Universidad
Rey Juan Carlos

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

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 puesto que esto proporciona la función objetivo del algoritmo. A continuación, se pasa 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 instancias, constructivos y generadores de vecindarios.

lo reescribimos al final.

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. Este robot puede ser parametrizado de forma sencilla a través de una interfaz como se puede observar en la figura X.

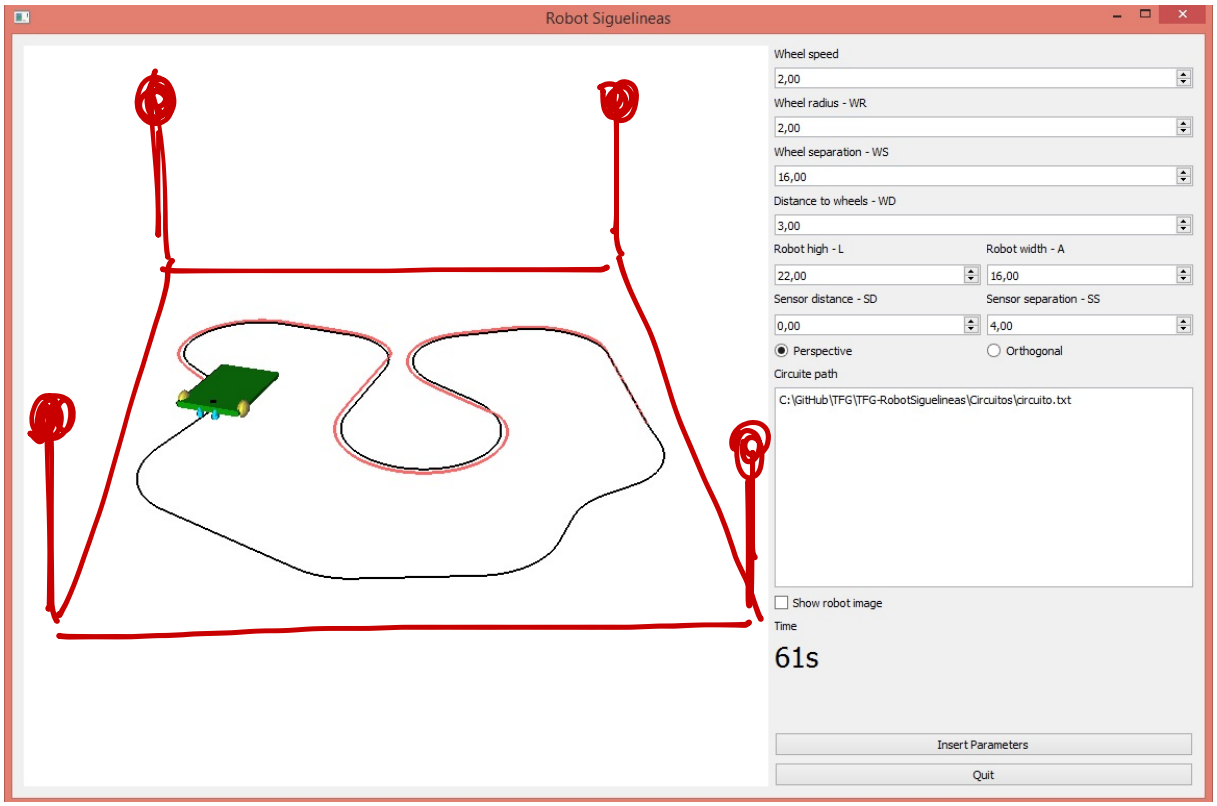


Figura X. Aplicación de simulación de robot con direccionamiento diferencial.

Dado que se manejan diferentes parámetros geométricos de forma simultánea, puede llegar a ser complicado 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 robot para 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 más relevantes son:

- **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 base de muchas metaheurísticas más 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. En caso de que la primera elección optima no sea factible, se descarta el elemento.

- **Algoritmos genéticos:** se inspiran en la evolución biológica y su base genético-molecular. Se trata de una metaheurística de evolución que, a lo largo de las diferentes iteraciones, mantienen un conjunto de posibles soluciones del problema, cuyos valores evolucionan hacia las mejores soluciones 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 último, 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. Para finalizar 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 último, se describen los detalles algorítmicos de bajo nivel.

En el Capítulo 5, ***Resultados***, se comparan los diferentes resultados en tiempo de ejecución y solución final, obtenidos sobre las diferentes instancias y búsquedas locales mediante gráficas y tablas.

Capítulo 2 Descripción del problema

1. Direccionamiento diferencial

Los robots móviles son robots que, gracias a los progresos entre la inteligencia artificial y los robots físicos, son capaces de moverse a través de cualquier entorno físico. Estos normalmente son controlados por software y usan sensores y otros equipos para identificar la zona de su alrededor.

Los robots móviles se pueden diferenciar en dos tipos, autónomos y no autónomos. Los robots móviles autónomos pueden explorar el entorno sin ninguna directriz externa a él, al contrario que los no autónomos, que necesitan algún tipo de sistema de guía para moverse.

Los vehículos con ruedas son un tipo de robot móvil los cuales proporcionan 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.

Otro problema que tienen este tipo de vehículos se halla 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.

Los vehículos con ruedas emplean distintos tipos de locomoción que les da unas características y propiedades diferentes entre ellos en cuanto a eficiencia energética, dimensiones, maniobrabilidad y carga útil. El robot simulado en este programa tiene una configuración con direccionamiento diferencial.

En este sistema las ruedas se sitúan de forma paralela y no realizan giros. 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 8 se muestra una imagen de dicho esquema.

Figura 8: Robot con direccionamiento diferencial.

Para poder realizar la simulación gráfica del robot y pintarlo se debe conocer primeramente el estado del sistema. El estado del robot viene dado por su posición y orientación. Puesto que el robot se va a mover, se necesita saber el estado del mismo en los diferentes instantes de tiempo, por lo tanto, se debe definir un modelo de cambios de estado, es decir, una ecuación que a partir del estado actual en *(t)* y unas entradas en *(t)* se pase al estado en el siguiente instante *(t+1)*. En la ecuación X se puede observar la ecuación que define el sistema, siendo $\overline{s}$ el estado y $\overline{r}$ las entradas. Asimismo, en esta misma ecuación se muestran las variables que definen el estado del robot en un instante *(t)* dado, posición *(x(t), z(t))* y su orientación $\varphi(t)$ en ese instante:

$$\begin{aligned} \overline{s}(t+1) &= f(\overline{s}(t),\overline{r}(t)) \\ \overline{s}(t) &= (x(t), z(t), \varphi(t)) \end{aligned} \qquad (1)$$

Para obtener la ecuación de cambio de estado para un robot con direccionamiento diferencial se deben definir las entradas como las velocidades izquierda y derecha. Por lo tanto, la ecuación de las entradas resultante para un robot con direccionamiento diferencial, siendo w_i la velocidad de la rueda izquierda y w_d la velocidad de la rueda derecha, es la siguiente:

$$\overline{r} = (w_i(t), w_d(t)) \qquad (2)$$

En la figura 9 se pueden observar los diferentes datos de entrada y del sistema puestos sobre un robot con direccionamiento diferencial.

