritmo genético genérico que trabaja con números reales. Tiene un funcionamiento bastante eficiente y su manejo es muy fácil. Consta de los siguientes ficheros principales:

- ***GA.p:*** Este fichero es el núcleo del algoritmo genético. Su extensión corresponde a la de archivos encriptados Matlab. Por lo tanto no puede ser examinado ni editado. Por esta razón desconocemos cómo trabaja interiormente este algoritmo: generación de individuos, métodos de selección y cruce…

- ***GAparam.m:*** En este fichero se establecen los parámetros para el funcionamiento del algoritmo genético. Este algoritmo trabaja con **codificación real** y dentro de este fichero podemos establecer:

    1) ***Tamaño de la población:*** número de individuos o soluciones que se evaluarán en cada iteración.
    2) ***Número de generaciones:*** es el número máximo de poblaciones o conjuntos de soluciones que se evaluarán hasta el final de la optimización.
    3) ***Cromosoma del individuo con rangos:*** Aquí establecemos tanto el número de genes o variables de cada solución como su rango (si se desea).
    4) ***Probabilidad de cruce:*** De esta probabilidad dependerá el número de individuos respecto al total de la población que serán seleccionados para la reproducción. Por defecto este valor está establecido a una tasa del 80%.
    5) ***Probabilidad de mutación:*** De esta probabilidad dependerá el número de individuos respecto al total de la población que serán seleccionados para mutar alguno de sus genes. Por defecto este valor está establecido a una tasa del 10%.
    6) ***Individuos introducidos en la población inicial:*** Mediante este parámetro se permite nos permite introducir los individuos que deseamos para evaluarlos en la primera generación. Según diversos autores como [Bezdek *et.al.*, 1994], esta heurística permite una convergencia más rápida del algoritmo introduciendo soluciones de mediana calidad.

- ***GAresults.m:*** En este fichero contiene la función que guarda los resultados de la optimización en la estructura de datos general del algoritmo gaDat.

- ***GAiteración.m:*** Este fichero controla qué mensajes se muestran por pantalla al finalizar cada generación. Entre ellos se puede mostrar el mejor individuo, su cromosoma, la media de la población, la variación estándar, …

- ***Coste.m:*** Dentro de este fichero introducimos nuestra función objetivo que servirá para evaluar la bondad de los individuos o soluciones al problema. Hay que tener en cuenta que este **algoritmo sólo minimiza, por tanto, para problemas de maximización deberemos hacer negativa nuestra función objetivo.**

Con los parámetros establecidos, podemos iniciar la simulación tecleando la palabra "ga" en la ventana de comandos de Matlab. Este comando realiza la llamada a la función principal ***GA.p***. En la siguiente tabla se detallan las características conocidas del algoritmo. La mayoría de ellas, como comentábamos anteriormente son desconocidas al estar su código encriptado.

CARACTERÍSTICAS DEL ALGORITMO	
Nombre	Algoritmo genético básico del CPOH
Tipo de codificación utilizada	Números reales
Operador de selección	-
Operador de reproducción	-
Operador de mutación	-
Elitismo	Utiliza elitismo pero se desconoce su %
Mecanismos de reemplazo	-
Criterio de terminación	Por nº máximo de generaciones

9.1.- ALGORITMO GENÉTICO BÁSICO.

En una primera aproximación al problema de la creación de un algoritmo genético, se creyó más conveniente comenzar con la variante más simple que se pudiera implementar para la realización de optimizaciones. Esta variante consiste únicamente en encontrar el rango óptimo de revoluciones a las que el motor funcionará **durante todas las arrancadas** realizadas en el transcurso de la vuelta al circuito. Esto implica que al constar nuestro vehículo de una sola relación de transmisión que se fija previamente a la prueba, las soluciones obtenidas vendrán dadas en forma de velocidad mínima a la cual se arranca y velocidad máxima, la cual una vez es rebasada, indica el punto de parada del motor hasta el próximo ciclo.

Este hecho se aprecia en los datos de la simulación que se muestran en la figura 9.1, donde se puede distinguir la franja de velocidades en las cuales va a oscilar nuestro vehículo durante la vuelta al circuito.

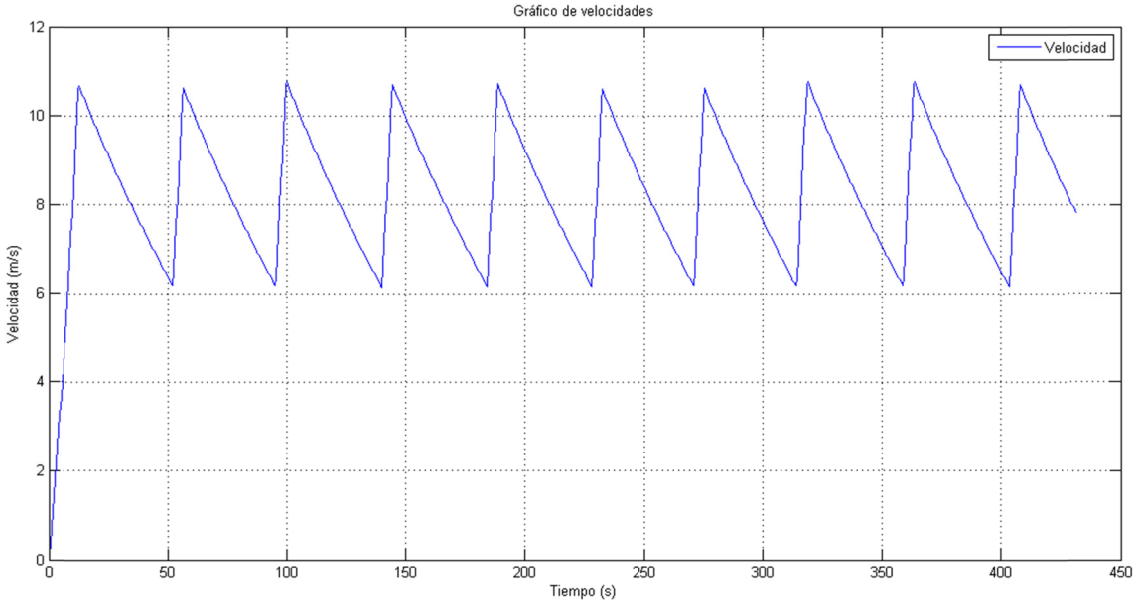


Figura 9.1.- Gráfica velocidades frente al tiempo utilizando optimización de V_mín y V_máx fijas.

Es lógico pensar que este tipo de optimización, aun siendo mejorable, pueda obtener buenos valores de kilometraje, en circuitos cuya topografía ni con curvas muy cerradas, donde además de penalizar con una alta resistencia a la rodadura las altas velocidades de paso por ella, corremos el riesgo de vuelco (factor este que no tiene en cuenta el simulador).

La razón de esta hipótesis, es que mediante este tipo de estrategia de carrera, sólo haríamos funcionar el motor siempre dentro de un rango óptimo de revoluciones, ya que los motores de explosión interna siempre alcanzan un valor consumo específico menor cuando funcionan en regímenes cercanos al par máximo. Realizando esta optimización para distintas relaciones de transmisión, estaríamos preparados para determinar este rango óptimo.

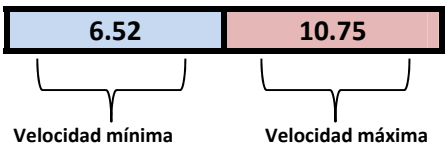
9.1.1.- Codificación utilizada.

La codificación utilizada en este algoritmo genético es muy sencilla. Como vimos en el apartado anterior, nuestro problema a resolver consta sólo de 2 variables a determinar:

- Velocidad de arranque del motor.
- Velocidad de parada del motor.

Utilizaremos codificación por números reales, por ser una solución más natural al problema que la codificación binaria, que en este caso concreto no aportaría ventajas, pero sí dificultaría la labor de programación del algoritmo genético.

El aspecto general de cada individuo puede ser el siguiente (los valores que aparecen son aleatorios):



El primer gen de cada individuo representará la velocidad mínima a la que podrá desplazarse sin poner el motor en marcha y su segundo gen la velocidad máxima a la que podrá desplazarse con el motor en marcha.

9.1.2.- Generación de la población inicial.

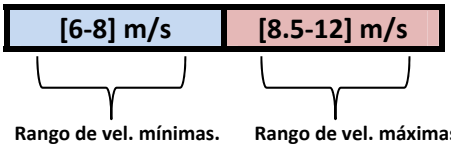
Una vez establecido el criterio de codificación de los individuos, pasamos a estudiar el método de generación de individuos. Dado que queríamos verificar el buen funcionamiento del algoritmo, se optó por no incluir ninguna solución en la primera generación. De esta forma podemos analizar la convergencia y o no convergencia del mismo.

Dado que hay ciertos rangos de velocidades a los cuales no es aconsejable hacer funcionar el vehículo durante la competición, podemos limitar los valores que tomarán ambas variables:

- ***Velocidad de arranque del motor:*** Limitaremos los valores de esta variable entre 6 y 8 m/s que corresponden aproximadamente a 21 – 29 km/h. Velocidades más bajas de 6 m/s no son aconsejables, ya que en ese rango de revoluciones el embrague centrífugo no estaría abierto completamente, provocando deslizamientos y pérdidas de energía.

- ***Velocidad de parada del motor:*** El rango establecido en este caso va de 8.5 a 12 m/s (aproximadamente 31-43 km/h). Escogemos 8.5 como mínima para que bajo ningún concepto se pueda solapar con alguna velocidad mínima y 12 como máximo, ya que es valor bastante por encima de la media requerida, y a esa velocidad el motor con la mayoría de las relaciones de transmisión va girando muy por encima de su régimen óptimo, consumiendo mucha más cantidad de combustible.

En el siguiente esquema vemos los rangos de valores que podrá tomar cada individuo:



La generación de estos valores es puramente aleatoria y para ello ha sido utilizada la función ***rand()*** de Matlab.

9.1.3.- Función objetivo.

La función objetivo que utilizaremos para evaluar a la población será el kilometraje obtenido por cada individuo mediante su simulación dentro del modelo Simulink del vehículo. Para llevar a cabo correctamente esta simulación necesitamos introducir el cromosoma de cada individuo dentro de nuestro modelo y lanzar la simulación. En este caso, al tener sólo una velocidad mínima y otra máxima, el submodelo *"Sistema de arranque"* utilizado en este caso será el más simple de los creados. Lo podemos ver en la figura 9.2:

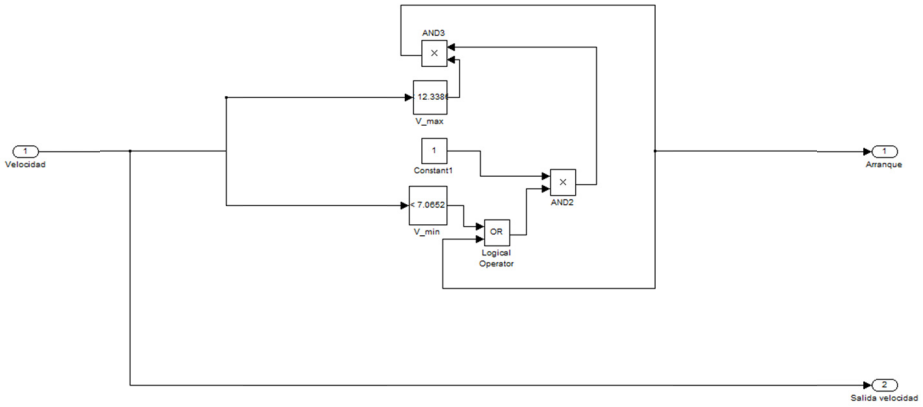


Figura 9.2.- Modelo de arranque para el AG básico.

El algoritmo genético, previo a la evaluación de cada individuo, cambiará los valores de las constantes V_min y V_max presentes en el modelo de la figura 9.2 por el primer y segundo gen del individuo respectivamente. Una vez la simulación del individuo ha sido terminada, se leen las salidas proporcionadas por el modelo general:

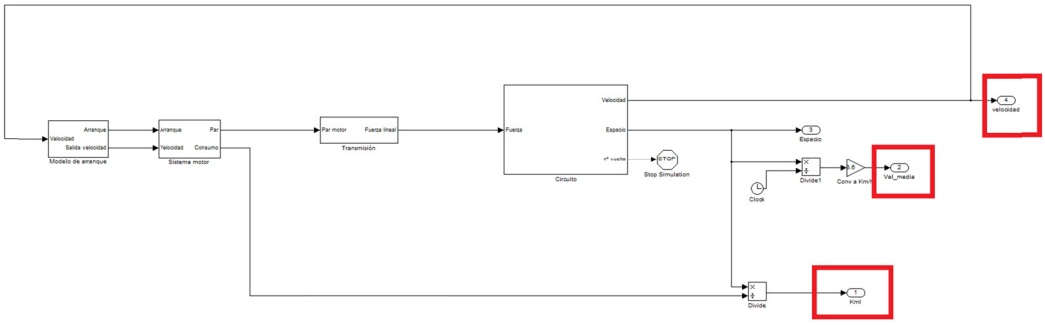


Figura 9.3.- Salidas del modelo general para el AG básico.

La función objetivo será por tanto maximizar el kilometraje, que es obtenido al dividir el espacio en Km entre el volumen de combustible en l.

$$\boldsymbol{Kilometraje = \frac{S}{V_{combustible}}}$$

La evaluación de esta función está sujeta a dos restricciones:

$$\boldsymbol{V_{media} \geq 30\ km/h}$$

$$\boldsymbol{V_{final} \geq V_{inicial}}$$

El no cumplimiento de alguna de estas dos restricciones, invalidaría el kilometraje obtenido por el individuo pasando su valor a ser de 100km/l, un valor lo suficientemente bajo como para descartarlo casi por completo de las operaciones de reproducción mediante el operador de selección que daría una probabilidad muy baja de ser escogidos a estos individuos. Utilizamos por tanto una ***técnica de penalización*** para el tratamiento de restricciones en este problema. Dentro de las técnicas de penalización se probó a realizar una penalización parabólica en función de la desviación con respecto a las restricciones obtenida por el individuo, pero hacía muy difícil la convergencia del algoritmo al convertir la función objetivo original en otra del tipo multiobjetivo ponderado.

Las técnicas de reparación han sido descartadas en este problema, porque conllevarían un costo computacional importante, ya que implicarían volver a comprobar varias veces cada individuo no válido hasta que cumpliera ambas restricciones y pudiera ser reintroducido en la población.

9.1.4.- Operador de selección.

Como operador de selección utilizamos el muestreo estocástico universal. Es un procedimiento similar al de selección por ruleta, pero en este caso se genera un sólo número aleatorio y a partir de él se generan los K números que se necesitan, para generar K individuos espaciados de igual forma. Los números se calculan de la siguiente forma:

$$a_j := \frac{a + j - 1}{k} \quad (\forall j = 1, \dots, k) \qquad (1)$$

Una vez generados estos números, el método funciona de la misma forma que la selección por ruleta, pero de una manera más eficiente, al generar sólo un número aleatorio. Podemos ver en la siguiente tabla cómo funciona este método:

Individuo	1	2	3	4	5	6	7	8
Kilometraje	800	100	100	900	700	500	100	400
P_i	0.22	0.028	0.028	0.25	0.19	0.14	0.028	0.11
q_i	0.22	0.248	0.276	0.526	0.716	0.856	0.884	1

Se calculan las P_i dividiendo el kilometraje de cada individuo entre la suma total de todos los kilometrajes. Las Q_i son las P_i acumuladas desde el individuo 1 al 8. Supongamos que queremos escoger a 3 individuos (k=3), y que el número aleatorio obtenido entre 0-1 es a= 0.2254. Sustituyendo en la ecuación (1) y calculando para j=1,2,3 tenemos que:

- a_1 = 0.075 < 0.22 → Escogemos individuo 1.
- a_2 = 0.408 < 0.526 → Escogemos individuo 4.
- a_3 = 0.742 < 0.856 → Escogemos individuo 6.

Como hemos podido apreciar, debido a la aleatoriedad se ha quedado fuera de la selección el individuo 5 que tenía un valor más alto que el 6 que sí ha sido escogido. En general este operador suele dar muy buen resultado y mantiene las dos características que no deben faltar en un buen algoritmo genético: la característica de selección guiada y la aleatoria.

9.1.5.- Operador de cruce.

Utilizaremos dos operadores de cruce en este algoritmo que iremos alternando cada 10 generaciones:

- ***Operador de cruce monopunto:*** En este caso al contar los individuos con sólo 2 genes, el único punto de cruce será el que los divide. Cada uno de los hijos heredará la mitad del cromosoma del padre y la otra mitad del de la madre. Lo vemos en la figura 9.4.

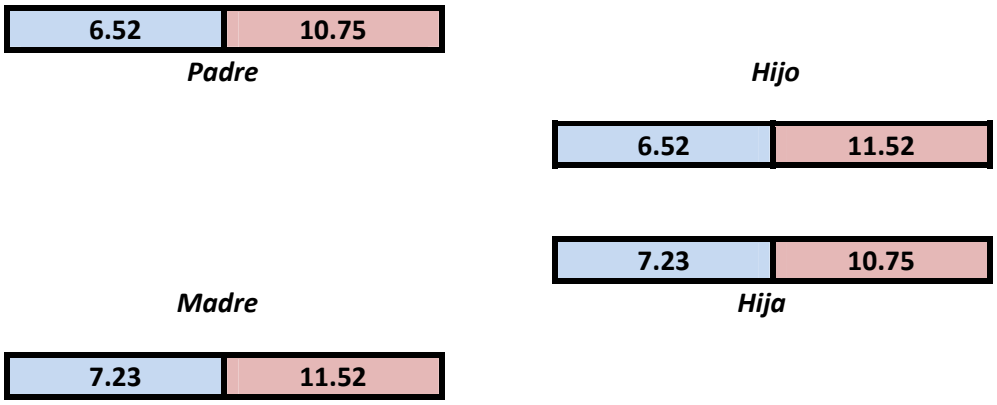


Figura 9.4.- Cruce monopunto.

- ***Operador de cruce aritmético:*** Generaremos dos números aleatorios α y β comprendidos entre 0 y 1. Los hijos los obtendremos de la siguiente manera:

$$h_{1i} = \alpha \cdot P_{1i} + (1 - \alpha) \cdot P_{2i}$$

$$h_{2i} = \beta \cdot P_{1i} + (1 - \beta) \cdot P_{2i}$$

9.1.6.- Operador de mutación.

El operador de mutación, en este caso, tendrá una tasa del 10%. El funcionamiento es el siguiente: una vez los individuos han sido seleccionados y cruzados, se escoge aleatoriamente el 10% de la población. De los individuos escogidos, escogeremos al azar uno de sus genes y lo sustituiremos por un valor aleatorio dentro del rango permisible. Podemos ver su funcionamiento en la figura 9.5:

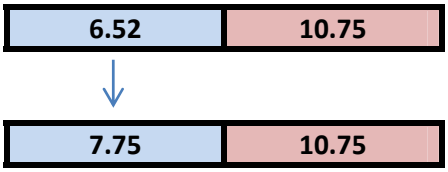


Figura 9.5.- Operador de mutación aplicado al primer gen.

9.1.7.- Reemplazo de la población.

El algoritmo genético que hemos diseñado es un AG con estado estacionario, que como describimos en el apartado 5.7, implica que ciertos individuos de la población pueden pasar a la siguiente sin sufrir cambios y otra parte de la población es renovada y sustituida por la anterior. Para el reemplazo tendremos en cuenta:

- **La élite de la población será del 2%:** Estará formada por los mejores individuos que pasaran duplicados a la siguiente generación para garantizar la permanencia de los mejores genes encontrados.

- **Los hijos sustituirán a sus padres aunque tengan un valor de adaptación menor que el de ellos:** De esta forma garantizaremos la diversidad en la población.

