Documento Final
Proyecto PGILearn

Marcos Sander

Tutor
Facundo Benavides

Instituto de Computación
Facultad de Ingeniería
Universidad de la República
Montevideo-Uruguay

2015

Resumen

Este documento describe el trabajo realizado en el proyecto de grado PGILearn, en el cual se aborda la temática del aprendizaje por demostración/imitación en robots.

Naturalmente tanto los hombres como los animales utilizan la observación y la imitación como medio de aprendizaje de nuevos comportamientos o habilidades. Esta forma de aprendizaje aplicada en los robots trae consigo algunas ventajas como son una mayor eficiencia en el proceso de aprendizaje de nuevos comportamientos y una mayor autonomía del robot, permitiendo que una persona sin conocimiento de programación pero sí de la habilidad en cuestión, pueda enseñarle al mismo a ponerla en práctica. Debido principalmente a las ventajas mencionadas que tiene el uso de este método de aprendizaje, es que desde hace algunos años se observa una creciente aplicación del mismo en el campo de la robótica.

En el presente proyecto se llevó a cabo primeramente un estudio del estado del arte del aprendizaje por demostración/imitación. Luego se aplicaron esos conocimientos a la resolución de un problema de aprendizaje de habilidades (de un gesto de atajada y gesto de pateo de pelota como en el fútbol, y saludo), implementadas por secuencias motoras del robot que participa en el rol de aprendiz, a través de la demostración de las mismas por parte de una persona, que participa en el rol de maestro. Entre otras dificultades que surgen con el enfoque planteado se destaca la del problema de correspondencia, que se debe a la diferencia morfológica entre el aprendiz y el maestro, lo que implica que probablemente una tarea demostrada por el maestro no pueda ser realizada por el aprendiz sin previa adaptación de la misma.

En la demostración de la tarea se utilizó equipamiento del laboratorio de Biomecánica de la Facultad de Medicina, el cual permite captar posiciones de las articulaciones de la persona a medida que la misma demuestra la tarea. Para adaptar la demostración de la persona al robot se utilizó un proceso evolutivo con algoritmos genéticos. En la evaluación de las distintas secuencias que realizan la habilidad se utilizó un simulador físico, que simula tanto el robot como su entorno. Cabe destacar que para evaluar a las distintas secuencias solamente se utilizó el concepto de estabilidad, o sea que una secuencia es valorada positivamente si al ejecutarla en el robot este no se cae. Por último se hicieron pruebas de concepto en el robot de forma de evaluar el resultado obtenido al aplicar el algoritmo de aprendizaje. Se observó que el robot realiza los movimientos de la tarea en cuestión aunque no realiza la tarea de forma estable (probablemente debido a las diferencias que existen entre el entorno simulado y la realidad).

Como resultado se observó que el sistema construido permite realizar el aprendizaje por demostración/imitación de secuencias motoras. Esas secuencias aprendidas en un entorno simulado deben ser posteriormente adaptadas al robot real para que el mismo pueda realizarlas de forma correcta.

Agradecimientos

Después de tan larga jornada quisiera agradecer a las personas que de una forma u otra colaboraron e hicieron posible que este proyecto pudiera ser realizado.

En primer lugar a mi tutor Facundo Benavides por los consejos, guía, paciencia y por el tiempo dedicado a esta empresa.

A Andrés Aguirre por su participación inicial en el proyecto, aporte de ideas y discusiones que ayudaron a la realización del mismo.

A Federico Andrade quien gentilmente se ofreció para hacer las grabaciones de las demostraciones.

A la gente de Biomecánica de la Facultad de Medicina de la UdelaR: Patricia Polero, Gabriel Fábrica y Darío Santos que nos permitieron utilizar el equipamiento y lugar para la grabación de las demostraciones, y también por el apoyo y conocimiento brindado en el uso de las herramientas.

A mi amada esposa Amy por el aguante todo este tiempo y ayuda en la corrección de este documento.

A mis padres Marta y Raúl, que a lo largo, no solamente de este proyecto sino de la vida siempre me dieron su apoyo incondicional.

A familiares, amigos y compañeros que de una forma u otra me acompañaron durante la realización del proyecto.

A todos, gracias!!

Índice general

1. Introducción **1**
1.1. Objetivos del proyecto . . . 1
1.2. Organización del documento . . . 2

2. Marco de trabajo **3**
2.1. Aprendizaje por demostración-imitación . . . 3
2.1.1. Definición . . . 3
2.1.2. Procesos de aprendizaje . . . 3
2.1.2.1. Aprendizaje con proceso de automejora . . . 4
2.1.3. Métrica . . . 4
2.1.3.1. Similaridad de caminos . . . 5
2.1.4. Mapeo maestro-aprendiz . . . 6
2.1.4.1. Demostración . . . 7
2.1.4.2. Imitación . . . 8
2.1.5. Derivación de una política . . . 9
2.1.5.1. Función de mapeo . . . 9
2.1.5.2. Modelos de sistema . . . 10
2.1.5.3. Planes . . . 11
2.2. Algoritmos genéticos . . . 12
2.2.1. Representación de los individuos . . . 12
2.2.2. Función de fitness . . . 12
2.2.3. Operadores genéticos . . . 13
2.2.3.1. Selección . . . 13
2.2.3.2. Cruzamiento . . . 14
2.2.3.3. Mutación . . . 15
2.2.3.4. Otros operadores . . . 15
2.2.4. Búsqueda en espacio de soluciones . . . 15
2.3. Técnicas de representación y manipulación de los datos . . . 16
2.3.1. PCA . . . 17
2.3.2. Splines . . . 18

