Práctica: Optimización de controlador PID y ruta de transporte

Sandra Gómez Gálvez

Conducción cooperativa, conectada y autónoma
Universidad Politécnica de Madrid
2020/21

10 de enero de 2021

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Esta práctica de optimización de controladores correspondiente al *Tema 3, Optimización de parámetros de control mediante técnicas metaheúristicas*, de la asignatura *Conducción cooperativa, conectada y autónoma*, tiene como objetivo optimizar un controlador PID y una ruta de transporte. Para la realización de la práctica unos fitness son aportados, dos para la optimización del PID (*pid_graph.py* y *pid_graph2.py*) que es el control de un motor, y otro para la optimización de la ruta de transporte (*tsp-2d.py*).

Para el objetivo de optimizar un controlador PID, se tratará de encontrar controladores que tengan un buen equilibrio entre prestaciones y consumo para el motor modelado en cada uno de los fitness aportados para la realización de la práctica.

Para cumplir el objetivo habrá que probar diferentes pesos para We y optimizar en cada caso.

Para el segundo objetivo de la optimización de la ruta de transporte se requiere encontrar la ruta más corta para entregar 100 paquetes en una ciudad. La distancia que se supondrá será la euclídea. Además, se suministran los puntos de entrega de los 100 paquetes. También se requiere contestar a las siguientes cuestiones:

- ¿Encuentra siempre la mejor solución? ¿Por qué?
- ¿Crees que se puede estar seguro de haber encontrado la solución óptima?
- ¿Es útil repetir la optimización varias veces? ¿Por qué?
- Calcula el número de soluciones posibles y el tiempo que tardaría una máquina en encontrar la óptima probando todas las soluciones si generar cada una y comprobar su longitud le llevara 1 microsegundo (en un ordenador actual en python y con multiproceso lleva unos 100 microsegundos)

En los siguientes capítulos se cumplirán con ambos objetivos. En el capítulo 2 se tratará el del controlador PID y en el 3 el de la ruta de transporte.

Para ello, en ambos capítulos, lo que primero se hará será abrir SALGA situándonos en la CMD (esta práctica se ha realizado en Windows) en la ubicación donde se encuentre el programa SALGA y con el comando *py salga.pyc -s* lo abriremos.

Seguidamente, SALGA se mostrará como se puede ver en la figura 1.1.

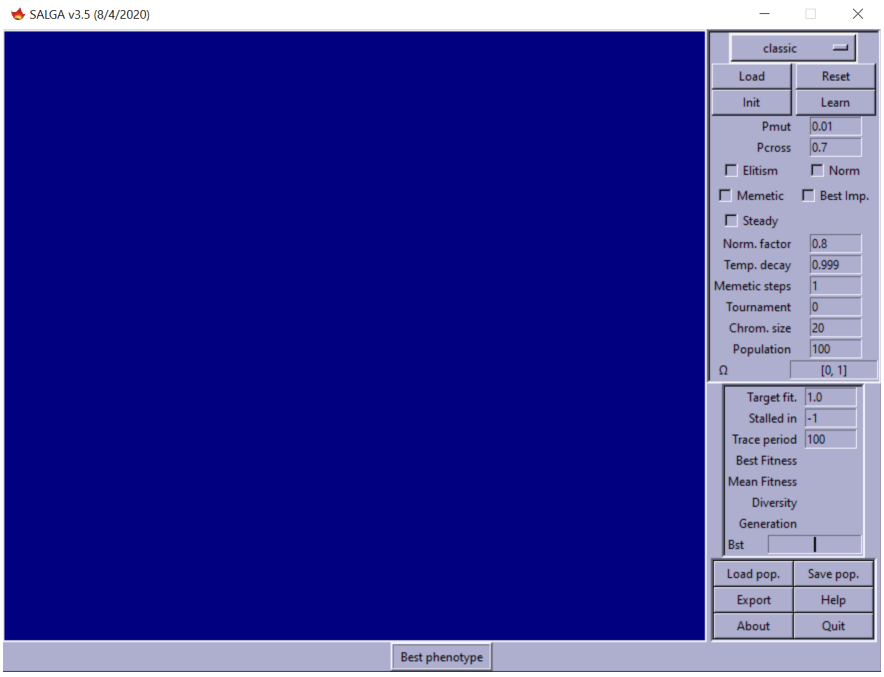

Figura 1.1: SALGA

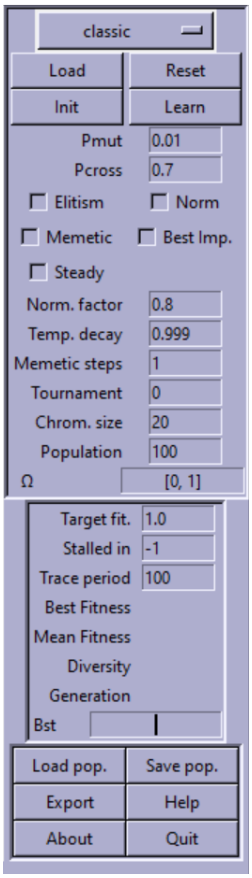

Figura 1.2: Cuadro de instanciación de SALGA

A lo primero que prestaremos atención es al cuadrado de instanciación de datos y configu-

ración, que se puede ver en 1.2. Aquí, se comenzará cargando los correspondientes fitness para cada objetivo con el botón Load. Seguidamente se elegirá el tipo de problema que es: *classic, floating, permutation*... Aunque normalmente este tipo se autoseleccionará al cargar el fitness.

Se continuará instanciando una tasa de probabilidad de permutación (*Pmut*), será importante fijar esta tasa bien. También se podrá instanciar una tasa de probabilidad de cruce o emparejamiento(*Pcross*), aunque se recomienda dejarla en 0.7.

A continuación se podrá seleccionar si se quiere que el algoritmo tenga:

- *Elitism*: El elitismo hace que el mejor individuo de una población se salve para la siguiente población. Produce una convergencia más rápida pero también hace que sea más probable caer en mínimos locales.
- *Normalization*: Exagera la probabilidad de selección de los mejores individuos. En problemas complejos si no se selecciona normalización o torneo (se verá más adelante) será difícil que la población evolucione.
- *Memetic*: Utilizará algoritmos meméticos. Si este se activa tardará más porque tiene una fase de optimización local (favorece la explotación y perjudica la exploración).
- *Best Imp.*: Elige cualquier paso que mejore.
- *Steady*: Activa el steady state. En algún caso puede dar un problema de sincronización si se activa.

Si la normalización ha sido activada, hay que marcar un factor de normalización (*Norm. factor*), en el que cuánto más bajo sea entonces más probabilidad le estaremos dando a los más adaptados, y cuánto más alto más probabilidad a los menos adaptados. Es decir, más bajo más explotación y más alto más exploración. Seguidamente se puede dar un valor a *Temp. decay* que para esta práctica no se tendrá mucho en cuenta. Si en el paso de arriba se ha seleccionado la opción *Memetic*, se puede elegir un número de *Memetic steps*, y cuanto más alto sea más lento irá. A continuación, se puede elegir un valor de *Tournament*. Como se ha explicado en el apartado de normalización si este valor está en 0 y/o la normalización no es marcada, entonces en problemas complejos no funcionará muy bien. En este apartado un valor de 1 no tendría sentido, el valor mínimo es 2 (si fuera 2 sería un torneo de 2 y lo mismo si aumentamos el número), cuanto más alto el torneo los menos adaptados tendrán menos probabilidad y, por tanto, más explotación. Seguidamente se instanciará el tamaño del cromosoma que dependerá del tamaño del problema (3 para el PID y 100 para las rutas). Después, se seleccionará un tamaño de población. 100 será un número aceptable, aunque esto habrá que experimentarlo. Cuanto mayor sea más variabilidad habrá y si es menor el aprendizaje irá más rápido aunque se podrá caer en mínimos locales más fácilmente. Luego se instanciará el alfa que será el alfabeto: los valores mínimos y máximos que pueden tomar los genes, en general se usará un rango de 0 a 100.

Ahora se continuará instanciando los valores del cuadro pequeño inferior. Primero, el *Target fit.*, que será el valor máximo del fitness para parar el fitness en el valor que le asignemos. Hay que tener cuidado con este valor porque podemos parar el fitness sin encontrar una solución aceptable. El siguiente apartado *Stalled in* no se tendrá en cuenta en esta práctica. A continuación, el apartado *Trace period* será el valor que indicará cada cuántas iteraciones pintará los valores. En problemas sencillos un valor de 100 es aceptable, pero en el caso del PID habrá que bajarlo a 1 ya que tendrá que realizar muchos cálculos.

Para comenzar el proceso se pulsará el botón *Learn*.

Capítulo 2

Optimización del PID

En este capítulo se tratará de optimizar un controlador PID buscando un buen equilibrio entre prestaciones y consumo para el motor. El objetivo será maximizar el rendimiento a la vez que se minimiza el consumo del motor.

Como material para realizar la práctica dos fitness *pid_graph.py* y *pid_graph2.py* son entregados y se tratarán en las dos siguientes secciones.

2.0.1. Problema encontrado a la hora de realizar la práctica

En el momento de realizar este apartado de la práctica se ha encontrado un error que no ha permitido actualizar las gráficas mostradas por cada generación de la población.

El error se ha dado por TKinter con Matplotlib y es el siguiente: "ValueError: shape mismatch: objects cannot be broadcast to a single shape".

Este error se ha intentado solucionar de muchas maneras pero no se ha conseguido. No es un problema de configuración o instalación ya que todo está configurado e instalado como debería y es lanzado como se indica en las instrucciones de la práctica (en Windows py salga.pyc -s).

Por tanto, se ha realizado la práctica únicamente con la gráfica de la población inicial (por ejemplo, la de la figura 2.3) , ya que es la única que se muestra al iniciar SALGA, pero esta no se actualiza al pulsar el botón de *Learn.*

2.1. *pid_graph.py*

Se comenzará tratando de optimizar el PID del fitness *pid_graph.py*. Para ello se probarán diversos pesos de energía We. Primero, la estrategia que se seguirá será cargar el fitness, y únicamente se cambiará el parámetro *target fitness* de 1 a 0.94 dejando todos los demás parámetros sin cambiar, es decir, como vienen definidos por el código. Se probarán diferentes We. En principio se pretende probar We = 0.00, We = 0.25, We = 0.50, We = 0.75, We = 1.00, y We = 10.00. Seguidamente, se elegirá el mejor controlador con un buen equilibrio entre prestaciones y consumo. Para más tarde, teniendo en cuenta el mejor, realizar cambios de otros parámetros como mutación, normalización, etc., para buscar mejores resultados.

2.1.1. Configuración inicial

Como lo único que modificaremos al principio es el *target fitness* y este será igual para todos los distintos valores de We, la configuración inicial de los parámetros será la siguiente:

Tipo de problema	Floating
Pmut	0.33333333
Pcross	0.7
Elitims	No
Norm	Sí
Memetic	No
Best Imp.	No
Steady	No
Norm. factor	0.95
Temp. decay	0.999
Memetic steps	1
Tournament	0
Chrom. size	3
Population	100
Ω	[0.0001, 100.0]
Target fit.	0.94
Stalled in	-1
Trace period	1

Tabla 2.1: Estanciación parámetros

Se ha decidido poner al inicio un *Target Fitness* de 0.94 porque, después de realizar pruebas iniciales para observar el funcionamiento de SALGA, se ha observado que para este problema no le es fácil converger al 1 en un tiempo razonable, sin embargo si alcanza el 0.94.

2.1.2. We = 0.00

Se ha comenzado seleccionando un peso de We = 0.00 (figura 2.1) , por tanto no se considerará la energía, solo se tendrá en cuenta el rendimiento del motor.

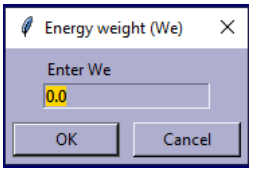

Figura 2.1: We

Tarda en generar la población inicial porque para evaluar cada fitness tiene que realizar muchas operaciones. Los parámetros de antes de comenzar el proceso evolutivo con *Learn* se pueden ver en la figura 2.2 y son los mismos que se han visto en la subsección 2.1.1. También en la figura 2.2 se observa como en esta población inicial de 100 individuos hay un individuo que

tiene un fitness de 0.022409 considerado como el mejor, que el fitness medio es de 0.017884, y que la diversidad es aceptable con un valor de 0.294737.

Si observamos la gráfica 2.3 y los valores de la figura 2.4, se puede ver como este PID tiene unos valores de Kp = 98.06, Ki= 9.20 y Kd=5.99, que hay un error de 44.177, y como el error en test es 41.732. Los errores son altísimos. También, tiene un valor de control 52.165 y una energía de 436.986 kws, lo que es una barbaridad. Que tenga un gasto de energía tan alto es debido a que acelera mucho y acaba sobrepasando los niveles lo que hace que se frene muy rápido y así en bucle, lo que produce un gasto de energía enorme.

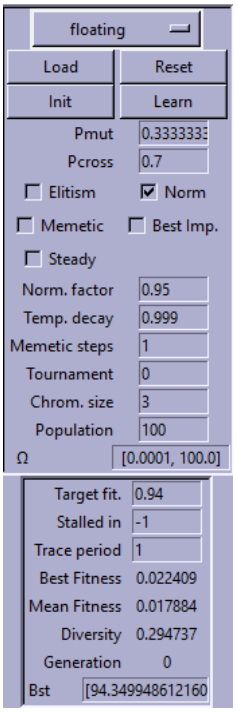

Figura 2.2: Instanciación

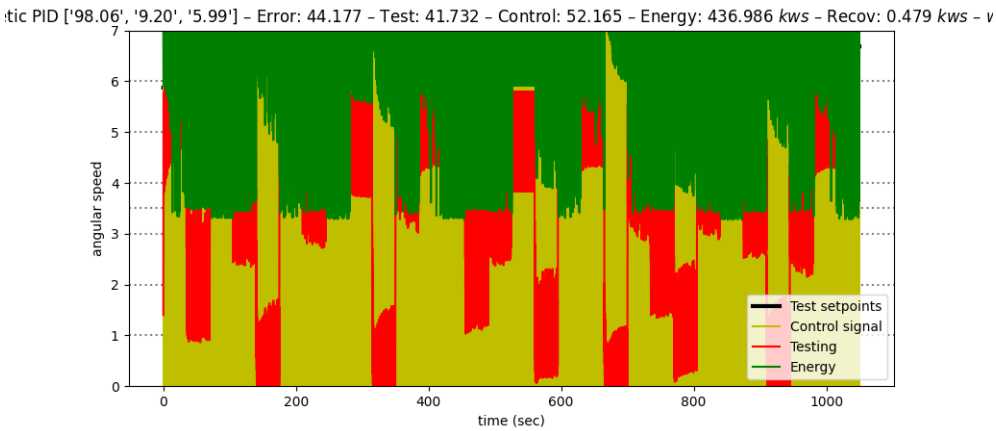

Figura 2.3: Gráfica inicial

Figura 2.4: Valores iniciales

Al comenzar la evolución el fitness pasa rápidamente de un *fitness* de 0.02 a valores cercanos a 0.80 ya en la tercera generación, pero no alcanza el *Target Fitness* (figura 2.5) hasta la décima generación. Esto es debido a que la diversidad disminuye considerablemente a valores cercanos a 0.2 en la tercera generación y de 0.30 a 0.17 en la décima. Que la diversidad disminuya tanto propicia la explotación y por ende, el algoritmo explota las zonas cercanas que proporcionan un *fitness* de 0.8, y por eso tarda tantas generaciones en llegar al 0.94. Sin embargo, hasta la generación 3ª, el algoritmo es claramente explorativo.

Estos valores que se han comentado sobre la 10ª generación se pueden observar en las figuras 2.5 y 2.6.

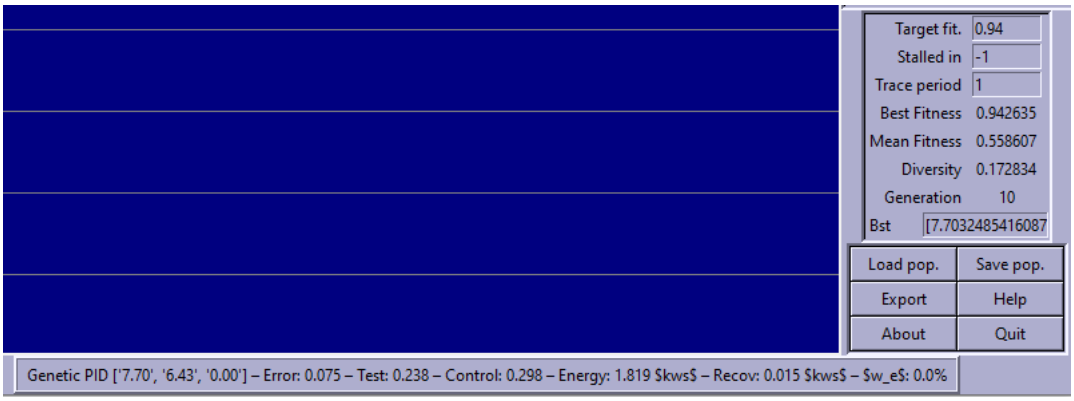

Figura 2.5: Valores al alcanzar el *Target Fitness* - Generación 10

```
gen:    10; meanf: 0.559; bestf: 0.94263; div: 0.173; time: 14.89
Temperature: 0.9910
1.8191690404144396
NPars: 3, Levels: 1, Error: 0.075, Test: 0.238, Control: 0.298
[7.703248541608707, 6.43308886931235, 0.0001]
```

Figura 2.6: Valores completos al alcanzar el *Target Fitness* - Generación 10

Los valores de PID, errores, control y energía, en la generación inicial y en la generación que contiene un individuo que alcanza el *Target Fitness*, se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	Generación 10
Kp	98.06	7.70
Ki	9.20	6.43
Kd	5.99	0.0001
Error	44.177	0.075
E. Test	41.732	0.238
Control	52.165	0.298
Energy	436.986kws	1.819kws
Recov.	0.479kws	0.015 kws
We	0 %	0 %

Tabla 2.2: Resultados We = 0.00

Por tanto, podemos observar como los resultados en la generación 10ª son mucho mejores que los resultados en la generación inicial. Los datos nos proporcionan que la velocidad será más lenta al haber disminuido la constante de acción proporcional Kp pero a pesar de ello, el sistema será más estable y no tendremos tanta inestabilidad. La disminución (aunque en este caso ha sido leve) de la constante de acción integral Ki hará que el sistema sea menos estable pero que la velocidad aumente. Y finalmente, en cuanto a la constante de control derivativa Kd, su disminución hace que la estabilidad sea menor pero que la velocidad aumente. Por tanto, podríamos decir que en la generación 10 tanto Kp, Ki y Kd se complementan y hacen que el sistema esté estable y tenga una velocidad aceptable.

En cuanto a la energía, se ha notado una bajada enorme desde la generación inicial hasta la generación 10, esto es debido a la estabilidad del sistema y de la velocidad.

Pese a no poder ver las gráficas por el problema de versiones con Tkinter, matplotlib, y Windows, los datos nos hacen ver que no hay un gran overshoot. Además, la siguiente generación 11ª que se muestra antes de finalizar el algoritmo, muestra datos bastante similares, lo que significa que el error estacionario es bajo.

Por tanto, hemos obtenido un buen rendimiento pero no hemos tenido en cuenta la energía, que será nuestro objetivo en las siguientes subsecciones.

2.1.3. We = 0.25

Como se han tenido valores aceptables sin tener en cuenta la energía, vamos a continuar aumentando el We poco a poco, por ello, se selecciona un We = 0.25 (figura 2.7).

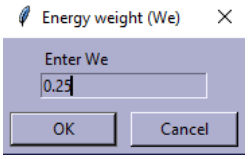

Figura 2.7: We

Al igual que con el We = 0, tarda en generar la población inicial por la cantidad de operaciones que tiene que realizar para evaluar cada fitness. Los parámetros siguen sin cambiar y se pueden ver en la figura 2.8 o en la subsección 2.1.1. En cuanto a la población inicial de 100 individuos que se puede ver en la figura 2.8, el *fitness* medio es de 0.017 y hay un individuo

con el mejor *fitness* con un valor de 0.02. Además la población tiene una diversidad de 0.28, un valor también aceptable parecido a la población inicial del We = 0.

La gráfica inicial 2.9 y la figura 2.10 nos muestran los valores del PID de esta población inicial, donde obtenemos un Kp = 46.00, Ki = 24.48, y Kd = 0.00, además hay un error de 44.712 y un error de test de 42.244. De nuevo, los errores son altísimos y habrá que disminuirlos. El control tiene un valor de 50.972 y la energía un valor de 289.472kws, que aunque es menor que con el We = 0 (436.986kws), sigue siendo un valor muy alto que no se puede asumir. De nuevo, este valor tan alto de energía es debido a las aceleraciones y deceleraciones tan rápidas que sufre, creando una gran inestabilidad. Por tanto, se buscará una estabilidad en el sistema para poder reducir la energía.

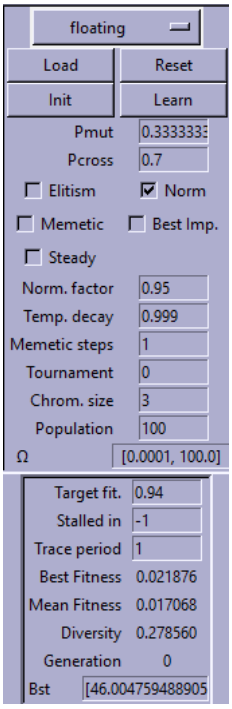

Figura 2.8: Instanciación

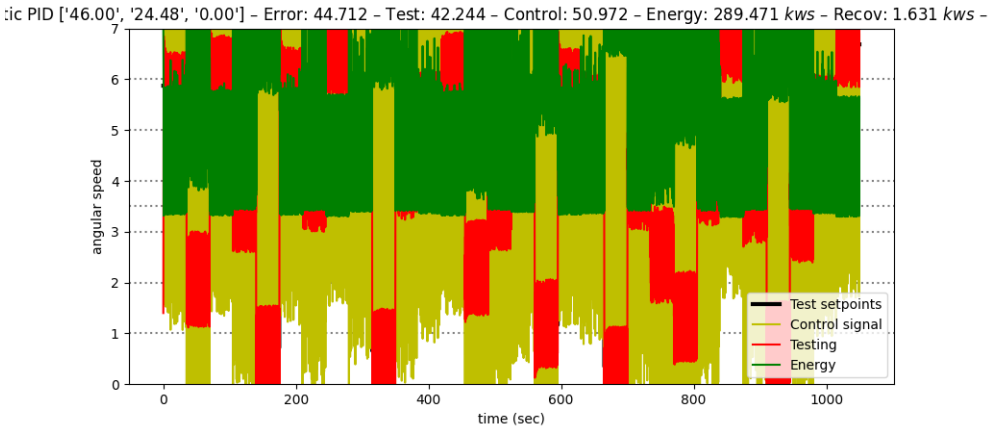

Figura 2.9: Gráfica inicial

Figura 2.10: Valores iniciales

Los resultados obtenidos en cuanto al fitness son bastante parecidos a los resultados del We = 0, ya que al comenzar la evolución, al igual que antes, el fitness pasa rápidamente de un *fitness* de 0.018 a valores cercanos a 0.80 en la tercera generación pero no alcanza el *Target Fitness* hasta la generación 13ª (figura 2.12). Al igual que antes, esto se debe a la diversidad, ya que disminuye un promedio de 0.10, y menos diversidad implica más explotación y menos exploración. Prácticamente el algoritmo se vuelve explotativo desde la tercera generación hasta el final. Cuando cambiemos los parámetros se tratará que el algoritmo no tenga tanta explotación tan pronto para poder converger más rápido al óptimo.

