|  |  |
| --- | --- |
|  | **niversidad**  **eyuanarlos** |

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERIA INFORMATICA**

**Curso Académico 2017/2018**

**Trabajo Fin de Grado**

## **CONFIGURACIÓN OPTIMA DE LOS PARÁMETROS GEOMÉTRICOS DE UN ROBOT MÓVIL CON DIRECCIONAMIENTO DIFERENCIAL**

### **Autor:** David Vacas Miguel

### **Director/Tutor:** Alberto Herrán González

Índice

[Índice 2](#_Toc512858237)

[Agradecimientos 3](#_Toc512858238)

[Resumen 4](#_Toc512858239)

[Capítulo 1 Introducción 5](#_Toc512858240)

[1. Motivación 5](#_Toc512858241)

[2. Objetivos 6](#_Toc512858242)

[3. Estado del arte 6](#_Toc512858243)

[4. Estructura de la memoria 8](#_Toc512858244)

[Capítulo 2 Descripción del problema 10](#_Toc512858245)

[1. Direccionamiento diferencial 10](#_Toc512858246)

[2. Navegación autónoma 13](#_Toc512858247)

[3. Función objetivo 15](#_Toc512858248)

[Capítulo 3 Descripción algorítmica 16](#_Toc512858249)

[1. Métodos constructivos 16](#_Toc512858250)

[2. Generación de vecindarios 16](#_Toc512858251)

[3. Búsquedas locales 17](#_Toc512858252)

[4. Búsquedas globales 17](#_Toc512858253)

[Capítulo 4 Implementación 19](#_Toc512858254)

[1. Metodología 19](#_Toc512858255)

[2. Diseño 20](#_Toc512858256)

[3. Estructuras de datos 21](#_Toc512858257)

[Capítulo 5 Resultados 23](#_Toc512858258)

[1. Descripción de las instancias 23](#_Toc512858259)

[2. Constructivos 23](#_Toc512858260)

[3. Búsquedas locales 23](#_Toc512858261)

[4. Resultados finales 23](#_Toc512858262)

[Conclusiones 24](#_Toc512858263)

Agradecimientos

Me gustaría agradecer a la universidad por el conocimiento recibido y en especial a mi tutor, Alberto Herrán, por su ayuda y apoyo durante el desarrollo del proyecto.

Resumen

En este TFG se ha desarrollado un algoritmo que calcula los parámetros óptimos de un robot con direccionamiento diferencial para obtener el mejor tiempo en un determinado circuito.

Se ha comenzado estudiando el funcionamiento del robot sobre el circuito dado que esto proporciona la función objetivo del algoritmo. A continuación, se pasó al análisis del algoritmo utilizado: métodos constructivos, generación y búsqueda de vecindarios, etc. Inmediatamente después se realizó la implementación del algoritmo. Finalmente se implementaron diferentes tipos de constructivos y generadores de vecindarios.

Capítulo 1 Introducción

1. Motivación

Este trabajo nace motivado del problema que resulta tratar de buscar los parámetros óptimos de un robot con direccionamiento diferencial sobre un circuito determinado.

Este trabajo se basa en una aplicación anterior la cual simula la navegación autónoma de un robot con direccionamiento diferencial sobre un circuito determinado. Este robot podía ser parametrizado de forma sencilla a través de una interfaz como se puede observar en la figura X.



**Figura X.** Aplicación de simulación de robot con direccionamiento diferencial.

Dado que se manejan diferentes parámetros geométricos de forma simultánea, puede llegar a ser complicado el intentar tratar de ajustarlos de forma manual.

Debido a esto se ha decidido desarrollar un algoritmo que realice la búsqueda de los parámetros óptimos de forma automática

2. Objetivos

Una vez vistos los problemas por los cuales se realiza este proyecto, los objetivos con los cuales subsanarlos, además de otros adicionales son:

* **Obtención de parámetros óptimos:** el objetivo principal del algoritmo se basa en obtener los parámetros óptimos del robotpara un circuito determinado ya que la obtención de estos de forma manual es realmente costosa en esfuerzo y tiempo.