3. Problema y solución propuesta **21**
3.1. Introducción . . . 21
3.2. Mapeo maestro-aprendiz . . . 23
3.2.1. Mapeo desde el maestro . . . 23

3.2.1.1. Técnica y sistema utilizado para obtención de datos . . . 23
3.2.1.2. Cálculo de ángulos de las articulaciones . . . 26
3.2.2. Mapeo hacia el aprendiz . . . 28
3.2.2.1. Plataforma Robótica . . . 29
3.2.2.2. Mapeo de datos registrados hacia el aprendiz . . . 30
3.3. Implementación del algoritmo genético . . . 33
3.3.1. Metaheurística utilizada . . . 33
3.3.2. Reducción del dominio de búsqueda . . . 34
3.3.2.1. Cálculo del número de componentes principales . . . 37
3.3.3. Representación de los individuos . . . 38
3.3.4. Generación de individuos de la población inicial . . . 39
3.3.4.1. Inicialización de la matriz de posturas . . . 39
3.3.4.2. Inicialización del vector de ejecución de posturas . . . 40
3.3.4.3. Mapa de estabilidad . . . 40
3.3.5. Función de fitness . . . 44
3.3.5.1. Simulador Bioloid Control . . . 45
3.3.5.2. Función de fitness por tiempo . . . 46
3.3.5.3. Función de fitness por posturas . . . 47
3.3.6. Operadores genéticos . . . 47
3.3.6.1. Operador de cruzamiento . . . 47
3.3.6.2. Operador de mutación . . . 48
3.3.6.3. Operador de selección . . . 51
3.3.7. Condiciones de terminación . . . 51
3.3.7.1. Cantidad de generaciones . . . 51
3.3.7.2. Fitness promedio y mejor individuo . . . 51

4. Entrenamiento, testeo y resultados 53
4.1. Puesta a punto del algoritmo . . . 53
4.1.1. Ajustes y modificaciones realizados . . . 53
4.1.1.1. Testeo y modificación de operadores . . . 53
4.1.1.2. Testeo y modificación de la función de evaluación . . . 56
4.1.1.3. Búsqueda de entornos de valores de parámetros para calibración . . . 56
4.1.1.4. No determinismo de las soluciones . . . 56
4.1.2. Conclusiones . . . 57
4.2. Aprendizaje del algoritmo . . . 57
4.2.1. Calibración . . . 57
4.2.1.1. Aproximación . . . 58
4.2.1.2. Refinamiento . . . 59
4.2.2. Testeo de parámetros aprendidos . . . 59
4.2.3. Conclusiones . . . 63
4.3. Testeo con otras tareas . . . 63
4.3.1. Resultados . . . 64
4.3.2. Conclusiones . . . 64
4.4. Pruebas en robot . . . 64
4.4.1. Determinación del parámetro de velocidad . . . 66
4.4.2. Resultados y conclusiones . . . 67

5. Discusión, conclusiones y trabajo a futuro **69**
5.1. Discusión 69
5.1.1. Proceso de aprendizaje por demostración-imitación 69
5.1.2. Herramientas utilizadas 70
5.2. Conclusiones 70
5.3. Trabajo a futuro 71
5.3.1. Aprendizaje de otras tareas 71
5.3.2. Aplicación de otro simulador/plataforma robótica 71
5.3.3. Utilización de medio distinto para la obtención de datos 71
5.3.4. Automatización de todo el proceso de aprendizaje 71

Índice de algoritmos

2.1. Aprendizaje con proceso de auto-mejora 5
3.1. Resumen de metaheurística gGA 34
3.2. Implementación de modificaciones sobre la metaheurística gGA 34
3.3. Cálculo del número de componentes principales 37
3.4. Inicialización del vector de ejecución de posturas 41
3.5. Algoritmo de estabilidad para un punto de R^3 42
4.1. Criterio de selección de algoritmo 58

Índice de figuras

2.1. Aprendizaje con proceso de auto-mejora 5
2.2. Mapeo de la ejecución del maestro al aprendiz 6
2.3. Matriz de mapeos maestro-aprendiz 7
2.4. Categorización de enfoques de aprendizaje de políticas 9
2.5. Representación del individuo 13
2.6. Operadores de cruzamiento 15
2.7. Operadores de mutación 16
2.8. Ilustración de puntos en R^2 y reproyecciones sobre el robot 17
2.9. Comparación de métodos de reducción no lineales 18
2.10. Aplicación de splines a la función seno y raíz cuadrada 20