Estos valores que se han comentado sobre la 10ª y la 13ª generación se pueden observar en las figuras 2.11 y 2.12.

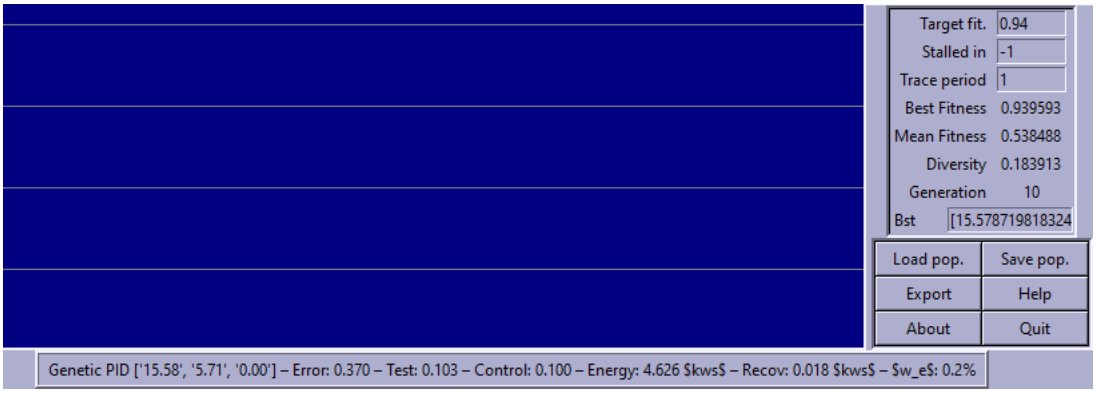

Figura 2.11: Valores al alcanzar la Generación 10

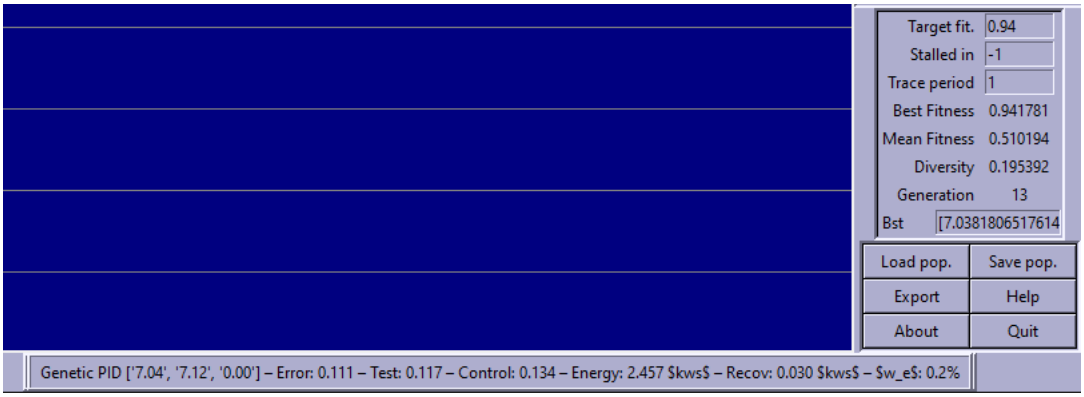

Figura 2.12: Valores al alcanzar el *Target Fitness* - Generación 13

En cuanto a los valores de PID, errores, control y energía, vamos a verlos en la generación inicial (figura 2.10), en la generación 10ª (figura 2.11), y en la generación 13ª que contiene un individuo que alcanza el *Target Fitness*. Estos valores se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	Generación 10	Generación 13 (alcanza el *Target Fitness*)
Kp	46.00	15.58	7.04
Ki	24.48	5.71	7.12
Kd	0.00	0.00	0.00
Error	44.712	0.370	0.111
E. Test	42.244	0.103	0.117
Control	50.972	0.100	0.134
Energy	289.471kws	4.626kws	2.457kws
Recov.	1.631kws	0.018kws	0.030kws
We	0.2 %	0.2 %	0.2 %

Tabla 2.3: Resultados We = 0.25

De la generación inicial a la 13ª pasando por la 10ª la constante de acción proporcional Kp no ha dejado de disminuir. Esta disminución propociona mayor lentitud en la velocidad pero mucha más estabilidad. Veamos si las constantes Ki y Kd aumentan esta disminución de velocidad para hacer un buen sistema. De la generación inicial a la generación 10ª se puede observar como la constante de acción integral Ki disminuye considerablemente, esto hace que el sistema sea menos estable pero que la velocidad aumente. Por tanto, Ki y Kp se complementan y hacen un sistema estable en la generación 10ª. En la generación 13ª el Ki aumenta levemente, esto hace que la estabilidad aumente un poco más y que la velocidad disminuya, aunque agrega un poco de error estacionario. En cuanto a la constante de control derivativa Kd ni aumenta ni disminuye considerablemente, por tanto, el sistema se equilibra mayormente con Kp y Ki.

Los errores desde la generación inicial hasta la generación 13ª no hacen más que disminuir, y esa leve disminución de la Kp y aumento de la Ki de la generación 10ª a la generación 13ª hace que el error disminuya y que además también se note en la bajada de energía.

Por tanto, en cuanto a la energía, consigue reducirse desde ese nivel altísimo de 289.471kws de la generación inicial (dado por las aceleraciones y frenazos rápidos) a 2.475kws, que aunque es más alto que con We = 0, es bastante aceptable. Esto indica que el sistema y la velocidad es estable.

Al igual que antes, pese a no poder ver las gráficas por el problema de versiones con Tkinter, matplotlib, y Windows, los datos nos hacen ver que no hay un gran overshoot en esta última generación. Y la siguiente generación 14ª que se muestra antes de finalizar el algoritmo, muestra datos bastante similares, lo que significa que el error estacionario es bajo. Esto también se debe a la falta de exploración del algoritmo.

Por tanto, para un We = 0.25 obtenemos un controlador con un buen compromiso entre energía y rendimiento, encontramos un controlador equilibrado aunque nos gustaría poder reducir aún más la energía. Para reducirlo, probaremos aumentando la constante de energía We.

2.1.4. We = 0.50

Para intentar reducir la energía, se probará aumentando la constante de energía We a 0.50 (figura 2.13) y se verá si se sigue consiguiendo esa estabilidad.

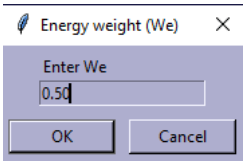

Figura 2.13: We

Como ocurrirá todo el rato para este problema sea el We que sea, la población inicial tardará en generarse por la cantidad de operaciones que tiene que realizar para evaluar el *fitness*. Los parámetros tampoco cambian y se pueden ver en la figura 2.14 o en la subsección 2.1.1.

La población estará compuesta de 100 individuos y en la población inicial (figura 2.14) se encuentra un *fitness* medio de 0.016 y el individuo con el mejor *fitness* de la población tiene un valor de 0.019. En cuanto a la diversidad, encontramos en la población inicial una diversidad de 0.26.

Los valores del PID de la población inicial los podemos observar en la gráfica 2.15 y en la imágen 2.16. Se puede ver que el Kp tiene un valor de 59.00, el Ki = 51.64, y el Kd = 0.92. También hay un error de 49.115 y un error de test de 45.805. Valores de error, al igual que con los otros We en la población inicial, muy muy altos. En cuanto al control, tiene un valor de 52.252. La energía es 392.251kws, un valor que no se puede asumir, y de la misma forma que con los anteriores We, se debe a la inestabilidad que produce que se acelere y se frene muy rápidamente produciendo grandes oscilaciones. Por tanto, se buscará reducir la energía lo que significará que hay una estabilidad en el sistema.

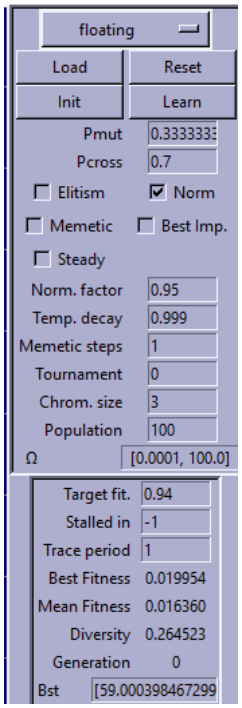

Figura 2.14: Instanciación

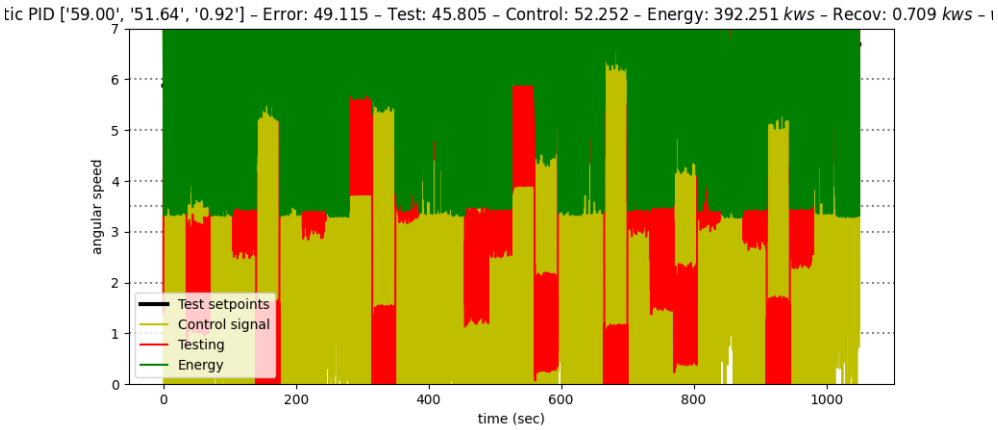

Figura 2.15: Gráfica inicial

Figura 2.16: Valores iniciales

Con We = 0.5 se obtienen mejores resultados más rápidamente. En la tercera generación ya obtiene individuos con un mejor *fitness* de 0.9 y es en la 6ª generación cuando uno de sus individuos alcanza el *Target Fitness*. Lo alcanza mucho más rápido (6ª generación contra 10ª con We=0 y 13ª con We=0.25) pero el *fitness* medio de la población es ligeramente inferior en comparación a los de los otros We. Además, la diversidad también es pequeña lo que hace que el *fitness* medio varíe poco. Como se ha comentado en el apartado del We = 0.25, más adelante se intentará hacer un algoritmo más explorativo y menos explotativo para que haya más variedad en los individuos y el *fitness* medio mejore, todo ello tratando de no perder la convergencia rápida hacia el óptimo global.

Estos valores que se han comentado sobre la 6ª generación se pueden observar en las figuras 2.17 y 2.18.

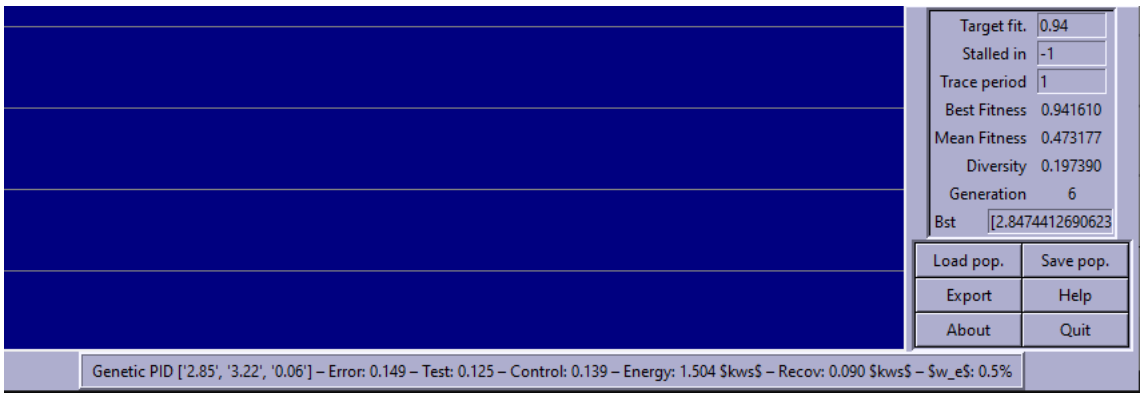

Figura 2.17: Valores al alcanzar el *Target Fitness* - Generación 6

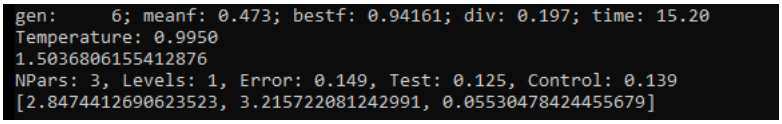

Figura 2.18: Valores completos al alcanzar el *Target Fitness* - Generación 6

En cuanto a los valores de PID, errores, control y energía, vamos a verlos en la generación inicial (figura 2.16) y en la generación 6ª (figura 2.17) que contiene un individuo que alcanza el *Target Fitness*. Estos valores se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	**Generación 6**
Kp	59.00	2.85
Ki	51.64	3.22
Kd	0.92	0.06
Error	49.115	0.149
E. Test	45.805	0.125
Control	52.252	0.139
Energy	392.251kws	1.504kws
Recov.	0.709kws	0.090kws
We	0.5 %	0.5 %

Tabla 2.4: Resultados We = 0.50

Vamos a comenzar analizando los valores del PID: Kp,Ki y Kd. En cuanto a la constante de acción proporcional Kp, ha disminuido muy rápidamente (solo 6 generaciones) lo que hace que la velocidad sea más lenta que en la generación inicial pero mucho más estable. La constante de acción integral Ki disminuye un poco menos que la Kp, esto hace que el sistema sea menos estable pero que la velocidad aumente. Lo único que para estabilizar el sistema ahora Kd tendrá que disminuir. Observando la tabla, se puede ver como afirmativamente la constante de control derivativa Kd disminuye lo que hace que complemente a Ki haciendo que la estabilidad disminuya y la velocidad aumente. Así Kp, Ki y Kd se encuentran en equilibrio. Obteniéndose un buen rendimiento en el motor.

Si observamos los errores, ambos bajan a unos niveles muy aceptables.

Al igual que el control, la energía baja justo a donde apuntábamos nuestro objetivo. Esto es gracias a que la velocidad, frenazos y la aceleración es estable. Además, si observamos las energías de We = 0 (1.819kws gen10ª) y We = 0.25 (2.547kws gen13ª), el valor obtenido para We = 0.5 (1.504kws gen6ª) es el valor más bajo y además es el que más rápido se ha obtenido.

Para finalizar, observamos lo mismo que en los anteriores We, pese a no poder ver las gráficas por el problema de versiones con Tkinter, matplotlib, y Windows, los datos nos hacen ver que no hay un gran overshoot en esta última generación. Y la siguiente generación 7ª que se muestra antes de finalizar el algoritmo, muestra datos bastante similares, lo que significa que el error estacionario es bajo. Esto también se debe a la falta de exploración del algoritmo.

Tenemos que concluir, que de los 3 We que hemos probado, por ahora este es con el que mejor controlador aporta con un buen compromiso energía/rendimiento. Por tanto, We = 0.5 es el mejor peso de energía actual ya que tenemos un nivel muy bajo de energía, tenemos en cuenta un We y el PID es estable, además de que se alcanza muy rápidamente el objetivo.

Aún así vamos a continuar aumentando la constante de energía We por si obtenemos mejores resultados.

2.1.5. We = 0.75

Para ver si se pueden obtener mejores resultados se aumenta la constante de energía We a 0.75 (figura 2.19).

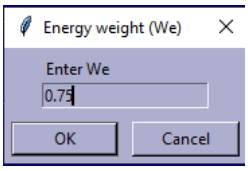

Figura 2.19: We

De igual modo a los otros We, la población inicial tarda en generarse por la cantidad de operaciones a realizar para evaluar el fitness. En cuanto a los parámetros, estos tampoco cambian y se pueden ver en la figura 2.20 o en la subsección 2.1.1.

Si analizamos la población inicial, observamos que está compuesta de 100 individuos y que tiene un *fitness* medio de 0.016 y tiene un individuo con el mejor *fitness* de la población con un valor de 0.019. Además, tiene una diversidad aceptable cercana a 0.3, lo que hace que la siguiente generación tenga mejores valores al facilitar la exploración.

Sus valores con respecto al PID se pueden observar en la gráfica 2.21 y en la figura 2.22. Aquí se puede observar un Kp = 95.98, Ki = 48.35 y Kd = 8.76. También un error de 51.593 y un error de test 48.065. Valores de error, una vez más, muy altos. El control tiene un valor de 51.693. Y la energía esta vez se encuentra en un valor altísimo de 439.613kws. Por tanto, tiene una fluctuaciones que no podemos asumir y hay que reducir. Para ello necesitaremos estabilizar el sistema teniendo en cuenta Kp, Ki, Kd y We.

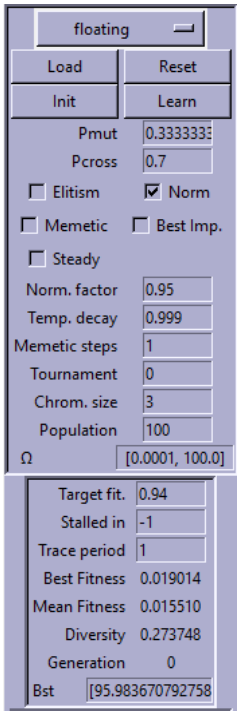

Figura 2.20: Instanciación

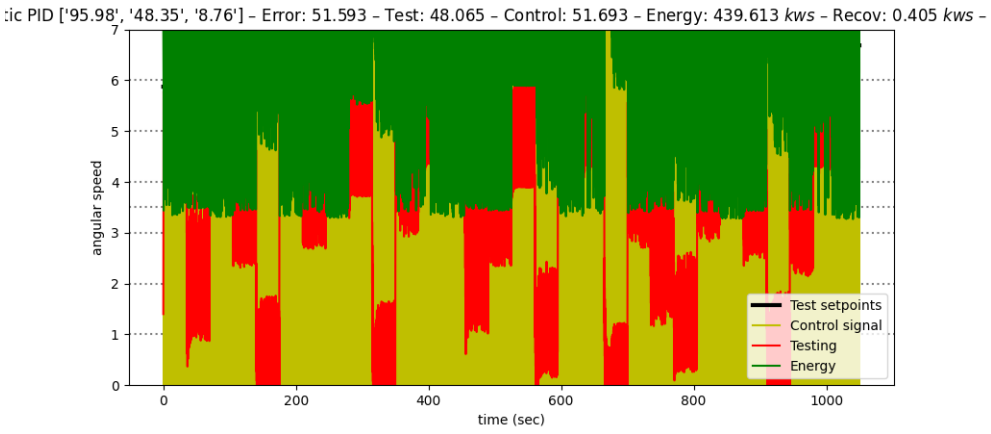

Figura 2.21: Gráfica inicial

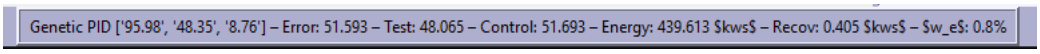

Figura 2.22: Valores iniciales

Se deja correr el algoritmo durante más de 20 generaciones pero finalmente se para manualmente por cuestión de tiempo (eficiencia) y se observa como ningún individuo de la población de ninguna generación alcanza el *Target Fitness*.

Al igual que otras veces en la generación 3ª ya había alcanzado el 0.8 de fitness pero el algoritmo ha encontrado dificultades a la hora de seguir convergiendo hacia el óptimo rápidamente.

Si observamos la generación 10ª el individuo con mejor *fitness* tiene un valor de 0.91 y la población una media de 0.53, pero la diversidad de la población es muy baja. Por tanto, le costará a la población explorar rápidamente.

Por otro lado, si observamos la generación 20ª, el inviduo con mejor *fitness* de la población ya está cerca del *Target Fitness*, pero tenemos que tener en cuenta que es la generación 20ª y ha sido demasiado lenta la convergencia. Además, en la siguientes generaciones sigue sin alcanzar el *Target*. También en esta generación, el *fitness* medio ha variado muy poco en comparación a la generación 10ª y la diversidad sigue baja.

Estos valores que se han comentado sobre la 10ª y la 20ª generación se pueden observar en las figuras 2.23 y 2.24.

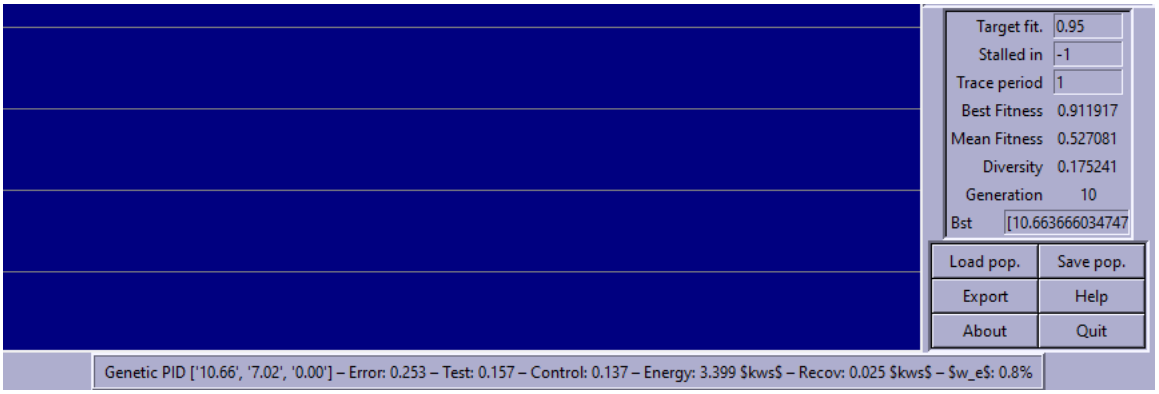

Figura 2.23: Valores al alcanzar la Generación 10

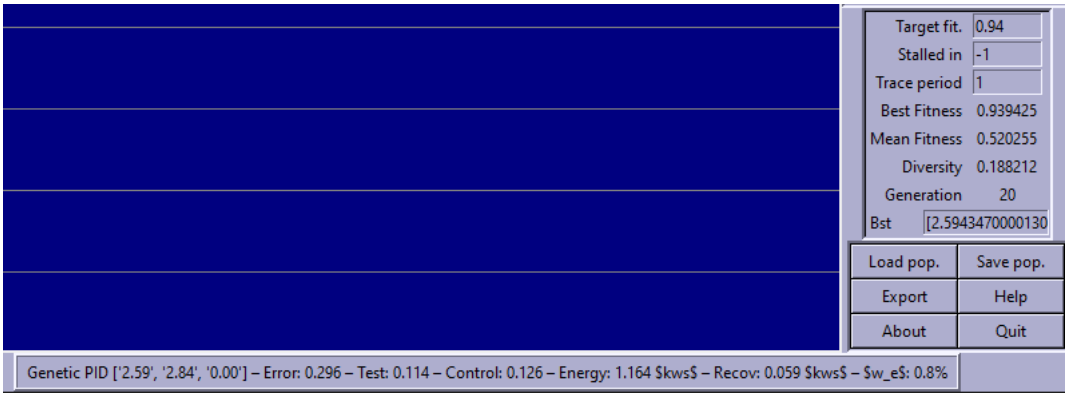

Figura 2.24: Valores al alcanzar la Generación 20

Es el turno de analizar los valores de PID, errores, control y energía, vamos a verlos en la generación inicial (figura 2.22), en la generación 10ª (figura 2.23) y en la generación 20ª (figura 2.24). Observamos como en la generación 20ª hay un individuo que está a punto de alcanzar el *Target Fitness* pero no se alcanza y además se considera que tantas generaciones son demasiadas para este problema. Estos valores se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	Generación 10	Generación 20
Kp	95.98	10.66	2.59
Ki	48.35	7.02	2.84
Kd	8.76	0.00	0.00
Error	51.593	0.253	0.296
E. Test	48.065	0.157	0.114
Control	51.693	0.137	0.126
Energy	439.613kws	3.399kws	1.164kws
Recov.	0.405kws	0.025kws	0.059kws
We	0.8 %	0.8 %	0.8 %

Tabla 2.5: Resultados We = 0.75

Si analizamos los valores del PID (Kp,Ki y Kd) observamos que la constante de acción proporcional Kp disminuye en cuanto aumentamos de generación lo que hace que la velocidad sea más lenta que en la generación inicial pero muchísimo más estable (generación 20ª). Al igual que en el We = 0.5, la constante de acción integral Ki disminuye un poco menos que la Kp, por tanto el sistema es menos estable y la velocidad aumenta. Para estabilizar el sistema la constante de control derivativa Kd disminuye lo que hace que complemente a Ki haciendo que la estabilidad disminuya y la velocidad aumente. Así, igual que en con el anterior We, Kp, Ki y Kd se encuentran en equilibrio. Obteniéndose un buen rendimiento en el motor.