- **El 20% de la población será renovada por individuos creados aleatoriamente:** Así garantizamos una vez más la diversidad en la población, al mismo tiempo que continuamos con la búsqueda aleatoria por el espacio de soluciones, un factor que no debe faltar en ningún AG. Los individuos sustituidos por los nuevos serán los que obtuvieron peores valores de adaptación en la generación anterior.

9.1.8.- Criterio de terminación.

El criterio de terminación que ha sido implementado en el algoritmo es únicamente el alcanzar el número de generaciones máximo que ha sido fijado por el usuario previo a lanzar la optimización.

9.1.9.- Resumen de características.

CARACTERÍSTICAS DEL ALGORITMO	
Nombre	Algoritmo genético básico
Tipo de codificación utilizada	Números reales
Operador de selección	Muestreo estocástico universal
Operador de reproducción	Cruce monopunto y cruce aritmético
Operador de mutación	Reemplazo aleatorio
Elitismo	2 %
Mecanismos de reemplazo	Sustitución incondicional
Criterio de terminación	Por nº máximo de generaciones

9.2.- ALGORITMO GENÉTICO AVANZADO.

En los anteriores apartados describimos la primera solución encontrada al problema de optimización de la estrategia de carrera. Su principal virtud era su simplicidad, al contar sólo los individuos con dos genes, lo que es ideal de cara a programar la primera versión del algoritmo genético, facilitando mucho las labores de depuración de posibles errores y comprobación de la correcta convergencia.

Una vez corregidos los errores encontrados en la primera fase, nos lanzamos a la programación de la versión avanzada de este algoritmo genético. Éste comparte varios aspectos con el anterior y por ello no volveremos a enunciarlos en los siguientes apartados.

Esta segunda variante del problema consistirá en independizar los valores de las velocidades de arrancada y parada del motor de manera que podremos hacer arrancadas de diferente duración en diferentes puntos del circuito. Con la anterior solución del problema nos encontrábamos que éstas tenían un rango fijo independiente de la topografía del circuito, lo que implicaba que se ejecutaba la arrancada de igual forma si la pendiente del circuito era 0, ascendente o descendente.

Aunque la hipótesis de mejorar el consumo haciendo trabajar al motor siempre en su rango óptimo es válida, esta puede ser mejorada evitando que el motor no trabaje cuando por ejemplo está cercano a una pendiente descendente. En este punto, lo ideal sería dejar descender la velocidad un poco más sin realizar una arrancada, para posteriormente aprovechar el impulso que nos da la pendiente descendente para acelerar el vehículo. El anterior algoritmo no contemplaba esta situación, ni por ejemplo el realizar arrancadas en otro rango cuando el vehículo se desplaza en pendiente ascendente.

Este hecho se aprecia en los datos de la simulación que se muestran en la figura 9.6, donde se puede distinguir las arrancadas desiguales que realiza ahora nuestro simulador.

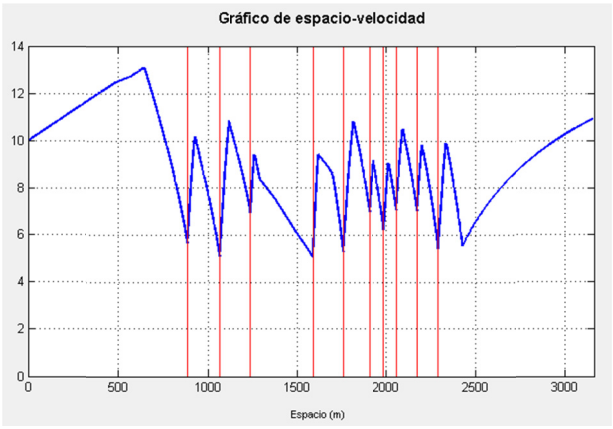


Figura 9.6.- Gráfica velocidades frente al tiempo utilizando arrancadas desiguales.

Es lógico pensar que este tipo de optimización pueda llegar a obtener resultados superiores a la anterior, sobre todo en circuitos con topografía irregular, con varias pendientes, curvas más cerradas que desaconsejen pasos a altas velocidades con el objetivo de no aumentar en exceso la resistencia a rodadura al desplazarnos por ellas,…

9.2.1.- Codificación utilizada.

La codificación utilizada en este algoritmo genético es una ampliación de la que tratamos en el apartado 9.1.1. En este problema, la longitud de los cromosomas será variable, dependiendo del número de arrancadas que fije el usuario. Las variables que tenemos que codificar son una vez más:

- Velocidades de arranque del motor.
- Velocidades de parada del motor.

Utilizaremos codificación por números reales, por ser una solución más natural al problema que la codificación binaria, que en este caso concreto no aportaría ventajas, pero sí dificultaría la labor de programación del algoritmo genético.

El aspecto general de cada individuo puede ser el siguiente (la longitud del cromosoma y los valores que aparecen son aleatorios):

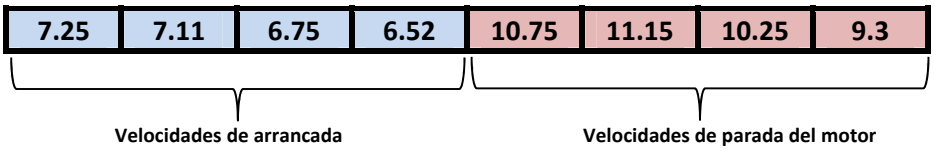


Figura 9.7.- Aspecto del cromosoma de los individuos en el AG avanzado.

Como vemos en la figura 9.7, la primera mitad del cromosoma está compuesta por las velocidades de arranque del motor y la otra mitad del cromosoma (parte coloreada en rojo) está formada por las velocidades de parada del motor. El orden de las arrancadas se establece de izquierda a derecha para ambas mitades del cromosoma.

9.2.2.- Generación de la población inicial.

La generación de la población inicial se realiza de la misma forma que fue descrita en el apartado 9.1.2., pero ahora generando números aleatorios independientes para cada posición del cromosoma (figura 9.8). Para la generación de estos números, utilizaremos la función ***rand()*** de Matlab como hicimos en el anterior algoritmo.

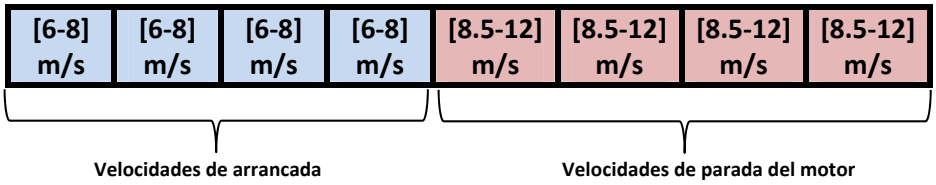


Figura 9.8.- Rangos posibles de los genes en el AG avanzado.

9.2.3.- Función objetivo.

La función objetivo que utilizaremos para evaluar a la población será, al igual que en algoritmo anterior, el kilometraje obtenido por cada individuo mediante su simulación dentro del modelo Simulink del vehículo. Para llevar a cabo correctamente esta simulación necesitamos introducir el cromosoma de cada individuo dentro de nuestro modelo y lanzar la simulación. En este caso a diferencia del presentado en el apartado 9.1.3, al tener arrancadas independientes, el diagrama ***"Modelo de arranque"*** es sustancialmente más complejo que el anterior. En lugar de introducir dos constantes correspondientes a la velocidad mínima y máxima, ahora introducimos medio cromosoma en cada uno de los bloques llamados ***"TablaGA"*** (figura 9.9), correspondientes a las variables de velocidades mínimas y máximas:

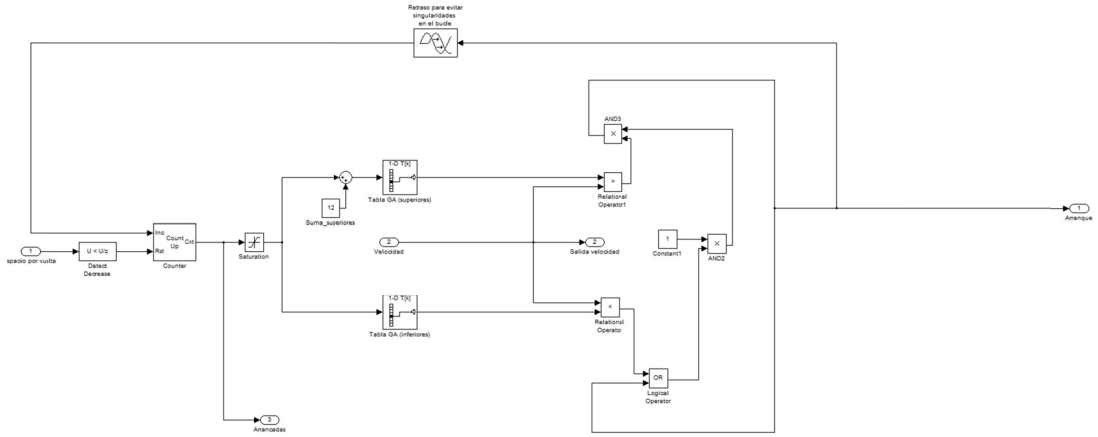


Figura 9.9.- Modelo de arranque para el AG avanzado.

El algoritmo genético, previo a la evaluación de cada individuo, cambiará los valores de los vectores correspondientes a los genes de cada individuo en los bloques ***"TablaGA(inferiores)"*** y ***"TablaGA(superiores)"*** presentes en el modelo de la figura 9.9 por la primera y segunda mitad del cromosoma del individuo respectivamente. Una vez la simulación del individuo ha sido terminada, se leen las salidas proporcionadas por el modelo general:

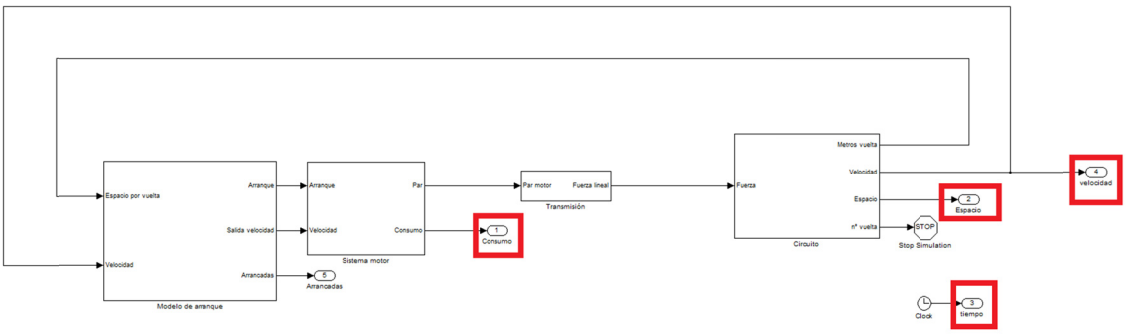


Figura 9.10.- Salidas del modelo general para el AG avanzado.

La función objetivo será por tanto maximizar el kilometraje, que es obtenido al dividir el espacio en Km entre el volumen de combustible en l.

$$\boldsymbol{Kilometraje = \frac{S}{V_{combustible}}}$$

La aceptación o no del valor obtenido por cada individuo está sujeto, como en el caso descrito en el apartado 9.1.3. a las siguientes restricciones:

$$\boldsymbol{V_{media} \geq 30\ km/h}$$

$$\boldsymbol{V_{final} \geq V_{inicial}}$$

El no cumplimiento de alguna de estas dos restricciones, invalidaría el kilometraje obtenido por el individuo pasando su valor a ser de 100km/l, un valor lo suficientemente bajo como para descartarlo casi por completo de las operaciones de reproducción mediante el operador de selección que daría una probabilidad muy baja de ser escogidos a estos individuos. Utilizamos por tanto una ***técnica de penalización*** para el tratamiento de restricciones en este problema. Dentro de las técnicas de penalización se probó a realizar una penalización parabólica en función de la desviación con respecto a las restricciones obtenida por el individuo, pero hacía muy difícil la convergencia del algoritmo al convertir la función objetivo original en otra del tipo multiobjetivo ponderado.

Las técnicas de reparación han sido descartadas en este problema, porque conllevarían un costo computacional importante, ya que implicarían volver a comprobar varias veces cada individuo no válido hasta que cumpliera ambas restricciones y pudiera ser reintroducido en la población. En este caso la reparación sería mucho más compleja que en el caso del algoritmo genético básico ya que ahora la cantidad de variables es mayor haciendo mucho más difícil su ajuste.

9.2.4.- Operador de selección.

Como operador de selección utilizamos al igual que el algoritmo genético básico el muestreo estocástico universal. Para más detalles acerca de su funcionamiento consultar el apartado 9.1.4.

9.2.5.- Operador de cruce.

Utilizaremos dos operadores de cruce en este algoritmo que iremos alternando cada 10 generaciones:

- ***Operador de cruce de dos puntos:*** Se trata de la generalización del cruce de 1 punto. En vez de cortar por un único punto los cromosomas de los padres como en el caso anterior se realizan dos cortes. Deberá tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se origen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre. La particularidad de la aplicación de este operador a nuestro problema es que se aplicarán de igual manera a cada mitad del cromosoma, realizando la transferencia genética de arrancadas completas de padres a hijos en lugar de alguna de las velocidades máximas o mínimas (figura 9.11).

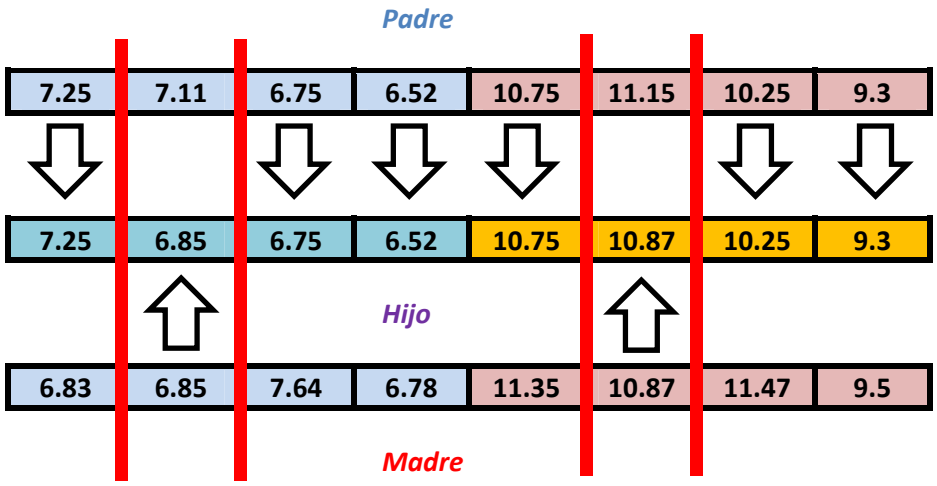


Figura 9.11.- Creación del hijo por el cruce de dos puntos.

- ***Operador de cruce aritmético de dos puntos:*** En este caso utilizamos un sistema similar al anterior, aunque los hijos ahora, en lugar de heredar genes completos de sus padres, éstos se obtienen como consecuencia de la suma de los mismos multiplicados por un número aleatorio entre 0 y 1 (ver apartado 9.1.5). En la figura 9.12 estudiamos el caso para $\alpha = 0.3$.

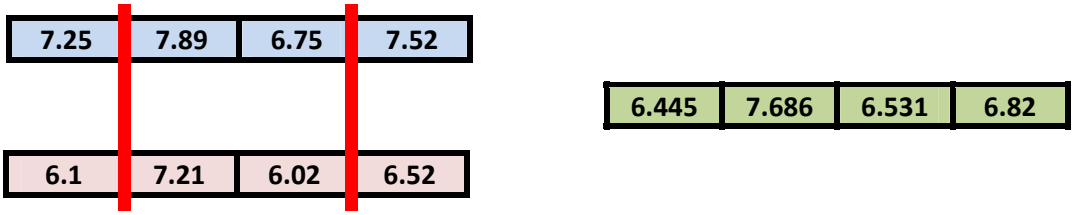


Figura 9.12.- Creación del hijo por el cruce aritmético de dos puntos.

9.2.6.- Operador de mutación.

El operador de mutación, como en el caso anterior, tendrá una tasa del 10%. El funcionamiento es el siguiente: una vez los individuos han sido seleccionados y cruzados, se escoge aleatoriamente el 10% de la población. El algoritmo genético avanzado incluye dos tipos de mutación que se escogen de manera aleatoria al finalizar la reproducción. Cada uno tiene la misma probabilidad de ser escogido que el otro.

- ***Mutación aleatoria:*** De cada individuo escogido se selecciona al azar uno de sus genes y se sustituye por un número aleatorio dentro del rango establecido.

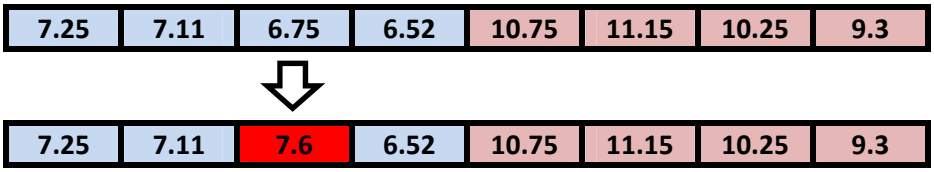


Figura 9.13.- Operador de mutación aplicado al tercer gen.

- ***Mutación por intercambio repetido:*** En este caso se escogen dos genes correspondientes del individuo y se intercambian sus posiciones. Hay que tener en cuenta que el intercambio afectará a los pares de velocidades que forman una arrancada. Se intercambiarán por igual velocidades mínimas y máximas (figura 9.14).

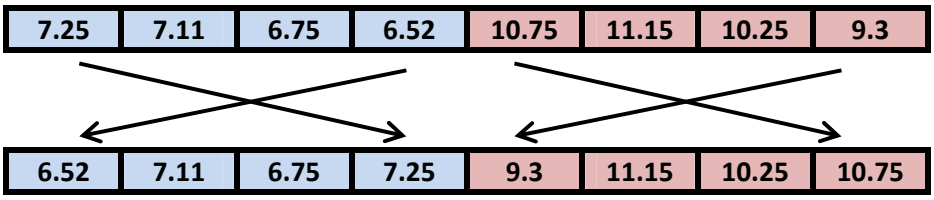


Figura 9.14.- Mutación por intercambio repetido.

9.2.7.- Reemplazo de la población.

El algoritmo genético avanzado, utiliza el mismo mecanismo de reemplazo que el básico. Es un AG con estado estacionario, que como describimos en el apartado 5.7, implica que ciertos individuos de la población pueden pasar a la siguiente sin sufrir cambios y otra parte de la población es renovada y sustituida por la anterior. Para el reemplazo tendremos en cuenta:

- **La élite de la población será del 2%:** Estará formada por los mejores individuos que pasaran duplicados a la siguiente generación para garantizar la permanencia de los mejores genes encontrados.

- **Los hijos sustituirán a sus padres aunque tengan un valor de adaptación menor que el de ellos:** De esta forma garantizaremos la diversidad en la población.

- **El 20% de la población será renovada por individuos creados aleatoriamente:** Así garantizamos una vez más la diversidad en la población, al mismo tiempo que continuamos con la búsqueda aleatoria por el espacio de soluciones, un factor que no debe faltar en ningún AG. Los individuos sustituidos por los nuevos serán los que obtuvieron peores valores de adaptación en la generación anterior.

9.2.8.- Criterio de terminación.

El criterio de terminación que ha sido implementado en el algoritmo es únicamente el alcanzar el número de generaciones máximo que ha sido fijado por el usuario previo a lanzar la optimización.

9.2.9.- Resumen de características.

CARACTERÍSTICAS DEL ALGORITMO	
Nombre	Algoritmo genético avanzado
Tipo de codificación utilizada	Números reales
Operador de selección	Muestreo estocástico universal
Operador de reproducción	Cruce de dos puntos y cruce aritmético
Operador de mutación	Reemplazo aleatorio y mutación por intercambio repetido
Elitismo	2 %
Mecanismos de reemplazo	Sustitución incondicional
Criterio de terminación	Por nº máximo de generaciones

10.- INTERFAZ GRÁFICA.