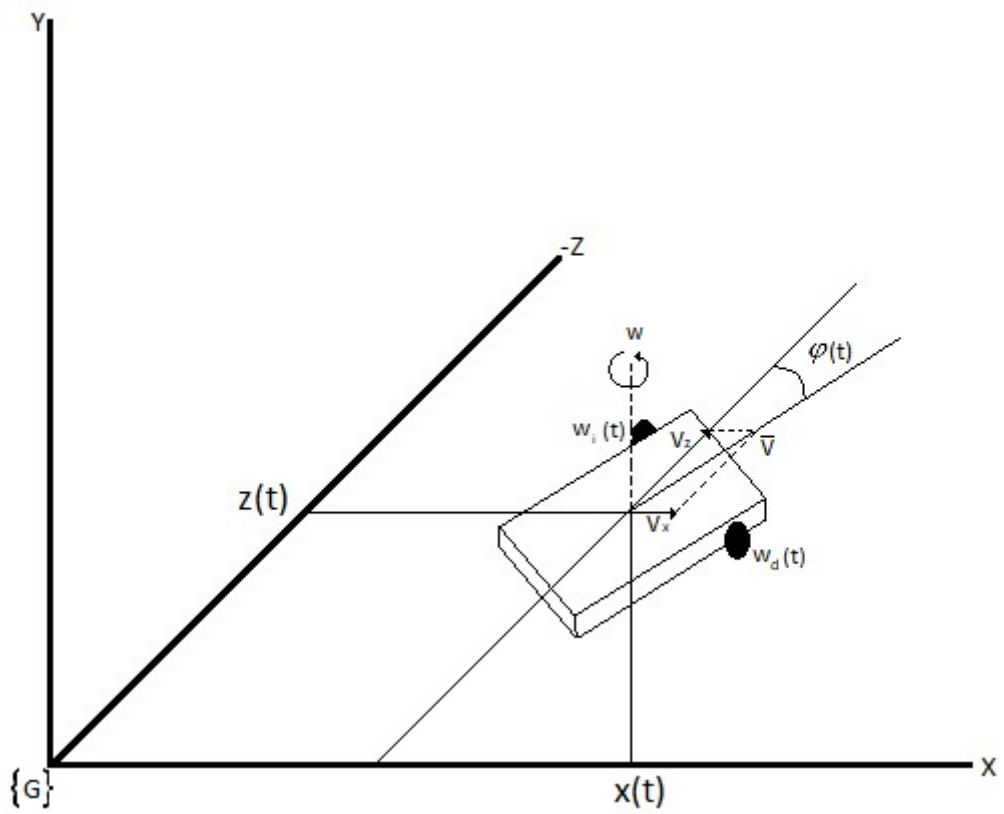


Figura 9. Sistema 3D de un robot con direccionamiento diferencial.

Una vez obtenido esto vamos a realizar el cálculo de las ecuaciones específicas.

A partir de la figura 9, si el vehículo tiene una velocidad de desplazamiento v y de rotación w, se obtiene que:

$$\begin{aligned} v_x &= v \cdot \sin(\varphi) \\ v_z &= v \cdot \cos(\varphi) \\ v_\varphi &= w \end{aligned} \tag{3}$$

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

$$v_x = \frac{dx}{dt} = \frac{x(t+1) - x(t)}{dt} \tag{4}$$

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:

$$x(t+1) = x(t) + v_x \cdot \Delta t \tag{5}$$

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

$$\begin{aligned} x(t+1) &= x(t) + v_x \cdot \Delta t \\ z(t+1) &= z(t) + v_z \cdot \Delta t \\ \varphi(t+1) &= \varphi(t) + v_\varphi \cdot \Delta t \end{aligned} \tag{6}$$

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

Note que para especificar la configuración hay que indicar los valores de las tres variables (x, z, φ), siendo las variables de control las velocidades de las ruedas laterales.

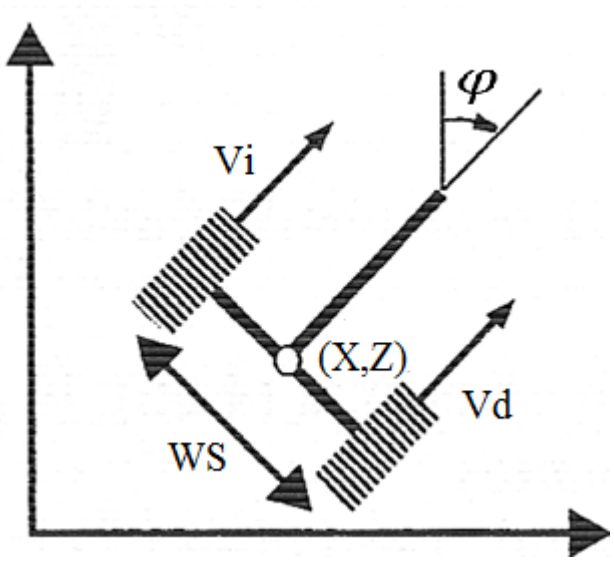


Figura 10. Locomoción mediante guiado diferencial.

Sean w_i y w_d las velocidades de giro de las ruedas izquierda y derecha, respectivamente. Si el radio de la rueda es WR, las velocidades lineales

correspondientes son $v_i = w_i \cdot WR$ y $v_d = w_d \cdot WR$. Es este caso, la velocidad lineal y velocidad angular correspondientes en el modelo vienen dadas por:

$$\begin{aligned} v &= \frac{v_d + v_i}{2} = \frac{(\mathrm{w}_d + v_i)\cdot \mathrm{WR}}{2} \\ w &= \frac{v_d - v_i}{WS} = \frac{(\mathrm{w}_d - \mathrm{w}_i)\cdot \mathrm{WR}}{WS} \end{aligned} \tag{7}$$

Sustituyendo estas expresiones en las obtenidas a partir de la Figura 10, se obtienen las velocidades de las coordenadas del robot en el sistema *{G}* a partir de la velocidad de giro de cada rueda:

$$\begin{aligned} v_x &= \frac{-(\mathrm{w}_d + \mathrm{w}_i)\cdot \mathrm{WR}}{2}\cdot \sin(\varphi) = -(\mathrm{w}_d + \mathrm{w}_i)\cdot \frac{\mathrm{WR}\cdot \sin(\varphi)}{2} \\ v_z &= \frac{(\mathrm{w}_d + \mathrm{w}_i)\cdot \mathrm{WR}}{2}\cdot \cos(\varphi) = (\mathrm{w}_d + \mathrm{w}_i)\cdot \frac{\mathrm{WR}\cdot \cos(\varphi)}{2} \\ v_\varphi &= \frac{(\mathrm{w}_d - \mathrm{w}_i)\cdot \mathrm{WR}}{WS} = (\mathrm{w}_d - \mathrm{w}_i)\cdot \frac{\mathrm{WR}}{WS} \end{aligned} \tag{8}$$