Por lo tanto, los requisitos que debe tener el algoritmo son:

* Conseguir los mejores parámetros para un determinado circuito.

3. Estado del arte

Para la resolución de problemas de optimización existen diferentes tipos de metaheurísticas, las mas relevantes

* **Búsqueda local:** se basa en a partir de una solución inicial, la búsqueda de una mejor solución en un vecindario. Esto se repite de forma iterativa. La búsqueda local es la basa de muchas metaheurísticas mas complejas, en las cuales se modifican la forma de generar la solución inicial, la generación de vecindarios, la elección del vecino óptimo, la condición de finalización, etc.
* **Algoritmos voraces:** se basa en la elección optima en cada paso local, con la intención de conseguir así la solución general óptima.
* **Algoritmos genéticos:** se inspiran en la evolución biológica y su base genético-molecular. Se trata de una metaheurística de evolución, a lo largo de las diferentes iteraciones, mantienen un conjunto de posibles soluciones del problema, cuyos valores evolucionan hacia las mejores soluciones del problema mediante un proceso combinado de selección de individuos y operadores genéticos.
* **Algoritmo temple simulado:** es un tipo de búsqueda global capaz de escapar de mínimos locales mediante la inclusión de cierta probabilidad de aceptar peores soluciones a la actual durante la búsqueda.
* **Búsqueda Tabú:** al igual que el anterior, es un algoritmo capaz de escapar de mínimos locales debido a que recopila información durante la exploración, la cual se utiliza para restringir las elecciones de vecinos en los vecindarios.

4. Estructura de la memoria

A continuación, se describe brevemente la estructura del resto del documento:

En el Capítulo 2, ***Descripción del problema***, se comienza explicando que es un vehículo a motor y su configuración de direccionamiento diferencial, así como las ecuaciones que se sacan de las mismas para la función objetivo. A continuación, se explica la navegación autónoma que realiza el robot. Por ultimo se describe como se lleva a cabo la simulación del sistema.

En el Capítulo 3, ***Descripción algorítmica***, se expone el algoritmo, comenzando por los métodos constructivos utilizados seguidos de los generadores de vecindarios. Por ultimo se explica el algoritmo de búsqueda local básica y posibles metaheurísticas con las que escapar de los óptimos locales.

En el Capítulo 4, ***Implementación***, se comienza informando sobre la metodología utilizada en el proyecto, lenguaje de programación, sistema de control de versiones, etc. A continuación, se pasa a explicar el diseño de la aplicación mediante un diagrama de clases. Por ultimo se describen los detalles algorítmicos de bajo nivel.

En el Capítulo 5, ***Resultados***, se comparan los diferentes resultados obtenidos sobre los diferentes constructores y generadores de vecindarios.

Capítulo 2 Descripción del problema

1. Direccionamiento diferencial

Los vehículos con ruedas son una solución simple y eficiente para conseguir movilidad sobre terrenos duros y libres de obstáculos, con los que se permite conseguir velocidades más o menos altas.

Su limitación más importante es el deslizamiento en la impulsión, además dependiendo del tipo de terreno, puede aparecer deslizamiento y vibraciones en el mismo. Como se ha dicho en la definición, estos vehículos son eficientes en terrenos duros y libres de obstáculos, por lo tanto, en terrenos blandos son poco eficientes.

Otro problema que tienen este tipo de vehículos se haya en que no es posible modificar la estabilidad para adaptarse al terreno, excepto en configuraciones muy especiales, lo que limita los terrenos sobre los que es aceptable el vehículo.

A continuación, se pasa a calcular las ecuaciones del modelo cinemático y discreto.

Según se ve en la figura X, se supone un sistema de referencia {G} y un sistema {L} con origen en el punto de guiado del vehículo y eje YL en la dirección del eje longitudinal del vehículo.



**Figura X.** Modelo cinemático.

Por lo tanto, si el vehículo tiene una velocidad de desplazamiento y de rotación con respecto a {L}, con respecto a {G} la velocidad es:

(1)

La derivada de una función puede aproximarse por el cociente incremental:

(2)

Esto se conoce como derivada discreta hacia adelante, pero también puede aproximarse con el cociente incremental hacia atrás, la aproximación centrada, u otras aproximaciones más complicadas.