3.1. Aprendizaje por imitación utilizando algoritmos genéticos 22
3.2. Maestro realizando la tarea que luego será aprendida por el aprendiz 24
3.3. Posición de los marcadores colocados sobre el maestro 25
3.4. Ángulos generados a partir de la observación del maestro 28
3.5. Hardware del Bioloid Expert Kit 29
3.6. Robot Bioloid Expert humanoide ensamblado 30
3.7. Mapeo de ángulos desde el maestro al aprendiz 31
3.8. Comparativo de ángulos reales y obtenidos al aplicar PCA 36
3.9. Representación del individuo solución del algoritmo genético 38
3.10. Algoritmo de estabilidad 43
3.11. Matriz de estabilidad y trayectoria de la secuencia de posturas 44
3.12. Entorno gráfico del simulador Bioloid Control 46
3.13. Operador de cruzamiento 48
3.14. Mutación polinomial 49
3.15. Mutación de un punto y alisado de la curva utilizando splines 50

4.1. Calibración algoritmo genético 60
4.2. Testeo de secuencia 1 61
4.3. Testeo de secuencia 2 62
4.4. Testeo de tarea de saludo 65
4.5. Testeo de tarea de gesto de pateo 65
4.6. Programa Motion Editor de Bioloid 66
4.7. Gráfica de velocidades y tiempos de ejecución 67

Índice de cuadros

3.1. Matriz de posiciones de marcadores de dimensión $n \times m$ 26
3.2. Mapeo de ángulos desde el maestro al aprendiz 32
3.3. Errores promedio y desviación estándar al aplicar PCA 35
3.4. Porcentaje de varianza retenida 37

4.1. Implementaciones de operadores testeadas 54
4.2. Valores de parámetros utilizados en la etapa de aproximación 58
4.3. Valores de parámetros utilizados en la etapa de refinamiento 59
4.4. Promedio de fitness de mejor individuo y fitness promedios en calibración y testeo . . . 59
4.5. Promedio de fitness de mejor individuo y fitness promedios para las tareas de saludo y gesto de pateo 64
4.6. Velocidades y tiempo de ejecución de posturas en el robot 66
4.7. Velocidades de posturas calculadas para las secuencias de atajada 67

Capítulo 1

Introducción

1.1. Objetivos del proyecto

El objetivo del proyecto de grado PGILearn es entender los fundamentos del aprendizaje por demostración-imitación en robots (ADIR) y a partir de ese conocimiento desarrollar una solución robótica. Para llevar esto a cabo se relevó el estado del arte [San12] y luego se implementó una aplicación que utiliza las técnicas estudiadas.

El ADIR se puede definir como la adquisición de conocimientos utilizando la observación y repetición. El abordaje de este proyecto es estudiar qué procesos se han utilizado para programar a las máquinas, dotándolas de capacidad de aprender por ellas mismas a través de la observación e imitación de un experto realizando cierta tarea.

La habilidad de aprender imitando es natural tanto en las personas (en comportamientos como caminar, hablar o escribir) como en los animales (en actividades como la caza, protección frente a depredadores o aprendizaje de rutas migratorias). Estudios revelan que los niños dedican gran parte de su tiempo a imitar comportamientos previamente observados [RM03]. A su vez se han realizado estudios con animales que sugieren que estos también utilizan un proceso imitativo [ZLC11] [RFFG96].

Es deseable dotar a las máquinas de esta misma habilidad ya que trae consigo las siguientes ventajas:

- Se mejora el proceso de aprendizaje, ya que se reduce el espacio de búsqueda de soluciones.
- No se necesita expertise en programación para enseñarle a la máquina a realizar cierta tarea.
- Mayor autonomía del sistema robótico.

El ADIR tiene sus inicios a principios de los años 80, cuando nació el interés en el campo de los robots industriales. Era una forma de automatizar la programación y reducir los costos de desarrollo y mantenimiento, ya que no requería programación experta para cada nueva tarea a desarrollar. A fines de los años 90, el descubrimiento de neuronas especializadas en aprendizaje por imitación en monos, junto con avances en otras áreas (psicología, neurología, etc.), motivan el diseño de sistemas de aprendizaje por imitación inspirados en sistemas biológicos. A lo largo de este documento se observarán técnicas elaboradas y aplicaciones desarrolladas en el marco del aprendizaje por observación e imitación.

1.2. Organización del documento

El documento se organiza de la siguiente forma: en el capítulo 1 se hace una descripción de los objetivos del proyecto y se realiza un resumen del contenido de este documento. En el capítulo 2 se brinda el marco de trabajo del proyecto, donde se definen las técnicas utilizadas. En el capítulo 3 se define el problema a abordar y se presenta la solución aplicada para resolverlo utilizando las técnicas que se definieron en el capítulo anterior. En el capítulo 4 se explica el proceso realizado para poner a punto, entrenar y luego testear el algoritmo de aprendizaje implementado. Por último, en el capítulo 5, se discuten los aspectos relevantes del proyecto, conclusiones del mismo y perspectivas a futuro.