La creación de la interfaz gráfica que controle las optimizaciones tiene dos objetivos principales:

- Facilitar al usuario la tarea de definición de datos.
- Facilitar la observación de todos los resultados que el algoritmo va generando.

En un principio no estaba planteado dentro de los objetivos de la tesina que el modelo y el algoritmo genético contaran con una interfaz gráfica de usuario. Ya que Matlab incorpora la herramienta GUI (figura 10.1), se aprovechó la realización de este trabajo para tener una primera toma de contacto con ella y aprender a realizar pequeños proyectos cuya interfaz no sea excesivamente compleja, como es el caso que nos ocupa.

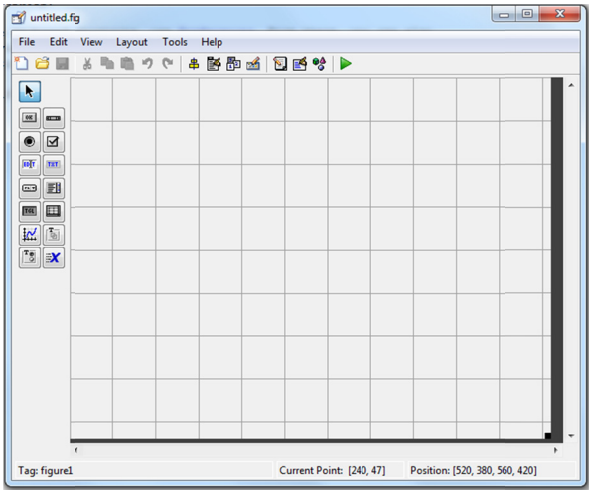


Figura 10.1.- Entorno de Matlab GUI

Aunque su entorno es parecido al conocido Visual Studio de Microsoft, su manejo y programación son mucho más complejos que en éste. Además esta herramienta, susceptible de ser muy mejorada en un futuro, adolece de una gran falta de elementos gráficos para añadir a nuestros proyectos. No es por tanto ésta, la herramienta ideal para crear complejas interfaces de usuario con varios formularios y multitud de elementos gráficos. Su aplicación es más bien, como hemos comentado, el proveer al usuario de un entorno más amigable que la ventana de comandos de Matlab y la capacidad de presentar los resultados de nuestros programas de una forma más directa.

En un principio de hizo de un estudio de los requerimientos que debería tener una buena interfaz para la aplicación de la optimización de estrategia de carrera. Son los siguientes:

- Capacidad para introducir los datos físicos más habituales para las simulaciones.
- Ocultar los datos fijos en el vehículo como son el C_x, los coeficientes de rozamiento por rodadura de neumáticos y rodamientos, la potencia del

motor,… todos ellos datos que normalmente no deberá cambiar el usuario en condiciones normales. De esta manera se simplifica su aspecto.

- Capacidad para elegir entre varios circuitos.
- Capacidad para escoger el tipo de AG deseado y variar sus parámetros: número de individuos, nº de generaciones máximas.
- Presentación adecuada de los resultados.

Con estos requerimientos como premisa se comenzó la construcción de la misma. El modo de construcción de la interfaz es similar al VB de Microsoft: basta con elegir los elementos de izquierda y arrastrarlos al escritorio. En la figura 10.2 podemos apreciar la apariencia de la interfaz en la primera fase de su creación:

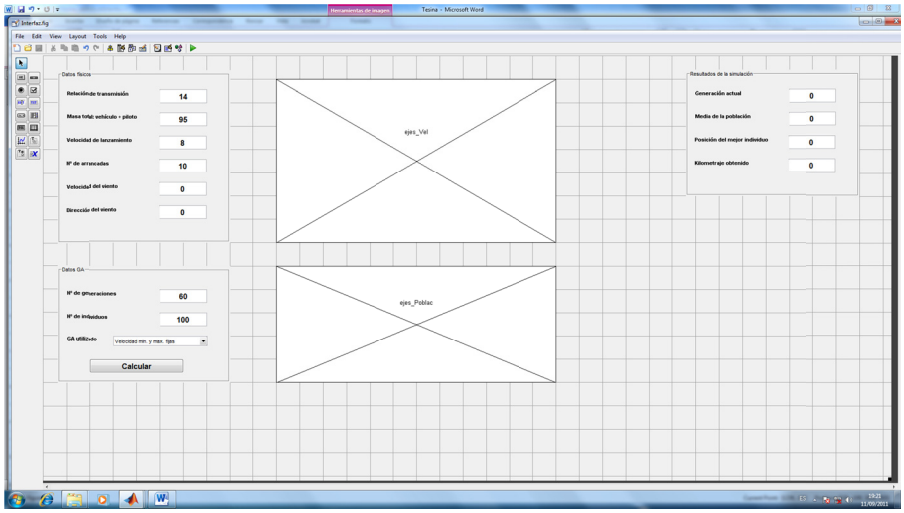


Figura 10.2.- Aspecto de la interfaz en su primera fase de diseño.

En la figura 10.3 podemos ver el aspecto final de la interfaz con los resultados correspondientes a una simulación.

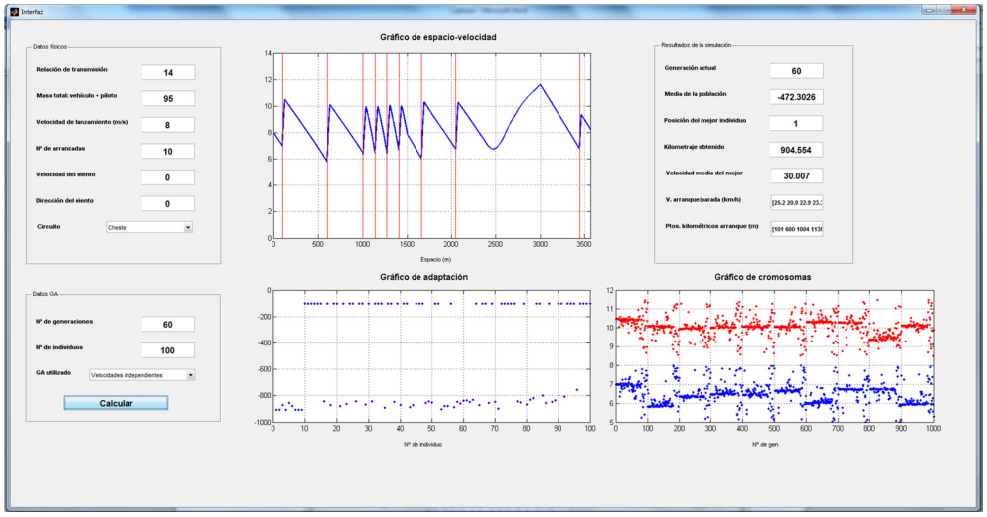


Figura 10.3.- Aspecto final de la interfaz al acabar una optimización.

10.1.- CARACTERÍSTICAS PRINCIPALES Y FUNCIONAMIENTO DE LA INTERFAZ GRÁFICA.

Observando la interfaz en su estado final (figura 10.3) podemos apreciar que las distintas cajas de texto se encuentran divididas en tres paneles principales. El primer panel, que se encuentra en la parte superior izquierda, corresponde a los datos físicos de entrada para la simulación (figura 10.4).

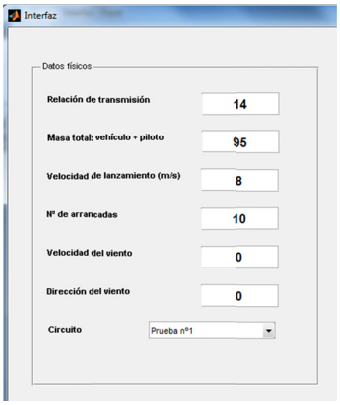


Figura 10.4.- Panel de datos físicos de la interfaz.

Podemos distinguir los siguientes campos:

- ***Relación de transmisión:*** En esta caja de texto introduciremos la relación de transmisión escogida para la prueba. Las relaciones disponibles van desde 12:1 a 20:1.

- ***Masa total:*** En este cuadro de texto introduciremos la masa en kg del pilo más el vehículo. El peso del vehículo actual ronda los 50 kg.

- ***Velocidad de lanzamiento:*** Aquí se establece la velocidad del vehículo en el paso por meta.

- ***Nº de arrancadas:*** Aquí pondremos el nº de arrancadas aproximado con el que trabajará nuestro vehículo en la prueba. **El valor del mismo debe ser establecido siempre igual o mayor que el deseado, pues la optimización más tarde tenderá a reducirlo si el valor inicial no era óptimo.**

- ***Velocidad del viento:*** En esta caja de texto debemos introducir el módulo de la velocidad del viento en (m/s).

- ***Dirección del viento:*** Aquí indicaremos la dirección del viento en grados respecto al Norte.

- ***Circuito:*** En este menú desplegable escogeremos el circuito donde queremos realizar la optimización.

La siguiente caja de texto es la correspondiente a los datos de entrada para la optimización. Está situada en la parte inferior izquierda de la interfaz (figura 10.5):

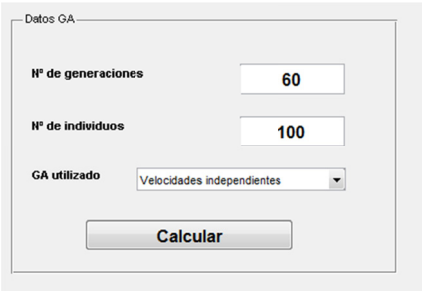


Figura 10.5.- Panel de datos de entrada para la optimización.

En este panel sólo tenemos que proporcionarle los siguientes datos:

- ***Nº de generaciones:*** Este dato marca el criterio de terminación para el algoritmo genético. 60 generaciones ha sido un número suficiente para que el algoritmo converja en todos los casos de prueba donde ha sido estudiado.

- ***Nº de individuos:*** Este dato marca el número de soluciones que se evaluarán en cada generación. Se ha comprobado que el valor óptimo de individuos y que tiene una mejor relación entre la calidad de la solución obtenida y el tiempo de cálculo es 100. Valores muy grandes podrían llegar quizá a una solución ligeramente mejor, pero por el contrario ralentizan mucho el proceso.

- ***GA utilizado:*** En este menú desplegable escogeremos entre los dos tipos de optimizaciones que hemos explicado en el capítulo 9: optimización basada en V_mín. y V_máx fijas durante toda la vuelta, y optimización con arrancadas de velocidades independientes.

Una vez que estos datos han sido completados, ya podemos lanzar la simulación pulsando el botón "***Calcular***". Durante la simulación, tendremos 3 gráficas con resultados y un cuadro de texto con información acerca de la mejor solución obtenida de todas las probadas e información acerca de la población del algoritmo genético.

La primera de las gráficas es la que mostramos en la figura 10.6, que nos mostrará la gráfica velocidad-espacio de la mejor solución encontrada hasta el momento. En ella tendremos información acerca del perfil de velocidades obtenido con la mejor solución (mostrado por la línea azul) y también los puntos kilométricos de arrancadas, delimitados por las bandas rojas verticales.

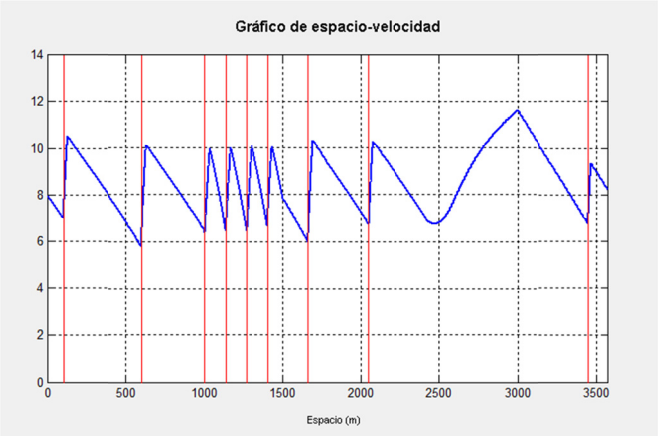


Figura 10.6.- Gráfica de velocidad-espacio.

La segunda de las gráficas (figura 10.7), nos da información acerca de la última población evaluada por el algoritmo. Cada punto en ella representa un individuo, siendo el eje horizontal el nº del mismo dentro de la matriz de población, y el eje vertical representa el valor obtenido en la optimización. Hay que recordar que los valores están en negativo, y por tanto individuos que se encuentren más hacia debajo, tendrán una mayor adaptación a nuestro problema. Todos los individuos que vemos situados en el valor -100 son aquellos que han sido descartados por violar alguna de las restricciones impuestas en el problema.

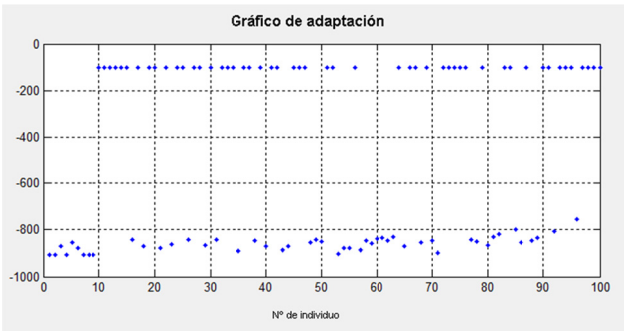


Figura 10.7.- Gráfico de adaptación de los individuos.

Por último, el gráfico de cromosomas (figura 10.8), representa las velocidades mínimas y máximas de arranques de los distintos individuos. Este gráfico nos da una idea del estado de convergencia en el cual se encuentra la optimización. Cuando los puntos se encuentran muy juntos, significa que la mayoría de los genes de los individuos están convergiendo al mismo valor. En este caso, la figura 10.8 muestra el estado en el que acabó la población al final de la optimización. Se puede apreciar la situación antes expuesta, concluyendo que la solución había convergido. Los puntos azules representan las velocidades mínimas y los rojos las velocidades máximas. Cada división de 100 separa una arrancada de otra, ya que el tamaño de individuos escogido fue de 100. **Nótese que siempre quedan algunos puntos dispersos, garantizando la búsqueda aleatoria del AG en todo el espacio de soluciones.**

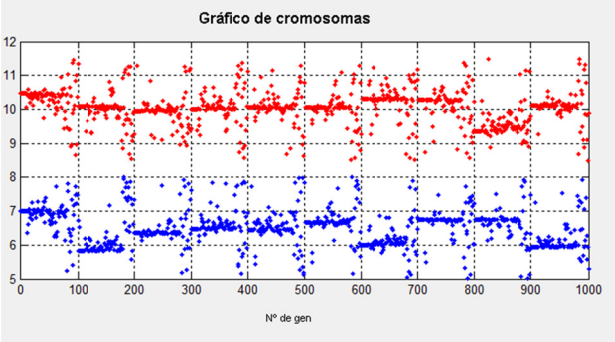


Figura 10.8.- Gráfico de cromosomas de los individuos

Por último el panel de datos de la simulación (figura 10.9), en el que se nos proporciona la siguiente información:

- ***Generación actual:*** En esta caja de texto la interfaz nos indicará el nº de la generación que está siendo evaluada.

- ***Media de la población:*** En este cuadro de texto se nos indica cuál es la media de los valores de adaptación de los individuos evaluados en la generación anterior

- ***Posición del mejor individuo:*** En este cuadro se nos informa de la posición del mejor individuo en la matriz de población. La élite la formarán los dos primeros individuos y por ello cada vez que se encuentre el mejor individuo este pasará a las primeras posiciones en la matriz para la generación siguiente.

- ***Kilometraje obtenido:*** Indica el valor de km/l obtenidos por el mejor individuo.

- ***Velocidad media:*** Indica la velocidad media del mejor individuo durante la vuelta al circuito simulada en (m/s).

- ***Velocidades de arranque y parada:*** En esta caja de texto se encuentra el vector con las velocidades mínimas y máximas en km/h del mejor individuo. Por cuestión de elegancia se ha dejado una caja pequeña y para ver el vector completo necesitaremos recorrerla con el cursor.

- ***Ptos. Kilométricos de arrancada:*** En esta caja de texto se nos mostrará el vector con los puntos correspondientes a los arranques realizados por el mejor individuo. Una vez más por cuestión de elegancia se ha evitado el poner un cuadro de texto de longitud suficiente para mostrar el vector de una vez, lo que hubiera supuesto "romper" la estética de la interfaz. Por ello aunque la caja de texto es pequeña, recorriéndola con el cursor podremos ver completamente dicho vector.

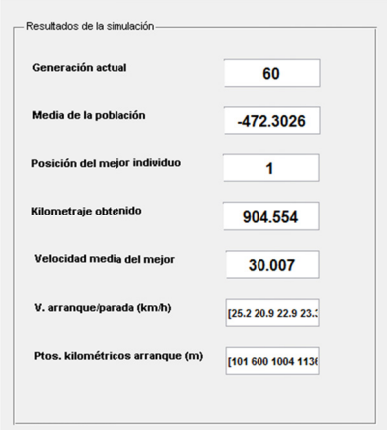


Figura 10.9.- Panel de resultados de la simulación.

11.- PRUEBAS DEL MODELO.

En el apartado anterior estudiamos todo lo referente a la implementación del modelo matemático del vehículo. Este modelo puede ser utilizado simplemente para simular una estrategia deseada por el usuario o lo que es más interesante, **utilizado para calcular la estrategia óptima de carrera**, dado un circuito y unas condiciones de carrera, reglajes del vehículo,…

Las etapas en las que podemos dividir las pruebas del modelo son las siguientes:

1) ***Pruebas del modelo sin optimización:*** Estas pruebas se realizaron para verificar la robustez del modelo y corregir los posibles errores en su funcionamiento, así como para ajustar diferentes parámetros.

2) ***Primeras pruebas de optimización con el GAToolbox del CPOH:*** En los primeros 2 meses de realización de la tesina, la totalidad de las simulaciones y las pruebas con distintos modelos de codificación se realizaron con el módulo para Matlab, GAToolbox del Grupo de Control Predictivo y Optimización Heurística de la UPV.

3) ***Realización del AG propio y comparativa con el anterior:*** Debido a algunas limitaciones de este AG, se decidió comenzar la creación de otro AG que se adaptara de manera más específica a nuestro problema.

4) ***Pruebas finales en diferentes trazados, con diferentes parámetros del AG y comparativa entre las distintas vertientes:*** Las pruebas en esta última fase tuvieron como objeto verificar la robustez del nuevo AG como de la interfaz gráfica, comparándolo en distintos trazados con el anterior para distintos parámetros.

En un principio y para simplificar el problema, se modificó el algoritmo para determinar el rango de velocidades óptimo para cada circuito. De esta forma, existe una velocidad mínima a la que el vehículo una vez que llega esta realiza un arranque, y una velocidad máxima, a la cual una vez ha sido rebasada, se para el motor. De este modo, se persigue el hacer funcionar al motor siempre en un rango óptimo, donde alcanza mayor eficiencia. Este rango depende de las características con las que fue diseñado el motor. Por tanto, la relación de transmisión será un factor muy a tener en cuenta de cara a hacer coincidir este rango óptimo con las velocidades necesarias para obtener una media ≥ 30 km/h.

De esta forma, en una primera aproximación del problema, haremos arrancadas similares sea cual sea el punto del circuito en el que nos encontramos, y estas sólo dependerán de la velocidad del vehículo y no de la topografía del circuito.

-Una vez resuelto este primer problema, y obtenidos resultados sobre distintos escenarios de pruebas de cara a poder compararlos en un futuro, se planteó la siguiente evolución: independizar las velocidades mínimas y máximas de arrancada/parada para poder obtener distintos valores a lo largo del trazado. Este planteamiento resultó ser complejo debido a diversos factores:

- Inestabilidad de las velocidades en diferentes vueltas.
- Tiempo de cálculo excesivo.
- Necesidad de un tamaño de población muy grande.