El resultado es que, con una ecuación de este tipo, la nueva coordenada x {en t+1} se puede calcular a partir de la anterior (en t) mediante la ecuación:

(3)

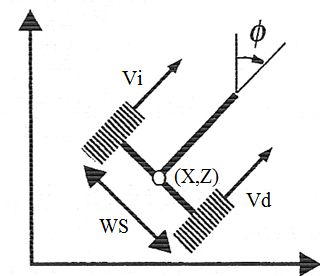
Aplicando el mismo resultado al resto de coordenadas del modelo cinemático directo, se obtiene:

(4)

Por tanto, si se dispone de las coordenadas en cada instante de un determinado horizonte temporal, se puede calcular la nueva posición y orientación del robot en dicho horizonte.

Los vehículos con ruedas tienen diferentes configuraciones, el direccionamiento diferencial es una de ellas. El robot que se ha simulado utiliza esta configuración, por lo tanto, a continuación, se explica la misma y se deducen las fórmulas utilizadas para la simulación.

El direccionamiento diferencial viene dado por la diferencia de velocidades de las ruedas laterales. La tracción se consigue con esas mismas ruedas. Adicionalmente, existen una o más ruedas para el soporte. En la figura X se muestra una imagen de dicho esquema. Nótese que para especificar la configuración hay que indicar los valores de las tres variables (x, z, ), siendo las variables de control las velocidades de las ruedas laterales.



**Figura X.** Locomoción mediante guiado diferencial.

Sean y las velocidades de giro de las ruedas izquierda y derecha, respectivamente. Si el radio de la rueda es WR, las velocidades lineales correspondientes son y . Es este caso, la velocidad lineal y velocidad angular correspondientes en el modelo vienen dadas por:

(5)

Sustituyendo estas expresiones en las obtenidas a partir de la Figura X, se obtienen las velocidades de las coordenadas del robot en el sistema {G} a partir de la velocidad de giro de cada rueda:

(6)

Finalmente, utilizando el modelo discreto, se obtiene:

(7)

Estas últimas ecuaciones se utilizarán en el proyecto para el cálculo de la posición del robot.

2. Navegación autónoma

El robot sigue líneas que se ha implementado realiza su movimiento de manera autónoma, se coloca el robot sobre un fondo blanco con una línea negra que representa el circuito, como se muestra en la figura X, y este deberá recorrer el circuito sin salirse del mismo. Esto se puede realizar gracias a dos sensores que son implantados en la parte posterior del robot los cuales son responsables de la detección de la línea del circuito. En función de lo que estos sensores recojan (están sobre el circuito o no) el robot realizara cambios en la velocidad de sus ruedas resultando en un movimiento recto, rotatorio hacia la izquierda o rotatorio hacia la derecha.

**Figura X.** Colocación del robot en un circuito negro sobre fondo blanco.

Los sensores que se implementan en este tipo de robots son sensores CNY70, los cuales se muestran en la Figura X.



**Figura X.** Sensor CNY70.

Estos son sensores ópticos reflexivos de corto alcance basados en un diodo de emisión de luz infrarroja y un receptor formado por un fototransistor que ambos apuntan en la misma dirección, esta estructura de forma simplificada se puede observar en la figura X. Cuando el sensor se haya sobre una línea negra la luz es absorbida y el fototransistor envía una señal (ya sea alta o baja dependiendo del montaje del sensor), sin embargo, cuando se haya sobre fondo blanco la luz es reflejada y por lo tanto el fototransistor envía la señal contraria a la enviada al estar sobre negro.



**Figura X.** Estructura simplificada del sensor CNY70.

La simulación de estos sensores se basa en el conocimiento de la posición de los sensores en todo momento y de los puntos que conforman el circuito, pudiendo así comprobar si cualquiera de los dos sensores está situado sobre el circuito.