Finalmente, utilizando el modelo discreto, se obtiene:

$$\begin{aligned} x(t+1) &= x(t) - (w_d + w_i)\cdot \frac{WR\cdot \sin(\varphi(t))}{2}\cdot \Delta t \\ z(t+1) &= z(t) + (w_d + w_i)\cdot \frac{WR\cdot \cos(\varphi(t))}{2}\cdot \Delta t \\ \varphi(t+1) &= \varphi(t) + (w_d - w_i)\cdot \frac{WR}{WS}\cdot \Delta t \end{aligned} \tag{9}$$

Estas ecuaciones son las que nos sirven para poder calcular el estado del robot en el instante siguiente, es decir, la posición y la orientación en *(t+1)*. Las variables necesarias para poder calcular este son: el estado anterior ya sea su posición $\mathrm{x(t)}, \mathrm{z(t)}$ u orientación $\varphi(\mathrm{t})$, el radio WR, la separación WS y la velocidad de las ruedas w_d, w_i

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 11, y este deberá recorrer el circuito sin salirse del mismo. Esto se puede realizar gracias a dos sensores que son

implantados en la parte delantera 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 realiza cambios en la velocidad de sus ruedas resultando en un movimiento recto, rotatorio hacia la izquierda o rotatorio hacia la derecha.

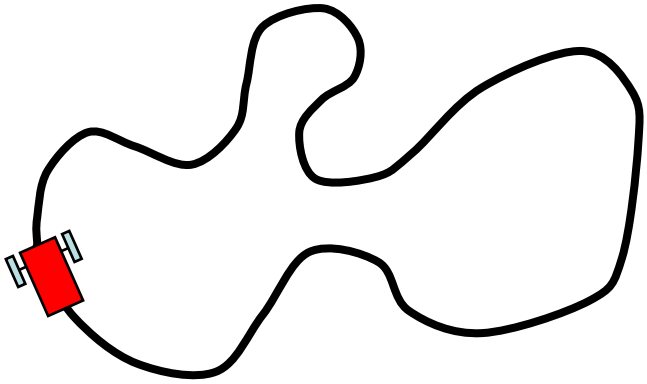


Figura 11. Colocación del robot en un circuito negro sobre fondo blanco.

Los sensores que se usan en este tipo de robots son sensores CNY70, los cuales se muestran en la Figura 12.

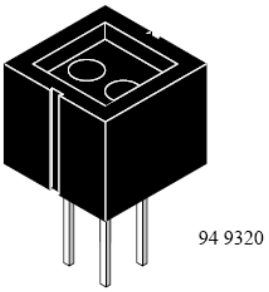


Figura 12. Sensor CNY70.

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 13. 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.

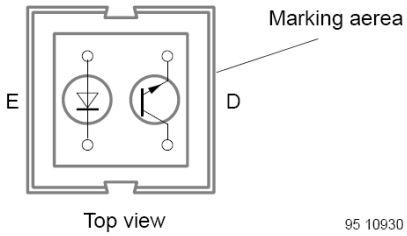


Figura 13. Estructura simplificada del sensor CNY70.

Para implementar dicho comportamiento en el simulador se ha seguido la siguiente lógica:

Si -> sensor izquierdo. | *Sd* -> sensor derecho. | *w* -> velocidad ruedas.

Algoritmo sensor

```
if(Si = circuito)
      Wi = 0;
if(Sd = circuito)
      Wd = 0;
```

Ecuacion de estado

```
[…]
Wi = w;
Wd = w;
```

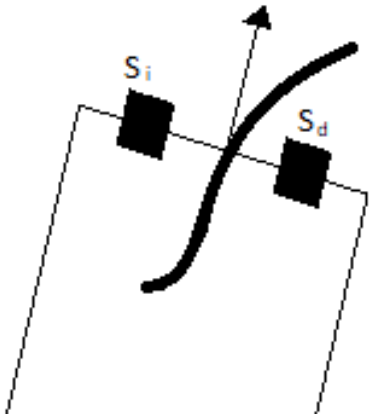


Figura 14. Posición sensores sobre robot.

Esta en una de las posibles implementaciones, otras opciones son: la rueda en vez de frenar realiza el giro hacia atrás $w_i = -w$ o disminuye la velocidad en vez de frenar $w_i = w - \Delta w$

(Descripción)

3. ~~Función objetivo~~ Formulación matemática (del problema de optimización)

La función objetivo de este algoritmo se trata de la simulación del robot realizando el circuito. Para poder implementarla se ha utilizado lo visto en este capítulo, es decir,

se necesitan las ecuaciones (9) del apartado *2.1 Direccionamiento diferencial* para el calculo del movimiento y posición del robot en cada instante, y la simulación de los sensores vista en el apartado *2.2 Navegación autónoma* que determinan cuando va a realizarse un giro.

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.

Esta función da como resultado un valor que es el utilizado para medir la calidad de cada solución. Dicho valor es el tiempo real que el robot tarda en realizar el circuito dado de principio a fin.

$\min_x f(x)$ s.t. $g(x) \leq 0$, $x \in \mathbb{R}$

x: vector de decisión
$f(x)$: función objetivo
$g(x)$: restricciones

En general.

AHORA, en particular:

$x = [d_{sensores}, r_{ruedas}, etc....]$

$f(x)$ ← qué se mide y con qué ecuaciones:

$g(x)$ ← existen límites en las variables? ¿Cuáles son?

Capítulo 3 Descripción algorítmica

falta un apartado de introduc... a la metaheuristica a utilizar.

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.

pseudocódigo

Puesto que se existen soluciones no factibles, es decir, robots que no pueden realizar el circuito, el método constructivo no para de sacar posibles robots hasta que uno de ellos sea factible y a partir de ese se continua con la ejecución del algoritmo.

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 variables para la generación del vecindario y el cambio de la magnitud de permutación de estas.

Explicar mucho más los estructuras de vecindarios

La opción elegida para realizar la búsqueda de vecinos se basa en que cada vecino se genera por las posibles combinaciones de adición y sustracción de una magnitud en tres valores distintos. Dicha magnitud toma los valores: 0.05, 0.1, 0.15 y 0.2.

Esta búsqueda genera una vecindad de 24 vecinos.

Dibujos, etc....

3. Búsquedas locales

Como se ha explicado en el punto *1.3 Estado del arte*, el algoritmo de búsqueda local básica se 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.

Algoritmo!

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 puede 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)***, se basa en el tratamiento con calor de la metalurgia. Esta metaheurística acepta soluciones peores para intentar llegar al óptimo.

Algoritmo!

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.