En cuanto a los errores, ambos bajan a niveles aceptables aunque no hay mucha diferencia entre la generación 10ª y 20ª, esto es debido a que no hay mucha variabilidad en los *fitness* de ambas generaciones ya que en estas generaciones se está dando explotación en vez de exploración. Lo mismo ocurre con el control, que baja mucho pero entre la generación 10ª y 20ª no hay mucha diferencia.

Si se observa la energía obtenemos el valor más bajo hasta ahora alcanzado:

- We = 0 => 1.819kws gen10ª.
- We = 0.25 => 2.547kws gen13ª
- We = 0.50 => 1.504kws gen6ª
- We = 0.75 => 1.164kws gen20ª

Pero pese a que es el más bajo, se da en la generación mayor. Así que se considera que es mucho mejor seleccionar un valor de We = 0.50 en vez de 0.75 por la eficiencia en tiempo y además porque con un We = 0.50 la población tiene individuos que son capaces de alcanzar el *Target Fitness*.

Para finalizar, observamos lo mismo que en los anteriores We, pese a no poder ver las gráficas por el problema de versiones con Tkinter, matplotlib, y Windows, los datos nos hacen ver que no hay un gran overshoot en esta generación 20ª ni en la generación 10ª. Además, si observamos la tendencia, en las siguientes generaciones se muestran datos bastante similares, lo que significa que el error estacionario es bajo. Esto también se debe a la falta de exploración del algoritmo.

Para concluir, con We = 0.75 obtenemos un controlador con el mejor valor de energía obtenido hasta ahora, pero ningún individuo de la población alcanza el *Target Fitness*. Por tanto,

se concluye que por ahora el mejor controlador que tenemos es el que nos proporciona el We con un valor 0.5 ya que es el que mejor compromiso energía/rendimiento mantiene. Como se tenía previsto se probará con un We = 1 y We = 10, aunque seguramente We = 1 empeore la situación siguiendo la tendencia de We = 0.75, si esto ocurre We = 10 no será probado ya que seguramente continue empeorando.

2.1.6. We = 1.00

Se quiere observar si aumentando más el We siguen empeorando los resultados y así confirmar que el mejor controlador obtenido se da con un We = 0.50. Por tanto, se aumenta la constante de energía We a 1.00 (figura 2.25).

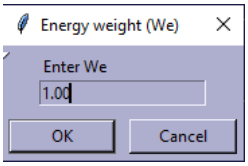

Figura 2.25: We

Igual a los otros We, la población inicial tarda en generarse por la cantidad de operaciones a realizar para evaluar el fitness. En cuanto a los parámetros, estos tampoco cambian y se pueden ver en la figura 2.26 o en la subsección 2.1.1.

Comenzaremos analizando la población inicial compuesta de 100 individuos y con un *fitness* medio de 0.015 y que contiene un individuo con el mejor *fitness* de la población con un valor de 0.018. La siguiente generación tendrá un mejor *fitness* ya que tiene un valor alto de diversidad cercano a 0.3.

Los valores con respecto al PID se pueden observar en la gráfica 2.27 y en la figura 2.28. Aquí se puede observar un Kp = 92.95, Ki = 33.83 y Kd = 10.54, un error de 53.901 y un error de test 51.411. Estos errores habrá que disminuirlos ya que son demasiado altos. El control tiene un valor de 53.119, y la energía al igual que con el We = 0.75 tiene un valor demasiado alto de 437.603kws. Igual que en los otros casos, este gasto de energía es demasiado grande y se debe a las fluctuaciones de velocidad, aceleración y freno muy rápidos. Es curioso, porque se puede observar, que esta vez la población inicial tiene todos sus valores muy parecidos a los valores de la población inicial del We = 0.75, por tanto, se pueden llegar a esperar resultados similares.

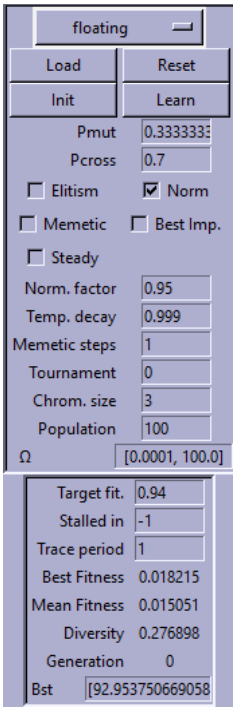

Figura 2.26: Instanciación

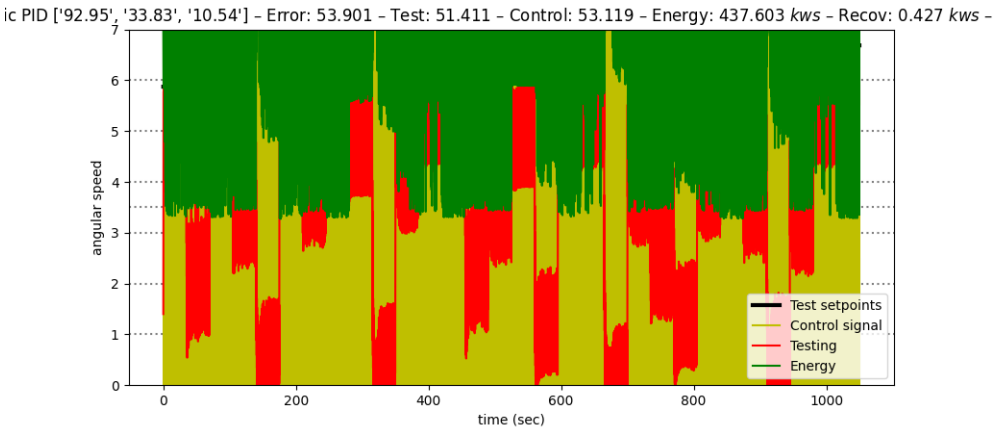

Figura 2.27: Gráfica inicial

Figura 2.28: Valores iniciales

Al igual que con el We = 0.75, el algoritmo alcanza más de 20 generaciones pero finalmente se para manualmente por cuestión de tiempo (eficiencia) y se observa como ningún individuo de la población de ninguna generación alcanza el *Target Fitness*.

Del mismo modo, en la generación 3ª ya había alcanzado el 0.8 de *fitness*, pero al igual que en el We = 0.75 no ha conseguido salir de esa zona de mínimo local en el transcurso de las generaciones debido a la falta de diversidad.

En cuanto a los *fitness* de las generaciones, en la generación 10ª el individuo con mejor *fitness* tiene un valor de 0.92 y el de la 20ª un valor de 0.93, cercano al *Target Fitness*, pero ya son demasiadas generaciones para este problema. Además, el *fitness* medio de ambas generaciones se encuentra cercano a 0.48 y tienen una diversidad baja.

Estos valores que se han comentado sobre la 10ª y la 20ª generación se pueden observar en las figuras 2.29 y 2.30.

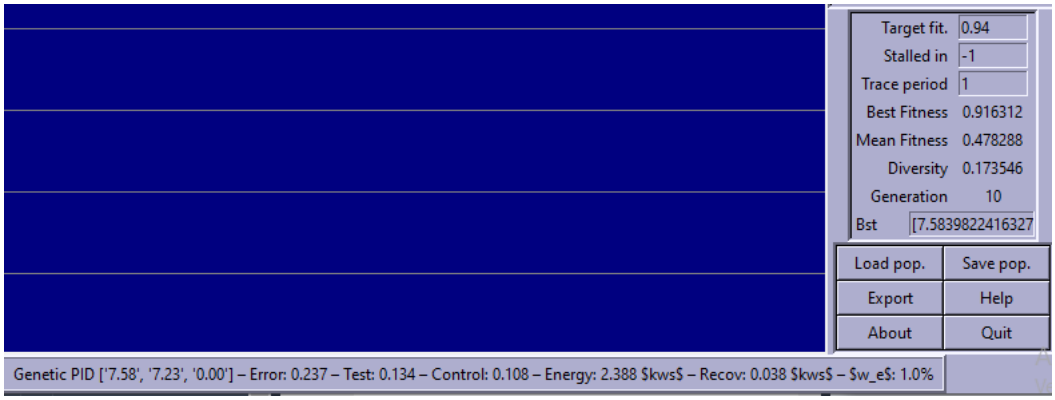

Figura 2.29: Valores al alcanzar la Generación 10

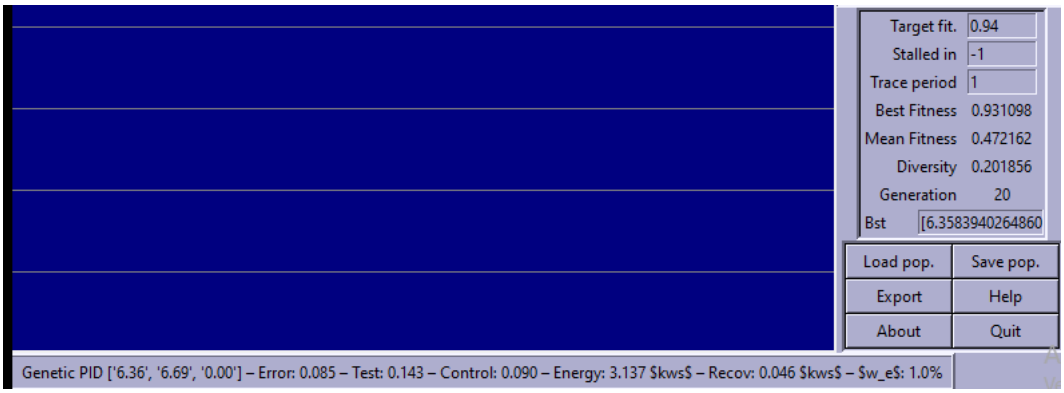

Figura 2.30: Valores al alcanzar la Generación 20

En cuanto a los valores de PID, errores, control y energía, vamos a verlos en la generación inicial (figura 2.28), en la generación 10ª (figura 2.29) y en la generación 20ª (figura 2.30). Al igual que en el We = 0.75, en la generación 20ª hay un individuo que está a punto de alcanzar el *Target Fitness* pero no se alcanza y además se considera que tantas generaciones son demasiadas para este problema. También, si se mantiene ejecutando generaciones, el algoritmo sigue sin converger, por tanto, es posible que esté estancado en un óptimo local. Estos valores se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	Generación 10	Generación 20
Kp	92.95	7.58	6.36
Ki	33.83	7.23	6.69
Kd	10.54	0.00	0.00
Error	53.901	0.237	0.085
E. Test	51.411	0.134	0.143
Control	53.119	0.108	0.090
Energy	437.603kws	2.388kws	3.137kws
Recov.	0.427kws	0.038kws	0.046kws
We	1 %	1 %	1 %

Tabla 2.6: Resultados We = 1.00

Analizando los valores de PID se observa que la constante de acción proporcional Kp y que la la constante de acción integral Ki disminuyen a valores muy cercanos de generación en generación lo que hace que el sistema esté equilibrado en la generación 10ª. Por ello, la constante de control derivativa Kd no cambia mucho a partir de la generación 10ª ya que el motor tiene una estabilidad aceptable.

Si se observan los errores ambos bajan a niveles aceptables, y al igual que en la We = 0.75, no hay mucha diferencia entre la generación 10ª y 20ª. Lo mismo ocurre con el control, que baja mucho pero entre la generación 10ª y 20ª no hay mucha diferencia.

Sin embargo, esta vez, el valor de energía es el mayor obtenido hasta ahora:

- We = 0 => 1.819kws gen10ª.
- We = 0.25 => 2.547kws gen13ª
- We = 0.50 => 1.504kws gen6ª
- We = 0.75 => 1.164kws gen20ª
- We = 1.00 => 3.137kws gen20ª

Por tanto, se descarta por completo el valor de We = 1.00 ya que no proporciona un controlador con un compromiso entre energía y rendimiento mejor que los anteriores.

2.1.7. We = 10.00

Se ha probado para el valor We = 10.00 pero la convergencia al 0.94 ha sido muy lenta y se ha observado que no vale la pena analizar este resultado.

2.1.8. Mejor controlador

Por tanto, se concluye que el mejor controlador alcanzado con las pruebas anteriores se obtiene con un valor We = 0.50 ya que se considera que de todos los controladores encontrados es el que proporciona un mejor compromiso entre energía y rendimiento. Además, se obtiene un PID estable y se alcanza rápidamente el objetivo porque ese valor de We hace que la función sea muy simple.

We = 0.50 es un valor que hace que no haya un gran overshoot, se reduce considerablemente desde las primeras generaciones ya que mejora la estabilidad, y también se dan pocos cambios de signos.

Pese a que no se puede observar gráficamente las gráficas en cada generación por el problema dado de Tkinter, matplotlib, y Windows, se puede observar la siguiente generación y ver que el error estacionario es bastante bajo.

También, hay que destacar que We = 0.50 ha funcionado muy bien para la población que se ha iniciado aleatoriamente en ese caso, pero es posible que al iniciar otra población aleatoria no se produzca un buen controlador para ese valor de We.

2.1.9. Cambiando la configuración inicial para el mejor controlador

Como se comentó al inicio de la sección, los resultados anteriores se han dado cambiando únicamente el *Target Fitness* de todos los posibles parámetros. A continuación, se cambiarán parámetros para el We = 0.5 para ver si es posible conseguir mejores resultados.

Es posible que no se obtengan mejores resultados, ya que también depende de la población inicial aleatoria. Por tanto, para este apartado se ha decidido probar con diversos parámetros pero con una misma población inicial que se ha cargado al inicio y se ha guardado. Por tanto, probaremos con diversos valores de mutación, torneo, etc. pero para la misma población inicial todo el rato. Para ello, se ha creado una nueva población y se ha guardado con el botón *Save pop.*, se ha probado su funcionamiento haciendo un pequeño cambio, y si se ha considerado que el algoritmo necesitaba otra cosa se ha vuelto a cargar la misma población con el botón *Load pop.* y se ha comprobado los resultados.

Por tanto, se ha comenzado con un We = 0.5 y creando la población:

Figura 2.31: Valores de la nueva población

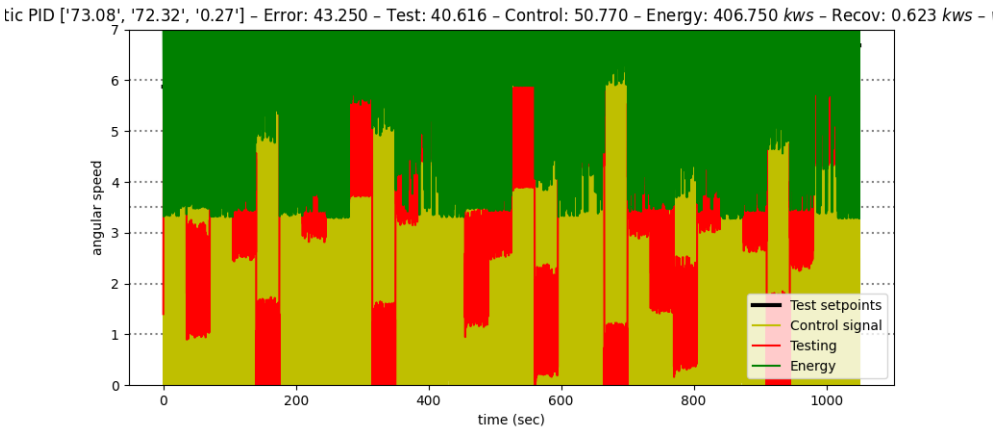

Figura 2.32: Gráfica de la nueva población

Para esta nueva población se ha probado a correr el programa sin modificar ningún parámetro como en las subsecciones anteriores:

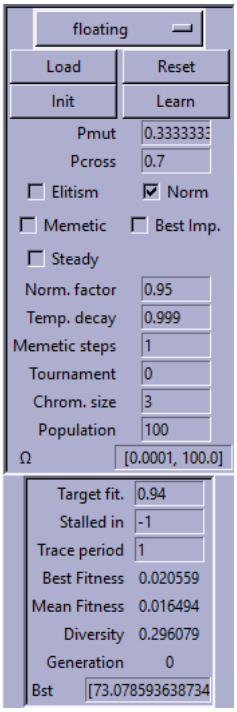

Figura 2.33: Parámetros sin modificar

Y se han obtenido los siguientes resultados:

Figura 2.34: Resultados We = 0.5. - Parámetros sin modificar

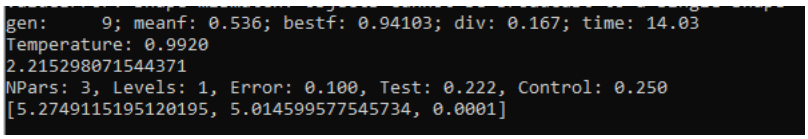

Figura 2.35: Resultados We = 0.5. - Parámetros sin modificar

La energía es alta, y el primer objetivo debe ser reducirla para conseguir un mejor controlador.

Para ello, se va a comenzar aumentando la diversidad ya que es muy baja. Por ello, se cambiará la Normalización por un Torneo de 2. Y también el *Target Fitness* se vuelve a colocar a 1.

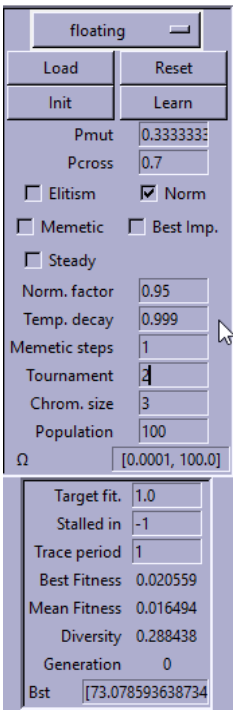

Figura 2.36: Parámetros Torneo = 2

Se realiza una parada manual porque los resultados que se estaban obteniendo no eran los que se buscaban.

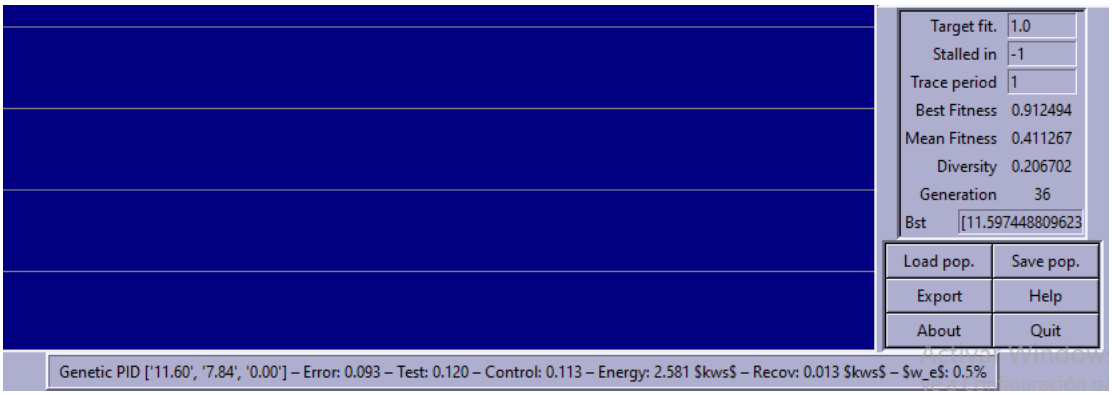

Figura 2.37: Resultados Torneo = 2 - Generación 36ª

Por tanto, se va a probar a aumentar la diversidad con la mutación. Se cambiará *Pmut* de 0.333333 a 0.5.

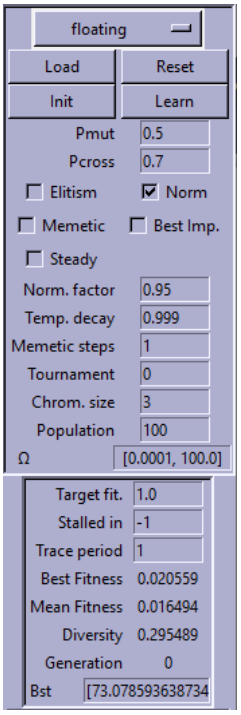

Figura 2.38: Parámetros Pmut = 0.5

En la generación 3 la población ya tiene un individuo que tiene un mejor fitness con valor de 0.94 pero el fitness medio de la población es bastante pésimo con un valor de 0.05, la diversidad se encuentra en este momento en un valor de 0.32. En la generación 10 aumenta un poco el fitness medio aunque la diversidad decrece. Al haber puesto una tasa de mutación relativamente alta hay muchas oscilaciones por la gran exploración.

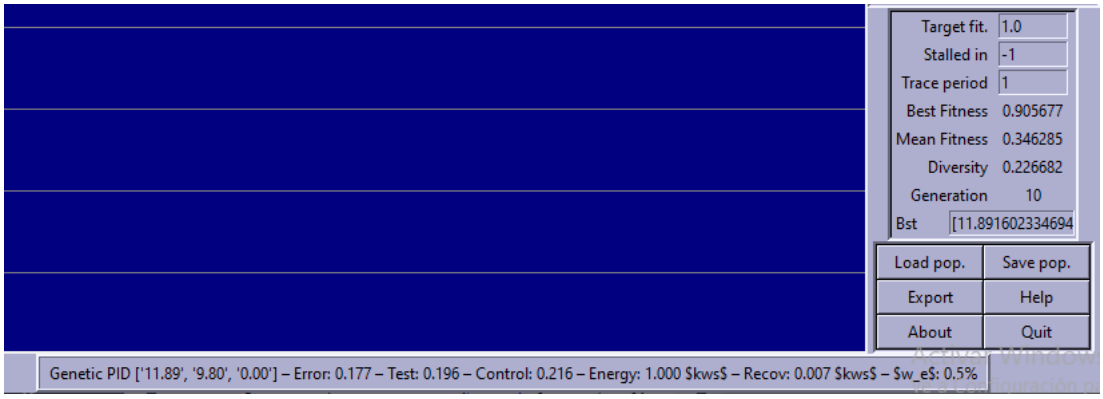

Figura 2.39: Resultados Pmut = 0.5 - Generación 10ª

En la generación 18ª un individuo de la población alcanza el target fitness de 0.94, pero el fitness medio es más bajo en comparación a otras configuraciones. En la generación 80ª se ha parado manualmente SALGA ya que al tener una alta tasa de mutación había muchas oscilaciones en el fitness, había mucha exploración. También hay que destacar que esta configuración aporta un buen controlador con un buen compromiso entre energía y rendimiento.