Capítulo 2

Marco de trabajo

2.1. Aprendizaje por demostración-imitación

2.1.1. Definición

La imitación es un proceso complejo que aplica el cerebro humano para extender su repertorio de soluciones y comportamientos para resolver diferentes problemas. Podemos definir al ADIR como el conjunto de técnicas y aplicación de las mismas en robots que se fundamentan en la observación para aprender. Observando a un individuo realizar cierta tarea, la máquina aprende a realizar la misma valiéndose de técnicas del ADIR.

A continuación se verán algunas definiciones del ADIR basado en el trabajo de Billard y Daniel Grollman [BG12]. Dado un comportamiento realizado por un humano o alguna máquina (el cual define de manera implícita cierta política, la cual denotaremos Ω^y), se desea que el mismo pueda ser ejecutado por un robot. El objetivo del ADIR, y también de otras técnicas de transferencia de política humano-robot tal como la programación o teleoperación es crear una política análoga Ω^x, que causará que el robot ejecute el comportamiento deseado. Luego de obtenida dicha política, en el caso ideal, instanciarla debería ser fácil e intuitivo.

En la programación explícita del robot, la mejora de la política Ω^x (aprendizaje) es realizada al reprogramar el robot o al cambiar los parámetros de forma manual. En cambio, en el ADIR el aprendizaje ocurre automáticamente a través de algún operador U, el cual realiza el algoritmo de aprendizaje. El objetivo del aprendizaje en ambos casos es encontrar un óptimo de la métrica M (ver subsección 2.1.3). Dicha métrica sirve como medida de qué tan bien se realizó la tarea.

2.1.2. Procesos de aprendizaje

El objetivo del ADIR es que el robot logre aprender un controlador que responda a una política Ω^x, al observar un conjunto de trayectorias T_M realizadas por el maestro (ejecutando una política implícita Ω^y), donde dicho conjunto de trayectorias se define según la expresión 2.1, donde $\eta_0 = \{y_0^n\}_{n=1}^{N_y}$ es el conjunto de N_y condiciones iniciales (estados del maestro), tal que cada condición inicial genera una trayectoria distinta. Cada trayectoria está formada por los estados por los cuales atraviesa el maestro al aplicar la política Ω^y a un estado inicial y_0^n. Dos trayectorias del conjunto de trayectorias T_M pueden diferir en tamaño. Cada estado y por el cual atraviesa el maestro, es tal que $y \in Y$, $Y \subset \Re^{D_y}$, donde

el espacio de estados Y del maestro es de dimensión D_y.

$$T_M = \Omega^y(\eta_0) = \{\Omega^y(y_0^n)\}_{n=1}^{N_y} \tag{2.1}$$

De manera similar podemos definir al conjunto de trayectorias generadas por el robot T_A según la expresión 2.2, donde $x_0 = \{x_0^n\}_{n=1}^{N_x}$ es el conjunto de N_x condiciones iniciales o estados iniciales del aprendiz, de forma tal que cada condición inicial genera una trayectoria del robot distinta. Se puede comparar el comportamiento del maestro con el del robot al evaluar el conjunto de trayectorias que ambos producen.

$$T_A = \Omega^x(x_0) = \{\Omega^x(x_0^n)\}_{n=1}^{N_x} \tag{2.2}$$

Al proceso de aprendizaje que permite al robot aprender la política Ω^x, lo denotamos $\Omega^x = \mathcal{L}(x, y)$. El mismo define cómo los controladores del aprendiz y del maestro son utilizados para generar datos y si son o no realizadas evaluaciones explícitas de la métrica M durante el mismo.

Entre los procesos de aprendizaje se destacan aprendizaje por lote, aprendizaje con proceso de automejora y aprendizaje interactivo, de los cuales se puede ver una descripción detallada en la sección 2.3 del documento del estado del arte [San12]. El proceso utilizado en el trabajo que describe este documento fue el de aprendizaje con proceso de automejora, del cual se hace una pequeña descripción a continuación.

2.1.2.1. Aprendizaje con proceso de automejora

En el aprendizaje con proceso de automejora se recolectan todas las muestras del maestro antes de realizarse el aprendizaje. Esto es así debido a que el proceso de recolección de datos puede ser difícil, tedioso o costoso, o debido a que el tiempo para procesar los datos es grande en comparación con el tiempo que lleva recolectarlos, lo que hace inviable recolectar los datos, procesarlos y luego recolectar más datos para realizar el aprendizaje.

En relación al aprendizaje por lote (ver subsección 2.3.1 del documento del Estado del Arte [San12]) se agrega al proceso la generación de comportamientos utilizando una política intermedia Ω_i^x (en una primera instancia se utiliza la política inicial Ω_0^x obtenida a partir de las demostraciones del maestro) que sirve para generar una política Ω_{i+1}^x. Una vez obtenida la política Ω_{i+1}^x, la misma es utilizada para generar nuevos ejemplos que llevarán a mejorar dicha política. Un enfoque sencillo de mejora es estimar el gradiente $\nabla M/\nabla\Omega^x$ desde las muestras de evaluación y luego cambiar la política Ω^x de acuerdo con eso.