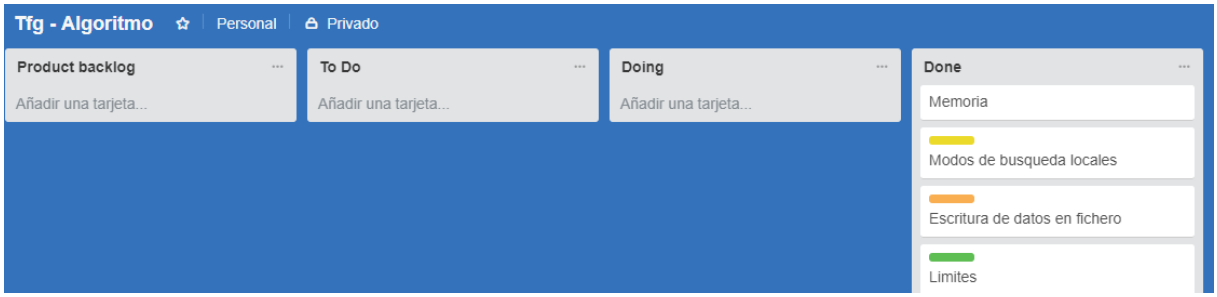


Figura X. Trello utilizado en el desarrollo del proyecto.

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.

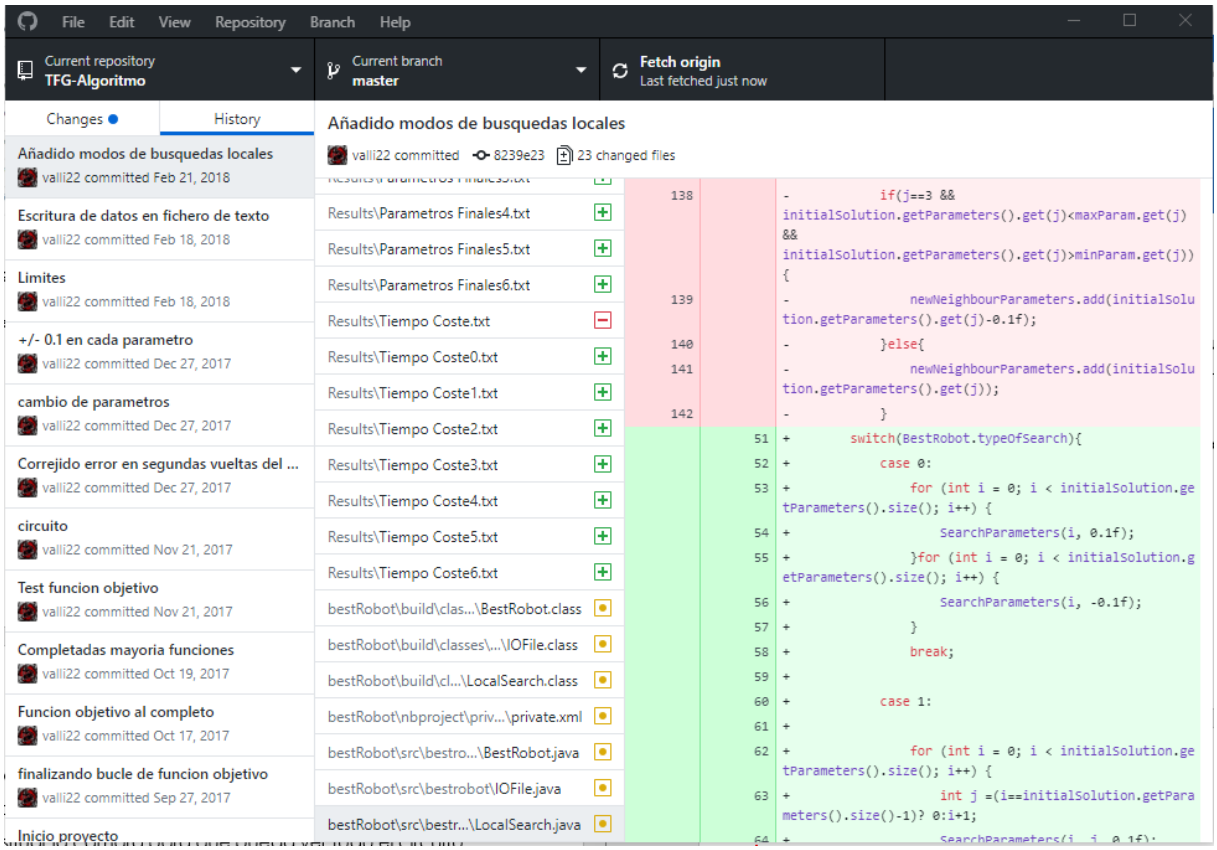


Figura X. Repositorio en GitHub desde la aplicación

de escritorio de Windows.

2. Diseño

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

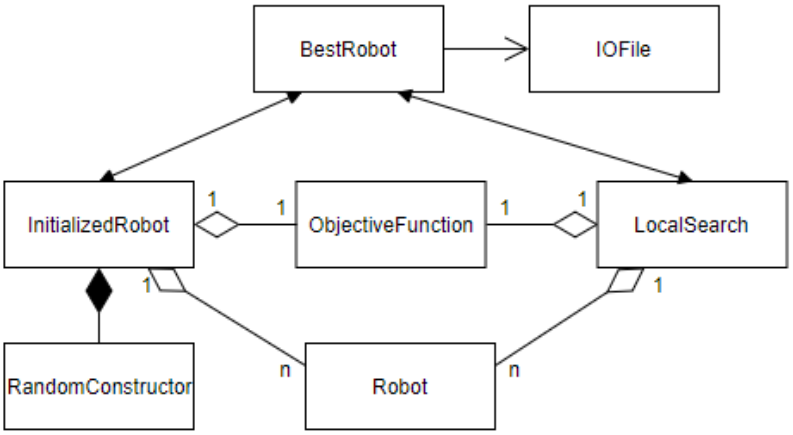


Figura X. UML

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.

Pasar esto al capítulo 2

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 realiza 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 9, calculadas en el apartado *2.1 Direccionamiento*

diferencial. Para saber si se debe mover cada rueda o dejarla parada se realiza la lógica explicada en el punto 2.2 *Navegación autónoma*. Además, se han creado funciones propias que replican el efecto de las funciones translate y rotate de OpenGL para el cálculo de las matrices de posición que son necesarias para saber la posición de los sensores en cada instante.

Esto 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.

Eliminar este apartado y explicar tu implementación con más detalle.
Entrada/salida / sistema de logs.
¿Cómo se ejecuta?
Se podría enlazar con la otra aplicación QT?
y devolver ahí la salida!

Capítulo 5 Resultados

1. Descripción de las instancias

El algoritmo se va a ejecutar sobre 9 instancias que resultan de la combinación de 3 circuitos distintos y 3 velocidades de ruedas diferentes. El primer circuito es un circuito estándar, tamaño medio y con algunas curvas, se puede observar en la figura X.