11.1.- CIRCUITOS DE PRUEBA

Para poder valorar la robustez del algoritmo y del modelo, se han creado 3 circuitos ficticios simples, con diferentes tipos de pendientes y radios de curvatura. Además se ha modelado el circuito "Ricardo Tormo" de Cheste, en su versión más corta, donde tuvimos ocasión de hacer unas pruebas en el mes de Mayo de este mismo año. A continuación vemos los circuitos de prueba y sus perfiles de altimetría:

- ***Circuito nº 1:*** Es el más simple de todos. Su plano podemos verlo en la figura 11.1. Es del tipo óvalo, con dos curvas de 200m de radio, suficiente como para no influir penalizando el coche por resistencia a rodadura adicional en éstas y **carece de pendientes en toda su longitud**.

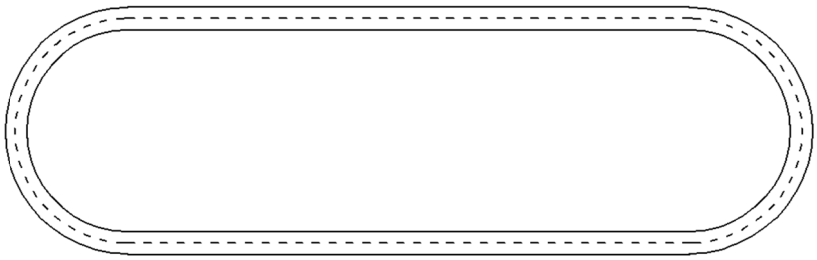


Figura 11.1.- Circuito nº 1 de forma oval y sin pendientes.

- ***Circuito nº 2:*** Tiene un plano similar al anterior, pero este consta ahora de dos pendientes, una de subida y otra de bajada situadas cada en las curvas. En la figura 11.2 vemos su plano y altimetría.

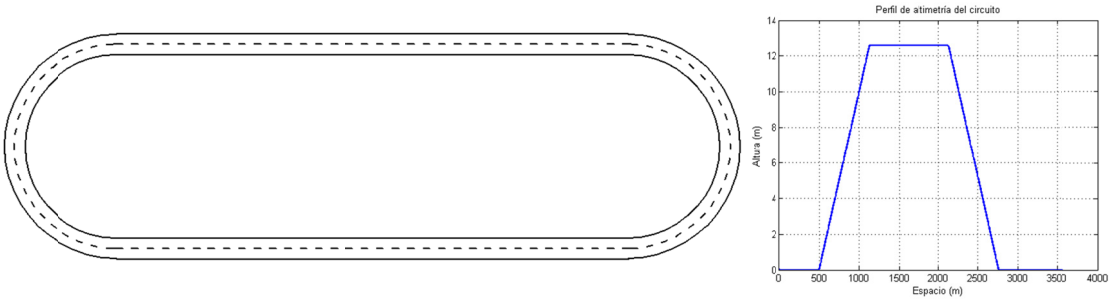


Figura 11.2.- Plano del circuito nº 2 y altimetría.

- ***Circuito nº 3:*** Este circuito consta de una forma más compleja, varias curvas con radios cercanos a los 100 en los que ya su influencia se dejará notar. Su perfil de altimetría es más irregular que el de los anteriores.

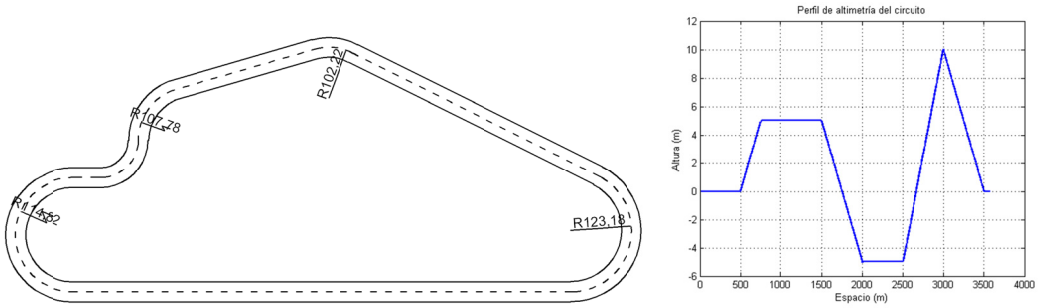


Figura 11.3.- Plano y perfil de altimetría del circuito de prueba nº 3

- ***Circuito Ricardo Tormo:*** Utilizaremos para las simulaciones la versión más corta de este. En la figura 11.4 tenemos representado su plano y en la figura 11.5 su perfil de altimetría.

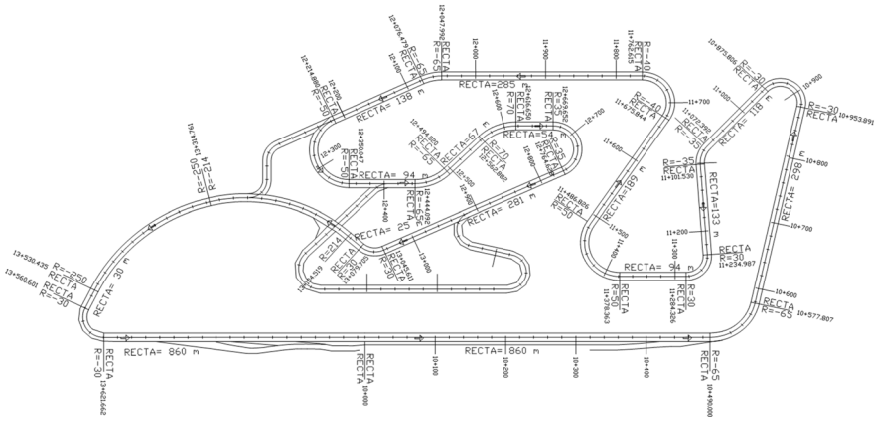


Figura 11.4.- Plano del circuito Ricardo Tormo de Cheste.

Figura 11.5.- Altimetría del Circuito Ricardo Tormo de Cheste.

11.2.- PRUEBAS DEL MODELO SIN OPTIMIZACIÓN

Estas pruebas fueron las primeras a las que sometimos el modelo matemático en Matlab (figura 11.6) del vehículo IDF Eco-Shell Marathon. Se encaminaron a verificar la robustez del modelo, hacer un análisis de los errores que se pudieran presentar y estudio de la estrategia más acertada para corregirlos.

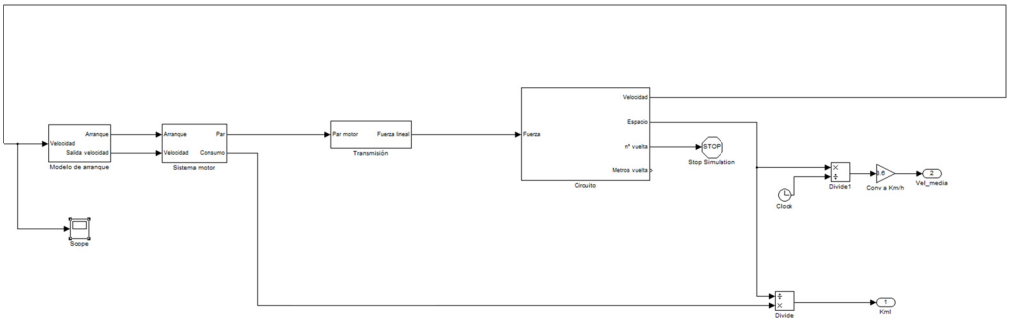


Figura 11.6.- Modelo general utilizado en las primeras pruebas.

El sistema de arranque utilizado en este primer modelo es el mismo que se utilizará en las primeras optimizaciones más simples (figura 11.7). Corresponde a fijar una velocidad mínima a la cual el vehículo enciende el motor una vez que su velocidad instantánea desciende por debajo de ese nivel, y una velocidad máxima a la cual el vehículo apaga el motor una vez que rebasa su valor. Podemos ver en la siguiente figura el detalle del diagrama de bloques que representa el "cerebro" del vehículo:

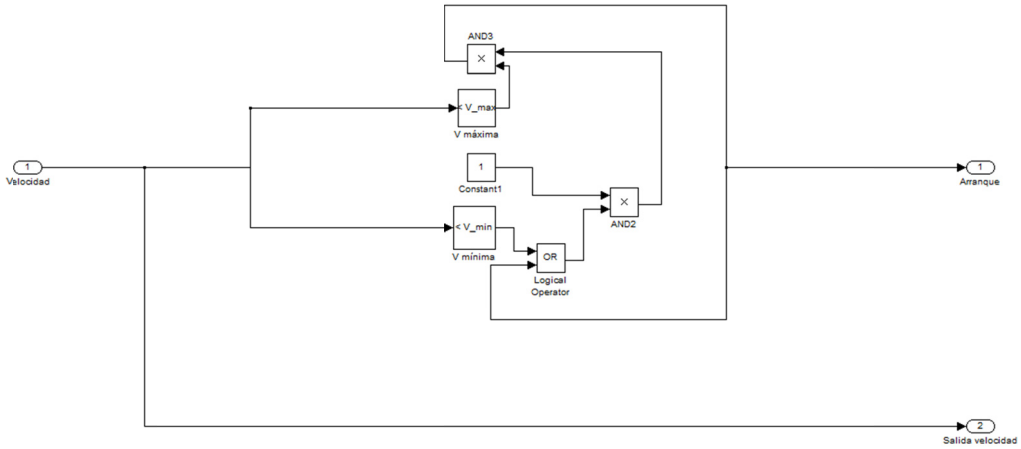


Figura 11.7.- Modelo de arranque simple utilizado en las primeras pruebas.

Toma como entrada únicamente la velocidad instantánea del vehículo, y mediante dos bloques "compare" va muestreando este valor con los valores de referencia previamente fijados. Con la ayuda de los bloques condicionales se genera la señal binaria de salida "0" ó "1" que representará "motor parado" o "motor encendido" respectivamente.

Mediante el uso de un gráfico compartido podemos ver las fluctuaciones que las distintas fuerzas de arrastre sufren a lo largo de la vuelta al circuito nº 1 simulada (figura 11.8). Se aprecian las fluctuaciones en la fuerza aerodinámica debido a los cambios de velocidad. En este circuito, la fuerza de arrastre es nula al ser completamente llano, y la resistencia a rodadura en curvas también es 0 al ser los radios de las mismas de 200m.

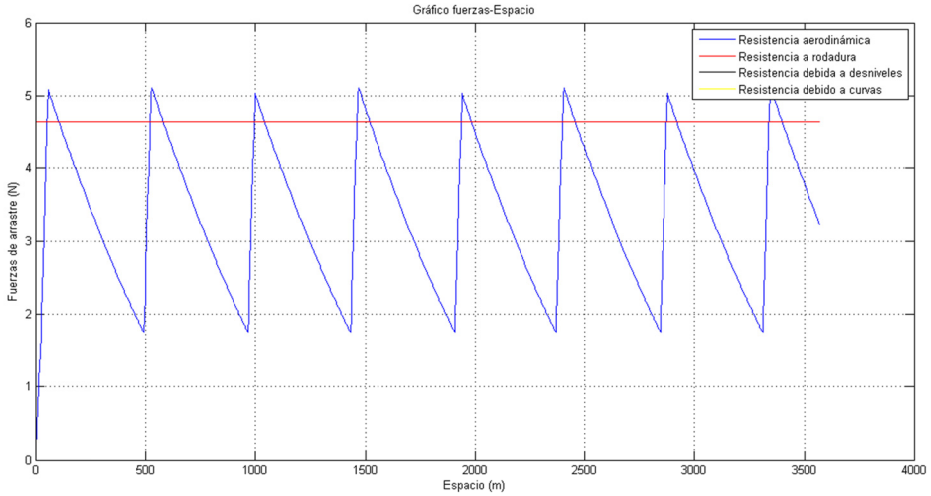


Figura 11.8.- Fuerzas de arrastre en el circuito nº 1

En la figura 11.9 se muestra las velocidades frente al tiempo. Se aprecia el funcionamiento del arranque por "histéresis", donde existe un valor de velocidad máximo a la cual el motor se para y un valor inferior a la cual el motor arranca. Se ve claramente que todas las crestas tienen igual anchura, lo que significa que las velocidades siempre aumentan y disminuyen a igual ritmo. Esto es causa de la planitud del circuito y de la ausencia de curvas muy cerradas.

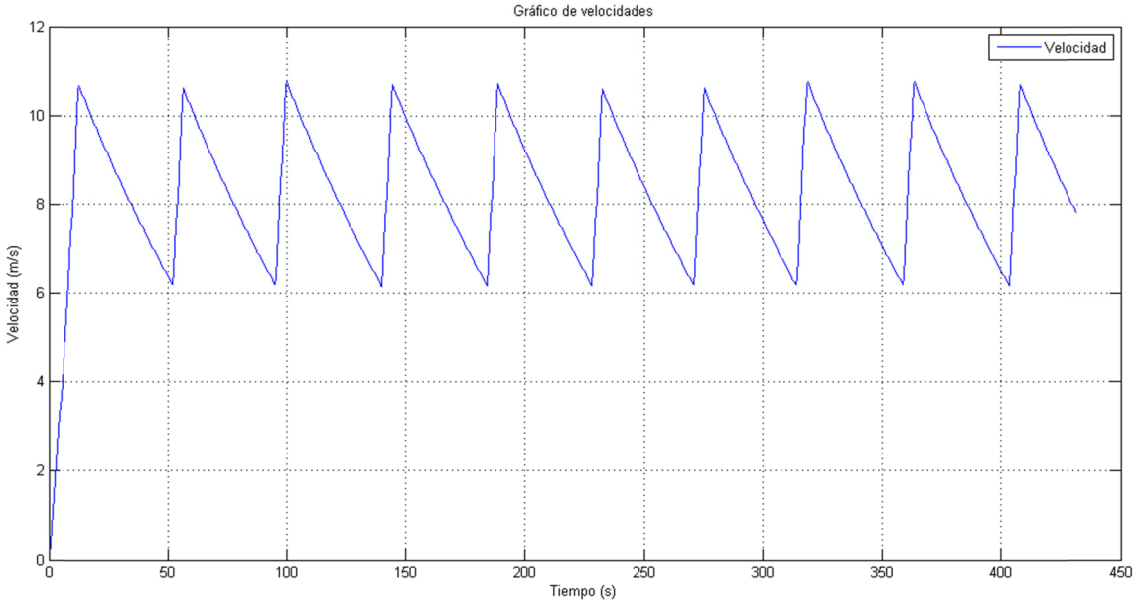


Figura 11.9.- Gráfico de velocidades

11.2.1.-Circuito nº 2

Este circuito es igual al anterior, a excepción de los desniveles, lo que se deja notar en las fuerzas de arrastre (figura 11.10). En ella se puede apreciar como justo al comenzar la pendiente ascendente la fuerza se dispara hacia casi los 20N manteniéndose constante durante la subida. También se puede ver como las crestas del arrastre aerodinámico frente al espacio se comprimen, siendo necesarias más arrancadas en esta zona de pendiente ascendente. En la pendiente descendente ocurre lo contrario, obteniendo ahora una fuerza de empuje y aumentando su velocidad sin necesidad de realizar arrancada.

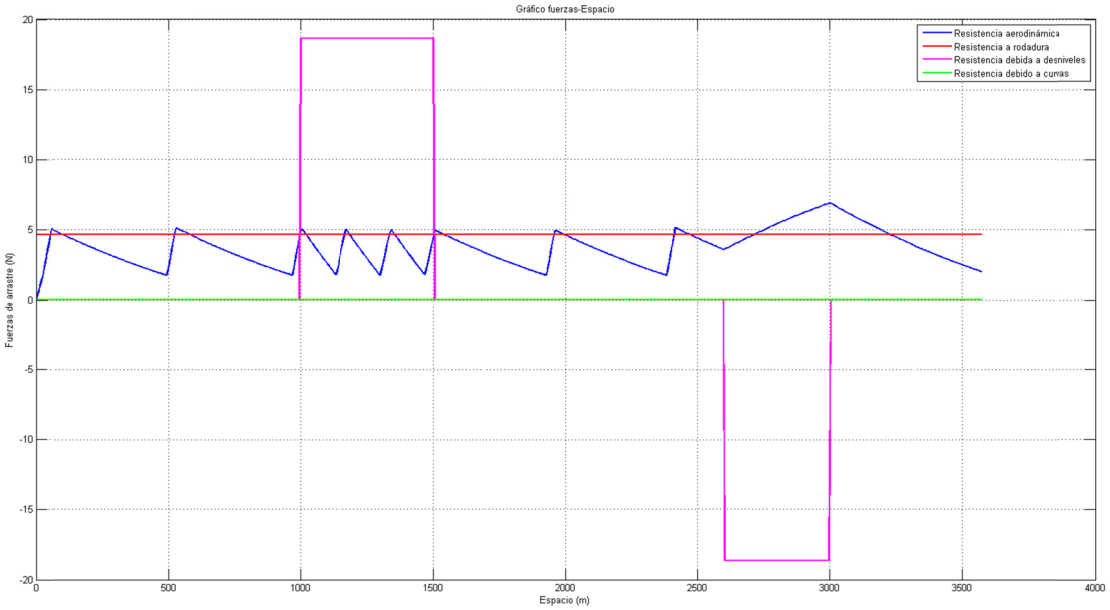


Figura 11.10.- Gráfica de fuerzas de arrastre en el circuito nº 2

En la gráfica de velocidades, figura 11.11, apreciamos cómo las crestas se comprimen debido a que en la pendiente ascendente son necesarias más arrancadas, al caer mucho más rápidamente la velocidad. En la pendiente ascendente, el aumento de la velocidad debido al motor ahora no es tan rápido y se puede apreciar cómo la pendiente de la primera parte de la cresta ha cambiado. En la pendiente descendente, la velocidad aumenta libremente.

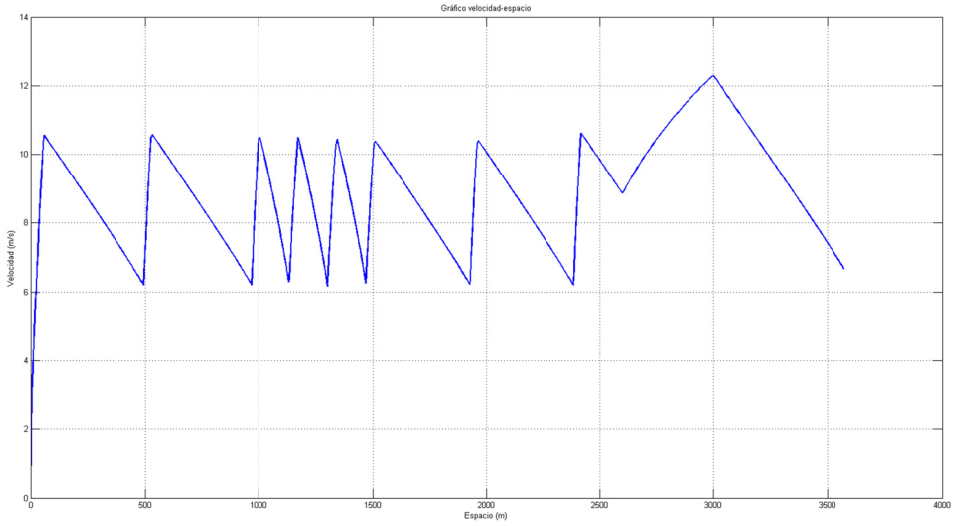


Figura 11.11.- Gráfico de velocidades frente al espacio en el circuito nº 12.