3. Función objetivo

El sistema simula la posición del robot en cada instante. Esta posición será determinada por las ecuaciones obtenidas anteriormente. Gracias a esta posición se puede obtener la posición de los sensores con los cuales se decide si se ha de parar una de las ruedas. En caso de que una de las ruedas se pare, el robot se gira pudiendo realizar así el circuito.

Los parámetros variables que se utilizan en la simulación son: la separación entre las ruedas, el radio de las ruedas, la distancia entre los ejes (distancia desde el centro de los sensores hasta el punto de rotación del robot) y la separación entre los sensores.

Para medir la calidad de cada solución se utiliza el tiempo que el robot tarda en realizar el circuito de principio a fin.

Capítulo 3 Descripción algorítmica

1. Métodos constructivos

Dado que se trata de un algoritmo que se basa en variables reales, el conjunto de valores iniciales tiene unos límites físicos, además puesto que se realiza una sola ejecución de la aplicación no se puede tener en cuenta valores de ejecuciones anteriores para la generación de esta primera solución.

Por esto el método constructivo que se ha utilizado se basa en la generación de valores aleatorios con limites superiores e inferiores distintos para cada una de las variables. Con esto cada vez que empiece el algoritmo se comienza desde una solución totalmente aleatoria y distinta.

2. Generación de vecindarios

En cuanto a la generación de vecindarios existen múltiples formas. Como se ha dicho anteriormente, debido a que se trata de valores reales, se ha optado por la permutación de una o más variables para la generación del vecindario.

Con esto se ha conseguido un total de 7 generadores diferentes:

1. Se genera una vecindad de 8. Cada vecino se genera por la adición y sustracción de 0.1 en uno de sus valores.
2. Se genera una vecindad de 12. Cada vecino se genera por la adición y sustracción de 0.1 en dos de sus valores.
3. Se genera una vecindad de 8. Cada vecino se genera por la adición y sustracción de 0.1 en tres de sus valores.
4. Se genera una vecindad de 2. Cada vecino se genera por la adición y sustracción de 0.1 en todos sus valores.
5. Se genera una vecindad de 12. Cada vecino se genera por las posibles combinaciones de adición y sustracción de 0.1 en dos valores distintos
6. Se genera una vecindad de 24. Cada vecino se genera por las posibles combinaciones de adición y sustracción de 0.1 en tres valores distintos.
7. Se genera una vecindad de 8. Cada vecino se genera por las posibles combinaciones de adición y sustracción de 0.1 en todos los valores.

3. Búsquedas locales

Como se ha explicado en el punto 1.3 Estado del arte, el algoritmo de búsqueda local básicase basa en explorar el entorno de una solución con el fin de encontrar otra mejor. Para esto se realizan cambios sobre los diferentes elementos de una solución lo que genera más soluciones diferentes de la anterior, que se conocen como vecinos. De entre estos vecinos se elige uno que se convierte en la solución y se vuelve a iterar sobre este.

Todos los vecinos del vecindario se pasan por la función objetivo. Esta función es la encargada de informar sobre que vecino es mejor.

Existen diferentes formas de escoger el vecino que se convierte en la siguiente solución, algunas de ellas son:

- ***First***, se escoge el primer vecino que mejore a la solución actual.

- ***Best***, se escoge al mejor vecino.

4. Búsquedas globales

El principal problema de las búsquedas locales se haya en que se pueden quedar fácilmente atrapadas en un óptimo local. Un óptimo local es una solución que no puede ser mejorada en el vecindario que ella genera y que, además, no es la solución óptima del problema. Las metaheurísticas que tratan de evitar este problema se las llama búsquedas globales.

Para solucionar el problema de los óptimos locales se podría utilizar:

* ***Multiarranque***, para tratar de evitar los óptimos locales, esta metaheurística reinicia la búsqueda desde una nueva solución construida con cualquiera de los métodos de construcción disponibles cuando se encuentra con un óptimo local.
* ***ILS (Iterated Local Search)***, esta metaheurística cuando se encuentra con un óptimo local reinicia la búsqueda desde la solución actual o una perturbación de esta.
* ***VNS (Variable Neighborhood Search)***, esta metaheurística va modificando en cada iteración la estructura del vecindario.
* ***SA (Simulated Annealing)***, está basado en el tratamiento con calor de la metalurgia. Esta metaheurística acepta soluciones peores para intentar llegar al óptimo.