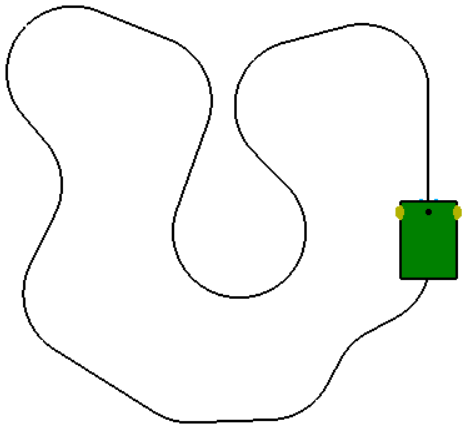


Figura X. Circuito 1 estándar

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.

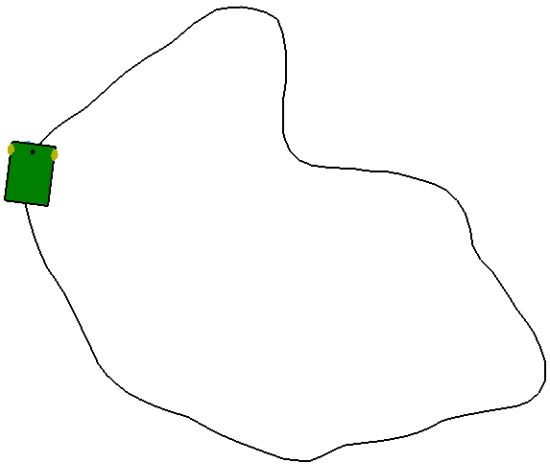


Figura X. Circuito 2 grande sin curvas

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.

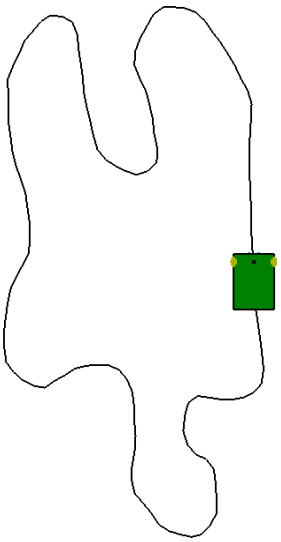


Figura X. Circuito 3 tamaño medio con curvas cerradas

Las velocidades del robot que se utilizan son 2,3 y 4 rps.

CIRCUITO / VEL. RUEDAS	CIRCUITO 1	CIRCUITO 2	CIRCUITO 3
1RPS	Instancia 1	Instancia 4	Instancia 7
2RPS	Instancia 2	Instancia 5	Instancia 8
3RPS	Instancia 3	Instancia 6	Instancia 9

Tabla X.

La ejecución del algoritmo se ha realizado sobre un ordenador con las siguientes especificaciones:

- Procesador: i7-6400 4GHz.
- RAM: 16 GB.
- SO: Windows 10 Home.

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 límites adecuados para su geometría real.

A continuación, se muestra una tabla en la que es expone el resultado de la ejecución del constructivo 10 veces sobre cada instancia:

	INSTANCIA 1	INSTANCIA 2	INSTANCIA 3	INSTANCIA 4	INSTANCIA 5	INSTANCIA 6	INSTANCIA 7	INSTANCIA 8	INSTANCIA 9
1	21,16	24,42	22,66	21,97	32,01	33,77	36,94	27,15	26,99
2	18,83	19,29	36,49	33,44	15,17	12,42	31,49	29,92	24,06
3	30,30	27,14	13,26	47,03	24,98	28,76	46,89	24,48	11,28
4	22,23	19,05	26,02	19,97	15,14	14,30	43,65	31,40	27,04
5	40,05	20,66	35,38	27,50	34,92	11,81	64,19	47,54	16,92
6	43,62	21,67	13,34	25,53	38,46	15,66	25,36	37,61	19,35
7	56,41	35,49	31,41	23,81	21,47	12,38	39,46	21,79	16,61
8	35,72	15,40	33,26	19,45	13,64	29,31	24,12	25,10	29,59
9	26,11	25,64	34,18	32,87	17,85	24,33	43,39	25,79	22,92
10	44,30	28,45	15,39	33,75	18,11	21,68	60,19	35,55	21,80
MEDIA	**33,87**	**23,72**	**26,13**	**28,53**	**23,17**	**20,44**	**41,56**	**30,63**	**21,65**

Tabla X. Valores del constructivo sobre cada instancia.

Como se puede observar, al aumentar la velocidad del robot disminuye el tiempo en el que realiza el circuito. Esto no pasa en el circuito 1, sin embargo, al ser un constructor aleatorio es posible que se generen irregularidades, y esta es una de ellas. Además, se puede observar que el circuito 2 de media es más rápido recorrerlo, sin embargo, el circuito 3 es el más lento.

3. Búsquedas locales

~~Una vez obtenido el mejor constructor (en este caso el único)~~, se elige como se va a realizar la búsqueda de vecindarios. Para ello se ha usado la generación de vecinos vista en el punto *3.3 Búsquedas locales*. A partir de esta se generan 8 posibles búsquedas locales combinando el cambio del parámetro de modificación de vecinos (0.05, 0.1, 0.15, 0.2) y los métodos de búsqueda locales best y first. Se puede observar dichas combinaciones en la tabla X.

	BEST	**FIRST**
0.05	Búsqueda local 1	Búsqueda local 5
0.1	Búsqueda local 2	Búsqueda local 6
0.15	Búsqueda local 3	Búsqueda local 7
0.2	Búsqueda local 4	Búsqueda local 8

Tabla X. Combinaciones de búsquedas locales

A continuación, se ha ejecutado el algoritmo sobre todas las búsquedas locales e instancias. Este proceso se ha repetido 10 veces y se ha sacado el tiempo medio. Se pueden observar los resultados en la tabla X.

	INSTANCIA 1	**INSTANCIA 2**	**INSTANCIA 3**	**INSTANCIA 4**	**INSTANCIA 5**	**INSTANCIA 6**	**INSTANCIA 7**	**INSTANCIA 8**	**INSTANCIA 9**	**MEDIA**
BUSQUEDA LOCAL 1	17,11	14,57	11,89	16,94	13,40	13,50	21,29	17	14,39	**15,56**
BUSQUEDA LOCAL 2	16,86	13,49	11,312	19,18	14,5	11,45	21,15	16,82	13,8	**15,39**
BUSQUEDA LOCAL 3	16,67	13,46	11,29	16,45	13,61	10,80	21,92	16,65	13,96	**14,98**

BUSQUEDA LOCAL 4	16,63	13,57	11,13	16,03	12,74	11,22	20,64	16,42	13,42	**14,65**
BUSQUEDA LOCAL 5	17,39	14,34	11,9	17,79	13,63	15,01	24,35	17,38	18,41	**16,69**
BUSQUEDA LOCAL 6	17,20	13,81	11,58	23,05	16,83	13,20	21,39	17,87	14,38	**16,59**
BUSQUEDA LOCAL 7	17,02	13,61	11,43	18,65	15,33	13,29	20,92	20,04	14,88	**16,13**
BUSQUEDA LOCAL 8	16,81	13,45	11,17	22,35	15,32	12,77	20,87	17,94	14	**16,07**
MEDIA	**16,99**	**13,79**	**11,46**	**18,8**	**14,42**	**12,66**	**21,56**	**17,5**	**14,6**	

Tabla X. Tiempo optimo medio obtenido.