El proceso queda descripto en el algoritmo 2.1 en el cual se puede ver que luego de los pasos 1 y 2 propios del aprendizaje por lote, los pasos 3 al 5 son aplicados reiteradamente hasta lograrse una política similar a la política del maestro. En la figura 2.1 ϕ_y y ϕ_x representan al mapeo entre el espacio del demostrador o el espacio del robot al espacio común a ambos, llamado espacio de tareas. A su vez los parámetros M y U refieren a la métrica de comparación y a la función de actualización de política del aprendiz Ω^x respectivamente.

2.1.3. Métrica

Debido a que los mecanismos de transferencia son imperfectos es importante poder evaluar la política del robot luego de instanciada. Para esto se puede utilizar una métrica, que conceptualmente computa la similaridad entre la política latente del humano y la del robot. Esta métrica es óptima cuando las dos políticas coinciden de manera exacta, o sea cuando el robot hace exactamente lo que el

Algoritmo 2.1 Aprendizaje con proceso de auto-mejora

1. Se recolectan datos T_M a partir de Ω^y, siendo Ω^y la política ejecutada por el maestro.
2. Se deriva Ω^x (la política del aprendiz) a partir de T_M basándose en un análisis de M.
3. Se genera T_A con la política actual Ω^x, siendo T_A el conjunto de datos generados por el aprendiz con la política actual Ω^x.
4. Se evalúa la métrica $M(\Omega^x, \Omega^y)$ donde se realiza la comparación entre lo demostrado por el maestro y lo ejecutado por el aprendiz.
5. Se actualiza Ω^x.
6. Se repite desde el punto 3.

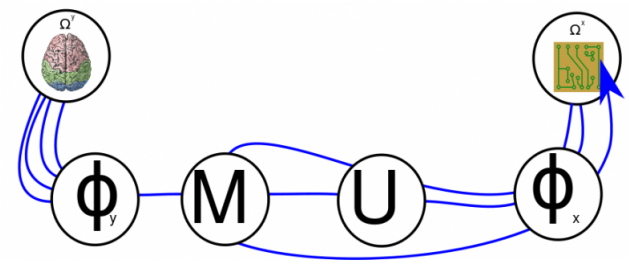


Figura 2.1: Aprendizaje con proceso de automejora, donde ϕ_y y ϕ_x representan al mapeo entre el espacio del demostrador o el espacio del robot al espacio común a ambos, llamado espacio de tareas. A su vez los parámetros M y U refieren a la métrica y la función de actualización respectivamente. Imagen extraída de [BG12]

humano haría en todas las situaciones. Hay muchas métricas posibles, desde evaluaciones subjetivas, hasta valores específicos tales como el número de objetivos cumplidos o el torque mínimo. Entre los tipos de métricas se destacan posición final del robot, coincidencia de caminos, similaridad de caminos, recompensas y probabilística, de las cuales se realiza una descripción detallada en la sección 2.4 del documento del estado del arte [San12]. En este trabajo se utilizó el tipo de métrica similaridad de caminos, de la cual se realiza una descripción a continuación.

2.1.3.1. Similaridad de caminos

En la métrica similaridad de caminos se definen ciertas características de la ejecución de la tarea -tal como el mínimo torque o la posición vertical relativa al piso de cierto motor del robot- las cuales se toman en cuenta para aplicar la métrica. La métrica se define según la ecuación 2.3 donde f es una función que mide características de las trayectorias definidas en el espacio de tareas, ϕ_x y ϕ_y son funciones que mapean desde el espacio del robot o el espacio del demostrador hasta el espacio de tareas, X_n y Y_n son las trayectorias definidas por una condición inicial entre las N_x y N_y posibles condiciones iniciales respectivamente. La métrica mide las diferencias de los promedios de f sobre las trayectorias generadas para las diferentes condiciones iniciales del aprendiz y el maestro. En otras palabras la

métrica mide la diferencia entre el maestro y el aprendiz en las caracteristicas seleccionadas de las trayectorias (medidas por la función f). Un valor positivo de la métrica indica que las características medidas por f en promedio son mayores en el maestro que en el aprendiz; por el contrario un valor negativo indica que en promedio las características medidas por f son menores en el maestro que en el aprendiz; y por último el valor nulo indica que las características medidas por f en promedio son iguales en ambos.

$$M(T_A, T_M) = -(\frac{1}{N_x}\sum_{n=1}^{N_x} f(\phi_x(X_n)) - \frac{1}{N_y}\sum_{n=1}^{N_y} f(\phi(Y_n))) \tag{2.3}$$

2.1.4. Mapeo maestro-aprendiz

En el artículo "A survey of robot learning from demostration" [ACVB09], Argall y colaboradores realizan un análisis de las distintas técnicas utilizadas al registrar las demostraciones realizadas por un maestro y la interpretación de las mismas por parte del aprendiz. Una de las dificultades que se da en este caso se debe al problema de correspondencia (definido en la sección 2.2 del documento de Estado del Arte [San12]). Se trata de resolver el mapeo entre el maestro y el aprendiz de forma tal que se permita la transferencia de información del uno al otro. A continuación se define la correspondencia con respecto a dos mapeos: mapeo de la ejecución del maestro hacia los datos registrados que llamaremos "mapeo desde el maestro" y el mapeo de esos datos que han sido registrados hacia la ejecución que debiera realizar el aprendiz, que llamaremos "mapeo hacia el aprendiz" (ver figura 2.2).

