Universidad
Carlos III de Madrid

Grado en Ingeniería Informática

Curso 2021-2022

Algoritmos Genéticos y Evolutivos

Práctica 2
Calibración de motores automática mediante Estrategias Evolutivas

Jorge Rodríguez Fraile 100405951 Grupo 83 100405951@alumnos.uc3m.es

Índice

Índice de figuras

Índice de tablas

Introducción

Este proyecto trata sobre encontrar una aproximación mediante estrategias evolutivas al problema de calibrar motores de un brazo robótico utilizando solo un láser, una cámara y un objeto de referencia. El robot deberá ser preciso en sus movimientos, dado que será utilizado para realizar soldaduras de alta precisión.

Para comenzar se trabajará sobre un brazo compuesto solo por 4 motores para podernos acercar al problema de una manera más gradual, en cuanto se haya comprobado que es factible esta simplificación del problema se pasara a un caso más realista con un brazo de 10 motores.

Codificación y Función de fitness

Al emplear estrategias evolutivas la codificación de los individuos de este problema consistirá en un vector de codificación compuesto por 4 o 10 números reales (según el número de motores) y un vector de varianzas de la misma longitud que el de codificación.

Los valores del vector de codificación estarán centrados en 0 representando que no se ha realizado ningún giro y podrán rotar tanto como quieran en sentido positivo como negativo, aunque llegado un punto girar mucho en un sentido equivale a movimientos más cortos. Lo ideal es que los valores se muevan entre -180 y 180 grados representando un giro completo de 360 grados, aunque no se restringirán los valores que puede tomar.

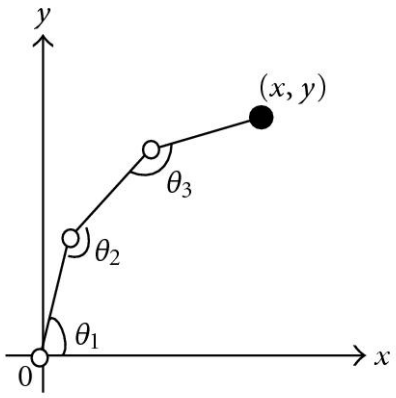

Figura 1: Diagrama brazo robótico

En cuanto a la función de fitness que se empleará para saber si una solución es buena o no, esta tendrá en cuenta el error que comente el soldador en un simulador, según como de dispersos y desviados sean con respecto al centro dará un valor mayor cuanto peor lo haga y menor cuanto más preciso sea. Por lo que el objetivo del problema es minimizar la función de fitness.

Para conocer el error para cada una se las soluciones se hará una llamada get a un servicio que se encuentra en un servidor de la universidad, http://memento.evannai.inf.uc3m.es/age/robot4?, seguido de cX=GradosDeX para cada uno de los 4 motores y separados por un &, en el caso de los 10 motores será igual, aunque con otro servicio http://memento.evannai.inf.uc3m.es/age/robot10? y 10 valores de grados.

Programación Estrategia Evolutiva (1+1)

Para este proyecto entre las posibles implementaciones de las estrategias evolutivas se ha elegido la (1+1)-EE y a continuación se describirán las características de esta.

El programa se ha desarrollado en el lenguaje Python, el procesamiento principal del programado se encuentra en la función *EE1plus1(c, cycles, test_iteration)*, donde recibirá como parámetros el valor por el que se multiplicaran las varianzas, el número de generaciones máximas del programa y la iteración en la que se encuentra el programa (se incluye para realizar las pruebas de ejecutar múltiples veces).

El proceso que sigue esta función es el siguiente:

1. Generar el individuo inicial.
2. Evaluar el individuo generado de la población inicial.
3. Repetir hasta cumplir el criterio de convergencia:
   a) Generar una nueva solución a partir del único individuo de la población, mutando el vector de codificación del descendiente.
   b) Evaluar el individuo generado.
   c) Eliminar el individuo cuyo valor de adecuación sea mayor, tratamos de minimizar el fitness.
   d) Si el individuo que queda es el nuevo, se aumenta la frecuencia de éxitos y si no se disminuye.
   e) Mutar el vector de varianzas del individuo elegido de acuerdo con la regla 1/5.

Además de lo relacionado con la generación de individuos, también se ha incluido código que nos permita almacenar la salida de los modelos para poderlos evaluar.