Si nos fijamos en las búsquedas locales, se puede ver una clara diferencia de tiempos entre las búsquedas con best (busq. 1- busq. 4) y first (busq. 5- busq. 8), donde las primeras alcanzan un valor mejor que las segundas. Además de esto también se puede observar como el aumento del valor de modificación de los vecinos implica una mejora en los tiempos de los robots (bq4<bq3<bq2<bq1 y bq8<bq7<bq6<bq5).

En cuanto a las instancias, se puede comprobar como el tiempo se ha mejorado considerablemente con respecto a los obtenidos en la tabla X donde se comprobaban simplemente usando el constructor.

En la tabla X se muestra el tiempo que se ha tardado en conseguir el mejor valor para cada uno de los casos expuestos en la tabla X.

	INSTANCIA 1	**INSTANCIA 2**	**INSTANCIA 3**	**INSTANCIA 4**	**INSTANCIA 5**	**INSTANCIA 6**	**INSTANCIA 7**	**INSTANCIA 8**	**INSTANCIA 9**	**MEDIA**
BUSQUEDA LOCAL 1	429,50	195,33	162,03	105,81	78,44	56,39	103,13	96,76	59,68	**143,01**
BUSQUEDA LOCAL 2	158,39	115,04	96,44	47,96	31,95	36,95	63,48	61,37	52,74	**73,81**
BUSQUEDA LOCAL 3	123,16	120,55	88,13	30,30	37,93	38,26	39,79	30,74	38,74	**64,18**
BUSQUEDA LOCAL 4	105,17	74,4	76,6	53,01	45,37	43,65	34,40	34,81	26,15	**54,84**
BUSQUEDA LOCAL 5	16,37	7	7,21	4,34	3,76	3,64	6,76	7,69	5,72	**6,94**
BUSQUEDA LOCAL 6	9,48	8,35	6,22	2,61	3,22	2,42	5,44	6,59	5,46	**5,53**

BUSQUEDA LOCAL 7	7,51	6,39	4,34	2,30	2,57	2,9	5,02	4,60	4,03	**4,41**
BUSA LOCAL 8	6,55	5,33	3,16	6,22	3,45	4,51	3,59	4,16	3,02	**4,48**
MEDIA	**107,02**	**66,55**	**55,52**	**35,32**	**25,84**	**23,59**	**32,70**	**30,84**	**24,44**	

Tabla X. Tiempo medio que se tarda en conseguir la solución óptima.

En esta tabla se puede observar cómo el tiempo que se tarda en encontrar el mejor valor es mucho menor utilizando first que best. También se puede comprobar como al aumentar el valor de modificación de vecinos se encuentra el mejor valor de manera más rápida.

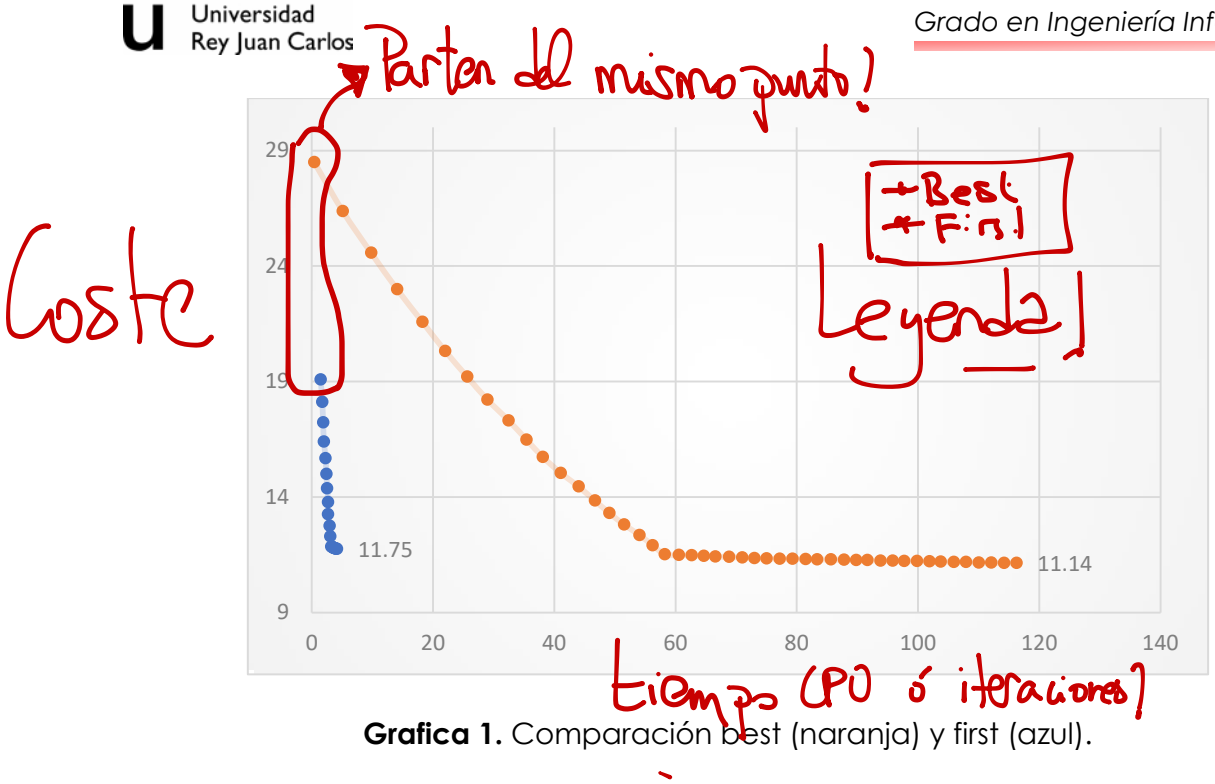
En cuanto a la relación calidad de la solución – tiempo de ejecución, best obtiene resultados un **8%** mejores que first, sin embargo, tarda **25.83** veces más en obtenerlos.

En lo respectivo al valor de modificación de los vecinos, su aumento implica una mejora tanto del valor obtenido como del tiempo de ejecución. En cuanto a la mejora obtenida, por cada aumento del valor en 0.05 se mejora el tiempo del robot entre un **1%-2%**. El porcentaje de mejora en cuanto al tiempo de ejecución se muestra en la tabla X.

	BEST	**FIRST**
0.05-0.1	93%	25%
0.1-0.15	15%	25%
0.15-0.2	17%	-2%

Tabla X. Mejora de tiempo de ejecución del algoritmo.