- El mapeo desde el maestro mapea la ejecución de la tarea por parte del maestro hacia datos del tipo estado-acción, que son los datos registrados de esa ejecución. Se refiere a cómo los estados y acciones realizados por el maestro durante la ejecución de la demostración son registrados en determinado conjunto de datos.
- El mapeo hacia el aprendiz mapea los datos anteriormente registrados al espacio de tareas del aprendiz (dicho espacio es el espacio de posibles acciones y estados del mismo). Se refiere a cómo los estados y acciones guardados en el paso anterior serán observados y ejecutados por el aprendiz.

Figura 2.2: Mapeo de la ejecución del maestro al aprendiz. Imagen extraída y modificada de [ACVB09].

Cuando el mapeo desde el maestro es la identidad $I(z, a)$ los estados (representados por la letra z) y acciones (representados por la letra a) experimentados por el maestro durante la ejecución son registrados directamente en el conjunto de datos. De otra forma, esta información es registrada luego de que los datos de la ejecución del maestro sean mapeados de acuerdo a alguna función de mapeo $g_M(z, a) \neq I(z, a)$. De manera similar, cuando el mapeo hacia el aprendiz es la identidad $I(z, a)$, los estados-acción registrados en el paso anterior se mapean en forma directa al aprendiz. De lo contrario, el mapeo hacia el aprendiz consiste de una función $g_A(z, a) \neq I(z, a)$. Para un sistema de aprendizaje dado, es posible que no exista ningún mapeo, que exista alguno o ambos.

Consideramos la tarea de mover una caja: un demostrador humano que utiliza su propio cuerpo para demostrar a un robot cómo mover una caja y una cámara que graba esa demostración. Sean las acciones del maestro A_M representadas como ángulos de articulaciones, y las acciones del aprendiz robot A_A representadas como articulaciones del robot. En este contexto, el robot observa la demostración del maestro a través de las imágenes percibidas por la cámara. Las acciones exactas del maestro son desconocidas por el robot, en cambio, esa información debe ser extraída desde las imágenes obtenidas por la cámara. Este es un ejemplo de un mapeo desde el maestro del tipo $g_M(z,a) \neq I(z,a)$, $A_M \rightarrow D$, donde D es el dominio del conjunto de datos registrados. Más aún, la estructura del cuerpo del maestro es diferente que la del robot y por lo tanto sus acciones (A_M) son distintas a las del robot (A_A). Entonces para que los datos de la demostración sean significativos para el robot, el mapeo $D \rightarrow A_A$ debe ser aplicado para convertir la demostración al marco de referencia del robot. Este es un ejemplo donde el mapeo hacia el aprendiz es del tipo $g_A(z,a) \neq I(z,a)$.

Argall y colaboradores presentan una categorización de la obtención de los datos de acuerdo a la ausencia o presencia de ambos mapeos: mapeo desde el maestro y mapeo hacia el aprendiz (ver figura 2.3). En esta categorización primeramente se separa la adquisición de los datos de acuerdo a dos categorías basadas en el mapeo hacia el aprendiz, y por tanto podemos decir separadas en su plataforma de ejecución:

- Demostración: Sucede cuando no hay mapeo de cuerpo, porque la demostración es efectuada sobre el robot aprendiz o en una plataforma fisicamente idéntica. O sea que $g_A(z,a) \equiv I(z,a)$.
- Imitación: Existe un mapeo de cuerpos, porque la demostración es realizada en una plataforma que no es la misma que la del robot aprendiz (o no es idéntica físicamente). Entonces $g_A(z,a) \neq I(z,a)$.

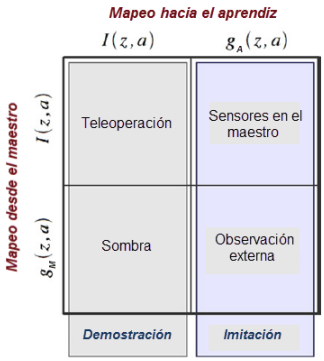


Figura 2.3: Matriz de mapeos maestro-aprendiz que indica la técnica utilizada de acuerdo a la presencia o ausencia de ambos mapeos: mapeo desde el maestro y mapeo hacia el aprendiz. Imagen extraída y modificada de [ACVB09].

2.1.4.1. Demostración

Según esta caracterización, cuando las ejecuciones son demostradas no existe un mapeo hacia el aprendiz, $g_A(z,a) \equiv I(z,a)$. Podría existir un mapeo desde el maestro para estados y acciones, si los estados experimentados (y acciones realizadas) por el demostrador no son registradas directamente y

deben ser inferidas de los datos. Basándose en esta distinción se identifican dos enfoques comunes para proveer datos de demostración al robot aprendiz: teleoperación y sombra.