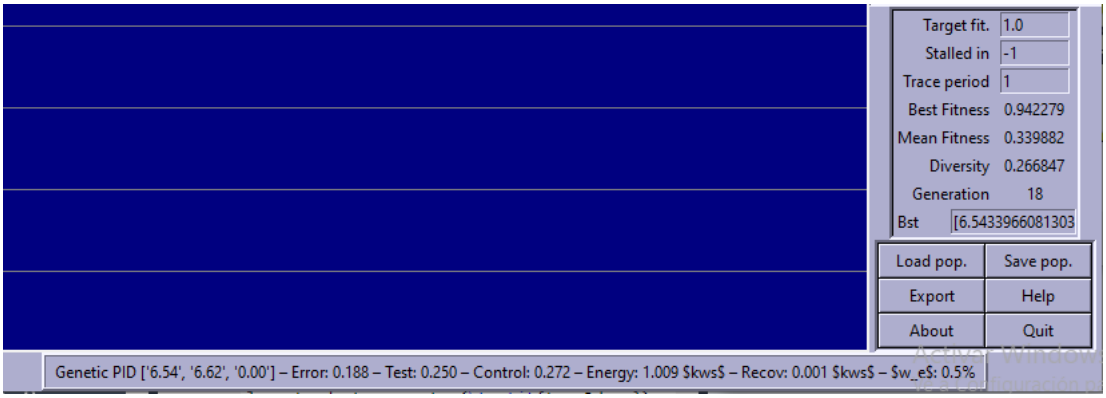

Figura 2.40: Resultados Pmut = 0.5 - Generación 18ª

Se quiere ver que ocurre si en vez de aumentar la mutación, se baja. Se sabe que la diversidad bajaría pero queremos que ver que resultados muestra, por tanto se cambiará *Pmut* de 0.333333 a 0.03.

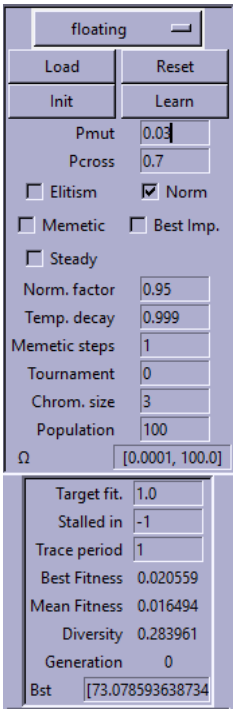

Figura 2.41: Parámetros Pmut = 0.03

Al llegar a un *Best Fitness* de 0.94 se decide parar manualmente por tiempo, ya que lleva muchas generaciones. Si que es cierto que se encuentra un buen controlador con estos valores pero tarda muchísimo en converger. Lo bueno es que toda la población tiene un fitness medio similar, esto se debe a la explotación. Este puede ser bueno, pero realmente no tanto, ya que es posible que esté estancado en un mínimo local ya que la diversidad se reduce considerablemente.

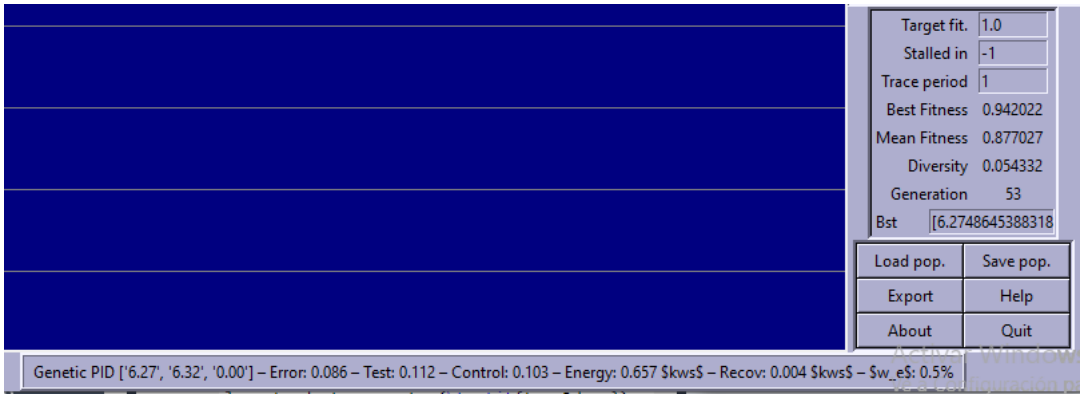

Figura 2.42: Resultados Pmut = 0.03

Se concluye que finalmente se prefiere seleccionar esta configuración final con respecto a las otras, con una mutación al 0.03 porque proporciona un mejor fitness aunque tenga una convergencia más lenta y nos arriesguemos a caer en minimos locales. Además, aporta un controlador con un compromiso muy bueno entre energía y rendimiento con una energía muy muy baja.

En cuanto a las demás posibles configuraciones, se decide no seleccionar los demás parámetros porque el elitismo podría hacer que se estanque en un mínimo local, y lo mismo con el paso de *best imp.*. También no se selecciona los algoritmos memeticos por eficiencia (en tiempo). Y finalmente se considera que la normalización es necesaria, ya que se ha decidido no utilizar torneo.

2.2. *pid_graph2.py*

En esta sección se tratará de optimizar el PID del segundo *fitness* proporcionado llamado *pid_graph2.py*. Se realizará la misma estrategia utilizada para el primer *fitness* *pid_graph.py* proporcionado que se ha visto en la sección 2.1. Al igual que con el anterior *fitness* se cargará y el único parámetro que se cambiará será el *Target Fitness* que pasará de 1 a 0.94, dejando los demás parámetros como vienen definidos en el código *pid_graph2.py*. Se probarán diferentes pesos de We. Al igual que en la sección anterior se probará con We = 0.00, We = 0.25, We = 0.50, We = 0.75, We = 1.00, y We = 10.00. Se continuará concluyendo cual es el mejor controlador con buen equilibrio entre prestaciones y consumo. Y se tratará de mejorar el resultado de optimización realizando cambios en los parámetros que habíamos dejado sin tocar como venían en el código, es decir, mutación, normalización, etc. para intentar encontrar mejores resultados.

2.2.1. Configuración inicial

La configuración inicial será la misma que en la subsección 2.1.1.

2.2.2. We = 0

Comenzaremos teniendo en cuenta únicamente el rendimiento del motor, sin considerar la energía. Por tanto, la constante de energía We (figura 2.43) será igual a 0.

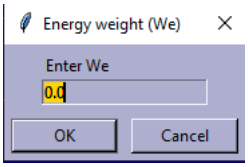

Figura 2.43: We

En los problemas de optimización del PID la generación de la población inicial tarda en generarse por la cantidad de operaciones que se tienen que realizar para evaluar cada *fitness*. Una vez cargado, se puede observar a la población inicial de 100 individuos. La población tiene un fitness con valor de 0.018 de media y el individuo de la población con mejor fitness tiene un valor de 0.022. Además, la población tiene una diversidad aceptable cercana a 0.3, lo que le proporcionará exploración a la siguiente generación.

Los valores de PID de la población inicial se pueden observar en la gráfica 2.45 y en la figura 2.46. Estos valores son Kp = 40.86, Ki= 95.52 y Kd=0.40. Además, se observa que hay un error de 43.645, y un error en test de 41.257. Estos errores son muy altos y hay que tratar de disminuirlos en las siguientes generaciones. También, tiene un valor de control de 51.571. En cuanto a la energía, su valor es de 318.197kws. Un valor muy alto, que como hemos explicado más arriba, estos valores altos de energía se deben a las altas velocidades, aceleraciones y frenazos que se producen alcanzando valores máximos y de una manera muy rápida. Es decir, hay grandes oscilaciones y hay que disminuirlas. Hay que reducir el overshoot.

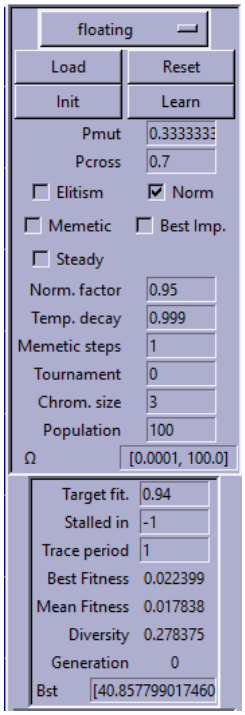

Figura 2.44: Instanciación

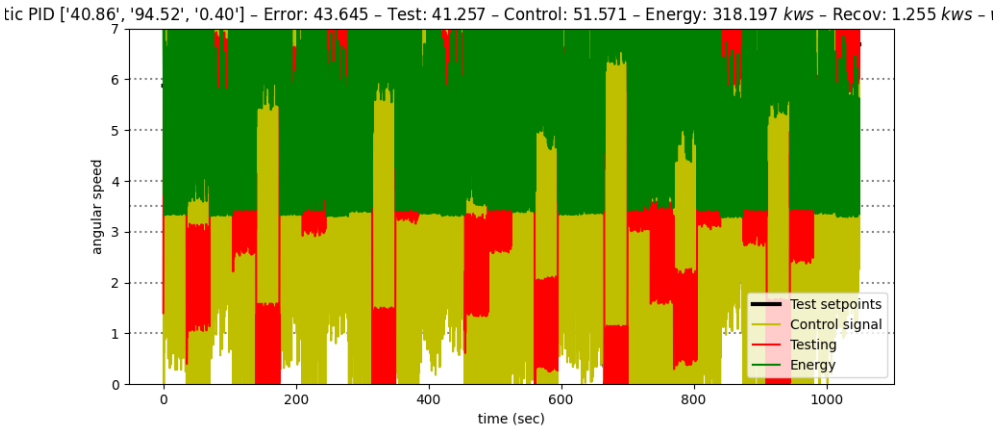

Figura 2.45: Gráfica inicial

Figura 2.46: Valores iniciales

Una vez se pulsa el botón de *Learn* y comienza la evolución, al igual que con el otro PID de la anterior sección, sobre la generación 3ª ya hay individuos que se acercan a un valor de 0.8 en fitness. Aún así, no es hasta la generación 9ª cuando consigue que uno de los individuos de la población alcance el *Target Fitness* instaurado en 0.94. La población en la generación 9ª tendrá un valor medio de *fitness* de 0.45. Y además, tendrá una diversidad tirando a baja de 0.21 que haría que si pusieramos el *Target Fitness* en 1 le costaría converger rápidamente, ya que una baja diversidad propicia más explotación que exploración. En este algoritmo al principio tenemos exploración y rápidamente comienza la explotación.

Estos valores que se han comentado sobre la 9ª generación se pueden observar en las figuras 2.47 y 2.48.

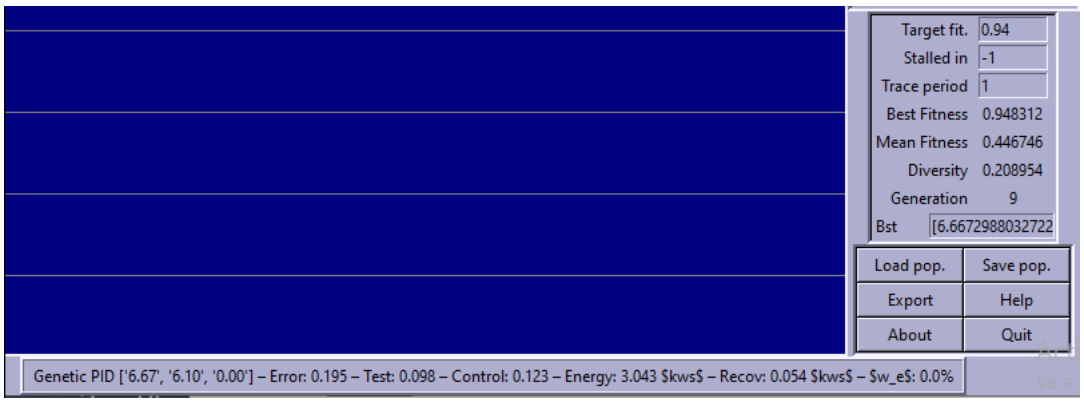

Figura 2.47: Valores al alcanzar el *Target Fitness* - Generacion 9

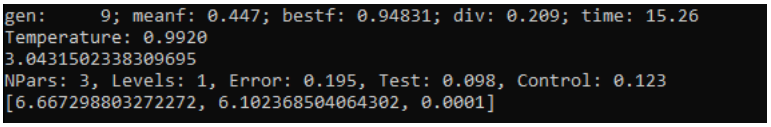

Figura 2.48: Valores completos al alcanzar el *Target Fitness* - Generacion 9

Es el turno de analizar los valores de PID, errores, control y energía, en la generación inicial y en la generación 9ª que contiene un individuo que alcanza el *Target Fitness*. Estos valores se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	**Generación 9**
Kp	40.86	6.67
Ki	94.52	6.10
Kd	0.40	0.00
Error	43.645	0.195
E. Test	41.257	0.098
Control	51.571	0.123
Energy	318.197kws	3.043kws
Recov.	1.255kws	0.054kws
We	0 %	0 %

Tabla 2.7: Resultados We = 0.00

Se observa como los resultados de la generación 9ª son mejores que los resultados de la generación inicial.

Comenzamos analizando las constantes del PID: Kp, Ki, y Kd. La gran disminución de la constante de acción proporcional Kp hace que la velocidad se vuelva más lenta pero proporciona una mayor estabilidad al sistema. Esta estabilidad se reducirá considerablemente por la gran disminución de la constante de acción integral Ki hará que el sistema sea menos estable pero que la velocidad aumente. Finalmente, la disminución de la constante de control derivativa Kd hace que también la estabilidad sea menor y que la velocidad también aumente. Como la Ki baja mucho más que la Kp hay más inestabilidad que estabilidad, por ello, la energía no alcanza niveles tan bajos como con el We = 0.00 (1.819kws gen10ª) de la sección anterior. Aunque la energía haya bajado considerablemente, el sistema no se encuentra tan estable y hace que el gasto de energía sea mayor.

Pese a no poder ver las gráficas por el problema de versiones con Tkinter, matplotlib, y Windows, los datos nos hacen ver que esta vez existe overshoot. Además, la siguiente generación 10ª que se muestra antes de finalizar el algoritmo, muestra que hay un error estacionario no tan bajo.

Por tanto, hemos obtenido un rendimiento que podría ser aceptable en comparación a la generación inicial, pero que no satisface completamente el objetivo de la práctica ya que se busca el controlador que aporte el mejor compromiso entre energía y rendimiento, y se considera que este no es muy bueno si lo comparamos con los resultados obtenidos del PID de la sección 1. Como no se ha tenido en cuenta la energía, esto se tratará en las siguientes subsecciones.

2.2.3. We = 0.25

El objetivo es buscar un controlador que aporte el mejor compromiso energía/rendimiento. Por ello, se probará teniendo en cuenta la energía con un We = 0.25 (figura 2.49).

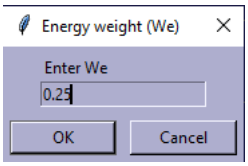

Figura 2.49: We

Al cargar el fitness tarda en generar la población inicial como ya se ha explicado. Además, los parámetros siguen sin cambiar como se pueden ver en la figura 2.50 o en la subsección 2.2.1.

Esta población inicial generada de 100 individuos que se puede ver en la figura 2.50 tiene un fitness medio de 0.017 y uno de sus individuos alcanza un fitness de 0.021. Además, tiene una diversidad aceptable con un valor de 0.29 que hará que sea un algoritmo exploratorio en las primeras generaciones.

Si se observa la gráfica 2.51 y la figura 2.52 podemos obtener los datos de PID de la población inicial. Estos datos son Kp = 57.83, Ki = 7.56 y Kd= 0.22. Además, podemos ver el valor de error = 42.825 y de Test = 42.713. Valores de error muy altos que hay que tratar de disminuir. En cuanto al control esta generación inicial tiene un valor de 51.056. Y el valor de la energía es un valor demasiado alto: 366.521kws. Esto se debe a que es un sistema muy inestable que cambia constantemente su velocidad acelerando y frenando alcanzando máximos. Esta energía se tratará de reducir en las siguientes generaciones.

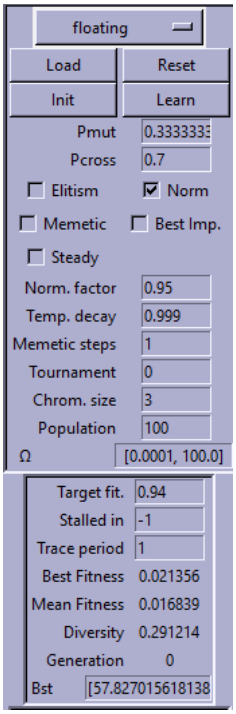

Figura 2.50: Instanciación

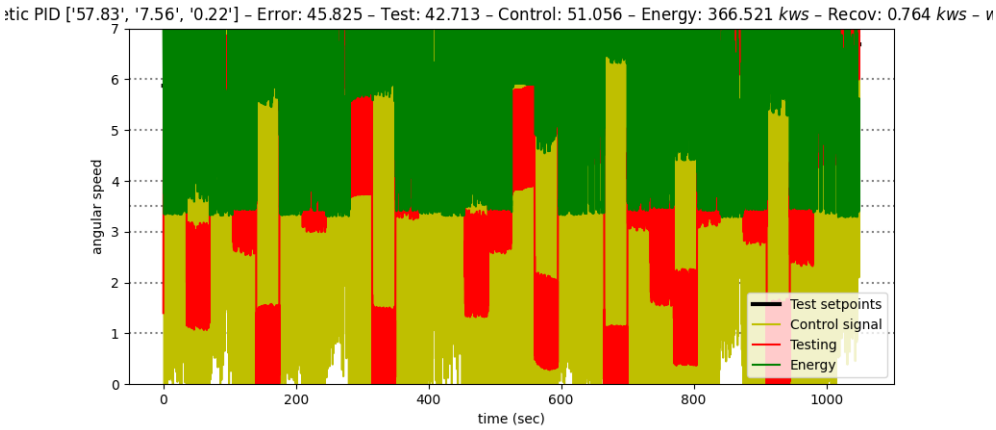

Figura 2.51: Gráfica inicial

Figura 2.52: Valores iniciales

Al comenzar la evolución con el botón *Learn* gracias a la diversidad de la población inicial se alcanza en la generación 3ª un valor muy alto de 0.9 en uno de los individuos de la población pero la diversidad baja considerablemente a 0.2. Por culpa de la diversidad baja, hasta la generación 7ª no se da ningún individuo en la población de las generaciones que alcance el *Target Fitness*. La generación 7ª tiene un *fitness* medio de 0.45 y una diversidad baja cercana a 0.19. Aún así, ha alcanzado el *Target Fitness* bastante rápido y, por tanto, We = 0.25 podría ser un buen candidato, aunque si proporciona unos malos resultados de PID y energía será totalmente descartado.

Estos valores que se han comentado sobre la 7º generación se pueden observar en las figuras 2.53 y 2.54.

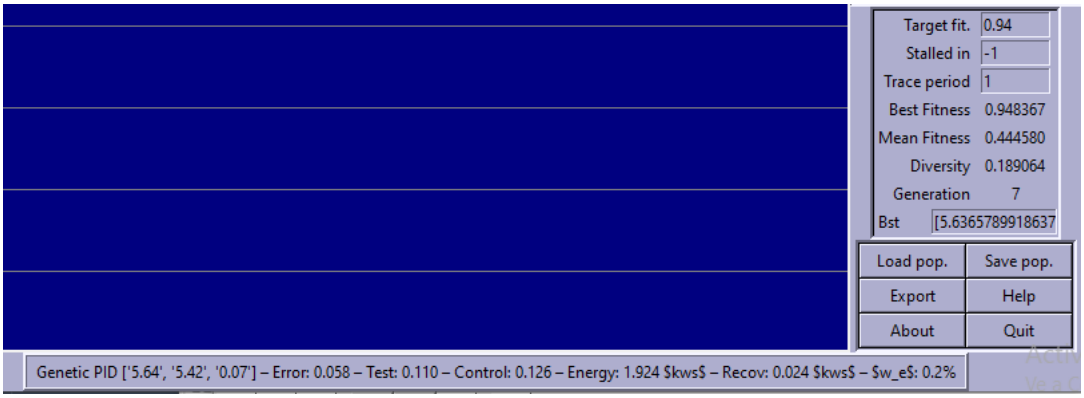

Figura 2.53: Valores al alcanzar el *Target Fitness* - Generacion 7

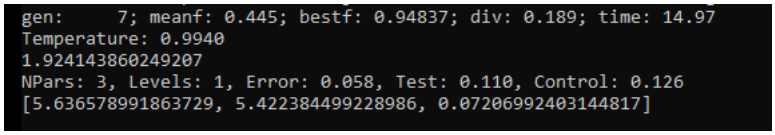

Figura 2.54: Valores completos al alcanzar el *Target Fitness* - Generacion 7

Vamos a ver los valores de PID, errores, control y energía, de la generación inicial (figura 2.52), y en la generación 7ª (figura 2.53) para ver si We = 0.25 es el We que mejor adapta nuestro objetivo. Estos valores se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	**Generación 7**
Kp	57.83	5.64
Ki	7.56	5.42
Kd	0.22	0.07
Error	45.825	0.058
E. Test	42.713	0.110
Control	51.056	0.126
Energy	366.521kws	1.924kws
Recov.	0.764kws	0.024kws
We	0.2 %	0.2 %

Tabla 2.8: Resultados We = 0.25

A primera vista se observa un resultado bastante bueno. Vamos a analizarlo.

Comenzaremos con las constantes del PDI: Kp, Ki y Kd.

La constante de acción proporcional Kp ha disminuido considerablemente de la generación inicial a la generación 7ª. Esta disminución propociona que la velocidad sea más lenta pero que la estabilidad del sistema aumente. En cuanto a la constante de acción integral Ki, disminuye muy poquito por tanto reduce muy poco la estabilidad y aumenta un poquito la velocidad. En cuanto a la la constante de control derivativa Kd disminuye poco, lo que hace que la estabilidad disminuya un poco y que la velocidad aumente. Por tanto, se observa que el sistema está bastante equilibrado. Por ello, se da ena bajada en el control, en los errores y en la energía. El gasto de energía, en comparación al We = 0.00 ha bajado considerablemente y es bastante aceptable. Esto indica que el sistema y la velocidad es estable.

A pesar de no poder ver las gráficas por el problema de versiones con Tkinter, matplotlib, y Windows, los datos nos hacen ver que no hay un gran overshoot en esta última generación, hay bastante estabilidad. Y la siguiente generación 8ª que se muestra antes de finalizar el algoritmo, muestra datos bastante similares, lo que significa que el error estacionario es bajo, además los datos Kp, Ki, Kd controlan este error.

Se concluye que en este caso un We = 0.25 es un buen valor de We ya el controlador tiene un compromiso realmente bueno entre energía y rendimiento. Se probará si es posible reducir aún más esta energía y mejorar el compromiso aumentando la constante de energía We en las siguientes subsecciones.

2.2.4. We = 0.50

Veamos que ocurre con un valor de We = 0.50 (figura 2.55). Quizás este sea el mejor valor al igual que en la anterior sección.

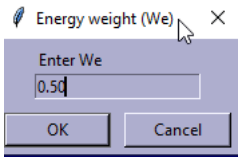

Figura 2.55: We

Al cargar en fitness con el botón *Load* en SALGA se genera la población inicial. La población inicial tarda en generarse por las operaciones necesarioas para el *fitness* cargado. Esta vez los parámetros tampoco cambian y se pueden observar en la figura 2.56 o en la subsección 2.2.1.

Esta población inicial está compuesta por 100 individuos con un *fitness* medio de 0.02 y contiene un individuo con un mejor *fitness* con valor 0.5. Además, tiene una diversidad alta cercana a 0.3 lo que producirá exploración en las primeras generaciones.