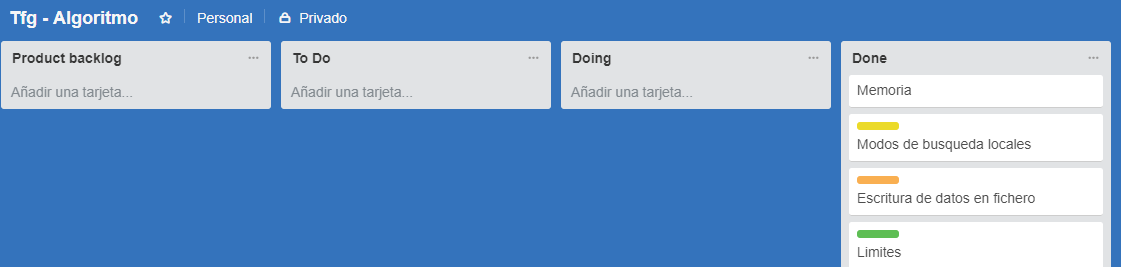
Capítulo 4 Implementación

1. Metodología

En cuanto a la tecnología utilizada para la realización de este proyecto se ha utilizado como lenguaje de programación **Java 8**, un lenguaje de programación de propósito general, concurrente y orientado a objetos. Se ha utilizado el IDE **NetBeans 8.2** puesto que tiene una buena integración con Java.

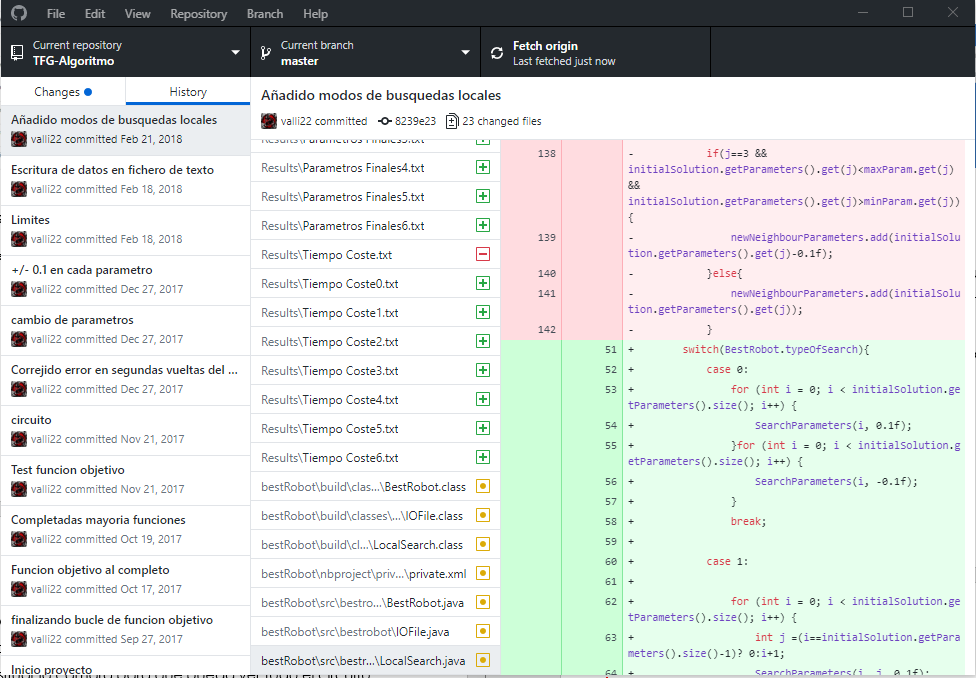
En lo referido a la metodología se han utilizado herramientas utilizadas habitualmente en proyectos en los que se aplica metodología ágil, principalmente Trello y GitHub.