Teleoperación

Es una técnica de demostración en que el maestro opera al robot aprendiz y sus sensores graban la ejecución. El mapeo desde el maestro es directo: $g_M(z,a) \equiv I(z,a)$. Provee el método de transferencia de información más directo dentro del aprendizaje por demostración, aunque requiere que el robot sea manejable por el maestro, lo que hace que no todos los sistemas sean adecuados para aplicar esta técnica. Por ejemplo demostraciones de movimiento de bajo nivel son difíciles en sistemas con control motor complejo tal como los humanoides con alto grado de libertad.

Sombra

Es una técnica de demostración en la que el robot aprendiz graba la ejecución utilizando sus propios sensores mientras intenta igualar o copiar el movimiento del maestro cuando ejecuta la tarea. El mapeo desde el maestro no es directo: $g_M(z,a) \neq I(z,a)$. Requiere un componente algorítmico extra que permite al robot seguir y copiar activamente al maestro (en vez de pasivamente como en el caso de teleoperación).

2.1.4.2. Imitación

En este caso existen diferencias en la plataforma de ejecución del aprendiz y el maestro. Existe un mapeo hacia el aprendiz, $g_A(z,a) \neq I(z,a)$. Por ejemplo si una persona tiene el rol de maestro y le enseña a un robot humanoide a realizar cierta tarea, posiblemente existan diferencias entre las articulaciones del robot y la persona, entre tamaño de brazos, piernas, etc. Se hará una distinción dependiendo de si existe también un mapeo desde el maestro o no. Basándose en esta distinción se identifican los siguientes enfoques: sensores en el maestro y observación externa.

Sensores en el maestro

Esta técnica de imitación implica que sean localizados en el cuerpo del maestro sensores para grabar su ejecución de la tarea. El mapeo desde el maestro es directo: $g_M(z,a) \equiv I(z,a)$. Al aplicarla se logra que el maestro provea de forma precisa los datos de la demostración, sin embargo, la sobrecarga que conlleva el uso de sensores, tales como trajes con sensores que limitan movimientos, o los entornos personalizados que requiere, tales como un lugar equipado con cámaras, dificulta y limita la aplicabilidad de la técnica.

Observación externa

Es una técnica de imitación donde existen sensores externos al cuerpo del maestro que son utilizados para grabar la ejecución de la tarea. Estos sensores podrían o no estar localizados en el robot aprendiz. Claramente existe un mapeo desde el maestro no directo: $g_M(z,a) \neq I(z,a)$.

2.1.5. Derivación de una política

Se utilizan los datos de las demostraciones aplicando algún método para derivar la política que ejecutará el aprendiz. En "A survey of robot learning from demonstration" [ACVB09] Argall y colaboradores realizan una categorización de los métodos utilizados en ADIR para lograr este cometido.

Aprender una política puede implicar aprender una aproximación al mapeo de estado-acción (función de mapeo), aprender un modelo dinámico del mundo y derivar una política de esa información (modelo del sistema), o también aprender un modelo de las precondiciones y postcondiciones de las acciones y ejecutar la secuencia de acciones por un planificador (ver figura 2.4).

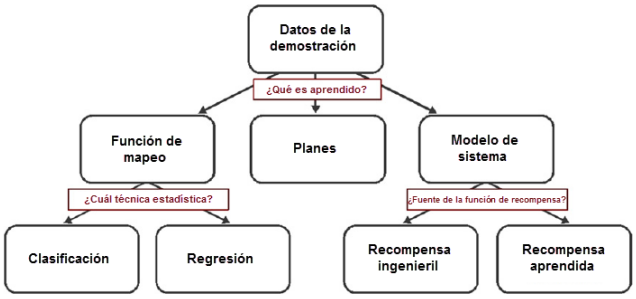


Figura 2.4: Categorización de enfoques de aprendizaje de la política desde los datos demostrados. Imagen extraída y modificada de [ACVB09]

2.1.5.1. Función de mapeo

Este enfoque de aprendizaje de política calcula una función que aproxima el mapeo de estado a acciones, $f() : Z \rightarrow A$, para el comportamiento demostrado. El objetivo de este tipo de algoritmo es reproducir la política del maestro, la cual es desconocida, y generalizar sobre el conjunto de datos de entrenamiento disponibles de tal forma que soluciones válidas también son adquiridas para estados similares que podrían no estar en las demostraciones.

En general, estas técnicas pertenecen a una de las siguientes categorías dependiendo de si la salida del algoritmo (espacio de acciones) es discreta o continua. Técnicas de clasificación producen una salida discreta, en cambio técnicas de regresión producen una salida continua. Cabe destacar que muchas de las técnicas para realizar clasificación y regresión han sido desarrolladas aparte del ADIR.

Clasificación

En este enfoque se categoriza la entrada en clases discretas. En el contexto de aprendizaje de políticas, la entrada al clasificador son estados del robot y las clases discretas de la salida son las acciones del robot. Dichas acciones pueden ser clasificadas en tres niveles de control de acciones: control de movimientos básico o de bajo nivel (incluyen comandos básicos tales como moverse hacia adelante o girar a la izquierda), primitivas de acción (ver [San12], sección 4.1) y comportamientos complejos (generalmente son desarrollados a mano o aprendidos con alguna otra técnica de aprendizaje previo al aprendizaje de la tarea).