Operadores genéticos

Población inicial

El individuo inicial es generado aleatoriamente, su vector de codificación siguiendo una gaussiana centrada en 0 y con desviación 100, de esta manera los valores serán tanto positivos como negativos y mayoritariamente centrado en 0, por otro lado el vector de varianzas se genera mediante valores reales positivos entre 5 y 20, aunque se realizarán pruebas con valores mayores.

Este proceso es realizado por la función *initial_individual()*, devolviendo un individuo en forma de matriz NumeroDeMotores x 2.

Mutación

Para los algoritmos evolutivos este operador se realiza en dos partes una para el vector de codificación y otra para el de varianzas.

Para el vector de codificación, el nuevo individuo tendrá como valores los del progenitor más un valor de la normal según la varianza de ese valor para el padre, es decir $x_s = x_p + N_0(\sigma_p)$, esta operación se repite para todos los valores del vector.

En cuanto al vector de varianzas, se realizará sobre el individuo seleccionado para la siguiente generación, y consiste en modificar sus varianzas en función de la regla 1/5, que se guia por el porcentaje de mejoras de las últimas generaciones.

Esta regla se aplicará teniendo en cuenta las 10 últimas generaciones, las variaciones vendrán dadas por:

- Si el número de mejoras es superior al 20 %, se aumenta la varianza $\frac{\sigma}{c}$.
- Si el número de mejoras es inferior al 20 %, se reduce la varianza $\sigma \cdot c$.
- Si el número de mejoras es igual al 20 %, se mantiene σ.

Lo que dice la regla es que si mejora con frecuencia es que todavía estamos lejos de la solución óptima y si no mejoramos es que estamos cerca y nos moveremos poco a poco.

Esta funcionalidad se encuentra en la función *mutate_codification(individual)* para el vector de codificación, aplica la operación para codificación y devuelve al sucesor, y en *individual_next_generation(individual, son, improvements_counter, iteration, c)* para el vector de codificación, se encuentran separadas porque hasta que no se conoce el individuo de la siguiente generación no se muta.

Inserción y Remplazo

Al tratarse del tipo (1+1) la población está formada por un solo individuo y también habrá un solo individuo, por lo que entre el individuo de la población y el nuevo se elegirá el mejor, en este caso el de menor fitness.

Cuando se elija al descendiente se considera una mejora y se tendrá en cuanta en la frecuencia de mejoras, si no se considera que no ha mejorado. Tras elegir el individuo que formará la población se mutará su vector de varianzas como se ha elegido antes.

La función que recoge la inserción y remplazo, además de la mutación de las varianzas es la que se mencionó antes, *individual_next_generation(individual, son, improvements_counter, iteration, c)*, que evalúa ambos individuos, elige el mejor, actualiza el contador de mejoras y aplica la mutación en función del parámetro c que se le pasa. La función devuelve el individuo elegido, su fitness y el contador de mejoras actualizado.

Aproximación 4 motores

Comenzaremos realizando pruebas en la versión simplificada del problema, en el que solo se consideraran 4 motores para el brazo soldador.

Estos primeros modelos que se generaran consistirán en la variación del valor c, el que determina cuanto cambiara la varianza de los individuos. Es conocido que el valor debe ser inferior a 1 y que el recomendable es 0,82, pero en busca de mejores resultados se ha probado además con 0,98, 0, 92, 0,72 y 0,62. No se han variado más estos valores, porque la selección de valores más bajos provocaría que los valores de varianza se movieran muy rápido.

Todos los modelos se han ejecutado durante 5000 generaciones, que son 10000 evaluaciones, y se han realizado 10 ejecuciones para cada modelo para evitar el sesgo de la aleatoriedad a la hora de generar los valores aleatorio.

Los modelos han sido nombrados siguiendo el siguiente patrón evolution_strategy_XX_YY_A-B_C:

- **XX:** Indica el número de motores que se consideran y para los que se ajustaran los ángulos.
- **YY:** Indica el valor de c, el cual determina como se modificarán las varianzas.
- **A-B:** Indica el rango de valores entre los que se generan aleatoriamente las varianzas.
- **C:** Indica el número de generaciones que tiene en cuanta para medir la frecuencia de mejora.

En la siguiente gráfica se pueden ver los resultados de fitness obtenidos en las 10 ejecuciones para los 5 modelos, junto a la media de cada modelo que es la que utilizaremos para comparar.