**Trello** como tablero de tareas, este se divide en las columnas habituales (Product backlog, To Do, Doing, Done). A su vez cada tarea tiene asignada una dificultad representada mediante colores. Las tareas no tienen asignadas personas puesto que hay una sola persona encargada de este tablero. A pesar de no ser relevante para la organización de un equipo y la división de tareas, puesto que solo se trata de una persona, ha resultado muy útil para no perder la visión de proyecto. Todo esto se puede ver en la figura X.



**Figura X.** Trello utilizado en el desarrollo del proyecto.

**Git** como repositorio y control de versiones (utilizado concretamente GitHub). Sobre este repositorio se ha ido subiendo los diferentes incrementos de funcionalidad de la aplicación de forma periódica y gracias a él se ha podido realizar un control de versiones. Se puede observar el repositorio desde la aplicación de Windows en la figura X.

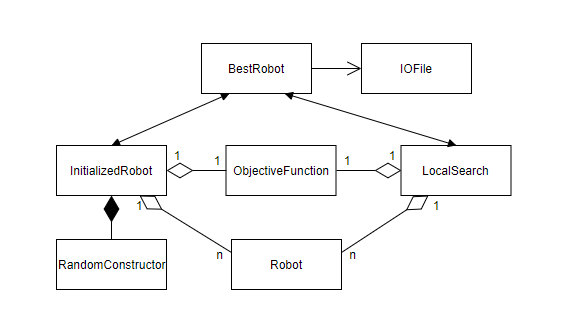


**Figura X.** Repositorio en GitHub desde la aplicación

de escritorio de Windows.

2. Diseño

El diseño de la aplicación se puede observar en el UML que se muestra en la figura X.



**Figura X.** UML

A continuación, se explica brevemente el funcionamiento de las clases expuestas en el UML:

* **BestRobot**, esta clase es la clase principal donde se haya la lógica del algoritmo, y la que ira llamando a las distintas clases.
* **InitializedRobot**, es la clase constructora del algoritmo. Construye la primera solución desde la que el algoritmo parte.
* **RandomConstructor**, esta clase se usa en la clase InitializedRobot para determinar los valores de la primera solución. Esta clase genera los valores aleatorios que tiene la primera solución del algoritmo y además tiene los limites sobre los cuales se pueden mover los valores durante el algoritmo.
* **LocalSearch**, como se indica en el nombre, esta clase es la encargada de generar los vecindarios de las diferentes soluciones.
* **ObjectiveFunction**, como se indica en el nombre, esta clase se encarga de obtener el valor por el cual se comparan los vecinos para comprobar cuál es mejor.
* **Robot**, esta clase guarda la información necesaria sobre un robot, es decir guarda los valores que serán permutados a lo largo del algoritmo.
* **IOFile**, se trata de la clase que escribe en ficheros de texto los datos que se quieren guardar.

3. Estructuras de datos

En este apartado se explican detalles sobre la implementación del algoritmo:

* **ObjectiveFunction**: Se va a explicar el funcionamiento de esta clase, es decir, como se obtiene el valor por el cual se decide que robot es mejor.

Se comienza leyendo el circuito de un fichero de texto para guardarlo en una variable. A continuación, se comienza con el cálculo de la posición del robot en función de la posición en el instante anterior, los parámetros de este, y la velocidad de las ruedas en ese instante. Esto se realiza mediante las ecuaciones número 7, calculadas en el apartado ***2.1 Direccionamiento diferencial***. Para saber si se debe mover cada rueda o dejarla parada se utilizan dos sensores que detectan si están sobre el circuito o no. En caso de que el sensor izquierdo se encuentre cerca del circuito la rueda izquierda se para, y lo mismo para el sensor y rueda derecha. Al parar una de las dos ruedas se consigue que el robot gire en dirección a la rueda detenida. Además, se han creado funciones propias que replican el efecto de las funciones translate y rotate de OpenGL para el calculo de las matrices de posición que son necesarias para saber la posición de los sensores en cada instante.

Este se ejecuta hasta que la posición del robot sea la inicial, es decir hasta que el robot ha realizado una vuelta completa al circuito. Al terminar se comprueba el tiempo que ha tardado en realizar el circuito y este es el que se devuelve como valor a comparar entre circuitos.