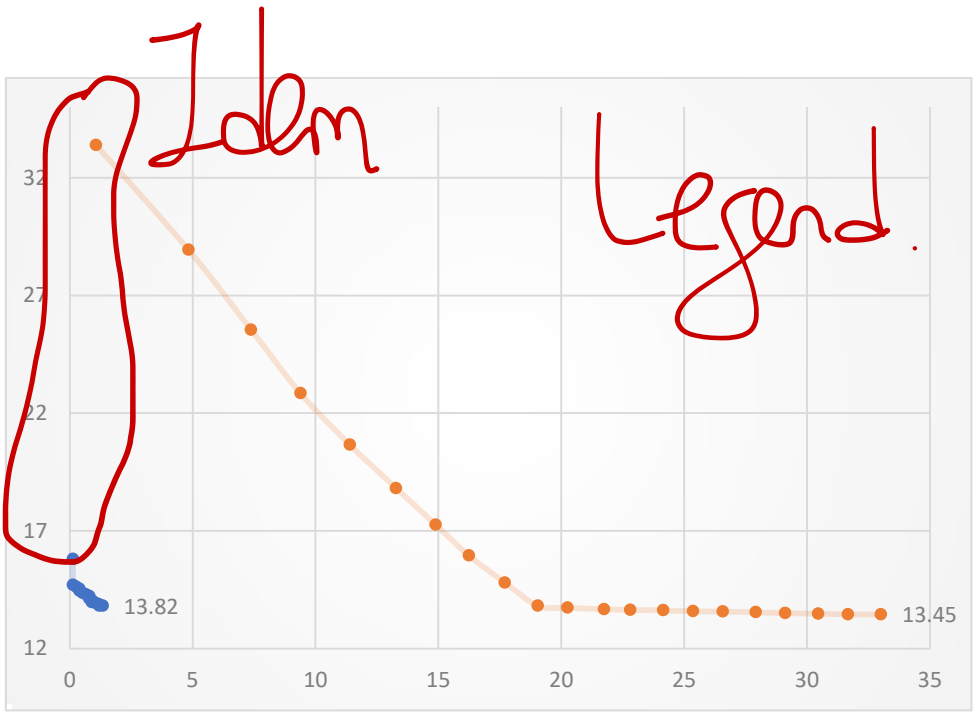
A continuación, se muestra una gráfica en la que se muestra un caso estándar (Búsqueda local 2 y 6 – Instancia 3), con best y first, y los valores obtenidos a lo largo del tiempo de ejecución.



Grafica 1. Comparación best (naranja) y first (azul).

En esta grafica se puede observar como mediante first se encuentra un valor optimo muy rápido, sin embargo, el valor que consigue best es mejor. Se puede comprobar también como best parece que se estanca, pero continúa sacando mejores valores, sin embargo, first se estanca y no consigue mejorar su resultado.

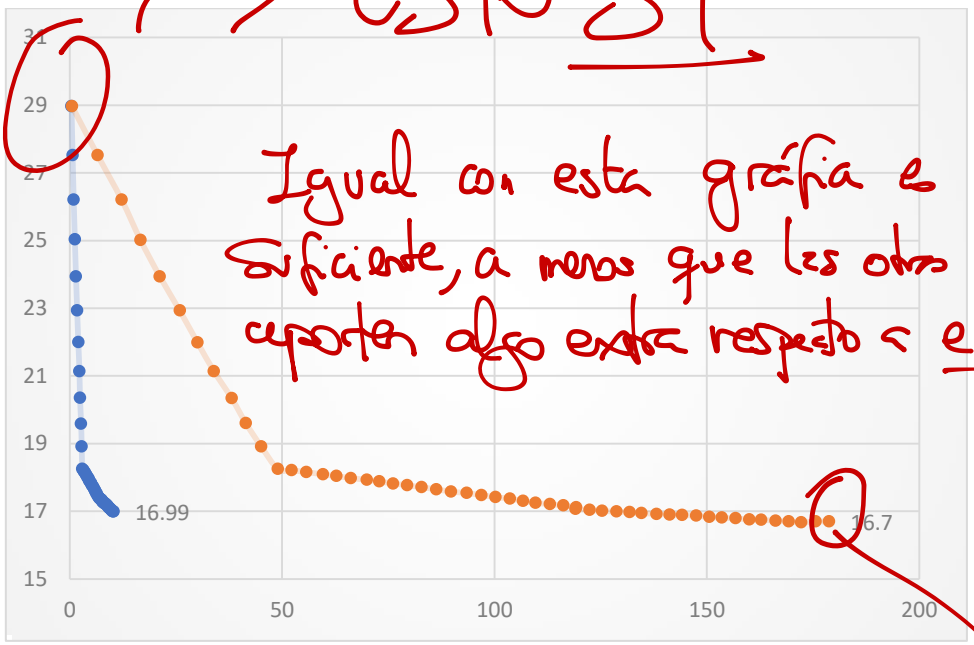
En la gráfica 2 se muestra otro ejemplo del funcionamiento de first y best (Búsqueda local 4 y 8 – Instancia 9).



Grafica 2. Comparación best (naranja) y first (azul).

Al igual que en la gráfica 1, en este caso best ofrece un mejor resultado a cambio de ser mucho más lento. La principal diferencia con el anterior ejemplo se nota en el estancamiento de first, que en este caso si consigue obtener mejores resultados cuando este parece estancarse.

Por último, se muestra una gráfica en la que se ha modificado el constructor para que best y first empiecen desde el mismo robot inicial (Búsqueda local 2 y 6 – Instancia 1).



Grafica 3. Comparación best (naranja) y first (azul) desde mismo robot inicial.

Gracias a esta grafica se puede comprobar lo dicho anteriormente, puesto que, a pesar de partir del mismo robot, ocurren los mismo hechos que en las anteriores, es decir, first obtiene un valor un poco peor pero mucho más rápido que first.

4. Resultados finales

Una vez comprobados los mejores métodos, se comprueban los cambios realizados en los valores geométricos del robot. A continuación, se muestra una tabla con los parámetros iniciales y los finales utilizados en la gráfica 3.

	INICIALES	BEST	FIRST
SEPARACION ENTRE RUEDAS	16	10,99	10,99
RADIO RUEDA	2	3,09	3,09
DISTANCIA ENTRE EJES	3	6,09	6,09
SEPARACION ENTRE SENSORES	4	1,9	4

Tabla X. Cambio en valores geométricos sobre búsqueda local 2 y 6 e instancia 1.

Como se puede observar, para la instancia 1 los mejores cambios son disminuir la separación entre ruedas y la separación entre sensores y aumentar el radio de las ruedas y la distancia entre ejes.

En la tabla X se muestra el mismo robot inicial pero ejecutado sobre la instancia 4 y los parámetros obtenidos.

	INICIALES	BEST	FIRST
SEPARACION ENTRE RUEDAS	16	14,19	15,69
RADIO RUEDA	2	3,09	2,29
DISTANCIA ENTRE EJES	3	3,39	3,29
SEPARACION ENTRE SENSORES	4	4,29	4

Tabla X. Cambio en valores geométricos sobre búsqueda local 2 y 6 e instancia 4.