11.3.- OPTIMIZACIÓN EN EL CIRCUITO Nº 1

En este apartado, estudiaremos los resultados obtenidos por las optimizaciones en el circuito nº 1 y los compararemos entre sí. Este circuito se asemeja a un llano perfecto, ya que sus curvas son lo suficientemente amplias como para no influir negativamente en las velocidades. Utilizaremos para todas las pruebas la relación de transmisión de 14 por ser la que ha obtenido mejores resultados.

Realizando la optimización con el AG básico los resultados obtenidos son los mostrados en la figura 11.12. Se obtuvo una mejor marca de 957 km/l de combustible, siendo las velocidades de arranque y parada 22.8 y 38.3 km/h.

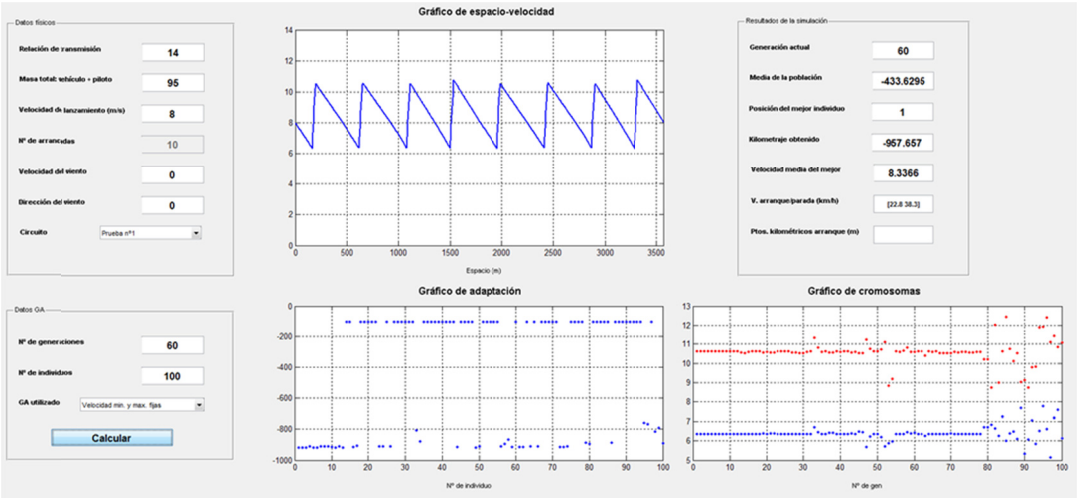


Figura 11.12.- Optimización mediante GA básico en circuito nº 1

Realizando la optimización con el GA avanzado se obtiene un kilometraje de sólo 5km más que en el caso anterior, confirmándonos que en el caso del llano, la optimización mediante velocidades independientes no es útil.

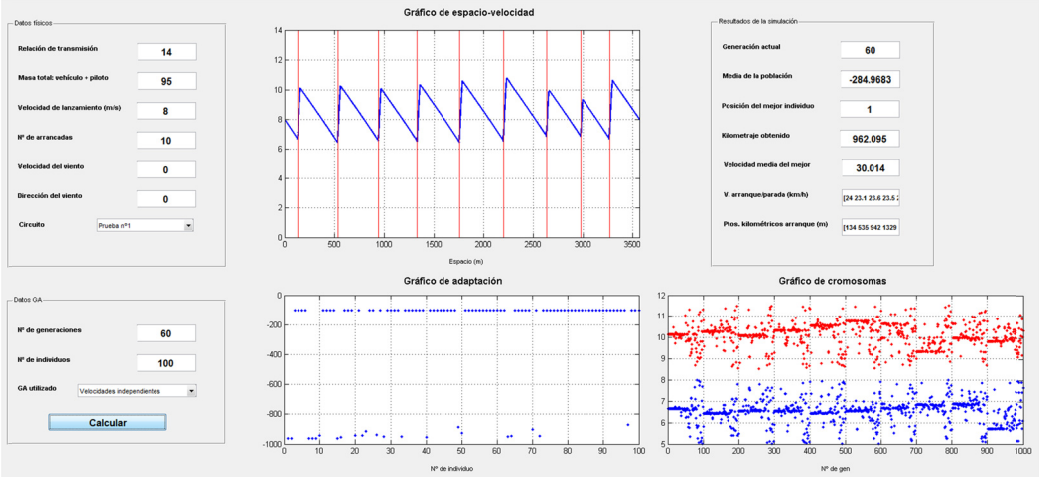


Figura 11.13 Optimización mediante el AG avanzado en el circuito nº 1.

Realizando la optimización mediante el AG básico del CPOH, el kilometraje obtenido es exactamente el mismo que con el AG básico nuestro, confirmando, como en el caso anterior, la no necesidad de optimizar mediante velocidades independientes circuitos en llano.

```
------------------------------------------------
Generación: 60
   Fun. objetivo: -957.2719 - Media población:-666.4848 -
   xmin: [6.63220446585127 6.97408704694071 6.776850336996
------------------------------------------------
######   RESULTADO   #########
   Función objetivo: -957.2719
   xmin: [6.63220446585127 6.97408704694071 6.776850336996
------------------------------------------------
```

Figura 11.14.- Resultados de la optimización mediante el GA del CPOH en el circuito nº 1.

En este caso de optimización en llano, la diferencia es inapreciable, y la pequeña variación de kilometrajes obtenido puede deberse más al tratamiento del programa de los decimales del consumo tan ínfimo durante una vuelta que a la verdadera ventaja de optimizaciones mediante velocidades independientes, ya que en el caso del GA básico y el del CPOH estos valores coinciden. Por tanto, podemos concluir que **no es aconsejable optimizar en circuitos llanos mediante los dos últimos algoritmos debido a que son computacionalmente más lentos y su uso no aporta ninguna ventaja.**

OPTIMIZACIÓN EN EL CIRCUITO Nº 1		
ALGORITMO UTILIZADO	*KILOMETRAJE OBTENIDO*	*% DE MEJORA*
GA básico	957	0
GA avanzado	962	0.5
GA CPOH	957	0

11.4.- OPTIMIZACIÓN EN EL CIRCUITO Nº 2

En este apartado, estudiaremos los resultados obtenidos por las optimizaciones en el circuito nº 2 y los compararemos entre sí. En este circuito, al contar con pendientes, se debería dejar notar la optimización con arrancadas desiguales, frente a la optimización mediante el AG simple. Utilizaremos para todas las pruebas la relación de transmisión de 14 por ser la que ha obtenido mejores resultados.

Realizando la optimización con el AG básico los resultados obtenidos son los mostrados en la figura 11.15. Se obtuvo una mejor marca de 849 km/l de combustible, siendo las velocidades de arranque y parada 23.6 y 36.1 km/h.

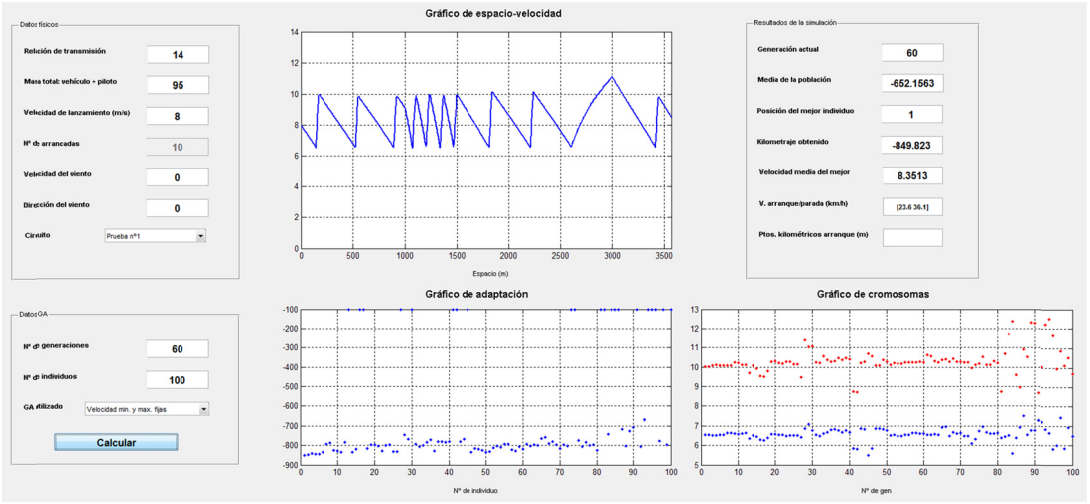


Figura 11.15.- Optimización realizada por el AG básico en el circuito nº 2

En este mismo circuito optimizando mediante el algoritmo genético avanzado se obtuvo una mejor marca de 899 km/l, mejorando en un 5.8 %

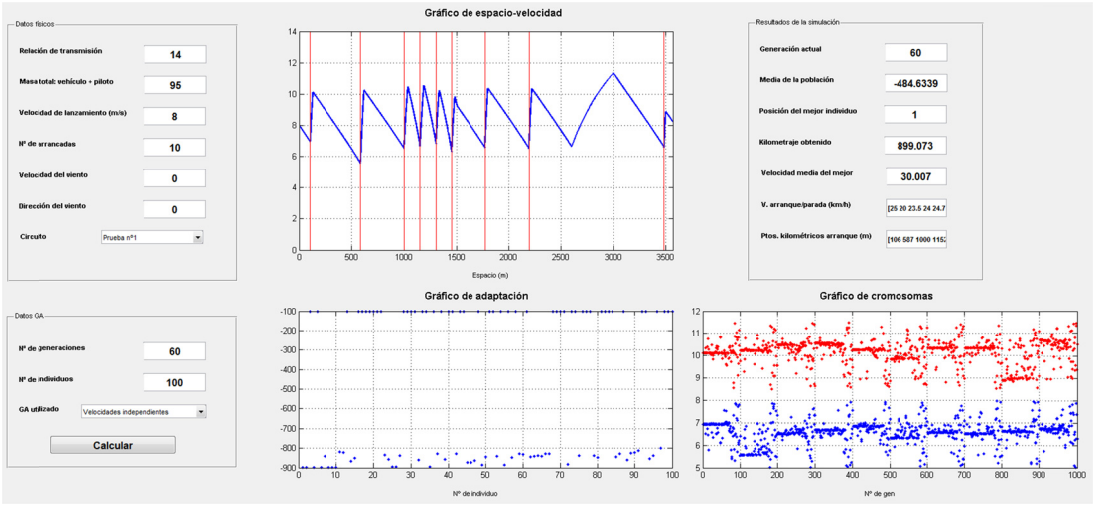


Figura 11.16.- Optimización realizada mediante el AG avanzado en el circuito nº 2.

Mediante la optimización con el AG básico del CPOH, el resultado obtenido fue de 896km/l, una marca ligeramente menor a la obtenida con el algoritmo genético avanzado desarrollado por nosotros.

```
------------------------------------------------
######    RESULTADO    #########
   Función objetivo: -896.5236
   xmin: [6.86363806905952 6.51726479797749
---------------------
```

Figura 11.17.- Resultados obtenidos por la optimización el circuito nº 2 mediante el GA CPOH

En este caso la marca obtenida por nuestro GA avanzado prácticamente coincide con la del GA básico del CPOH, lo que nos hace pensar que ambos hayan alcanzado el óptimo global en el problema de optimización en este circuito. En este circuito sí se deja notar ya una diferencia apreciable entre optimizar mediante un rango de velocidades fijo a un rango de velocidades variable como son los dos últimos casos.

OPTIMIZACIÓN EN EL CIRCUITO Nº 2		
ALGORITMO UTILIZADO	**KILOMETRAJE OBTENIDO**	**% DE MEJORA**
GA básico	**849**	**0**
GA avanzado	**899**	**5.8**
GA CPOH	**896**	**5.5**

11.5.- OPTIMIZACIÓN EN EL CIRCUITO Nº 3

En este apartado, estudiaremos los resultados obtenidos por las optimizaciones en el circuito nº 3 y los compararemos entre sí. En este circuito, aunque los radios de las curvas son amplios, cuenta con un perfil de altimetría más irregular con varias pendientes de diferente inclinación. Realizando la optimización con el AG básico los resultados obtenidos son los mostrados en la figura 11.17. Se obtuvo una mejor marca de 834 km/l de combustible, siendo las velocidades de arranque y parada 19.9 y 37.6 km/h.

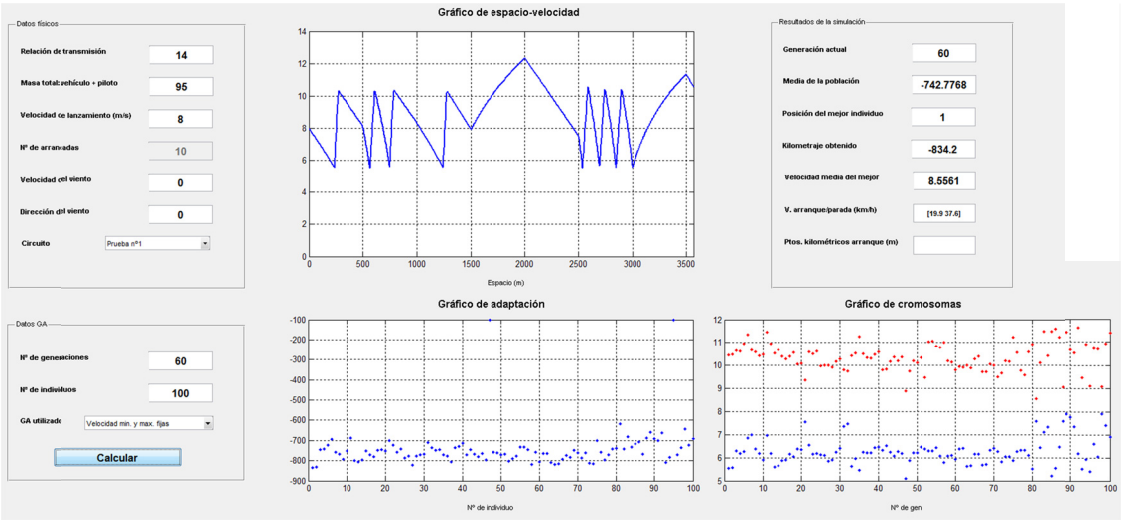


Figura 11.17.- Simulación con el GA básico en el circuito nº 3

Optimizando ahora mediante el GA avanzado, podemos ver que hemos obtenido un kilometraje de 870 km/l, un valor un 4.3 % mayor que el que obtuvimos mediante el GA básico, lo que confirma nuestras hipótesis de que en circuitos con perfiles irregulares se deja notar la optimización utilizando velocidades independientes. El nº de arrancadas en ambos casos ha sido igual a 8, pero en este caso se han realizado de manera más inteligente, evitando hacer arrancadas en puntos cercanos a una pendiente descencendente, aprovechando éstas para aumentar la velocidad sin consumir combustible.

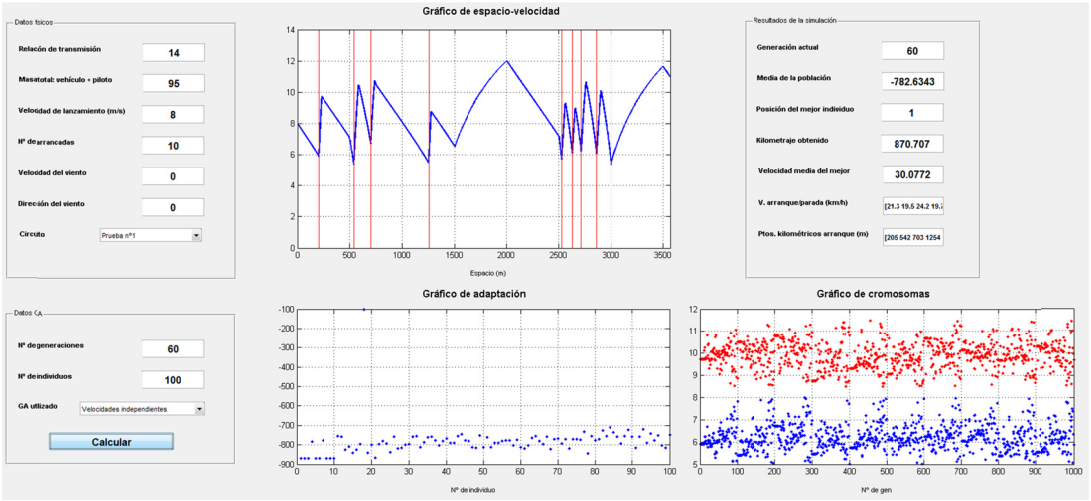


Figura 11.18.- Simulación con el GA avanzado en el circuito nº 3.

Optimizando mediante el GA básico del CPOH se ha obtenido una mejor marca de 854 km/l, un 2% inferior a la obtenida mediante nuestro algoritmo GA avanzado.

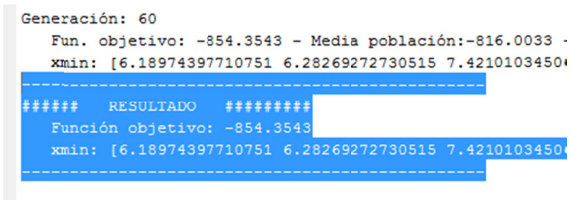


Figura 11.19.- Resultados obtenidos por el GA básico del CPOH para el circuito nº 3

En este circuito se aprecia una diferencia más acentuada entre la optimización obtenida por nuestro algoritmo GA avanzado y el resultado proporcionado por el algoritmo del CPOH, siendo un 1.9% más favorable para el nuestro. Se puede concluir que en este tipo de circuitos de perfiles más irregulares, el tiempo de cálculo mayor de los algoritmos más complejos se ve compensado con un resultado más refinado que mediante el AG básico.

OPTIMIZACIÓN EN EL CIRCUITO Nº 3		
ALGORITMO UTILIZADO	**KILOMETRAJE OBTENIDO**	**% DE MEJORA**
GA básico	**834**	**0**
GA avanzado	**870**	**4.3**
GA CPOH	**854**	**2.4**

11.6.- OPTIMIZACIÓN EN EL CIRCUITO RICARDO TORMO

En Mayo tuvimos la ocasión de visitar las instalaciones del Circuito Ricardo Tormo de Cheste, para la realización de los entrenamientos del equipo previos a su participación en la competición Shell Eco-Marathon 2011. En aquella ocasión utilizamos para las pruebas la versión corta del mismo. La principal diferencia de este circuito con los anteriores es que cuenta con unos radios de curva muy cerrados, que penalizan con mucha resistencia a rodadura en el paso por ellas. En la optimización mediante el GA básico la mejor marca es de 615 km (fig11.20), inferior a la que se obtuvo en aquellas pruebas.

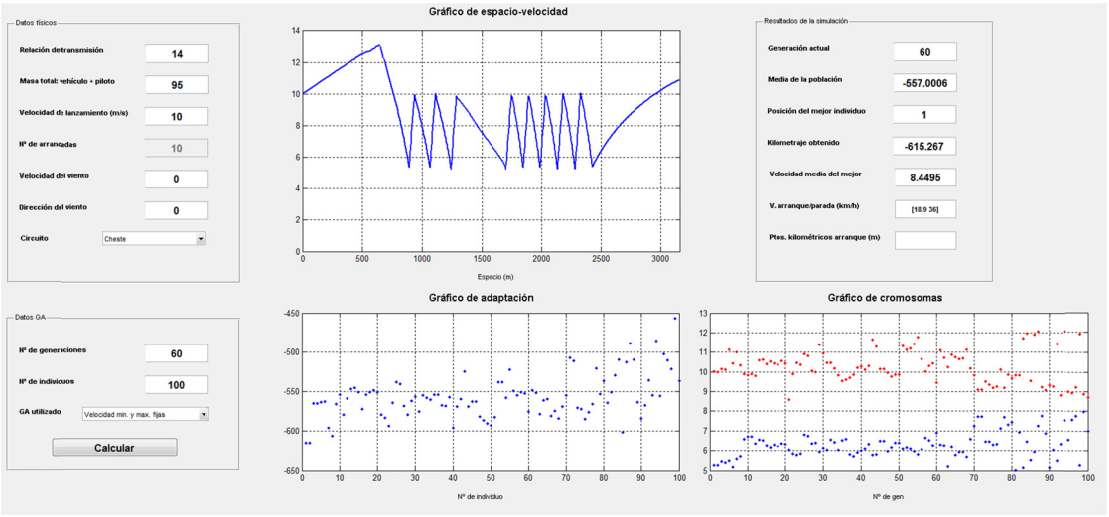


Figura 11.20.- Optimización del GA básico en el circuito de Cheste.