Como añadido se ha incluido una condición de tiempo en la que si el robot no ha realizado el circuito en ese tiempo se para la ejecución, esto se debe a que, al ser un problema real, dependiendo de los parámetros que se incluyan en el robot, es posible que el robot se quede parado al realizar el circuito y por tanto nunca acabaría el mismo y fallaría el algoritmo.















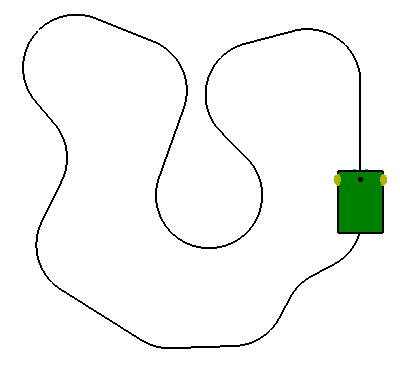




Capítulo 5 Resultados

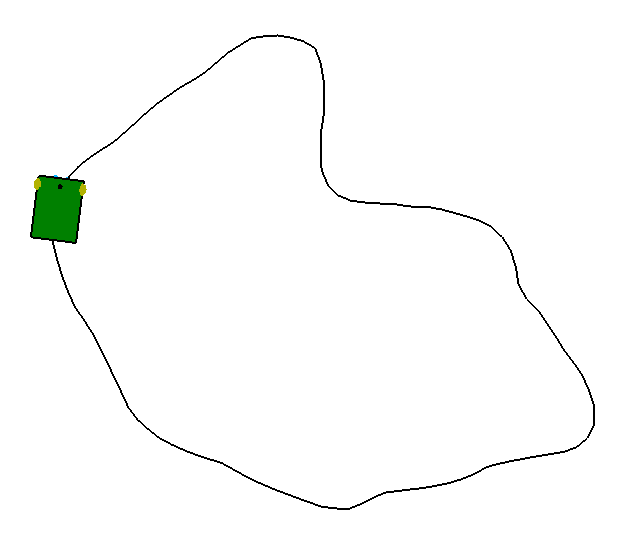
1. Descripción de las instancias

El algoritmo se va a ejecutar sobre 3 instancias que se diferencian en el circuito sobre el cual el robot se mueve. El primero de ellos es un circuito estándar, tamaño medio y con algunas curvas, se puede observar en la figura X.



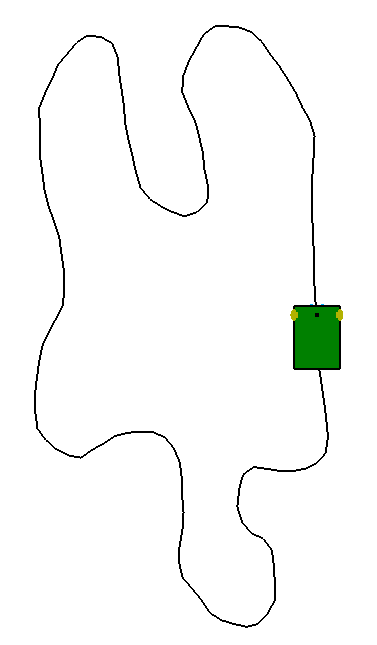
**Figura X.** Circuito 1 estándar

El segundo de los circuitos se trata de un circuito de gran tamaño sin gran cantidad de curvas, este se puede observar en la figura X.



**Figura X.** Circuito 2 grande sin curvas

El tercero de los circuitos se trata de un circuito de tamaño medio, pero con curvas muy cerradas que hace que varios de los posibles robots no puedan realizarlo, se puede observar este circuito en la figura X.



**Figura X.** Circuito 3 tamaño medio con curvas cerradas

2. Constructivos

En lo respectivo a los constructivos, se ha implementado uno debido a que al realizarse una sola ejecución del programa y al ser valores reales no se han encontrado otros constructivos adecuados para el problema expuesto.

El método constructivo implementado genera valores aleatorios dentro de un rango específico para cada valor, es decir, cada valor tiene unos limites adecuados para su geometría real.

3. Búsquedas locales

4. Resultados finales

Conclusiones

**Bibliografía**