Si observamos la gráfica 2.57 y la imágen 2.58, podemos ver los valores del PID de la población inicial: Kp = 15.38, Ki = 66.05 y Kd = 0.23.

Es un sistema que tiene unos errores bajos, pero considerables de 1.052 y en test de 1.507. El objetivo es conseguir errores por debajo de 1.

En cuanto al control tiene un valor de 1.899. También, tiene una energía ya en el principio casi baja de 4.552kws, pero sigue siendo un gasto alto que hay que tratar de reducir ya que se observa que sigue siendo un gasto de energía bastante alto dado por la inestabilidad.

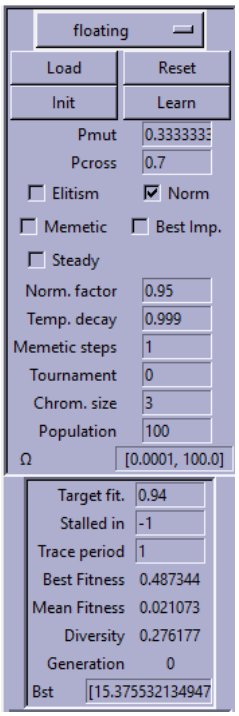

Figura 2.56: Instanciación

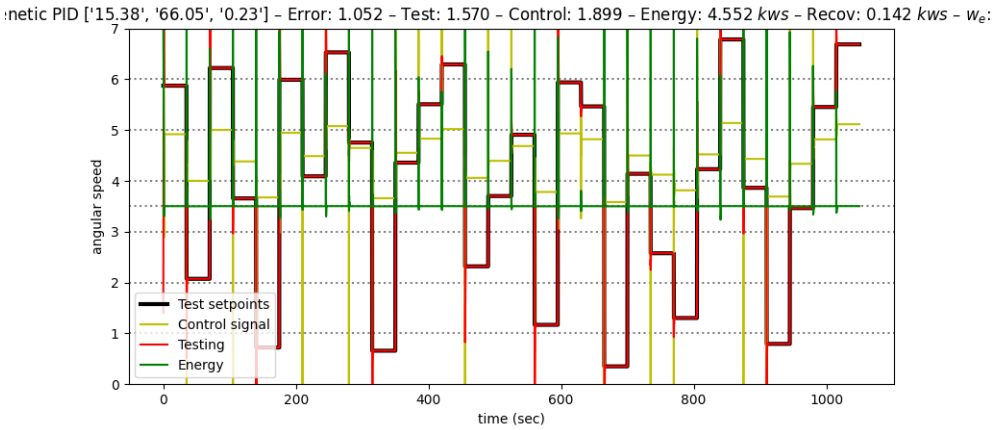

Figura 2.57: Gráfica inicial

Figura 2.58: Valores iniciales

Una vez pulsado el botón de *Learn*, no es hasta la generación 9ª que se da un individuo que alcanza el *Target Fitness*. Esta generación obtiene un *fitness* con valor de 0.45 de media y una diversidad baja de 0.20. Que lo alcance en la 9ª generación puede que indique que We = 0.50 de un peor resultado que We = 0.25. Esto puede ser debido a que ha tenido una convergencia más lenta. Aún así antes de descartar este resultado y poner a We = 0.25 como el mejor, habrá que analizar los demás resultados: energía, PID, etc.

Estos valores que se han comentado sobre la 9ª generacion se pueden observar en las figuras 2.59 y 2.60.

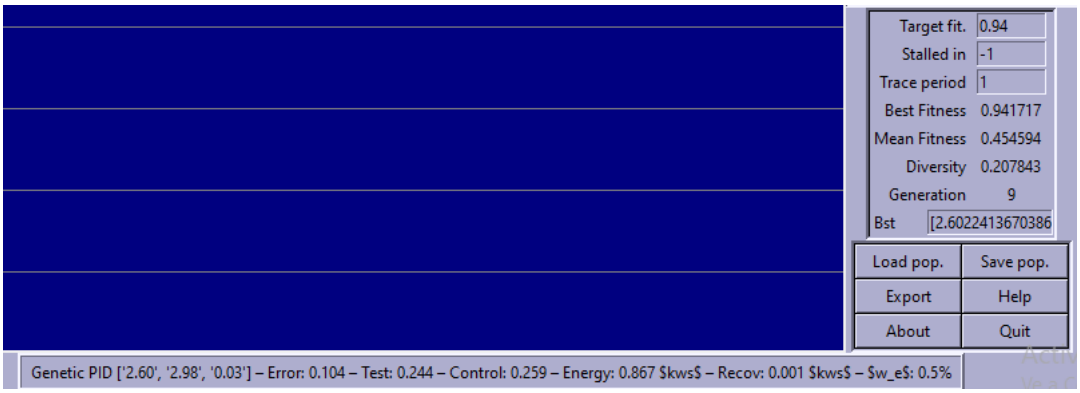

Figura 2.59: Valores al alcanzar el *Target Fitness* - Generación 9

```
gen:     9; meanf: 0.455; bestf: 0.94172; div: 0.208; time: 15.26
Temperature: 0.9920
0.8667432141000384
NPars: 3, Levels: 1, Error: 0.104, Test: 0.244, Control: 0.259
[2.602241367038603, 2.984757423614931, 0.031210043395800337]
```

Figura 2.60: Valores completos al alcanzar el *Target Fitness* - Generación 9

Por tanto, vamos a analizar los valores de PID, errores, control y energía, en la generación inicial (figura 2.16) y en la generación 9ª (figura 2.17) que contiene un individuo que alcanza el *Target Fitness* para ver si We = 0.50 es un buen valor o lo descartamos. Estos valores se pueden comparar en la tabla siguiente y se discutirán a continuación.

	Generación Inicial	**Generación 9**
Kp	15.38	2.60
Ki	66.05	2.98
Kd	0.23	0.03
Error	1.052	0.104
E. Test	1.570	0.244
Control	1.899	0.259
Energy	4.552kws	0.867kws
Recov.	0.142kws	0.001kws
We	0.5 %	0.5 %

Tabla 2.9: Resultados We = 0.50

Para ver si We = 0.50 es un buen valor, comenzaremos analizando los valores del PID: Kp, Ki y Kd. En cuanto a la constante de acción proporcional Kp, ha disminuido lo que hace que la velocidad sea más lenta que en la generación inicial pero mucho más estable. La constante de acción integral Ki disminuye muchísimo más que la Kp lo que hace que el sistema sea mucho menos estable pero que la velocidad aumente. Por tanto, ahora tendremos un sistema menos estable pero rápido. Al disminuir también Kd el sistema se vuelve mucho menos estable y más rápido. Si observamos los errores, ambos bajan a unos niveles muy aceptables, aunque los errores son algo mayores que para el We = 0.25. El control también se reduce pero también es mayor que el del We = 0.25.

Es en la energía donde encontramos una diferencia notable en comparación al We = 0.25. Ya que alcanza un valor de 0.867kws, realmente bajo. Por tanto, el sistema, pese a que es algo inestable (si tuvieramos acceso a las gráficas podríamos confirmar mejor esta suposición) obtiene un nivel de energía muy bajo lo que quiere decir que no hay overshoot y que realmente hay una estabilidad real ya que no gasta energía al acelerar, frenar, etc.

Por tanto, este valor de We = 0.5, por ahora podríamos considerarlo mejor que We = 0.25 y We = 0.00, aunque 0.25 también podría ser un buen valor. Se considera que We = 0.5 propociona el controlador con mejor compromiso energía/rendimiento en comparación a otros We.

Aún así vamos a continuar aumentando la constante de energía We por si obtenemos mejores resultados.

2.2.5. We = 0.75

Se cree que va a ser complicado mejorar los resultados que proporciona We = 0.5.

Pese a esta creencia, rápidamente se analizan los resultados de We = 0.75. El análisis para este resultado no va a ser tan profundo ya que solo queremos comprobar si hay una mejora o no. Por tanto, se muestran las imágenes correspondientes a We = 0.75 que también se han mostrado con los otros We, y al final de la subsección se realizará una conclusión.

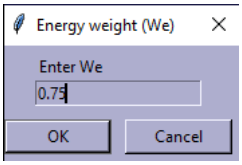

Figura 2.61: We

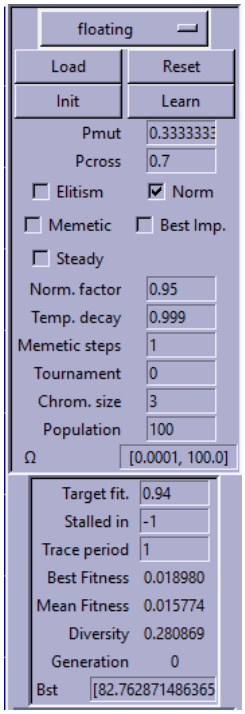

Figura 2.62: Instanciación

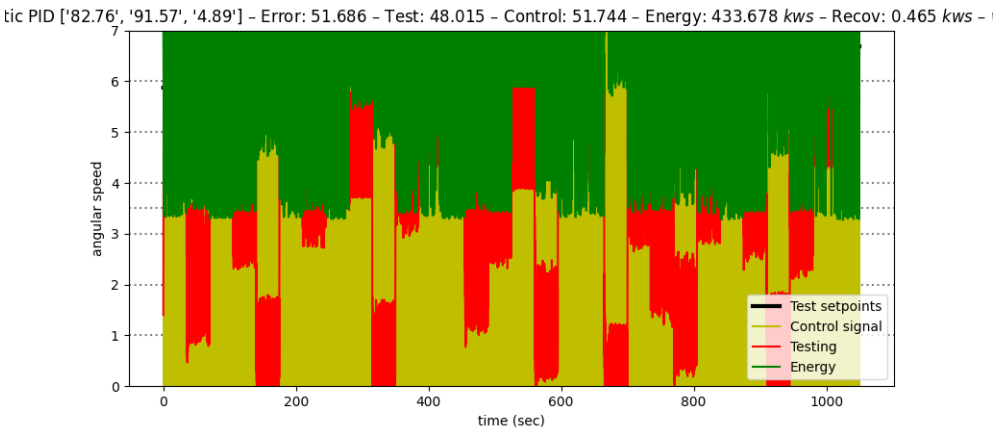

Figura 2.63: Gráfica inicial

Figura 2.64: Valores iniciales

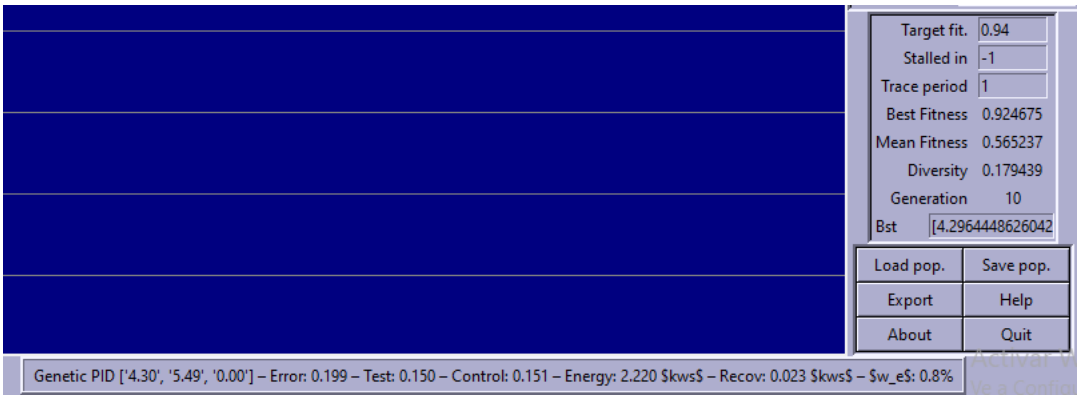

Figura 2.65: Valores al alcanzar la Generación 10

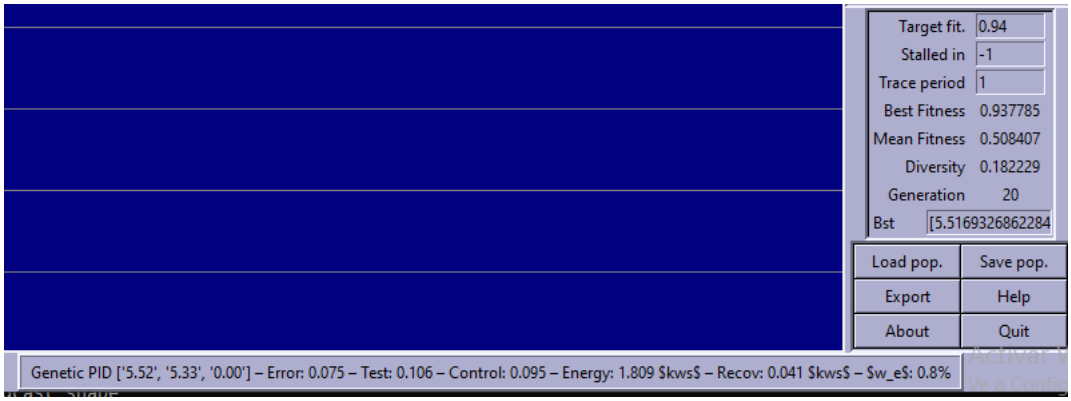

Figura 2.66: Valores al alcanzar la Generación 20

Se concluye que ocurre exactamente igual que con el PID del *fitness pid_grap.py* de la sección anterior.

No solo no se alcanza una generación (<20ªgen) que contenga un individuo que alcance el *Target Fitness*, sino que además no se mejora el PID, el error, ni la energía. Por tanto, se concluye que este valor de We = 0.75 no se adapta a nuestro objetivo como se pensaba desde el inicio.

2.2.6. We = 1.00

Al igual que con el valor de We = 0.75, se realiza para We = 1.00. Esto se realiza rápido para confirmar nuestras hipótesis. A continuación se pueden ver las figuras del proceso y al finalizar estas, se realizará una conclusión.

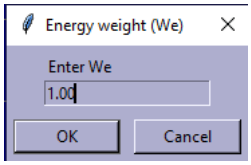

Figura 2.67: We

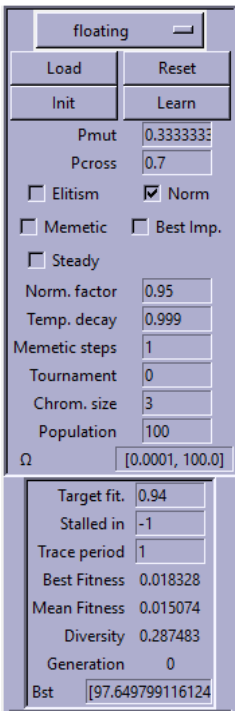

Figura 2.68: Instanciación

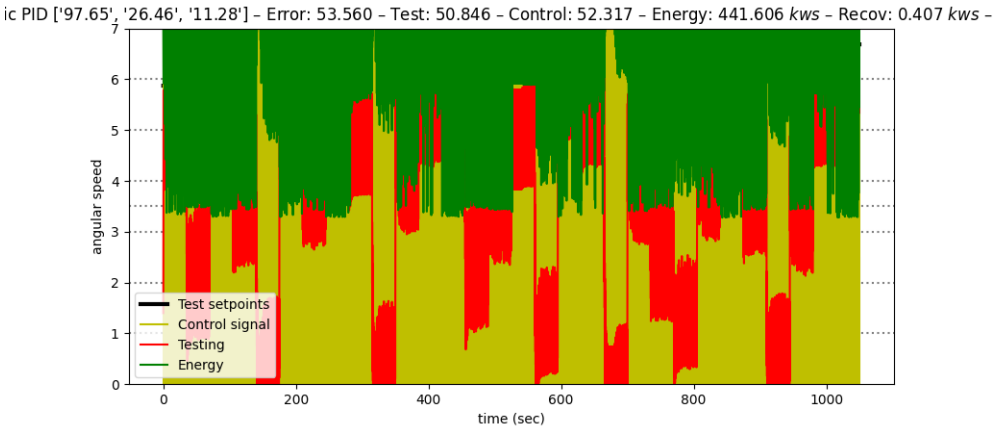

Figura 2.69: Gráfica inicial

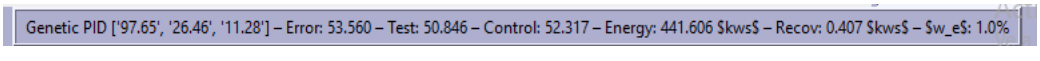

Figura 2.70: Valores iniciales

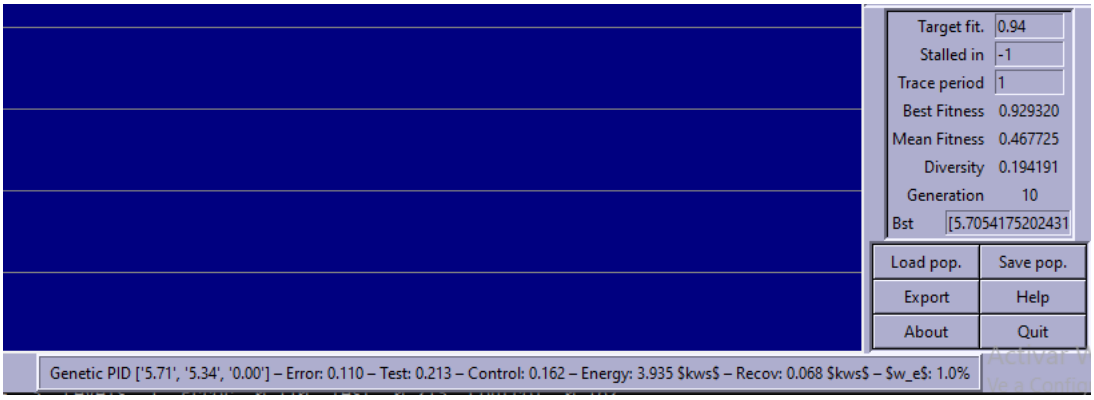

Figura 2.71: Valores al alcanzar la Generacion 10

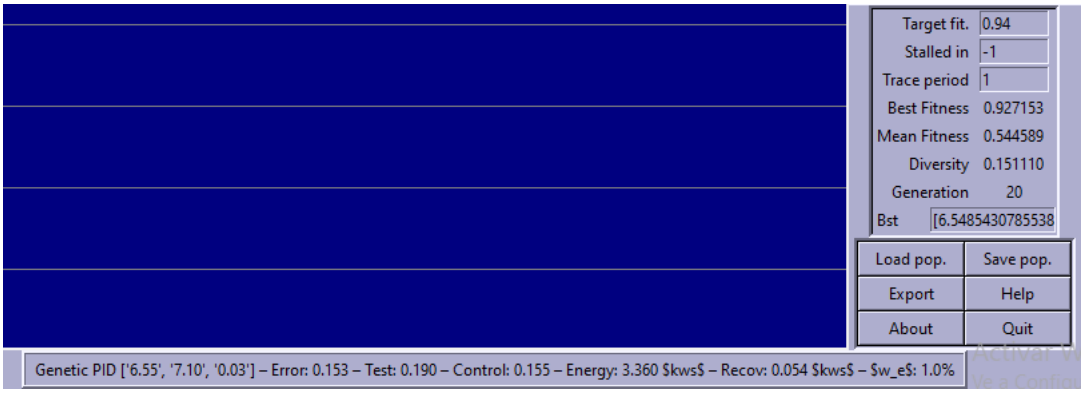

Figura 2.72: Valores al alcanzar la Generación 20

Por tanto, la hipótesis se confirma. Ocurre lo mismo que en el We = 0.75, y este valor no es el valor que se adapta a nuestro objetivo. ¿Por qué? Porque, al igual que antes no se alcanza una generación (<20ªgen) que contenga un individuo que alcance el *Target Fitness*, y lo más importante: no mejora el PID ni la energía, sino que la empeora considerablemente.

2.2.7. We = 10.00

Rápidamente se ha probado SALGA y este fitness para el valor de We = 10.00 pero los resultados han sido tan nefastos que no vale la pena analizarlos, al igual que en la sección anterior.

2.2.8. Mejor controlador

Se concluye que el mejor controlador alcanzado se obtiene con un valor de We = 0.50. Con este valor se consigue un controlador que proporciona un buen equilibrio entre prestaciones y consumo para el motor.

Además, el valor de energía alcanzado es mucho mejor (0.867kws) que el valor de energía alcanzado (1.504kws) para el mejor We (también We = 0.5) del PID de la sección anterior. Por tanto, este controlador sería mejor que el de la sección anterior ya que proporciona un mejor compromiso entre energía y rendimiento. Aunque esto no se puede afirmar como tal ya que esta

afirmación no tiene en cuenta que estos resultados pertenecen a distintas poblaciones iniciales, y de la población inicial y de otros factores dependen mucho los resultados. Por tanto, más que una afirmación, es una suposición.

También, el controlador obtenido con el We = 0.50 aporta un PID estable. En cuanto a la velocidad, es menos rápido que el mejor de la sección anterior, ya que en el anterior se alcanza el *Target Fitness* en la generación 6ª y en este en la Generación 9ª. De igual modo, es una suposición ya que se parte de puntos distintos.

Esta solución es una buena solución, pero nunca podemos estar seguros de que es la mejor solución.

2.2.9. Cambiando la configuración inicial para el mejor controlador

Al igual que en la sección anterior, se cambiarán parámetros (a parte del *Target Fitness*) para el We = 0.5 para ver si es posible conseguir mejores resultados.

Como los resultados dependen también de la inicialización de la población inicial, para que este apartado tenga sentido, *jugaremos* todo el rato con la misma población inicial que crearemos, juardaremos con el botón *save pop.*, probaremos configuraciones nuevas, y cada vez que queramos probar algo nuevo cargaremos de nuevo la población con el botón *Load Prob.*. Esto es lo mismo que se decidió realizar en el apartado 2.1.9.