La optimización con el GA avanzado supuso una mejora notable en este circuito obteniendo un kilometraje de 683 km/l, un 11% superior al GA básico. En la estrategia calculada se muestra claramente la tendencia a realizar pasos lentos cuando la curva es cerrada y la de utilizar velocidades de arranque más altas cuando la pendiente es ascendente. Estrategia esta que tuvimos ocasión de comprobar en los entreanmientos que se mostraba más efectiva.

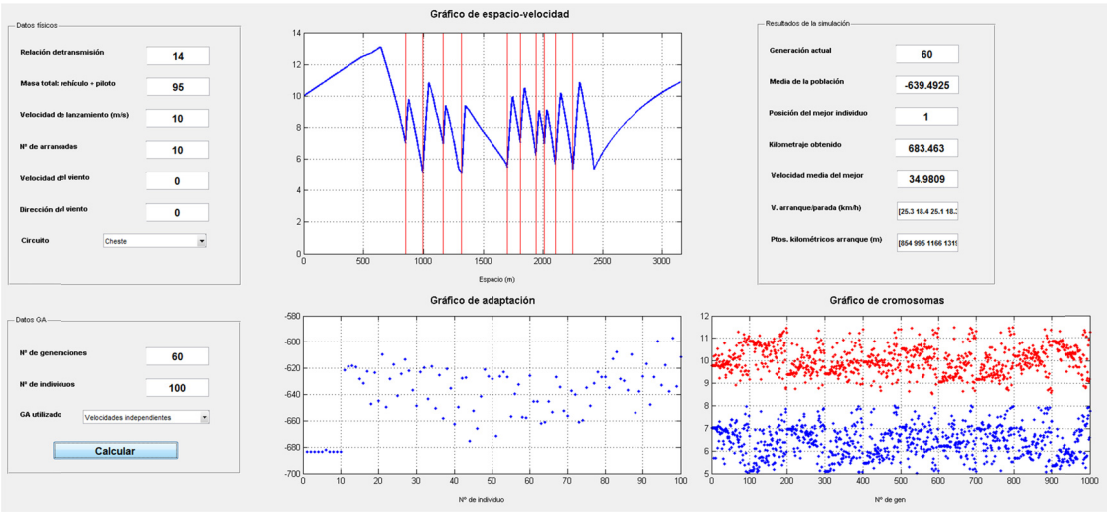


Figura 11.21.- Optimización del GA avanzado en el circuito de Cheste.

En estas simulaciones, el algoritmo del CPOH obtuvo unos resultados inferiores incluso a los que obtuvo nuestro GA básico. Obtuvo una mejor marca de 592 km/l de combustible, un **3.8% inferior al primer algoritmo y un 15.4% menos que nuestro GA avanzado**.

```
------------------------------------------------
Generación: 60
   Fun. objetivo: -592.3128 - Media población:-5
   xmin: [6.39970681641326 6.83256100295558 7.18
------------------------------------------------
######   RESULTADO   #########
   Función objetivo: -592.3128
   xmin: [6.39970681641326 6.83256100295558 7.18
------------------------------------------------
```

Figura 11.22.- Optimización mediante el GA del CPOH en el circuito de Cheste.

En este circuito es donde se han presentado las mayores diferencias entre la optimización mediante velocidades fijas y variables, confirmando nuestra hipótesis de que **cuanto más irregular es el plano del circuito y su altimetría, mayor importancia cobra la elección de una buena estrategia de carrera**. Este circuito ya contaba con unas curvas lo suficientemente cerradas como para penalizar con resistencia adicional el paso por ellas y por tanto se debe procurar en ellas que la velocidad de paso sea lo menor posible para evitar pérdidas de energía. La optimización nos indica también, que no se deben hacer arrancadas en puntos cercanos al comienzo de la gran pendiente descendente antes de meta.

OPTIMIZACIÓN EN EL CIRCUITO RICARDO TORMO		
ALGORITMO UTILIZADO	***KILOMETRAJE OBTENIDO***	***% DE MEJORA***
GA básico	**615**	**3.8**
GA avanzado	**683**	**15.4**
GA CPOH	**592**	**0**

12.- CONCLUSIONES.

Para la realización de esta tesina ha sido necesario el aprendizaje de varias disciplinas como son la optimización mediante AG y programación en Matlab. Durante la realización de la misma han sido innumerables los cambios que se han realizado tanto a los modelos Simulink del vehículo y circuito, como a los algoritmos genéticos utilizados. **Se desarrollaron dos AG aparte de los expuestos** en el apartado 9 de la tesina, aunque fueron descartados por no mejorar los resultados de los ya existentes. Estos fueron:

- Algoritmo AG con codificación de los individuos basados en punto kilométrico de arrancada y velocidad máxima de parada.

- Algoritmo AG con codificación binaria del espacio de la pista donde un 0 en el tramo estudiado suponía apagar el motor y un 1 implicaba el arranque del mismo.

El problema principal con esta codificación fue que la arrancada, al estar supeditada a puntos kilométricos en lugar de a velocidades mínimas, puede ocurrir que el vehículo quede parado en el circuito al no tener suficiente impulso como para llegar al punto de arrancada siguiente. Ello provocaba el descarte de muchos individuos y la proliferación de otros que aun no siendo buenas soluciones, al haber obtenido una cierta valoración en la función objetivo podían reproducirse rápidamente dando lugar a convergencias en mínimos locales.

El otro gran problema, con este tipo de codificación, fue que al ser los puntos kilométricos obtenidos mediante funciones generadoras de números aleatorias, el reparto de ellos sobre el rango de valores es más o menos uniforme, situación más o menos ideal para desplazamientos en llano, pero nefasta cuando se trata de circuitos con largas pendientes tanto ascendentes como descendentes. En las pendientes ascendentes las arrancadas deben comprimirse en el espacio, mientras que en pendientes descendentes normalmente no se necesita arrancar el motor. Una vez más, esta codificación tenía una tasa de fallo muy elevada por no cumplir con las arrancadas requeridas en pendientes ascendentes o realizar arrancadas en pendiente descendente malgastando energía.

Principalmente por estas razones, **la codificación según velocidades mínimas y máximas fue la escogida** presentando las siguientes ventajas principales con respecto a la anterior:

1) ***Imposibilidad de que el vehículo quede parado durante la simulación:*** Al tener siempre fijada una velocidad mínima según el rango establecido, ésta nunca podrá ser rebasada inferiormente.

2) ***Mayor robustez para evitar arrancadas en pendientes descendentes:*** Si la pendiente es lo suficientemente fuerte como para hacer acelerar el vehículo, durante ella nunca se viajará a velocidad menor que la mínima.

3) ***Relativa independencia del nº de arrancadas con respecto al fijado al iniciar la simulación:*** Podemos establecer por ejemplo un nº de 12 arrancadas, pero el vehículo, al cambiar de una a otra sólo por el criterio de nº de veces que se rebasen inferiormente las velocidades mínimas, si durante la simulación sólo se ha bajado de las mínimas 10 veces, el vehículo sólo utilizará 10 arrancadas de las 12 previstas si con ello obtiene un kilometraje mayor.

Durante la fase de pruebas del algoritmo genético avanzado se cambió varias veces de operadores de reproducción y mutación, concluyendo que la combinación de los expuestos en el apartado 9 fueron los que lograron converger a un valor de kilometraje más alto, evitando la caída en mínimos locales. Algunos de los mecanismos de reproducción que se desecharon son:

- ***Cruce monopunto:*** Debido a la complejidad del problema, el cruce monopunto, al cambiar de una vez prácticamente la mitad del cromosoma del individuo, en la mayoría de los casos conseguía desequilibrarlo, bien reduciendo mucho su kilometraje, o bien haciendo que violara alguna de las restricciones, siendo descartado de la evolución.

- ***Cruce uniforme:*** Este operador de cruce presenta un problema parecido al anterior y es que normalmente cambia muchos valores de arrancadas de una sola vez en los hijos, lo que provoca, que si los padres eran individuos que estaban muy cerca del óptimo, den lugar a hijos con mucho peor valor de adaptación.

Por ello finalmente se puede concluir con **respecto a los operadores de cruce** para este problema que los que mejor funcionamiento han obtenido han sido aquellos en los que alguno de los hijos hereda la mayor parte del cromosoma de uno de los padres y otra pequeña porción del otro progenitor. Aparte de esto, han demostrado mayor eficacia los cruces cuando para ello se intercambian conjuntamente velocidad mínima y máxima de cada arrancada, en lugar intercambiar independientemente sólo las máximas o las mínimas. Como medida para evitar la falta de diversidad, la alternancia cada 10 generaciones de operador de reproducción demostró su éxito.

Es este el factor que creo que marca la diferencia en tiempo y calidad de la solución obtenida de los algoritmos expuestos en este trabajo con respecto al Algoritmo Genético Básico del CPOH, que al ser un algoritmo de propósito general para ser aplicado en multitud de problemas, no contempla las incidencias que se puedan producir en los cruces de este tipo de soluciones.

Con respecto al **modo de reemplazo de los individuos**, también se probaron varias modalidades llegando a las siguientes conclusiones:

- ***La no convergencia cuando se utiliza un AG totalmente generacional:*** en esta modalidad de AG, se reemplaza **toda la población** por la nueva incondicionalmente. Debido a la aleatoriedad de los cruces, en algunas ocasiones los hijos están peor adaptados que los padres y sin la élite de la población, se dificulta la convergencia. Por ello es aconsejable mantener a la élite aunque sea de pocos individuos.

- ***La lentitud en la convergencia al utilizar reemplazo condicionado:*** en este problema, al condicionar la sustitución de un padre sólo si su hijo tenía un valor de adaptación más alto, se demostró que en un principio las soluciones evolucionaban más rápidamente, pero llegados a valores cercanos al óptimo se dificultaba mucho la convergencia debido a que los hijos al no poder mejorar a los padres no eran incluidos en la nueva generación y los siguientes cruces se volvían a realizar entre los mismos individuos de la población anterior ralentizando mucho la convergencia final hacia el óptimo. Esta situación además traía otro problema, y es que al realizarse cruces de mismos individuos a lo largo de varias generaciones, esto desembocaba la multiplicación copias exactas de individuos con alto valor de adaptación, desembocando rápidamente en situaciones de falta de diversidad en la población.

Con respecto al **método de optimización escogido**, los algoritmos genéticos, han demostrado ser un método bastante eficiente para resolver este problema a priori de difícil optimización. Se ha comprobado su convergencia y mediante la comparación con otro algoritmo genético de calidad (el AG básico del CPOH), hemos visto que los óptimos encontrados con ambos AG se encuentran muy cercanos, lo que nos hace pensar tales soluciones estén en el entorno del óptimo global del problema.

Mediante la lectura de artículos de investigación se descubrió otro método de optimización que quizá también se pueda adecuar a este problema: ***Simulated annealing*** o recocido simulado. Resumiendo mucho, este algoritmo hace evolucionar una serie de soluciones realizando pequeños cambios en sus variables y evaluando si éstos han sido satisfactorios o no. La diferencia principal entre este algoritmo y otros de búsqueda local es que éste permite con una cierta probabilidad cambios de variables que empeoren la calidad de la solución, con la esperanza de que en futuras modificaciones, ésta pueda tener una calidad mejor que la original.

Quizá sería buena idea el realizar un algoritmo híbrido (entre AG y Simulated Annealing), que en una primera etapa calculara buenas soluciones mediante el AG, para posteriormente, una vez que éste ya sufre el estancamiento en su solución óptima, mediante el Simulated Annealing refinar la búsqueda aún más para ver si aquéllas soluciones eran mejorables. **Y es que la realidad, cuando se tratan problemas con restricciones con AG, es que hay una línea muy fina que separa una buena solución del precipicio al cual caería ésta con la violación de una restricción, provocando que se descarten gran cantidad de soluciones muy cercanas al óptimo global, que quizá con pequeños cambios podrían mejorar el óptimo encontrado.**

Con respecto a las conclusiones obtenidas de las pruebas realizadas en diversos circuitos, se puede afirmar que en **circuitos llanos y con curvas de radio amplio, no aporta ventajas la utilización de la optimización con velocidades independientes** y por el contrario el tiempo de cálculo necesario para obtener la solución óptima puede multiplicarse por dos. Las optimizaciones con los algoritmos de velocidades independientes en circuitos con desniveles y curvas acusadas sí aportaron una mejora del kilometraje en torno del 5%, siendo aconsejable su uso en estos casos a pesar de su mayor tiempo de cálculo.

Dentro de los parámetros que influyen dentro del consumo del coche se detectó el gran impacto que tiene en el consumo la elección de una relación de transmisión adecuada. Los mejores consumos se obtienen siempre con relaciones de 13, 14 y 15 dependiendo de la altimetría del circuito. Con valores de RT mayores de 16 se apreció una caída muy notable del kilometraje obtenido que en muchos casos rondaba 100km menos. Esto se debe a que el motor trabaja por encima de su rango óptimo de eficiente, provocando un consumo excesivo.

Por todo lo expuesto anteriormente y con la esperanza de que la creación de este simulador pueda servir para mejorar las marcas del equipo en un futuro, se valora muy positivamente la realización de este trabajo por el esfuerzo y el reto que ha supuesto para el autor de esta tesina, que al haber estudiado I.T.I. Mecánica disponía de nulos conocimientos sobre optimización y programación en Matlab antes de la entrada a este Máster. Una vez más agradecer a **Sergio García-Nieto Rodríguez**, mi profesor de Optimización en este Máster, la ayuda prestada durante la realización de la misma y a **Vicente Colomer Romero**, mi tutor, por sus rápidas enseñanzas y explicaciones acerca de la técnica y el funcionamiento de un mundo totalmente desconocido para mí antes de la entrada a este Máster como es el de las carreras de bajo consumo

12.1.- LOGRO DE OBJETIVOS

En un principio, cuando surgió la idea de la realización de esta tesina sus objetivos eran:

- Generación de un simulador de carreras de bajo consumo basado en Matlab.
- Estudio de la estrategia de carrera en la Shell Eco-Marathon.
- Simulaciones con diversos circuitos.

Estos objetivos no sólo se han cumplido sino que además se han completado con otros como son:

- Cálculo de la estrategia de carrera óptima mediante algoritmos genéticos.
- Creación de una interfaz gráfica más intuitiva para el usuario que el entorno Matlab.

Con el modelo Simulink creado, se permitirá en un futuro, todas las modificaciones que se deseen sobre su base, ya que debido a la modularidad de los diseños en esta plataforma, la tarea resulta muy sencilla y rápida, ya que evita el engorro de tener que interpretar el código fuente de programación creado por otro usuario. Simulink, al ser un método de programación gráfico evita el problema citado anteriormente.

De igual forma, el algoritmo genético se ha programado de una manera clara y haciendo uso de diversas funciones para separar mejor el código y facilitar su futura edición.

A pesar del cumplimiento de los objetivos iniciales, queda en el aire el cumplimiento del objetivo más importante, aunque no fuera planteado como tal: Contrastar los resultados obtenidos por el simulador con la realidad y comprobar su eficacia.

12.2.- TAREAS FUTURAS.

La creación de un simulador es un proyecto ambicioso y con la realización de esta tesina se puede afirmar que este proyecto no ha hecho nada más que empezar. A continuación enumeramos los posibles trabajos que podrán tener lugar a partir de este:

- **Ajustar el modelo Matlab lo más posible al modelo físico**: Para ello será necesario disponer de los medios suficientes para la realización de experimentos fiables para la medida de los diversos coeficientes de rozamiento que afectan al vehículo, o como mínimo de datos de telemetría precisos en los que se pueda estudiar la caída de velocidad del vehículo cuando no actúa el motor y de esta forma ajustar manualmente los coeficientes en el modelo Matlab para que los perfiles de velocidades coincidan en ambos.

- **Mejora de la interfaz gráfica que permita la introducción de circuitos desde la misma:** Actualmente y por falta de tiempo, los circuitos son introducidos dentro del modelo Simulink por medio de matrices, cuyos datos representan la curvatura de la pista, su pendiente, la acción del viento, en función de la distancia recorrida. En un futuro sería interesante poder introducir estos datos de una forma más intuitiva, y que sea la propia interfaz la que convierta los datos de entrada en las matrices para Simulink. A este respecto, los ficheros de AutoCAD ****.dxf***, tienen una codificación conocida por medio de archivos de texto ASCII. Sería interesante crear un traductor mediante Matlab para tal función.

- **Estudio de otros algoritmos de optimización:** Como se enunció en el apartado de conclusiones, son numerosos los estudios realizados en otros campos de la ciencia que requieren optimización global y que utilizan algoritmos más avanzados que los genéticos o híbridos de éstos con otros tipos. Sería muy interesante el repetir estas optimizaciones con otros tipos de algoritmos para estudiar cuál de ellos obtiene mejores resultados o menores tiempos de cálculo.

- **Ampliación del AG genético a AG genético paralelo:** Uno de los problemas que tienen los AG es el tiempo relativamente alto de convergencia en problemas con gran número de variables como es nuestro caso. En implementaciones simples, los bucles de evaluación de individuos sólo hacen uso de uno de los núcleos del procesador del computador. Ya desde hace varios años, la totalidad de los ordenadores incorporan procesadores de 2, 4, 8 núcleos, y Matlab dispone de herramientas para computación en paralelo, y la adaptación de este algoritmo para soportarlas, supondría una reducción del tiempo de cálculo muy considerable. Con esta implementación, las grandes matrices de individuos se dividirían, y cada una de estas partes sería procesada en un núcleo del procesador.

13.- BIBLIOGRAFÍA.

[Alander, 92] J.T. Alander. On optimal population size of genetic algorithms. Proceedings CompEuro 1992, Computer Systems and Software Engineering, 6th Annual European Computer Conference, pág. 65-70.

[Araujo Cervigón, 09] Araujo, Lourdes. Cervigón, Carlos. Algoritmos Evolutivos. Un enfoque práctico, Ed. Ra-Ma, pág. 27-77. (2009)

[Baalen, 2004] J.V. Baalen. Modelling of the fuel consumption of a fuel cell powered car, pág 13-20. (2004)

[Bezdek, 1994] Bezdeck, J. C., Boggavaparu, S., Hall, L. O. y Bensaid, A. *Genetic Algorithm guided clustering*, in Proc. of the First IEEE Conference on Evolutionary Computation, 3440. (1994).

[Briet, 2009] Briet Blanes, Timoteo. Simulación CFD y túnel de viento, The F1.com 2009

[Cole, 1998] Cole, Rowena M. Clustering with Genetic Algorithms. Thesis for the degree of Master of Science, Department of Computer Science, University of Western Australia (1998)

[Deb, 1991] Deb, K. Goldberg, David. A comparative analysis of selection schemes used in genetic algorithms. In FGA1, pág 69-93 (1991).

[De Jong, 1975] De Jong, K.A. An analysis of the behavior of a class of genetic adaptative systems. PhD thesis. Ann Arbor, MI, USA (1975).

[Falkenauer, 1999] Falkenauer, Emanuel. Evolutionary Algorithms: Applying Genetic Algorithms to Real-World Problems. Springer, New York, pág 65-88 (1999).