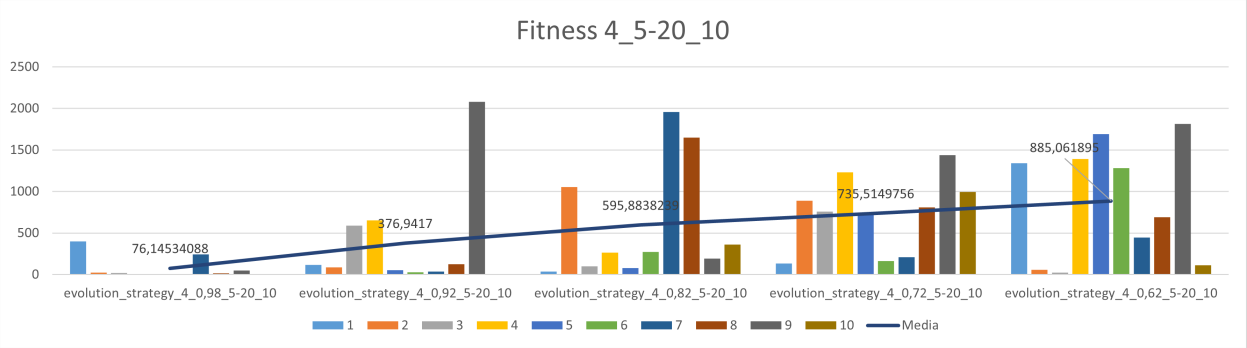

Figura 2: Valores de Fitness: 4_5-20_10

Modelo	1	2	3	4	5	6	7	8	9	10	Media
4_0,98_5-20_10	399,8967664	21,88903367	20,89403404	1,989918114	2,984877171	**0,994959057**	243,1577278	13,92939149	49,74695184	5,969749305	**76,14534088**
4_0,92_5-20_10	115,4108682	86,56039686	587,9440061	654,4753316	51,73723573	25,86868199	35,81824234	125,3603064	2081,267141	4,974790248	376,9417
4_0,82_5-20_10	34,8234203	1053,037537	98,49807057	263,6453657	79,59564625	270,6192328	1955,329393	1649,132003	193,0126789	361,1448922	595,8838239
4_0,72_5-20_10	134,6082641	887,2371193	758,3690516	1232,534402	731,1155627	161,1833393	208,9351831	808,6563643	1438,833559	993,6769101	735,5149756
4_0,62_5-20_10	1342,19067	58,71220968	24,87415781	1391,691525	1691,348176	1280,418402	443,7601042	692,0338572	1812,167475	113,4223726	885,061895

Tabla 1: Fitness iniciales 4 motores

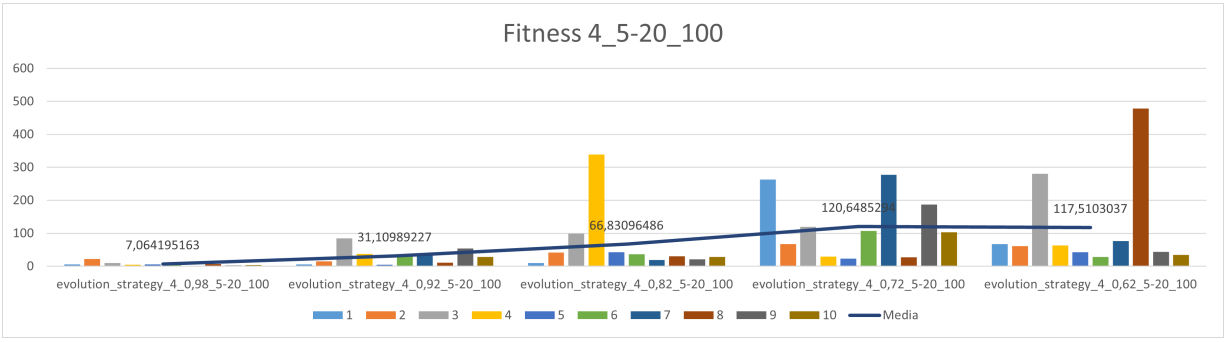

Figura 3: Valores de Fitness: 4_5-20_100

Aproximación 10 motores
Resultado
Conclusiones