En este caso la mejora del robot se basa en disminuir la separación entre las ruedas, pero aumentar el resto de los parámetros geométricos.

Por último, se muestra una tabla que indica el porcentaje de mejora del resultado en función de la búsqueda utilizada sobre cada una de las instancias.

	INSTANCIA 1	INSTANCIA 2	INSTANCIA 3	INSTANCIA 4	INSTANCIA 5	INSTANCIA 6	INSTANCIA 7	INSTANCIA 8	INSTANCIA 9	MEDIA
BUSQUEDA LOCAL 1	97%	62%	119%	68%	72%	51%	95%	80%	50%	**77%**
BUSQUEDA LOCAL 2	101%	75%	131%	48%	59%	78%	96%	82%	56%	**80%**
BUSQUEDA LOCAL 3	103%	76%	131%	73%	70%	89%	89%	83%	55%	**85%**
BUSQUEDA LOCAL 4	103%	74%	134%	77%	81%	82%	101%	86%	61%	**88%**
BUSQUEDA LOCAL 5	94%	65%	119%	60%	69%	36%	70%	76%	17%	**67%**
BUSQUEDA LOCAL 6	96%	71%	125%	23%	37%	54%	94%	71%	50%	**69%**
BUSQUEDA LOCAL 7	99%	74%	128%	52%	51%	53%	98%	52%	45%	**72%**
BUSQUEDA LOCAL 8	101%	76%	133%	27%	51%	60%	99%	70%	54%	**74%**
MEDIA	**99%**	**71%**	**127%**	**53%**	**61%**	**62%**	**92%**	**75%**	**48%**	

Tabla X. Porcentaje de mejora respecto a la primera solución.

En la tabla X, se puede comprobar como la instancia que mas se mejora utilizando este algoritmo es la instancia 3 con un **127%** de mejora, sin embargo, la que menos se mejora es la 4 con solo un **53%.**

En cuanto al tipo de búsqueda que mejor resultado proporciona es la búsqueda local 4 con un **88%** de mejora respecto al primer valor y la búsqueda que menos mejora es la 5 con un **67%** de mejora.

Una vez vistos todos los datos se puede llegar a la conclusión de que si lo que se necesita es una mayor precisión en conseguir el óptimo, el mejor método es best, sin embargo, si se busca una relación de mejora-tiempo, first es un método mucho mejor puesto que encuentra una buena solución en un tiempo mucho menor.

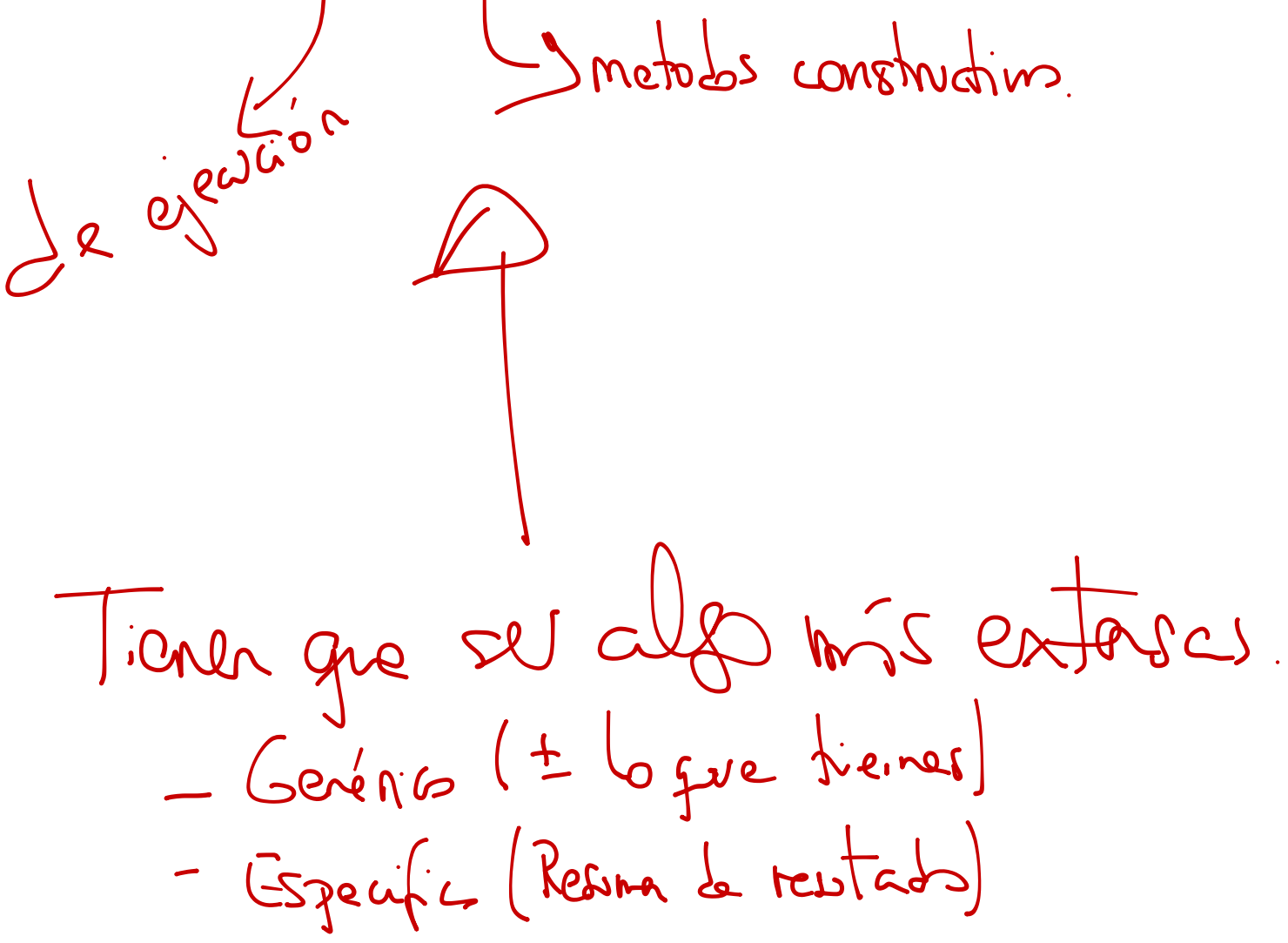
Conclusiones

Gracias a este proyecto he indagado un poco mas en metaheurísticas, mas concretamente en las búsquedas locales y su gran variabilidad para obtener mejoras.

El objetivo principal propuesto ha sido logrado con un resultado de mejora bastante grande.

Las posibles mejoras a realizar entran dentro de la mejora de este esté algoritmo, añadiendo búsquedas globales cuando te encuentras con un óptimo local o pensando diferentes constructores que mejoren el random.

Otra posible mejora es la utilización de otras metaheurísticas para estudiar la posible mejora de los tiempos sobre estas.



Bibliografía

Referencias!