[Figueiredo, 2002] de Figueiredo Vieira Carvalheira, Pedro. Simulation of the performance of an extra-low fuel consumption vehicle. Universidade de Coimbra, Portugal, pág 1-6 (2002)

[Forrest, 1985] Forrest, S. Documentation for prisoners dilemma and norms programs that use the genetic algorithm. Technical report, University of Michigan, Ann Arbor, Mi, (1985)

[Gil, 2006] Gil Londoño, Natyhelem. Algoritmos genéticos. Universidad de Colombia, pág 20-40 (2006)

[Gestal, 2004] Gestal Pose, Marcos. Introducción a los algoritmos genéticos. Dpto. de Técnicas de la Informática y la Comunicación. Universidade da Coruña, pág 5-16 (2004)

[Goldberg, 1989] Goldberg, David. Genetic algorithms in search, optimization and machine learning. Addison-Wesly (1989)

[Hancock, 1994] Hancock, P.J.B. And empirical comparison of selection methods in evolutionary algorithms. Evolutionary Computing, AISB Workshop, pág 80-94 (1994)

[Holland, 1975] Holland, John. Adaptation in natural and artificial systems. Ann Arbor. The University of Michigan Press.

[Jacob, 2010] Establecimiento de metodologías para la realización de CFD mediante NX y planteamiento de propuestas de re-diseño en la carrocería del vehículo para la competición "Shell eco-maratón 2010". Tesis de Máster en Diseño y Fabricación Asistidos por Computador. Universidad Politécnica de Valencia, pág 7-13 (2010)

[Koza, 2003] Koza, J.R. Genetic programming: On the programming of computers by means of natural selection. MIT Press, pág 560 (1992)

[Macgilvary, 1985] Macgilvary Gillies, A. Machine learning procedures for generating image domain feature detectors. PhD thesis, Ann Arbor, MI, USA, (1985)

[Magrab, 2011] Magrab, Edward B. An engineer´s Guide to Matlab. Ed. Pearson (2011)

[Mickalewicz, 1994] Mickalewicz, Z. Genetic algorithms + Data structures = Evolution programs. Springer-Verlag, 2nd edition, (1994)

[Moore, 2007] Moore, Holly. Matlab para ingenieros. Ed. Prentice Hall, (2007)

[Rossi, 2006] Rossi, F., van Beek, P., Walsh, T. Handbook of constraint programming (Foundations of artificial intelligence). Elsevier Science Inc., (2006)

[Santin et al., 2007] Santin, J.J., Onder, C.H., Bernard, J., Isler, D. The world´s most fuel efficient vehicle. Design and development of PAC Car II. VDF, (2007)

[Smith, 2006] Smith, Scott T. Matlab. Advanced GUI Development, Dog Ear Publishing, (2006)

[Tsang, 1999] Tsang, E. A glimpse of constraint satisfaction. Artificial intelligence review, pág 215-227 (1999)