Por tanto, se ha comenzado con un We = 0.5 y creando la población:

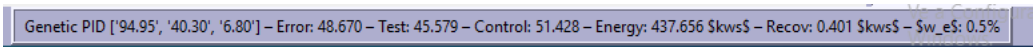

Figura 2.73: Valores de la nueva población

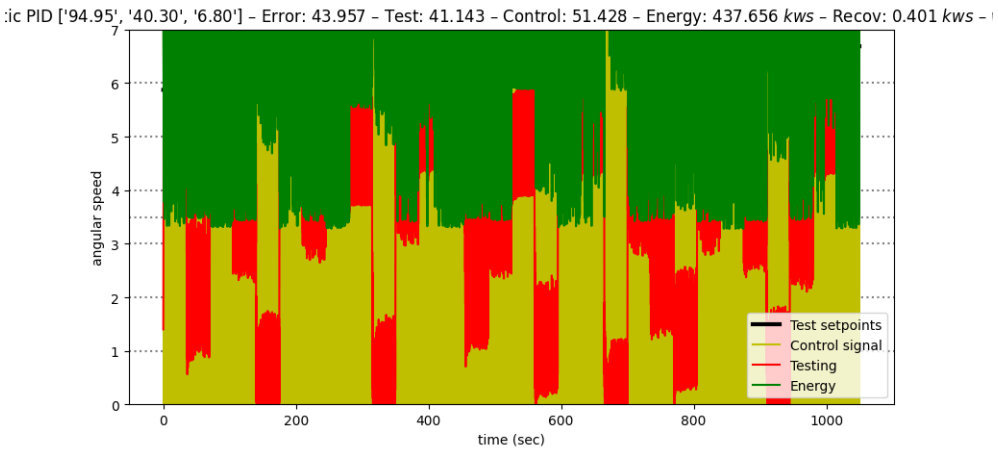

Figura 2.74: Gráfica de la nueva población

Al igual que en el fitness anterior, para esta nueva poblacion se ha probado a correr el programa sin modificar ningún parámetro como en las subsecciones anteriores:

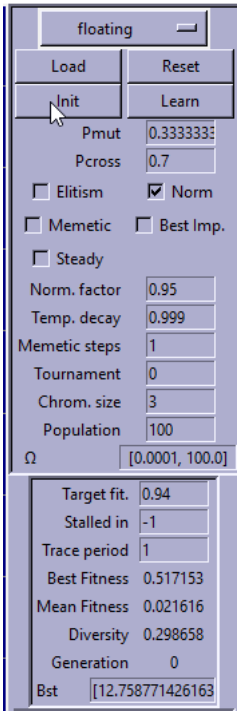

Figura 2.75: Parámetros sin modificar

Y se han obtenido los siguientes resultados:

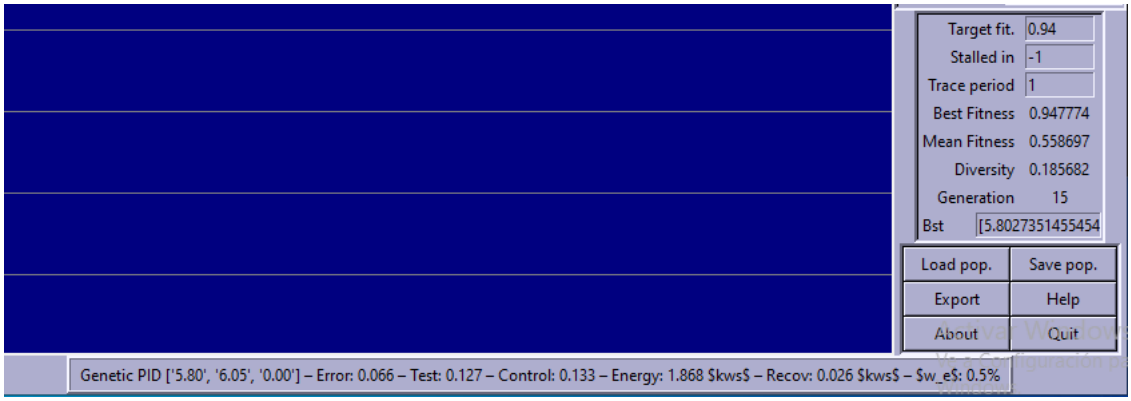

Figura 2.76: Resultados We = 0.5. - Parámetros sin modificar

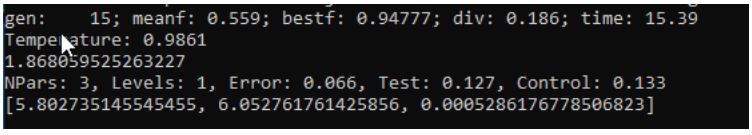

Figura 2.77: Resultados We = 0.5. - Parámetros sin modificar

Observamos como el gasto de energía es aceptable, aunque queremos intentar reducirlo para obtener un mejor controlador.

Por tanto, se tratará de aumentar la diversidad para ver si encuentra el óptimo más rápidamente. Para aumentarla, primero, se va a probar con un valor bajo de Torneo.

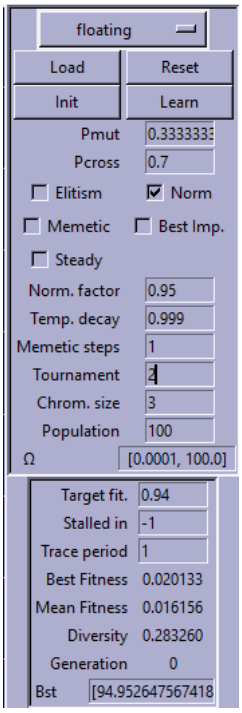

Figura 2.78: Parámetros Torneo = 2

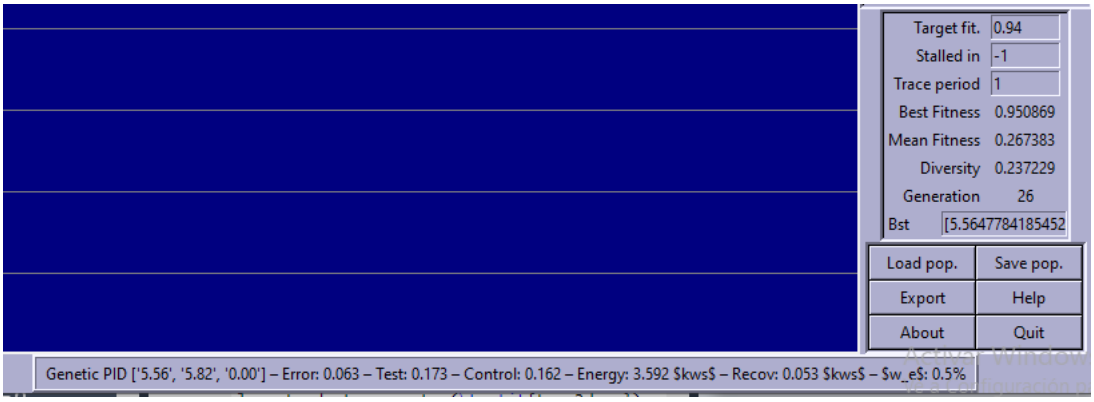

Figura 2.79: Resultados Torneo = 2 - Generación 26ª

Observamos que el controlador obtenido no ofrece un buen compromiso entre energía y rendimiento, ya que el gasto de energía se puede considerar alto.

Queremos ver que ocurre si aumentamos muchísimo más la exploración. Por tanto, instanciamos un valor de Torneo = 6.

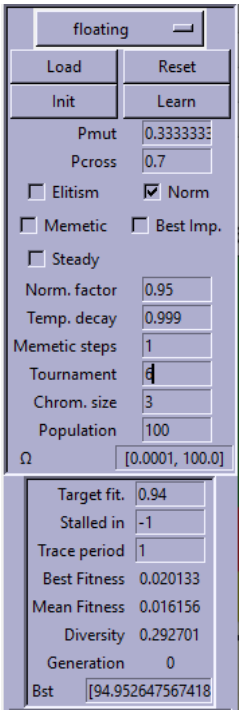

Figura 2.80: Parámetros Torneo = 6

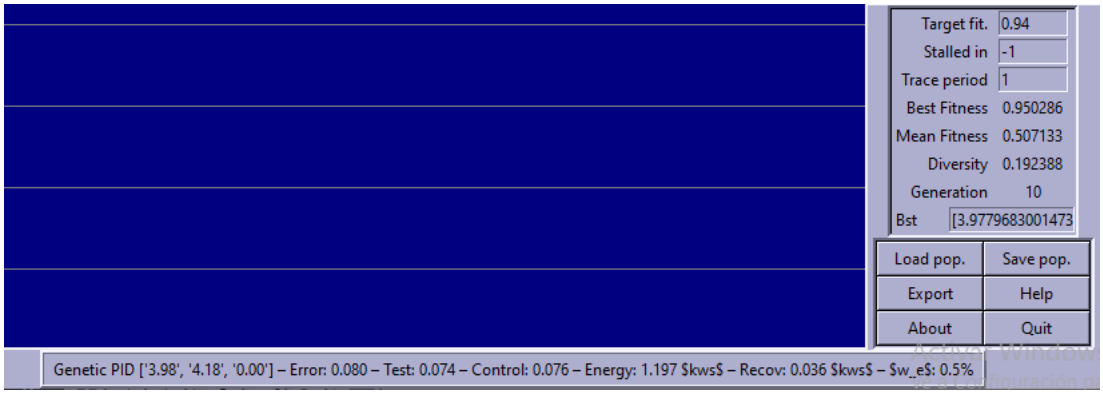

Figura 2.81: Resultados Torneo = 6 - Generación 10ª

Con un valor de Torneo = 6 los resultados obtenidos son bastante buenos ya que mejora el controlador inicial y reduce la energía.

Se quiere probar que ocurre si en vez de utilizar torneo se aumentar la diversidad con la mutación. Se instanciará un valor de *Pmut* = 0.5.

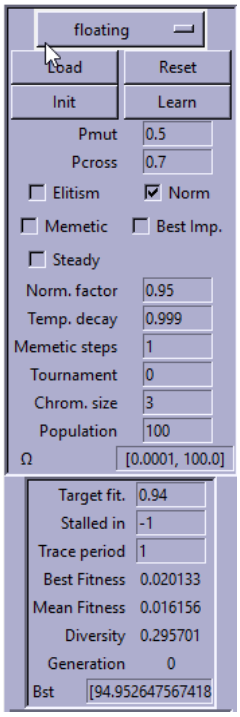

Figura 2.82: Parámetros Pmut = 0.5

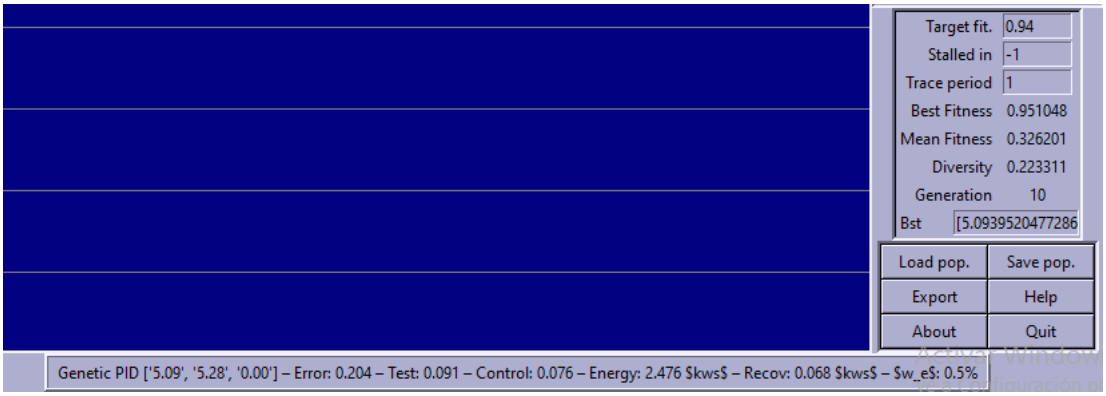

Figura 2.83: Resultados Pmut = 0.5

No obtenemos un buen controlador, así que esta configuración queda totalmente descartada.

Se probará si se obtiene un buen controlador en vez de aumentando la probabilidad de mutación, reduciéndola a 0.01. Así aumentamos la explotación. Aunque es posible que si se hace esto pueda converger a un óptimo local.

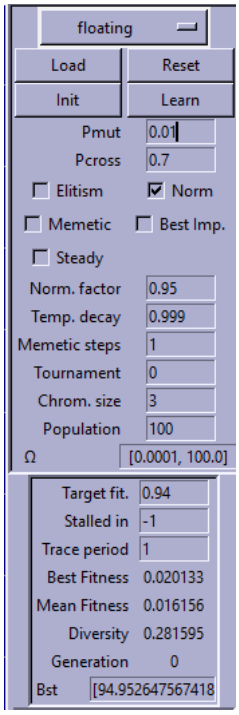

Figura 2.84: Parámetros Pmut = 0.01

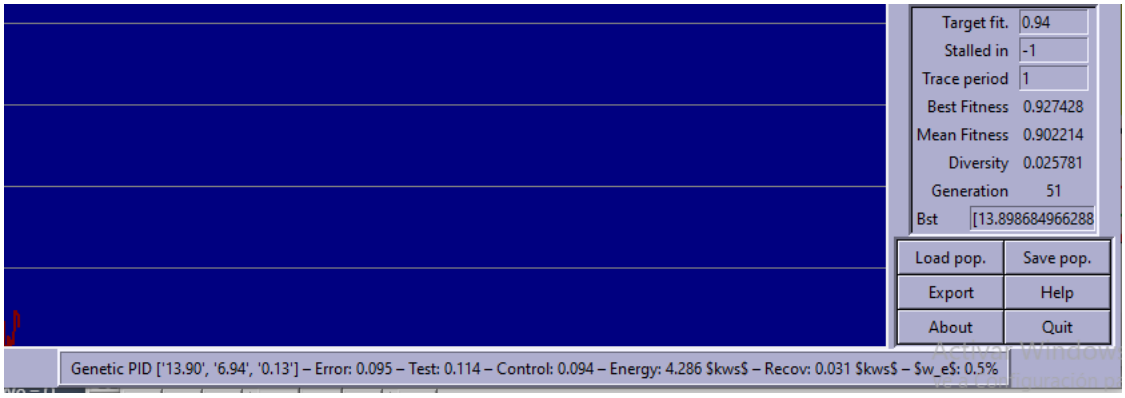

Figura 2.85: Resultados Pmut = 0.01 - Generación 51ª

Se decide parar SALGA en la generación 51ª para el Pmut bajo porque se estanca en un mínimo local y no es capaz de salir de él. Además, el controlador que se estaba obteniendo era bastante malo, y la diversidad muy muy baja, lo que hacía que fuese difícil obtener mejores resultados. Por tanto, se decide desechar esta configuración.

Se concluye, que la configuración que mejor controlador aporta es la del torneo = 6 (figura 2.80).

En cuanto a las demás posibles configuraciones, se decide no seleccionar los demás parámetros porque el elitismo podría hacer que se estanque en un mínimo local, y lo mismo con el paso de *best imp.*. También no se selecciona los algoritmos memeticos por eficiencia (en tiempo).

Capítulo 3

Optimización de la ruta de transporte

En este capítulo se tratará de alcanzar el objetivo de optimizar una ruta de transporte (*fitness tsp2d.py*). Se requiere encontrar la ruta más corta para entregar 100 paquetes en una ciudad. La distancia que se supondrá será la euclídea. Además, se suministran los puntos de entrega de los 100 paquetes.

3.1. Optimización - Parámetros por defecto

Se va a realizar la optimización varias veces con los parámetros que vienen por defecto. Se realizará varias veces para encontrar el mejor resultado ya que este depende de muchos factores, por ejemplo, depende de la inicialización de la población inicial, ya que esta es aleatoria.

A continuación se muestran los datos de las poblaciones iniciales y sus resultados, en la última subsección se analizarán los resultados.

Optimización 1

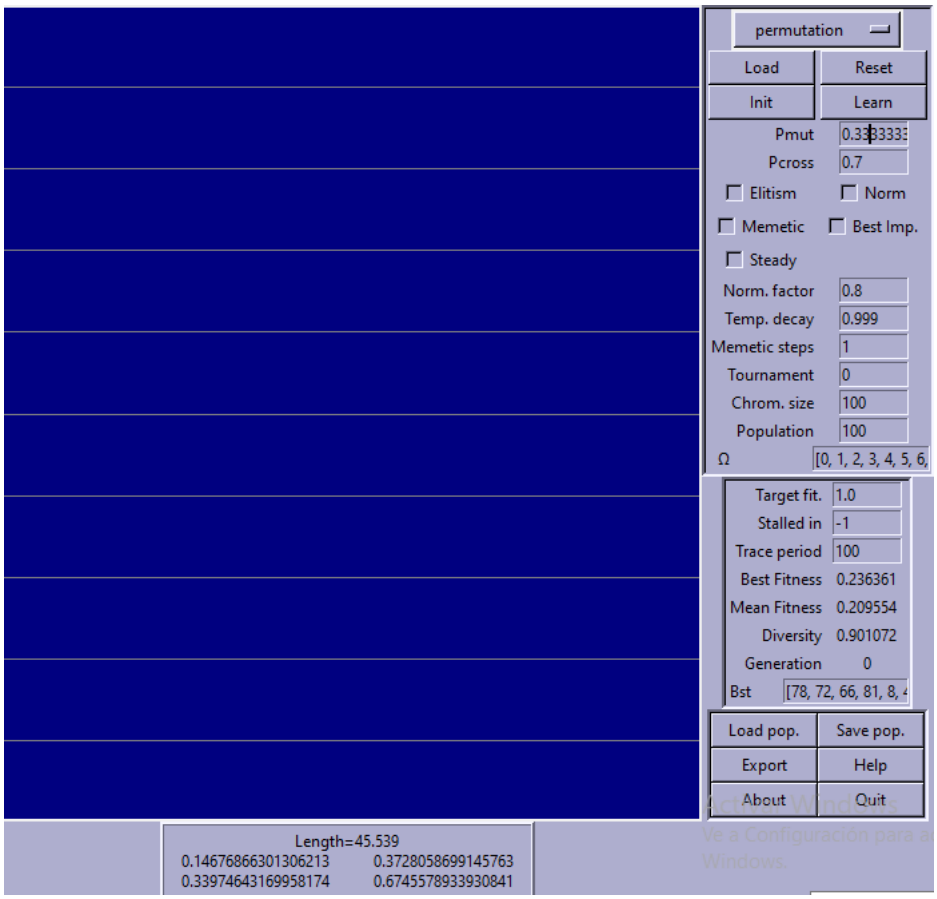

Figura 3.1: Población inicial y configuración

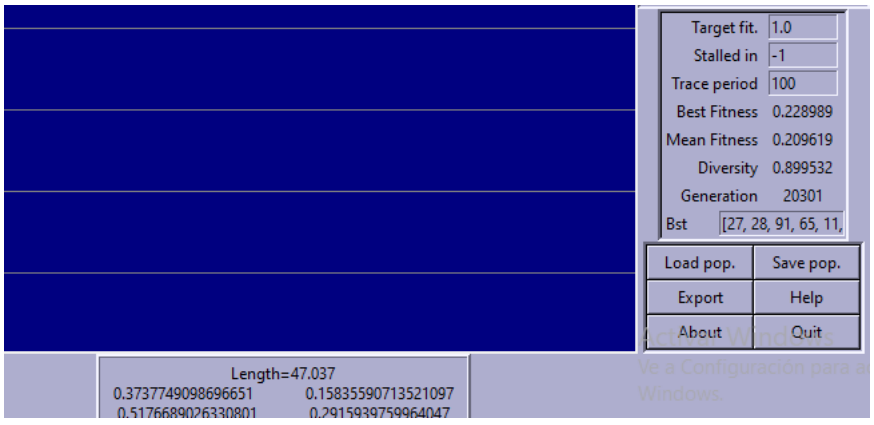

Figura 3.2: Resultados optimización 1

Optimización 2

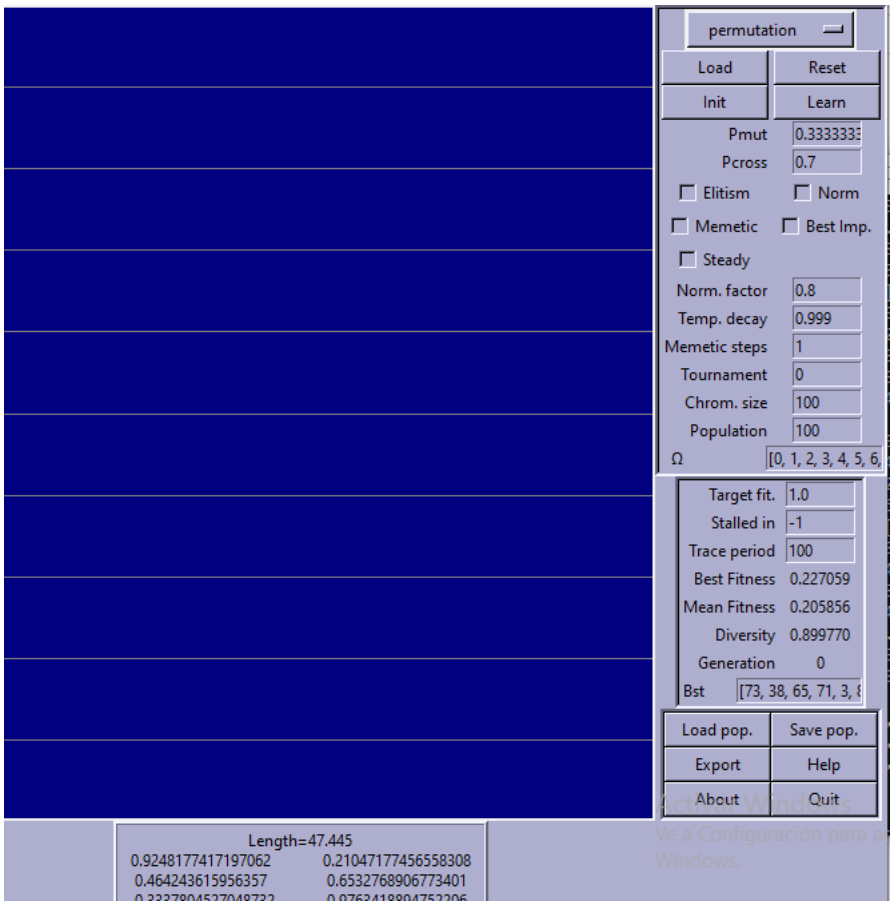

Figura 3.3: Población inicial y configuración

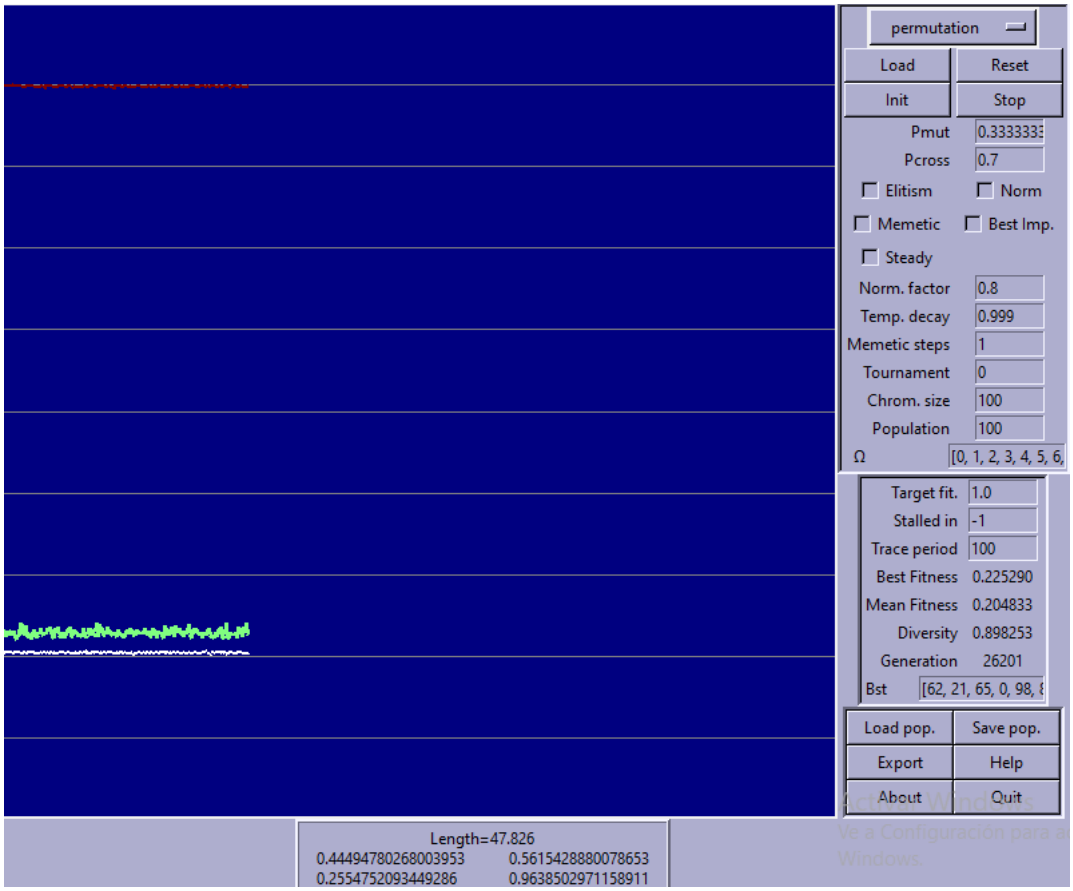

Figura 3.4: Resultados optimización 2

Optimización 3

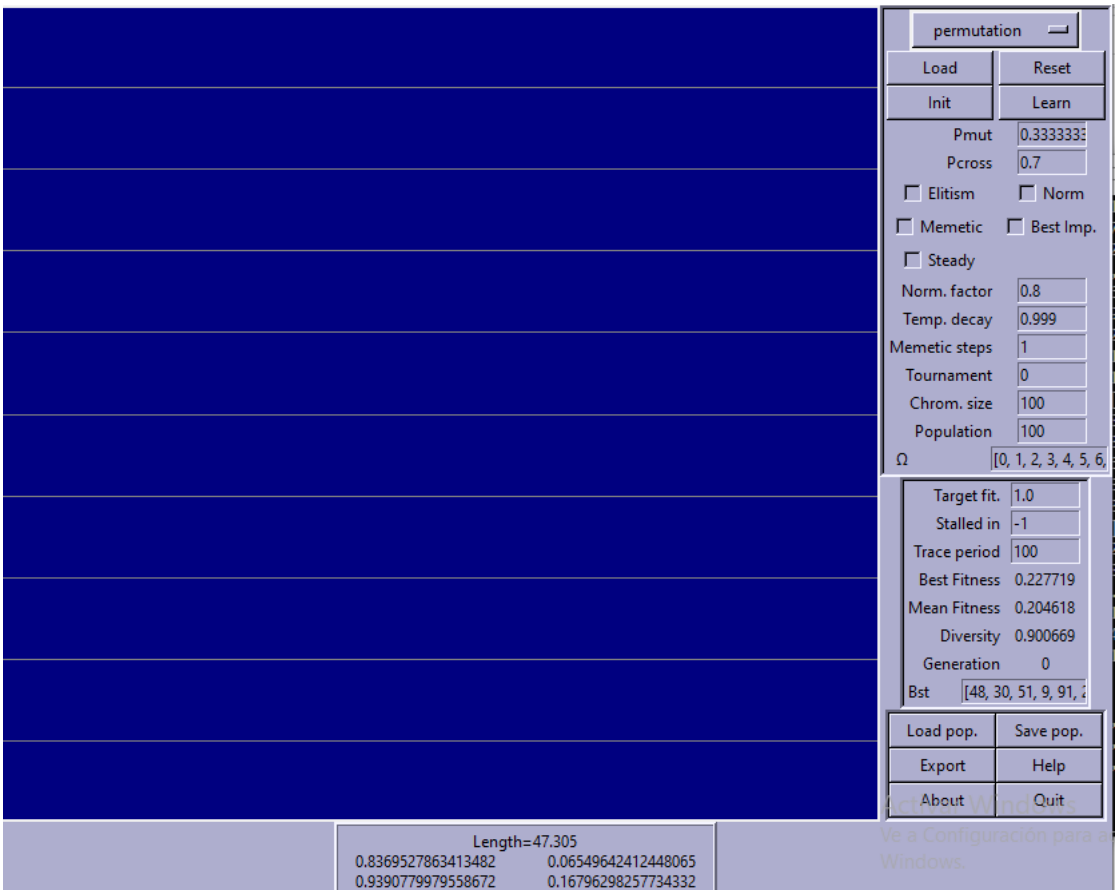

Figura 3.5: Población inicial y configuración

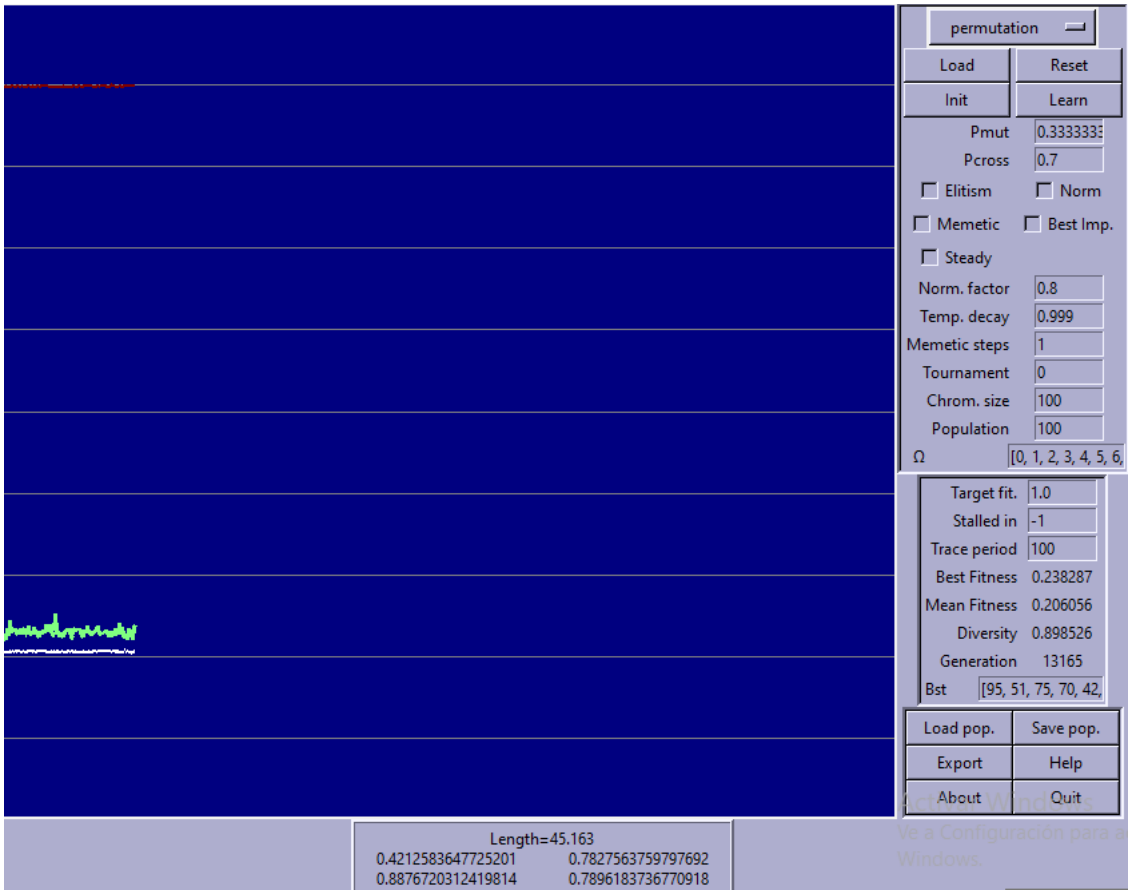

Figura 3.6: Resultados optimización 3

Resultados

En las 3 optimizaciones se ha tenido que parar manualmente en un tiempo considerablemente alto ya que en las 3 el *best fitness* se ha mantenido cercano a 0.2 practicamente desde el inicio. Esto es curioso porque la diversidad de la población es bastante alta. Es posible que se este dando mucha explotación y esté estancado en un mínimo local o en una meseta, así que en la siguiente sección se probará aumentando la exploración aunque la diversidad ya es alta de por sí.

Aún así, en este proceso la mejor ruta encontrada tiene una longitud de 45.163.

3.2. Optimización - Aumento de exploración

Como se ha comentado en el apartado de resultados de la sección anterior, se va a intentar aumentar la exploración ya que es posible que haya un estancamiento en un óptimo local o que haya una meseta.

Por tanto, se ha decidido aumentar la probabilidad de mutación a 0.5 y además poner normalización.

Optimización 1

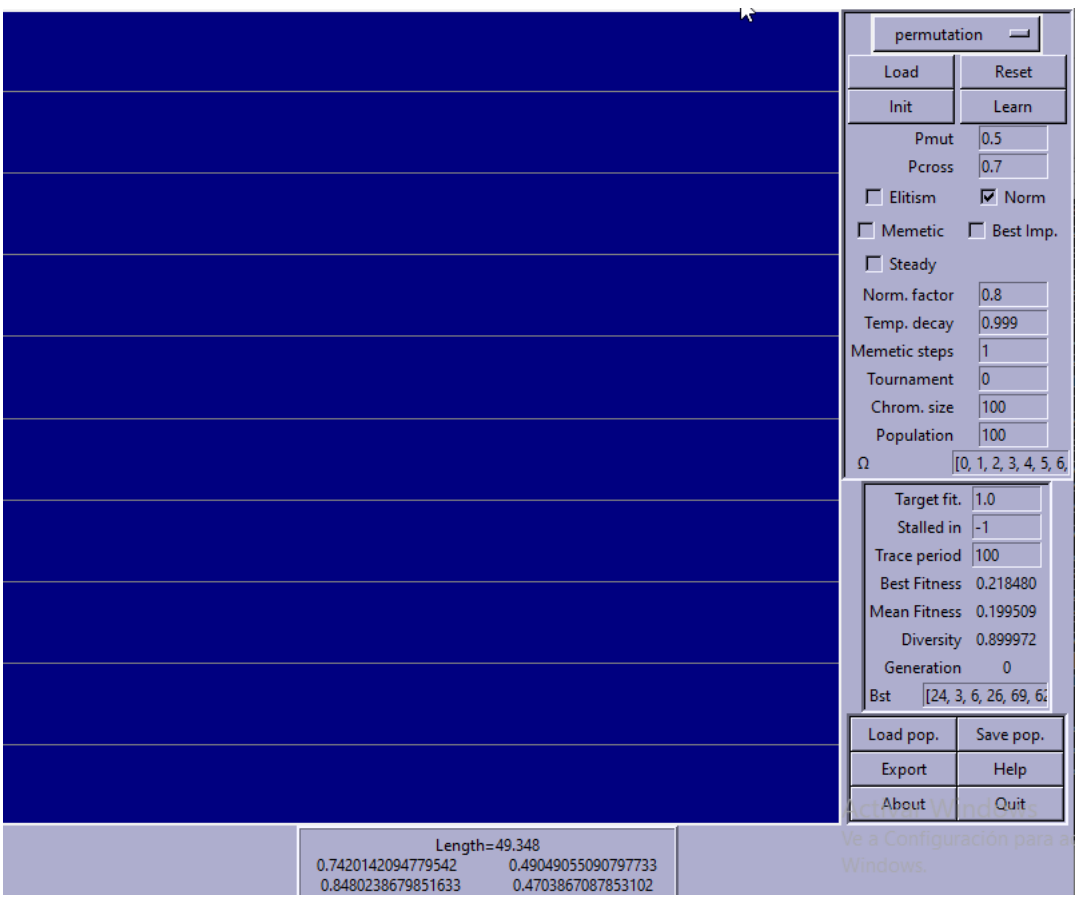

Figura 3.7: Población inicial y configuración

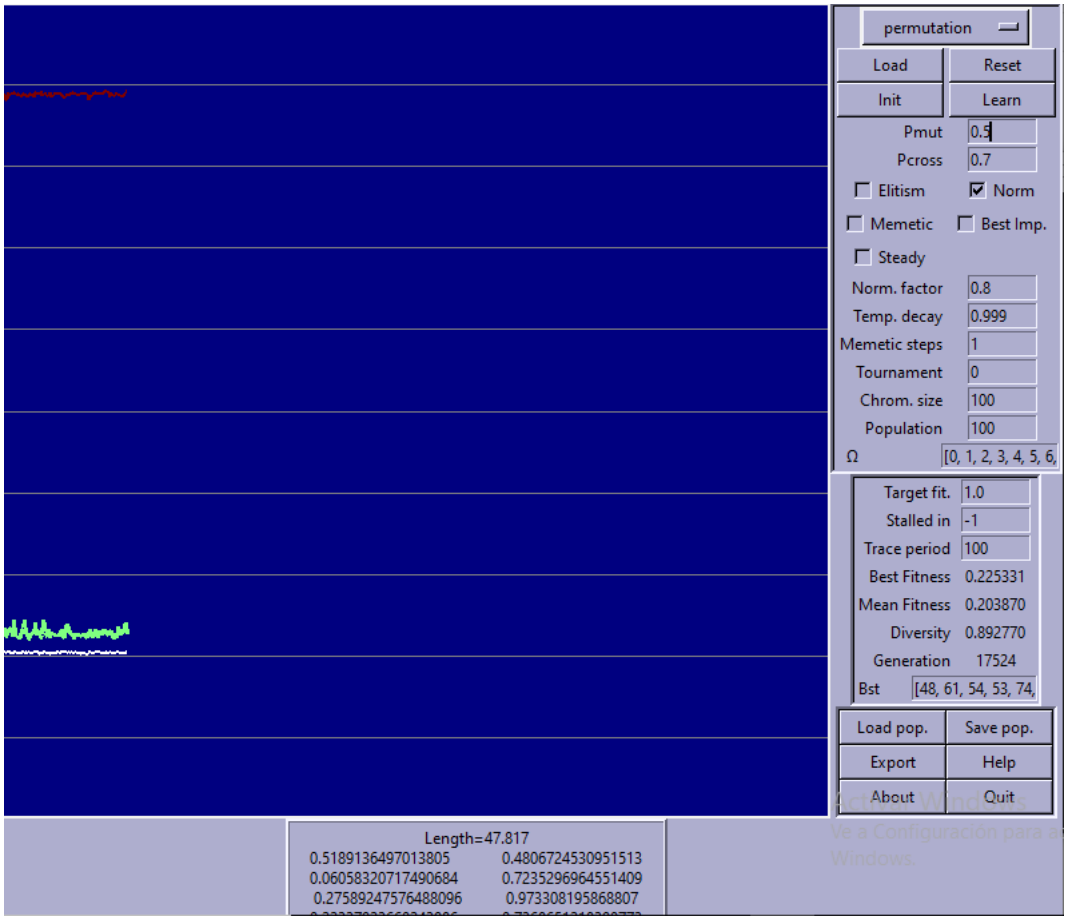

Figura 3.8: Resultados optimización 1

Optimización 2

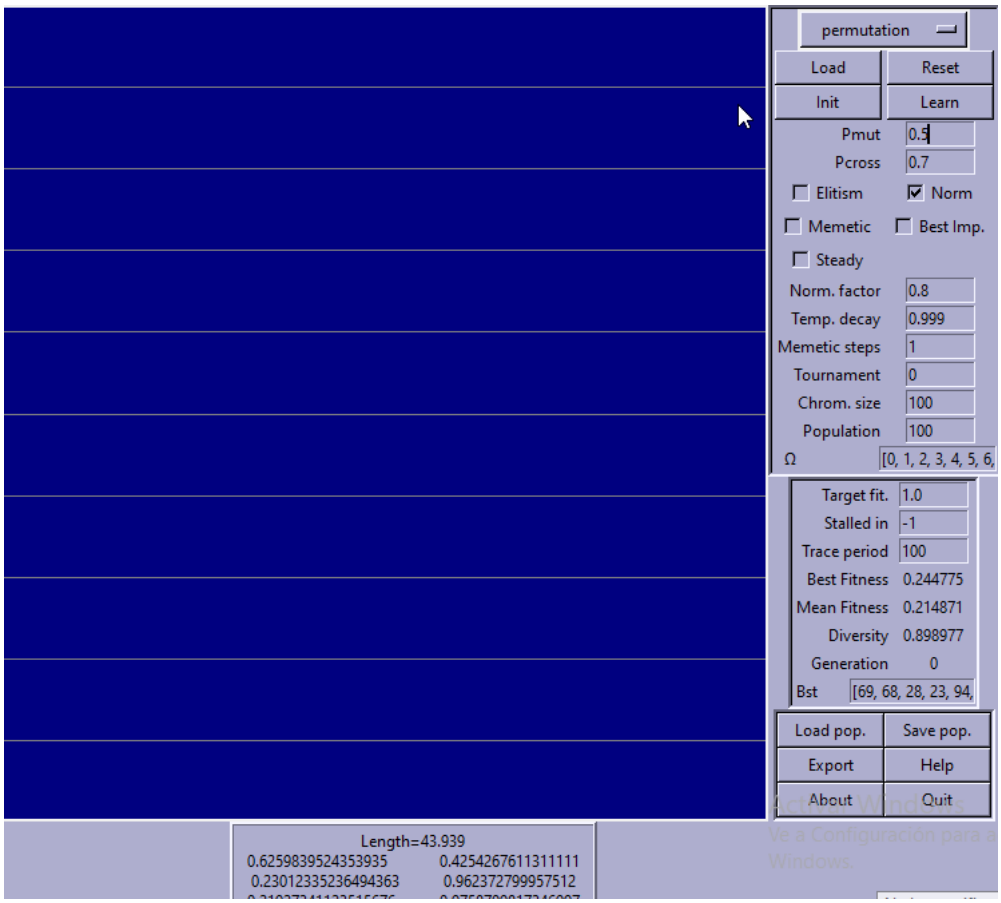

Figura 3.9: Población inicial y configuración

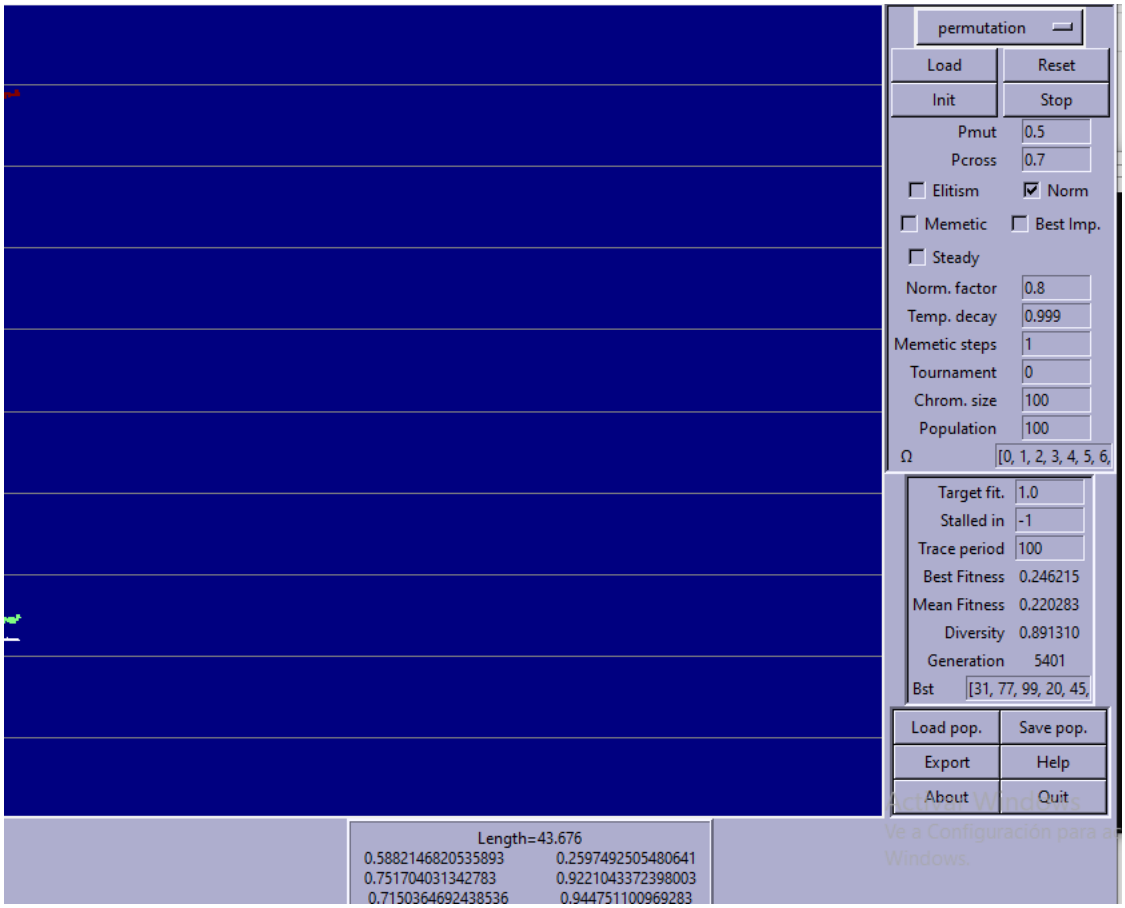

Figura 3.10: Resultados optimización 2

Optimización 3

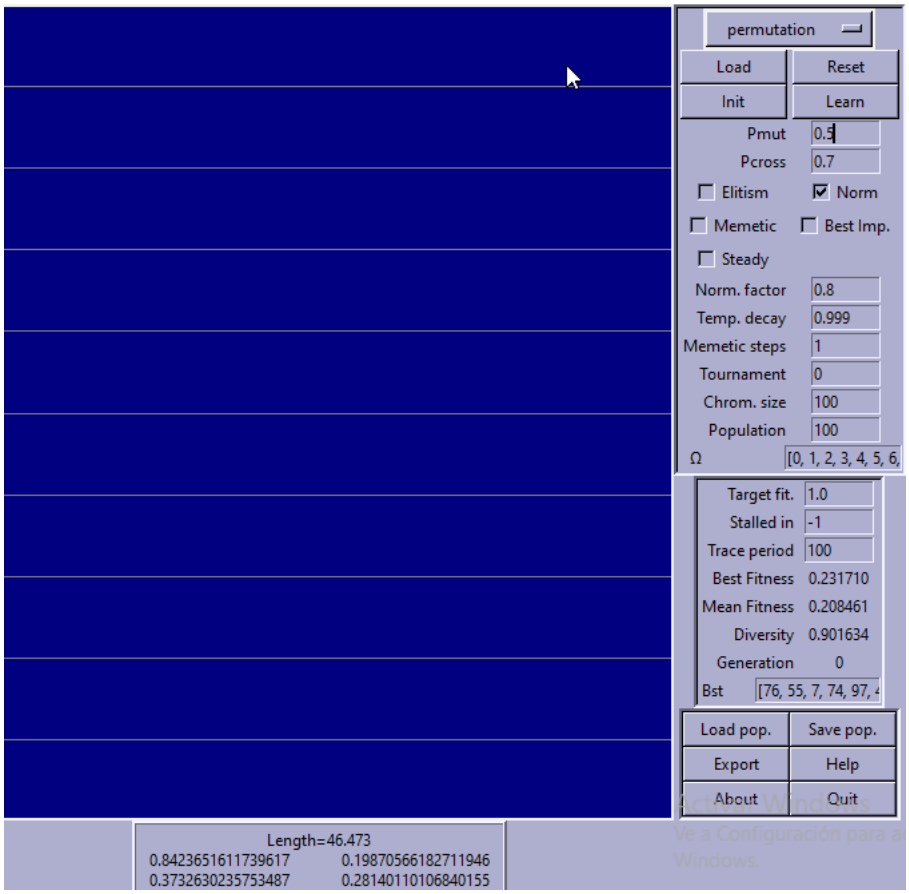

Figura 3.11: Población inicial y configuración

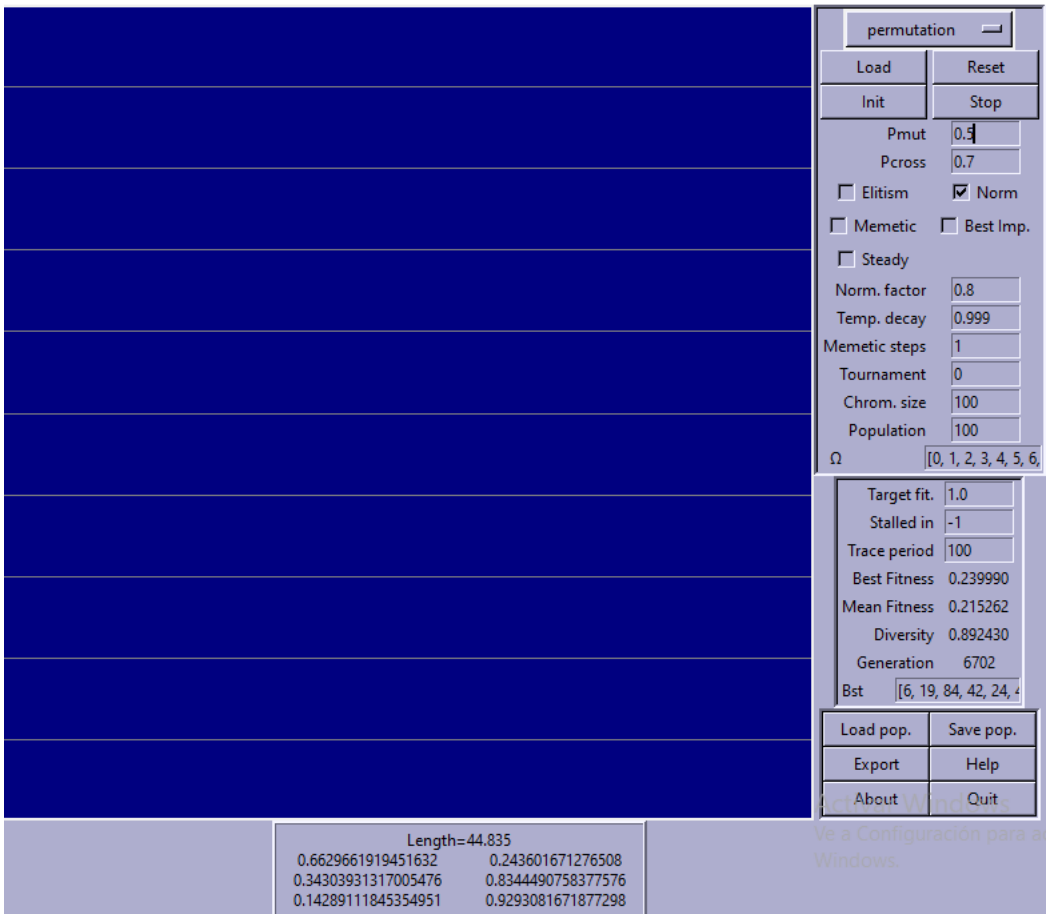

Figura 3.12: Resultados optimización 3

Resultados

Practicamente ocurre lo mismo que con la otra configuración.