APÉNDICE DE CÓDIGO

```
function varargout = Interfaz(varargin)
% INTERFAZ M-file for Interfaz.fig
%      INTERFAZ, by itself, creates a new INTERFAZ or raises the existing
%      singleton*.
%
%      H = INTERFAZ returns the handle to a new INTERFAZ or the handle to
%      the existing singleton*.
%
%      INTERFAZ('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in INTERFAZ.M with the given input
arguments.
%
%      INTERFAZ('Property','Value',...) creates a new INTERFAZ or raises
the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Interfaz_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Interfaz_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Interfaz

% Last Modified by GUIDE v2.5 14-Sep-2011 18:50:11

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Interfaz is made visible.
function Interfaz_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Interfaz (see VARARGIN)

% Choose default command line output for Interfaz
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Interfaz wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Interfaz_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function txt_RT_Callback(hObject, eventdata, handles)
% hObject    handle to txt_RT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_RT as text
%        str2double(get(hObject,'String')) returns contents of txt_RT as
a double

RT = str2double(get(handles.txt_RT, 'String'));
handles.RT = RT;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function txt_RT_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_RT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Mv_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Mv (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of txt_Mv as text
%        str2double(get(hObject,'String')) returns contents of txt_Mv as
a double

Mv = str2double(get(handles.txt_Mv, 'String'));
handles.Mv = Mv;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function txt_Mv_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Mv (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Vel_lanz_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Vel_lanz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Vel_lanz as text
%        str2double(get(hObject,'String')) returns contents of
txt_Vel_lanz as a double

Vel_lanz = str2double(get(handles.txt_Vel_lanz, 'String'));
handles.Vel_lanz = Vel_lanz;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function txt_Vel_lanz_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Vel_lanz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Arrancadas_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Arrancadas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of txt_Arrancadas as text
%        str2double(get(hObject,'String')) returns contents of
txt_Arrancadas as a double

Arrancadas = str2double(get(handles.txt_Arrancadas, 'String'));
handles.Arrancadas = Arrancadas;
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function txt_Arrancadas_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Arrancadas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_V_viento_Callback(hObject, eventdata, handles)
% hObject    handle to txt_V_viento (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_V_viento as text
%        str2double(get(hObject,'String')) returns contents of
txt_V_viento as a double

Vel_viento = str2double(get(handles.txt_V_viento, 'String'));
handles.Vel_viento = Vel_viento;
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function txt_V_viento_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_V_viento (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Dir_viento_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Dir_viento (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Dir_viento as text
%        str2double(get(hObject,'String')) returns contents of
txt_Dir_viento as a double

Dir_viento = str2double(get(handles.txt_Dir_viento, 'String'));
handles.Dir_viento = Dir_viento;
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function txt_Dir_viento_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Dir_viento (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_n_gen_Callback(hObject, eventdata, handles)
% hObject    handle to txt_n_gen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_n_gen as text
%        str2double(get(hObject,'String')) returns contents of txt_n_gen
as a double

n_gen = str2double(get(handles.txt_n_gen, 'String'));
handles.n_gen = n_gen;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function txt_n_gen_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_n_gen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_n_ind_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to txt_n_ind (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_n_ind as text
%        str2double(get(hObject,'String')) returns contents of txt_n_ind
as a double

n_ind = str2double(get(handles.txt_n_ind, 'String'));
handles.n_ind = n_ind;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function txt_n_ind_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_n_ind (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in menu_GA.
function menu_GA_Callback(hObject, eventdata, handles)
% hObject    handle to menu_GA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns menu_GA
contents as cell array
%        contents{get(hObject,'Value')} returns selected item from
menu_GA
valor = get(handles.menu_GA, 'Value');
handles.valor = valor;
guidata(hObject, handles);

switch handles.valor
    case 1
        set(handles.txt_Arrancadas, 'Enable', 'off');
    case 2
        set(handles.txt_Arrancadas, 'Enable', 'on');
end


GA_utilizado = get(handles.menu_GA, 'Value');
handles.GA_utilizado = GA_utilizado;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function menu_GA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_GA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Gen_actual_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Gen_actual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Gen_actual as text
%        str2double(get(hObject,'String')) returns contents of
txt_Gen_actual as a double


% --- Executes during object creation, after setting all properties.
function txt_Gen_actual_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Gen_actual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Media_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Media (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Media as text
%        str2double(get(hObject,'String')) returns contents of txt_Media
as a double


% --- Executes during object creation, after setting all properties.
function txt_Media_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Media (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function txt_Pos_mejor_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Pos_mejor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Pos_mejor as text
%        str2double(get(hObject,'String')) returns contents of
txt_Pos_mejor as a double


% --- Executes during object creation, after setting all properties.
function txt_Pos_mejor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Pos_mejor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Kilometraje_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Kilometraje (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Kilometraje as text
%        str2double(get(hObject,'String')) returns contents of
txt_Kilometraje as a double


% --- Executes during object creation, after setting all properties.
function txt_Kilometraje_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Kilometraje (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in menu_circuito.
function menu_circuito_Callback(hObject, eventdata, handles)
% hObject    handle to menu_circuito (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns menu_circuito
contents as cell array
```

```
%        contents{get(hObject,'Value')} returns selected item from
menu_circuito


% --- Executes during object creation, after setting all properties.
function menu_circuito_CreateFcn(hObject, eventdata, handles)
% hObject    handle to menu_circuito (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function txt_V_media_Callback(hObject, eventdata, handles)
% hObject    handle to txt_V_media (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_V_media as text
%        str2double(get(hObject,'String')) returns contents of
txt_V_media as a double


% --- Executes during object creation, after setting all properties.
function txt_V_media_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_V_media (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Resultado_vel_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Resultado_vel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Resultado_vel as
text
%        str2double(get(hObject,'String')) returns contents of
txt_Resultado_vel as a double


% --- Executes during object creation, after setting all properties.
function txt_Resultado_vel_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Resultado_vel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function txt_Resultado_ptos_Callback(hObject, eventdata, handles)
% hObject    handle to txt_Resultado_ptos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of txt_Resultado_ptos as
text
%        str2double(get(hObject,'String')) returns contents of
txt_Resultado_ptos as a double


% --- Executes during object creation, after setting all properties.
function txt_Resultado_ptos_CreateFcn(hObject, eventdata, handles)
% hObject    handle to txt_Resultado_ptos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in bt_Calcular.
function bt_Calcular_Callback(hObject, eventdata, handles)
% hObject    handle to bt_Calcular (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%///// COMENZAMOS DECLARANDO LAS VARIABLES GENERALES PARA EL MODELO

S = 0.304;              %Área frontal del vehículo en m^2
Cx = 0.25;              %Coef. aerodinámico
ro = 1.2;               %Densidad del aire
D = 0.48;               %Diámetro de la rueda
d = 0.02;               %Diámetro interior del rodamiento
mu = 0.0015;            %Coef. fricción rodamiento
g = 9.81;               %Aceleración de la gravedad
fr = 0.0016;            %Coef. fricción neumáticos a rodadura
Ef_trans = 0.94;        %Eficiencia de la transmisión


%/////CREAMOS LAS VARIABLES CON LOS DATOS FÍSICOS DE PARTIDA

RT = str2double(get(handles.txt_RT, 'String'));
```

```
Mv = str2double(get(handles.txt_Mv, 'String'));
Vel_lanz = str2double(get(handles.txt_Vel_lanz, 'String'));
Arrancadas = str2double(get(handles.txt_Arrancadas, 'String'));
Vel_viento = str2double(get(handles.txt_V_viento, 'String'));
Dir_viento = str2double(get(handles.txt_Dir_viento, 'String'));



%/////CALCULAMOS LOS VALORES PARA LAS GANANCIAS Y CONSTANTES

Conversor = (60*RT)/(pi*D);
Inversa_Radio_Rueda = 1/(D/2);
Peso = Mv * g;
Roz_Rod = 1/3 * (mu*Mv*g*d/D) + (fr*Mv*g);
Arrastre_Aero = 1/2 * ro * S * Cx;
Inversa_Masa = 1/Mv;
Inversa_Longitud = 1/3570;
Longitud = 3570;



%/////VARIABLES CORRESPONDIENTES AL GA

n_gen = str2double(get(handles.txt_n_gen, 'String'));
n_ind = str2double(get(handles.txt_n_ind, 'String'));
GA_utilizado = get(handles.menu_GA, 'Value');

switch GA_utilizado
    case 1
        find_system('Name', 'modelo');
        load_system('modelo');
        warning off;

        %/////ESTABLECEMOS LAS CONSTANTES Y GANANCIAS EN EL MODELO

        set_param('modelo/Sistema motor/Conversor', 'Gain',
num2str(Conversor));

        set_param('modelo/Transmisión/RT', 'Gain', num2str(RT));
        set_param('modelo/Transmisión/Eficiencia', 'Gain',
num2str(Ef_trans));
        set_param('modelo/Transmisión/Inversa_Radio_Rueda', 'Gain',
num2str(Inversa_Radio_Rueda));

        set_param('modelo/Circuito/Inversa_Masa', 'Gain',
num2str(Inversa_Masa));
        set_param('modelo/Circuito/V_ini', 'Value', num2str(Vel_lanz));
        set_param('modelo/Circuito/Peso', 'Gain', num2str(Peso));
        set_param('modelo/Circuito/Arrastre_Aero', 'Gain',
num2str(Arrastre_Aero));
        set_param('modelo/Circuito/Roz_Rod', 'Value',
num2str(Roz_Rod));

        set_param('modelo/Circuito/Circuito/Inversa_Longitud', 'Gain',
num2str(Inversa_Longitud));
        set_param('modelo/Circuito/Circuito/Longitud', 'Gain',
num2str(Longitud));
        set_param('modelo/Circuito/Circuito/Velocidad viento', 'Gain',
num2str(Vel_viento));
```

```
        %/////FIN DE ESTABLECIMIENTO DE CONSTANTES Y GANANCIAS EN EL MODELO
        poblac = crea_aleatoria(n_ind);

        for j=1:n_gen
            valor = zeros(n_ind,1);
            mejor = 0;
            for i=1:n_ind

                set_param('modelo/Modelo de arranque/V_min', 'const',
num2str(poblac(i,1)));
                set_param('modelo/Modelo de arranque/V_max', 'const',
num2str(poblac(i,2)));

                [T,X,Y1,Y2,Y3,Y4,Y5]=sim('modelo');


                if (Y2(end,1) > 30) && (Y4(end,1) > Vel_lanz)
                    valor(i)= -Y1(1,1,end);
                else
                    valor(i)= -100;
                end

                if valor(i)<mejor
                    mejor = valor(i);
                    vector_velocidades = Y4;
                    vector_espacio = Y3;
                    cromosoma_mejor = [poblac(i,1) poblac(i,2)];
                    media_mejor = Y3(end,1)/Y5(end,1);
                end

            end

            [valores, orden] = sort(valor,1);
            valores_ord = [valores orden];
            media = mean(valores_ord(:,1));
            velocidades_mejor = round(cromosoma_mejor * 3.6 * 10) / 10;


            set(handles.txt_Pos_mejor, 'String', valores_ord(1,2));
            set(handles.txt_Kilometraje, 'String', valores_ord(1,1));
            set(handles.txt_Gen_actual, 'String', num2str(j));
            set(handles.txt_Media, 'String', num2str(media));
            set(handles.txt_V_media, 'String', num2str(media_mejor));
            set(handles.txt_Resultado_vel, 'String',
mat2str(velocidades_mejor));


            plot(handles.ejes_Vel, vector_espacio(:,1),
vector_velocidades(:,1), 'LineWidth', 2);
            set(handles.ejes_Vel, 'XGrid', 'on', 'YGrid', 'on', 'XLim',
[0 3570], 'YLim', [0 14]);

            set(handles.ejes_Poblac, 'NextPlot', 'Replace');
            plot(handles.ejes_Poblac, 1:n_ind, valor, '.b');
            set(handles.ejes_Poblac, 'XGrid', 'on', 'YGrid', 'on');
```

```
            set(handles.ejes_Ind, 'NextPlot', 'Replace');
            plot(handles.ejes_Ind, 1:n_ind, poblac(:,1), '.b');
            set(handles.ejes_Ind, 'NextPlot', 'Add');
            plot(handles.ejes_Ind, 1:n_ind, poblac(:,2), '.r');
            set(handles.ejes_Ind, 'XGrid', 'on', 'YGrid', 'on');

            pause(0.5);

            nueva = cruzar(poblac, valores_ord, media);
            poblac =  nueva;

        end



    case 2
        find_system('Name', 'modelo2');
        load_system('modelo2');
        warning off;

        %/////ESTABLECEMOS LAS CONSTANTES Y GANANCIAS EN EL MODELO
        set_param('modelo2/Modelo de arranque/Suma_superiores', 'Value',
num2str(Arrancadas));
        set_param('modelo2/Modelo de arranque/Saturation', 'UpperLimit',
num2str(Arrancadas - 1));

        set_param('modelo2/Sistema motor/Conversor', 'Gain',
num2str(Conversor));

        set_param('modelo2/Transmisión/RT', 'Gain', num2str(RT));
        set_param('modelo2/Transmisión/Eficiencia', 'Gain',
num2str(Ef_trans));
        set_param('modelo2/Transmisión/Inversa_Radio_Rueda', 'Gain',
num2str(Inversa_Radio_Rueda));

        set_param('modelo2/Circuito/Inversa_Masa', 'Gain',
num2str(Inversa_Masa));
        set_param('modelo2/Circuito/V_ini', 'Value',
num2str(Vel_lanz));
        set_param('modelo2/Circuito/Peso', 'Gain', num2str(Peso));
        set_param('modelo2/Circuito/Arrastre_Aero', 'Gain',
num2str(Arrastre_Aero));
        set_param('modelo2/Circuito/Roz_Rod', 'Value',
num2str(Roz_Rod));

        set_param('modelo2/Circuito/Circuito/Inversa_Longitud', 'Gain',
num2str(Inversa_Longitud));
        set_param('modelo2/Circuito/Circuito/Longitud', 'Gain',
num2str(Longitud));
        set_param('modelo2/Circuito/Circuito/Velocidad viento', 'Gain',
num2str(Vel_viento));

        poblac = crea_aleatoria2(n_ind, Arrancadas);
        cambio_reproductivo = true;

        for j=1:n_gen
            valor = zeros(n_ind,1);
            mejor = 0;
```

```
            if (rem(j,10) == 0)
%Cada 10 iteraciones cambiamos el método de cruce de individuos
                cambio_reproductivo = not(cambio_reproductivo);
            end

            for i=1:n_ind

                MatrizGA(1,:) = poblac(i,:);
                set_param('modelo2/Modelo de arranque/Tabla GA
(inferiores)', 'Table', mat2str(MatrizGA'));
                set_param('modelo2/Modelo de arranque/Tabla GA
(superiores)', 'Table', mat2str(MatrizGA'));


                [T,X,Y1,Y2,Y3,Y4,Y5,Y6]=sim('modelo2');


                media = 3.6 * 3570 / Y3(end,1);          %Velocidad media


                    if (media >= 30) && Y4(1,1,end)>8
                        valor(i)= -(3570/Y1(1,1,end));          %metros
/ gramo =? kilómetros / litro  AVERIGUAR DENSIDAD!!!!!!!
                    else
                        valor(i) = -100;
                    end

                    if valor(i)<mejor
                        mejor = valor(i);
                        vector_velocidades = Y4(1,1,:);
                        vector_espacio = Y2(1,1,:);
                        media_mejor = media;
                        arrancadas_mejor = Y5(1,1,:);
                        momentos_arranque_mejor = Y6(1,1,:);
                        cromosoma_mejor = MatrizGA;
                    end


            end

            [valores, orden] = sort(valor,1);
            valores_ord = [valores orden];
            media = mean(valores_ord(:,1));
            kilometraje = -valores_ord(1,1);

            ptos_arranque = determina_ptos_arranque(vector_espacio,
momentos_arranque_mejor, arrancadas_mejor, Arrancadas);

            if arrancadas_mejor(1,1,end) > Arrancadas
                arranques_vuelta = Arrancadas;
            else
                arranques_vuelta = arrancadas_mejor(1,1,end);
            end

            clear vector_Resultados_vel;
            vector_Resultados_vel = zeros(2, arranques_vuelta);
            vector_Resultados_vel(1, 1:arranques_vuelta) =
cromosoma_mejor(1, 1:arranques_vuelta);
```

```
            vector_Resultados_vel(2, 1:arranques_vuelta) =
cromosoma_mejor(1, (Arrancadas+1):(Arrancadas+arranques_vuelta));
            vector_Resultados_vel = (round(vector_Resultados_vel * 3.6 *
10))/10;



            set(handles.txt_Pos_mejor, 'String', valores_ord(1,2));
            set(handles.txt_Kilometraje, 'String', kilometraje);
            set(handles.txt_Gen_actual, 'String', num2str(j));
            set(handles.txt_Media, 'String', num2str(media));
            set(handles.txt_V_media, 'String', num2str(media_mejor));
            set(handles.txt_Resultado_vel, 'String',
mat2str(vector_Resultados_vel));
            set(handles.txt_Resultado_ptos, 'String',
mat2str(round(ptos_arranque)));

            clear velocidades;
            clear espacio;

            velocidades(1,:) = vector_velocidades(1,1,:);
            espacio(1,:) = vector_espacio(1,1,:);

            set(handles.ejes_Vel, 'NextPlot', 'Replace');
            plot(handles.ejes_Vel, espacio(1,:), velocidades(1,:),
'LineWidth', 2);
            set(handles.ejes_Vel, 'NextPlot', 'Add');



            for k=1:arranques_vuelta
                plot(handles.ejes_Vel,[ptos_arranque(1, k),
ptos_arranque(1, k)], [0 14], 'r');
            end

            set(handles.ejes_Vel, 'XGrid', 'on', 'YGrid', 'on', 'XLim',
[0 3570], 'YLim', [0 14]);

            set(handles.ejes_Poblac, 'NextPlot', 'Replace');
            plot(handles.ejes_Poblac, 1:n_ind, valor, '.b');
            set(handles.ejes_Poblac, 'XGrid', 'on', 'YGrid', 'on');



            vector_x = 1:(n_ind * Arrancadas);
            vector_y_min = crea_vector_min(poblac);
            vector_y_max = crea_vector_max(poblac);

            set(handles.ejes_Ind, 'NextPlot', 'Replace');
            plot(handles.ejes_Ind, vector_x, vector_y_min, '.b');
            set(handles.ejes_Ind, 'NextPlot', 'Add');
            plot(handles.ejes_Ind, vector_x(1:end), vector_y_max, '.r');
            set(handles.ejes_Ind, 'XGrid', 'on', 'YGrid', 'on');



            pause(1);

            switch cambio_reproductivo
                case true
                    nueva = cruzar_2(poblac, valores_ord, media, j);
                    poblac =  nueva;
```

```
                case false
                    nueva = cruzar_3(poblac, valores_ord, media);
                    poblac =  nueva;
            end


        end

    case 3
        find_system('Name', 'modelo3');
        load_system('modelo3');
        warning off;
end


function poblac = crea_aleatoria(n_ind)
aux = zeros(n_ind,2);

aux(:,1) = rand(n_ind, 1) * 3 + 5;
aux(:,2) = rand(n_ind, 1) * 4 + 8.5;

poblac = aux;

function matriz = escalar_adapt(valores_ord, media)
a = (2 * media) / (valores_ord(1,1) - media);
b =  (1 - a) * media;

n_ind = size(valores_ord,1);
valores_ord2 = valores_ord;

for i = 1:n_ind
    valores_ord2(i,1) = valores_ord(i,1) * a + b;
end

matriz = valores_ord2;


function nueva = cruzar(poblac, valores_ord, media)
n_ind = size(poblac,1);
nuevos_ind = floor(0.8*n_ind);
aux = zeros(nuevos_ind,2);

puntuaciones = valores_ord(:,1)/media;

punt_acumulada = zeros(1, n_ind);
punt_acumulada(1) = puntuaciones(1);

for i=2:n_ind
    punt_acumulada(i) = punt_acumulada(i-1) + puntuaciones(i);
end

punt_acumulada = [punt_acumulada' valores_ord(:,2)];

puntuaciones_aleatorias = zeros(nuevos_ind, 1);
aleatorio = rand;

for i = 1:nuevos_ind
    puntuaciones_aleatorias(i) = n_ind * (aleatorio + i - 1) / nuevos_ind;
```

```
end


aux(1,:) = poblac((valores_ord(1,2)),:);
aux(2,:) = poblac((valores_ord(2,2)),:);

matriz_cruces = determina_cruces(puntuaciones_aleatorias,
punt_acumulada, nuevos_ind);

j = 1;

alfa = rand(1, nuevos_ind);

for i=1:2:(nuevos_ind-3)
    index_padre = matriz_cruces(j,1);

    index_madre = matriz_cruces(j,2);
    j = j + 1;

    aux(i+2, 1) = alfa(i) * poblac(index_padre, 1) + (1 - alfa(i)) *
poblac(index_madre, 1);
    aux(i+2, 2) = alfa(i) * poblac(index_padre, 2) + (1 - alfa(i)) *
poblac(index_madre, 2);

    aux(i+3, 1) = alfa(i+1) * poblac(index_padre, 1) + (1 - alfa(i+1)) *
poblac(index_madre, 1);
    aux(i+3, 2) = alfa(i+1) * poblac(index_padre, 2) + (1 - alfa(i+1)) *
poblac(index_madre, 2);

end


aleatorios = n_ind - size(aux,1);

restantes_aleatorios = crea_aleatoria(aleatorios);

nueva = [aux; restantes_aleatorios];


function matriz = determina_cruces(puntuaciones_aleatorias,
punt_acumulada, nuevos_ind)

i = 1;
j = 1;
vector_cruces = zeros(nuevos_ind, 1);

while i <= nuevos_ind
    if puntuaciones_aleatorias(i) < punt_acumulada(j,1)
        vector_cruces(i) = j;
        i = i + 1;

    else
        j = j + 1;
    end
end

matriz_cruces = zeros(nuevos_ind/2, 2);
```

```
ind_restantes = nuevos_ind;

for i = 1:nuevos_ind/2
    matriz_cruces(i,1) = vector_cruces(1,1);
    ind_restantes = ind_restantes - 1;

    auxiliar = vector_cruces(2:end,1);

    aleatorio =  randi(ind_restantes);

    matriz_cruces(i,2) = auxiliar(aleatorio);
    ind_restantes = ind_restantes - 1;

    if aleatorio == 1
        auxiliar = auxiliar(2:end,1);


    elseif aleatorio == ind_restantes
        auxiliar = auxiliar(1:end-1,1);

    else
        auxiliar(1:aleatorio) = auxiliar(1:aleatorio);
        auxiliar(aleatorio:ind_restantes) = auxiliar(aleatorio +
1:ind_restantes + 1);
    end

    vector_cruces = auxiliar;

end

matriz = matriz_cruces;


function poblac = crea_aleatoria2(n_ind, arrancadas)
aux = zeros(n_ind,arrancadas * 2);


aux(:,1:arrancadas) = rand(n_ind, arrancadas) * 3 + 5;
aux(:,arrancadas + 1:arrancadas * 2) = rand(n_ind, arrancadas) * 3 + 8.5;

poblac = aux;

function nueva = cruzar_2(poblac, valores_ord, media, k)
n_ind = size(poblac,1);
columnas = size(poblac, 2);
arrancadas = columnas / 2;
nuevos_ind = floor(n_ind*0.8);
aux = zeros(nuevos_ind,columnas);

puntuaciones = valores_ord(:,1)/media;

punt_acumulada = zeros(1, n_ind);
punt_acumulada(1) = puntuaciones(1);

for i=2:n_ind
    punt_acumulada(i) = punt_acumulada(i-1) + puntuaciones(i);
end
```

```
punt_acumulada = [punt_acumulada' valores_ord(:,2)];

puntuaciones_aleatorias = zeros(nuevos_ind, 1);
aleatorio = rand;

for i = 1:nuevos_ind
    puntuaciones_aleatorias(i) = n_ind * (aleatorio + i - 1) / nuevos_ind;
end


aux(1,:) = poblac((valores_ord(1,2)),:);
aux(2,:) = poblac((valores_ord(2,2)),:);

matriz_cruces = determina_cruces(puntuaciones_aleatorias,
punt_acumulada, nuevos_ind);


    j = 1;

    for i=1:2:(nuevos_ind-3)
        index_padre = matriz_cruces(j,1);

        index_madre = matriz_cruces(j,2);
        j = j + 1;

        pto_medio = floor(arrancadas/2);
        pto_corte1 = randi(pto_medio - 2) + 1;
        pto_corte2 = randi(pto_medio - 2) + pto_medio;
        pto_corte3 = pto_corte1 + arrancadas;
        pto_corte4 = pto_corte2 + arrancadas;



        aux(i+2, 1:pto_corte1) = poblac(index_padre, 1:pto_corte1);
%Creamos el hijo
        aux(i+2, pto_corte1+1:pto_corte2) = poblac(index_madre,
pto_corte1+1:pto_corte2);
        aux(i+2, pto_corte2+1:arrancadas) = poblac(index_padre,
pto_corte2+1:arrancadas);
        aux(i+2, arrancadas+1:pto_corte3) = poblac(index_padre,
arrancadas+1:pto_corte3);
        aux(i+2, pto_corte3+1:pto_corte4) = poblac(index_madre,
pto_corte3+1:pto_corte4);
        aux(i+2, pto_corte4+1:columnas) = poblac(index_padre,
pto_corte4+1:columnas);

        aux(i+3, 1:pto_corte1) = poblac(index_madre, 1:pto_corte1);
%Creamos la hija
        aux(i+3, pto_corte1+1:pto_corte2) = poblac(index_padre,
pto_corte1+1:pto_corte2);
        aux(i+3, pto_corte2+1:arrancadas) = poblac(index_madre,
pto_corte2+1:arrancadas);
        aux(i+3, arrancadas+1:pto_corte3) = poblac(index_madre,
arrancadas+1:pto_corte3);
        aux(i+3, pto_corte3+1:pto_corte4) = poblac(index_padre,
pto_corte3+1:pto_corte4);
```

```
        aux(i+3, pto_corte4+1:columnas) = poblac(index_madre,
pto_corte4+1:columnas);

    end

    if rand > 0.5
        mutada = mutar(aux, arrancadas);
    else
        mutada = mutar_inver(aux,arrancadas);
    end

    aux = mutada;



aleatorios = n_ind - size(aux,1);
restantes_aleatorios = crea_aleatoria2(aleatorios, arrancadas);
nueva = [aux; restantes_aleatorios];



function nueva = cruzar_3(poblac, valores_ord, media)
n_ind = size(poblac,1);
columnas = size(poblac, 2);
arrancadas = columnas / 2;
nuevos_ind = floor(n_ind*0.8);
aux = zeros(nuevos_ind,columnas);

puntuaciones = valores_ord(:,1)/media;

punt_acumulada = zeros(1, n_ind);
punt_acumulada(1) = puntuaciones(1);

for i=2:n_ind
    punt_acumulada(i) = punt_acumulada(i-1) + puntuaciones(i);
end

punt_acumulada = [punt_acumulada' valores_ord(:,2)];

puntuaciones_aleatorias = zeros(nuevos_ind, 1);
aleatorio = rand;

for i = 1:nuevos_ind
    puntuaciones_aleatorias(i) = n_ind * (aleatorio + i - 1) / nuevos_ind;
end

aux(1,:) = poblac((valores_ord(1,2)),:);
aux(2,:) = poblac((valores_ord(2,2)),:);
aux(3,:) = poblac((valores_ord(1,2)),:);
aux(4,:) = poblac((valores_ord(2,2)),:);
aux(5,:) = poblac((valores_ord(1,2)),:);
aux(6,:) = poblac((valores_ord(2,2)),:);
aux(7,:) = poblac((valores_ord(1,2)),:);
aux(8,:) = poblac((valores_ord(2,2)),:);
aux(9,:) = poblac((valores_ord(1,2)),:);
aux(10,:) = poblac((valores_ord(2,2)),:);
```

```
matriz_cruces = determina_cruces(puntuaciones_aleatorias,
punt_acumulada, nuevos_ind);

j = 1;

alfas = rand(1, nuevos_ind);

for i=1:2:(nuevos_ind-11)
    index_padre = matriz_cruces(j,1);

    index_madre = matriz_cruces(j,2);
    j = j + 1;


    pto_medio = floor(arrancadas/2);
    pto_corte1 = randi(pto_medio - 2) + 1;
    pto_corte2 = randi(pto_medio - 2) + pto_medio;
    pto_corte3 = pto_corte1 + arrancadas;
    pto_corte4 = pto_corte2 + arrancadas;

    alfa = alfas(1,i);
    beta = alfas(1,(i+1));

    aux(i+10, 1:pto_corte1) = alfa * poblac(index_padre, 1:pto_corte1) +
(1 - alfa) * poblac(index_madre, 1:pto_corte1);       %Creamos el hijo
    aux(i+10, pto_corte1+1:pto_corte2) = alfa * poblac(index_madre,
pto_corte1+1:pto_corte2) + (1 - alfa) * poblac(index_padre,
pto_corte1+1:pto_corte2);
    aux(i+10, pto_corte2+1:arrancadas) = alfa * poblac(index_padre,
pto_corte2+1:arrancadas) + (1 - alfa) * poblac(index_madre,
pto_corte2+1:arrancadas);
    aux(i+10, arrancadas+1:pto_corte3) = alfa * poblac(index_padre,
arrancadas+1:pto_corte3) + (1 - alfa) * poblac(index_madre,
arrancadas+1:pto_corte3);
    aux(i+10, pto_corte3+1:pto_corte4) = alfa * poblac(index_madre,
pto_corte3+1:pto_corte4) + (1 - alfa) * poblac(index_padre,
pto_corte3+1:pto_corte4);
    aux(i+10, pto_corte4+1:columnas) = alfa * poblac(index_padre,
pto_corte4+1:columnas) + (1 - alfa) * poblac(index_madre,
pto_corte4+1:columnas);

    aux(i+11, 1:pto_corte1) = beta * poblac(index_padre, 1:pto_corte1) +
(1 - beta) * poblac(index_madre, 1:pto_corte1);       %Creamos la hija
    aux(i+11, pto_corte1+1:pto_corte2) = beta * poblac(index_madre,
pto_corte1+1:pto_corte2) + (1 - beta) * poblac(index_padre,
pto_corte1+1:pto_corte2);
    aux(i+11, pto_corte2+1:arrancadas) = beta * poblac(index_padre,
pto_corte2+1:arrancadas) + (1 - beta) * poblac(index_madre,
pto_corte2+1:arrancadas);
    aux(i+11, arrancadas+1:pto_corte3) = beta * poblac(index_padre,
arrancadas+1:pto_corte3) + (1 - beta) * poblac(index_madre,
arrancadas+1:pto_corte3);
    aux(i+11, pto_corte3+1:pto_corte4) = beta * poblac(index_madre,
pto_corte3+1:pto_corte4) + (1 - beta) * poblac(index_padre,
pto_corte3+1:pto_corte4);
    aux(i+11, pto_corte4+1:columnas) = beta * poblac(index_padre,
pto_corte4+1:columnas) + (1 - beta) * poblac(index_madre,
pto_corte4+1:columnas);
```

```
end

if rand > 0.5
    mutada = mutar(aux, arrancadas);
else
    mutada = mutar_inver(aux,arrancadas);
end

aux = mutada;

aleatorios = n_ind - size(aux,1);

restantes_aleatorios = crea_aleatoria2(aleatorios, arrancadas);

nueva = [aux; restantes_aleatorios];

function nueva = cruzar_4(poblac, valores_ord, media)
n_ind = size(poblac,1);
columnas = size(poblac, 2);
arrancadas = columnas / 2;
nuevos_ind = floor(n_ind*0.8);
aux = zeros(nuevos_ind,columnas);

puntuaciones = valores_ord(:,1)/media;

punt_acumulada = zeros(1, n_ind);
punt_acumulada(1) = puntuaciones(1);

for i=2:n_ind
    punt_acumulada(i) = punt_acumulada(i-1) + puntuaciones(i);
end

punt_acumulada = [punt_acumulada' valores_ord(:,2)];

puntuaciones_aleatorias = zeros(nuevos_ind, 1);
aleatorio = rand;

for i = 1:nuevos_ind
    puntuaciones_aleatorias(i) = n_ind * (aleatorio + i - 1) / nuevos_ind;
end


aux(1,:) = poblac((valores_ord(1,2)),:);
aux(2,:) = poblac((valores_ord(2,2)),:);
aux(3,:) = poblac((valores_ord(1,2)),:);
aux(4,:) = poblac((valores_ord(2,2)),:);
aux(5,:) = poblac((valores_ord(1,2)),:);
aux(6,:) = poblac((valores_ord(2,2)),:);
aux(7,:) = poblac((valores_ord(1,2)),:);
aux(8,:) = poblac((valores_ord(2,2)),:);
aux(9,:) = poblac((valores_ord(1,2)),:);
aux(10,:) = poblac((valores_ord(2,2)),:);

matriz_cruces = determina_cruces(puntuaciones_aleatorias,
punt_acumulada, nuevos_ind);

j = 1;
```

```
for i=1:2:(nuevos_ind-11)
    index_padre = matriz_cruces(j,1);

    index_madre = matriz_cruces(j,2);
    j = j + 1;

    vector_binario = randi(2, 1, columnas) - 1;

    for k = 1:columnas

        if vector_binario(k) == 1
            aux(i+10, k) = poblac(index_padre, k);
            aux(i+11, k) = poblac(index_madre, k);
        else
            aux(i+10, k) = poblac(index_madre, k);
            aux(i+11, k) = poblac(index_padre, k);
        end
    end

end

if rand > 0.5
    mutada = mutar(aux, arrancadas);
else
    mutada = mutar_inver(aux,arrancadas);
end

aux = mutada;

aleatorios = n_ind - size(aux,1);

restantes_aleatorios = crea_aleatoria2(aleatorios, arrancadas);

nueva = [aux; restantes_aleatorios];

function matriz = mutar(aux, arrancadas)
n_ind = size(aux,1);
n_mutados = floor(0.8 * n_ind);

ind_sel = randi(n_ind-2, n_mutados, 1) + 2;
gen_sel = randi(arrancadas * 2, n_mutados, 1);

sel = [ind_sel gen_sel];

for i=1:n_mutados
    if sel(i,2) <= arrancadas
        aux(sel(i,1), sel(i,2)) = rand * 2 + 6;
    else
        aux(sel(i,1), sel(i,2)) = rand * 3 + 8.5;
    end
end

matriz = aux;

function matriz = mutar_inver(aux, arrancadas)
n_ind = size(aux,1);
n_mutados = floor(0.8 * n_ind);
```

```
temporal = aux;

ind_sel = randi(n_ind-2, n_mutados, 1) + 2;
gen_sel = randi(arrancadas, n_mutados, 2);

sel = [ind_sel gen_sel];

for i=1:n_mutados
    if sel(i,2) ~= sel(i,3)
        aux(sel(i,1), sel(i,2)) = aux(sel(i,1), sel(i,3));
        aux(sel(i,1), sel(i,3)) = temporal(sel(i,1), sel(i,2));

        aux(sel(i,1), sel(i,2) + arrancadas) = aux(sel(i,1), sel(i,3) +
arrancadas);
        aux(sel(i,1), sel(i,3) + arrancadas) = temporal(sel(i,1),
sel(i,2) + arrancadas);

    end
end

matriz = aux;

function vector = crea_vector_min(poblac)
n_ind = size(poblac,1);
arrancadas = size(poblac,2) / 2;
vector_plot = zeros(1, n_ind * arrancadas);

k = 1;

for i=1:arrancadas
    for j=1:n_ind
        vector_plot(k) = poblac(j,i);
        k = k + 1;
    end
end

vector = vector_plot;


function vector = crea_vector_max(poblac)
n_ind = size(poblac,1);
arrancadas = size(poblac,2) / 2;
vector_plot = zeros(1, n_ind * arrancadas);

k = 1;

for i=arrancadas + 1:arrancadas * 2
    for j=1:n_ind
        vector_plot(k) = poblac(j,i);
        k = k + 1;
    end
end

vector = vector_plot;

function vector = determina_ptos_arranque(vector_espacio,
momentos_arranque_mejor, arrancadas_mejor, Arrancadas)
filas = size(vector_espacio, 3);
```

```
arrancadas_hechas = arrancadas_mejor(1,1,end);

if arrancadas_hechas > Arrancadas
    arrancadas_hechas = Arrancadas;
end

aux = zeros(1,arrancadas_hechas);
contador = 1;

for i=2:filas
    if momentos_arranque_mejor(1,1,i)==1 &&
momentos_arranque_mejor(1,1,i-1)==0
        aux(contador) = vector_espacio(1,1,i);
        contador = contador + 1;
    end
end

vector = aux;
```