Se ha observado que algunas veces si que se obtienen longitudes bajas como, por ejemplo, de 42. Así que se va a probar qué ocurre manteniendo los mejores seleccionando elitismo.

3.3. Optimización - Elitismo

Como se ha comentado en el apartado de resultados de la sección anterior, se va a mantener los mejores individuos con elitismo y que así, además, haya una convergencia más rápida. Aunque con esto caeremos en mínimos locales si es que estamos estancados en uno.

También, reduciremos la mutación a 0.01.

Optimización 1

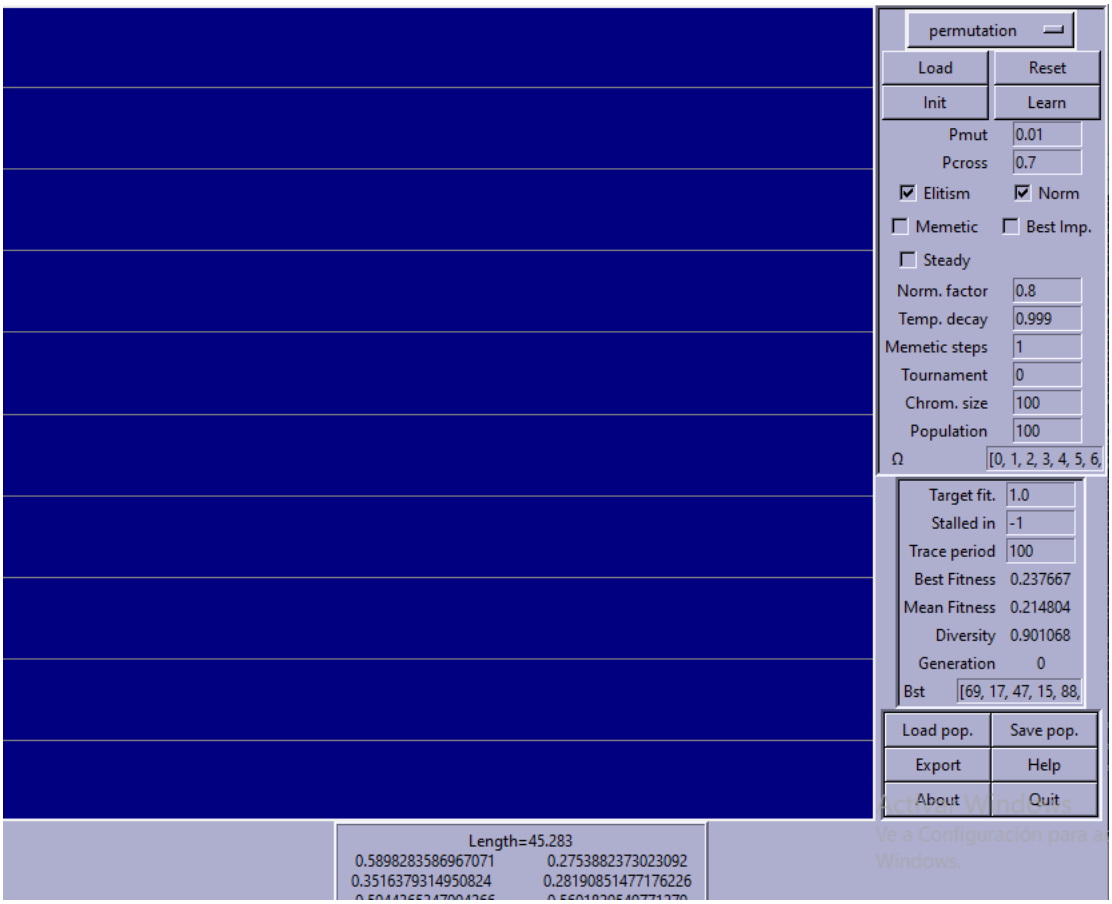

Figura 3.13: Población inicial y configuración

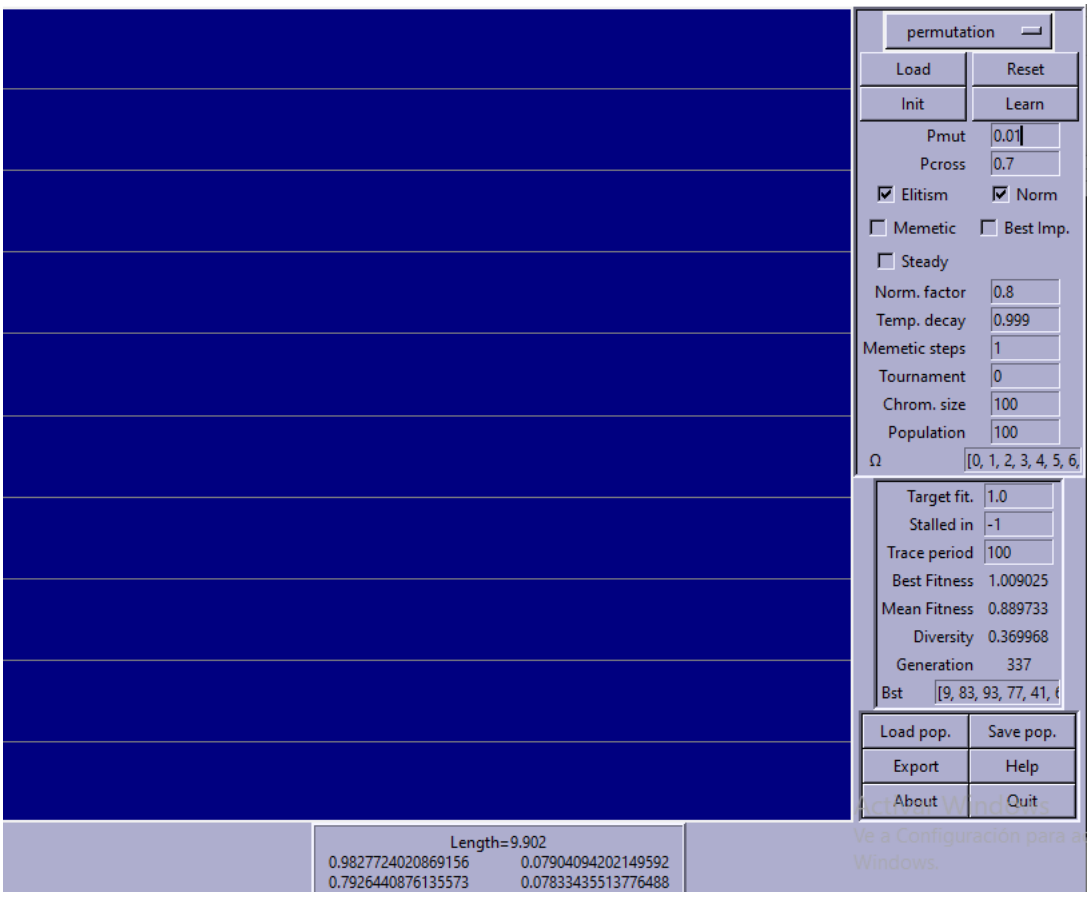

Figura 3.14: Resultados optimización 1

Optimización 2

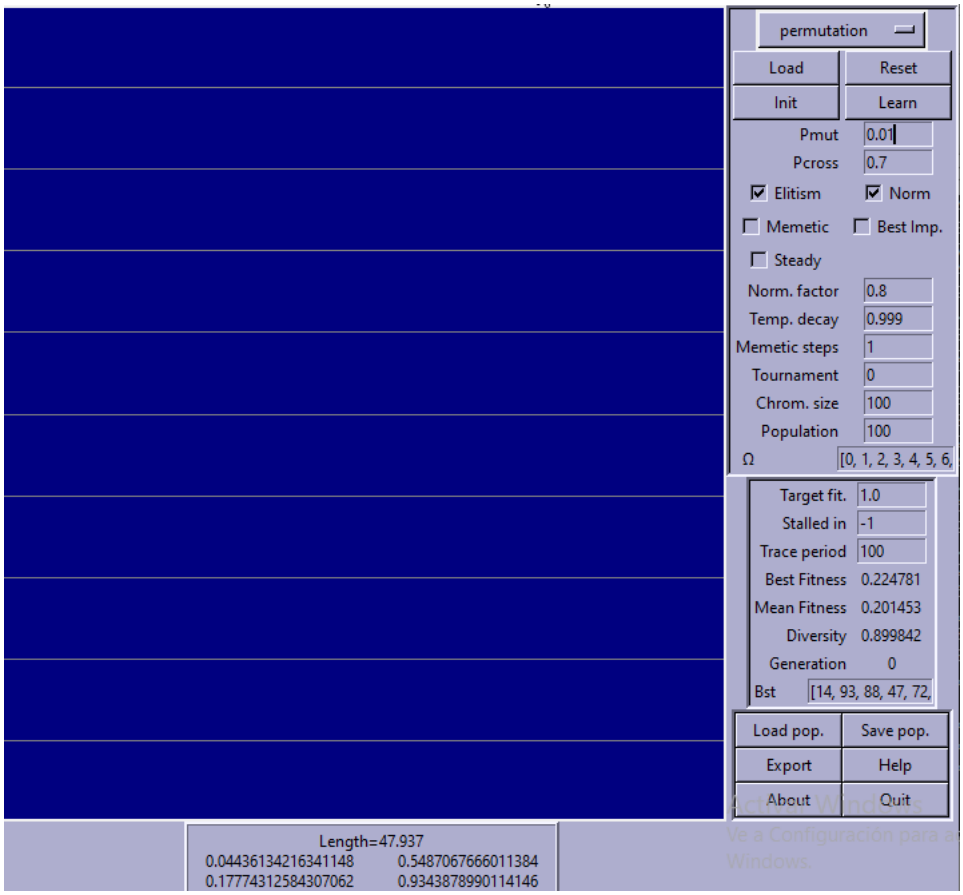

Figura 3.15: Población inicial y configuración

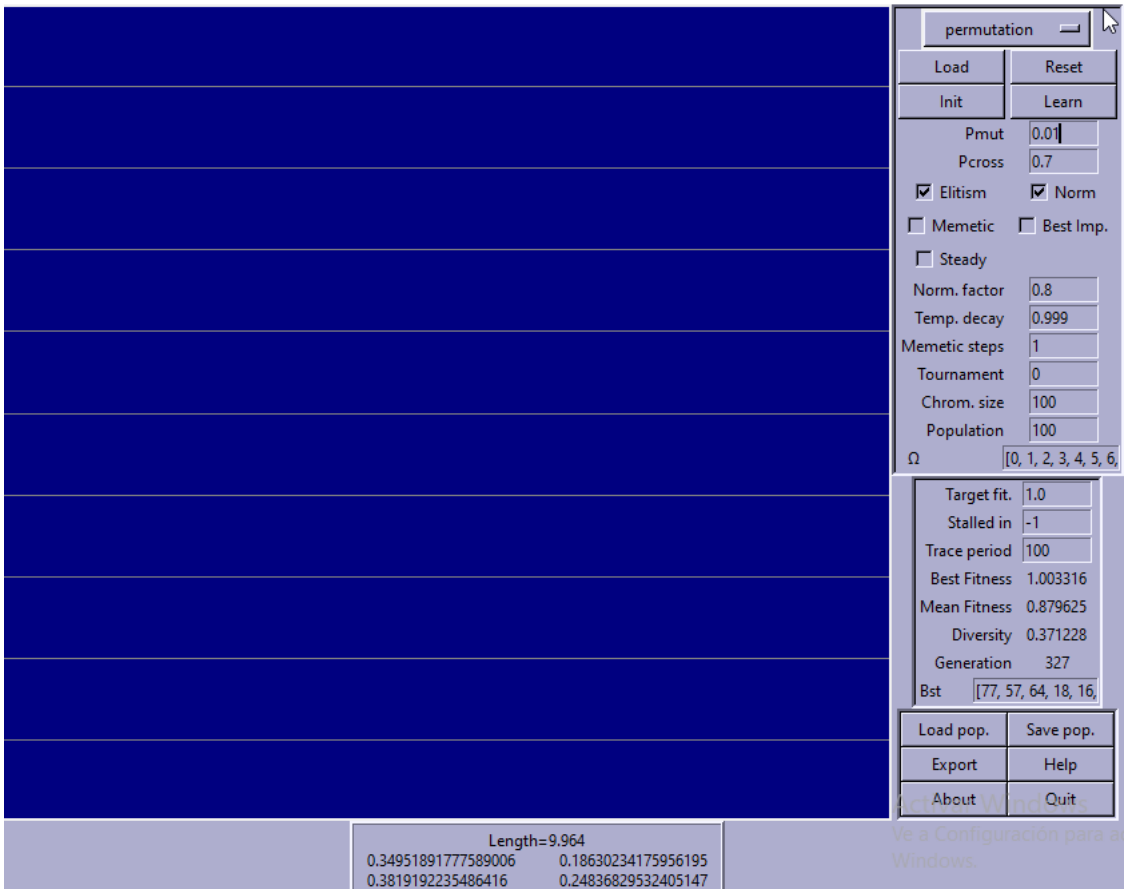

Figura 3.16: Resultados optimización 2

Optimización 3

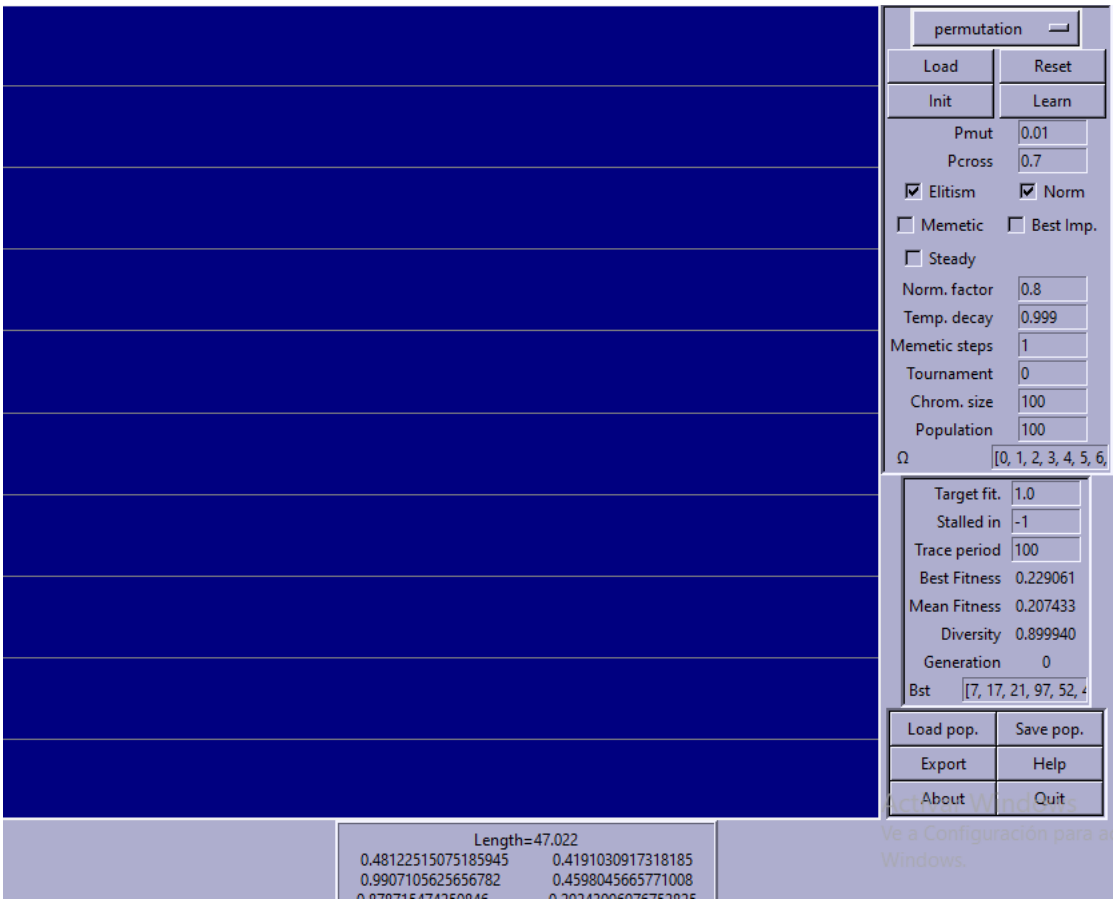

Figura 3.17: Población inicial y configuración

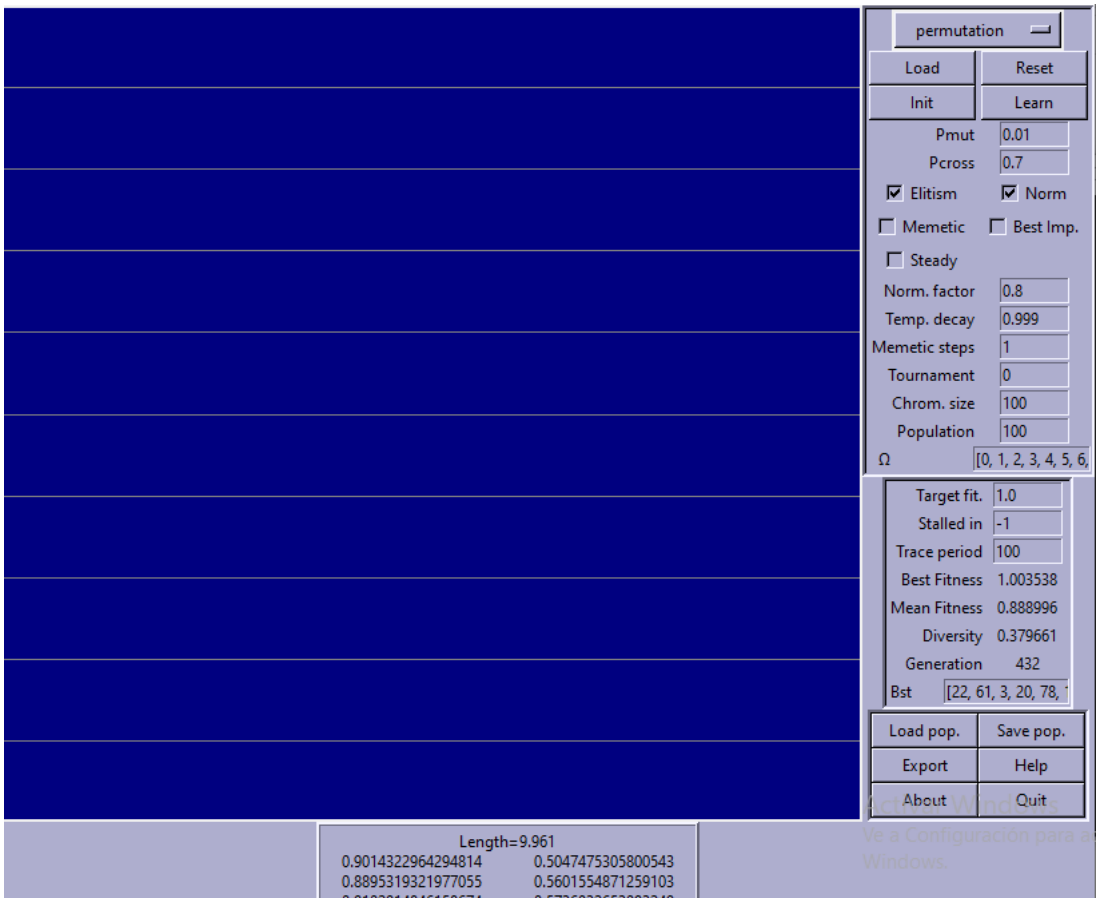

Figura 3.18: Resultados optimización 3

Resultados

Los resultados bajando la probabilidad de mutación y con elitismo son muy pero que muy buenos. Se consigue bajar la longitud de valores cercanos a 45.000 a valores cercanos a 9.900. La ruta más corta encontrada para el reparto de paquetes tiene una longitud de 9.902 y se puede ver en la figura 3.14.

Se considera que esta configuración es la más certera para este problema.

3.4. Resultados

También se requiere contestar a las siguientes cuestiones:

- **¿Encuentra siempre la mejor solución? ¿Por qué?**:
  - No, depende de la configuración de los parámetros (mutación, torneo, etc.) y de la generación de la población inicial. Como se ha podido ver con determinados parámetros se queda estancado en mínimos locales y le es muy complicado salir de ellos, no encontrando una buena solución para este problema. Pero, en este caso, seleccionando elitismo se ha conseguido llegar a un buen resultado.
- **¿Crees que se puede estar seguro de haber encontrado la solución óptima?**

- Nunca se puede estar seguro de haber encontrado la solución óptima si no se sabe la solución de antemano que nos permita corroborarlo.

  Es cierto que esta solución obtenida finalmente es satisfaciente, y para mí es óptima, pero no puedo estar segura al 100 % de no haber caído en un óptimo local.

- **¿Es útil repetir la optimización varias veces? ¿Por qué?**

  - Por supuesto que sí, porque como se ha podido ver con la misma configuración podemos obtener distintas longitudes finales ya que hay muchos parámetros que hacen que las poblaciones sean distintas: inicialización de la población inicial, mutación, etc.. Es la gracia de los algoritmos genéticos: no siempre obtenemos los mismos resultados.

- **Calcula el número de soluciones posibles y el tiempo que tardaría una máquina en encontrar la óptima probando todas las soluciones si generar cada una y comprobar su longitud le llevara 1 microsegundo (en un ordenador actual en python y con multiproceso lleva unos 100 microsegundos)**

  - La solución es $100! * 10^{-6}$