Regresión

Para este enfoque se mapean los estados de las demostraciones a espacios de acción continuos. De forma similar que para la clasificación, la entrada al algoritmo son los estados del robot y la salida continua son las acciones del mismo. Como los valores continuos de la salida a menudo son el resultado de combinar múltiples acciones de un conjunto de demostraciones, tipicamente los enfoques de regresión aplican a movimientos de bajo nivel y no a comportamientos de alto nivel. Una distinción clave entre los métodos de regresión es si la aproximación de la función de mapeo ocurre en tiempo de ejecución o previo al tiempo de ejecución. Argall realiza un resumen de técnicas de regresión tomando en cuenta este factor.

Primeramente están los métodos de regresión que son ejecutados puramente en tiempo de ejecución. Estos métodos son rápidos y no realizan esfuerzo en aproximar la función en áreas no visitadas, pero requieren mantener todos los datos de entrenamiento. Una técnica de este tipo es Lazy Learning.

Algunos otros métodos tienen como particularidad que los datos originales son convertidos a otra representación antes del tiempo de ejecución. Los datos convertidos son luego utilizados por técnicas Lazy Learning en tiempo de ejecución. Este enfoque tiene el beneficio de no necesitar evaluar todos los datos de entrenamiento en tiempo de ejecución pero al costo de necesitar computación extra y generalización antes de la ejecución. Técnicas que utilizan este enfoque son Regresión de Campo Receptivo Ponderado (RFWR) y Regresión de Proyección de Pesos Locales (LWPR), entre otras.

Por último, en el otro extremo se encuentran los métodos que generan una aproximación de función completa previo al tiempo de ejecución. El beneficio de este enfoque se encuentra en el hecho de que en la ejecución no se depende más de los datos demostrados o de alguna transformación de los mismos como en los anteriores casos. Una desventaja de estos tipos de métodos es que producen más gasto computacional previo a la ejecución que los enfoques anteriores. Dentro de estas técnicas se encuentran Redes Neuronales y enfoques estadísticos como Regresión Gausiana Mixta (GMR) y Procesos Gausianos Dispersos en Línea (SOGP).

2.1.5.2. Modelos de sistema

El enfoque de modelo de sistema para el aprendizaje de una política de aprendizaje utiliza un modelo de transición del mundo, $T(s'|s,a)$ y de ese modelo se deriva una política $\pi : Z \rightarrow A$. Este enfoque es tipicamente formulado dentro de la estructura de aprendizaje por refuerzo (ver apéndice B del documento del Estado del Arte [San12]). Tanto los datos demostrados como la exploración autónoma adicional que el robot pueda realizar son utilizados para determinar la función de transición $T(s'|s,a)$. Para derivar una política a partir de este modelo de transición, la función de recompensa $R(s)$, la cual asocia valores de recompensa r con estados del mundo s, es aprendida o definida por el usuario.

En este tipo de enfoque la recompensa es lo que motiva a la selección de acciones y estados y consecuentemente guía el proceso de ejecución de la tarea. Definir una función de recompensa que capture de manera correcta el comportamiento deseado es una tarea no tan obvia. Argall y colaboradores categorizan a las implementaciones del ADIR del método de modelo de sistema por la forma en que es obtenida la función de recompensa. Por un lado se utilizan funciones de recompensa definidas por el usuario (funciones de recompensa ingenieriles) y por otro se realiza el aprendizaje de la función de recompensa (funciones de recompensa aprendidas).

Funciones de recompensa ingenieriles

En muchas de las aplicaciones que utilizan el enfoque de modelos de sistema el usuario define manualmente la función de recompensa. Dichas funciones tienden a ser dispersas en el sentido de que el valor de recompensa es siempre cero excepto en algunos estados, tales como alrededor de ciertos obstáculos o cerca de la meta. Las técnicas basadas en demostración tratan de guiar al robot a estados que tienen recompensa y evitan períodos extensos de exploración sin resultados. Tanto las recompensas como las demostraciones de acciones influencian en el desempeño del aprendiz.

Funciones de recompensa aprendidas

Definir una función de recompensa en sistemas reales puede ser muy difícil en algunos casos. El aprendizaje por refuerzo inverso se ocupa de esto, aprendiendo la función de recompensa en vez de definirla a mano. En el ADIR, muchos algoritmos aprenden la función de recompensa a través de las demostraciones.

Una función de recompensa aprendida al asociar mayor recompensa a estados similares a los encontrados durante las demostraciones, combinado con aprendizaje por refuerzo, sirven para enseñar a un brazo robótico a realizar una tarea de equilibrio de péndulo hacia arriba (ver sección 5.1 del documento del Estado del Arte [San12]). Recompensar similaridades entre la ejecución demostrada por el maestro y las trayectorias ejecutadas por el robot es otro enfoque utilizado en el aprendizaje de la función de recompensa.

2.1.5.3. Planes

Una alternativa a realizar el mapeo directo de estados a acciones es representar el comportamiento deseado del robot como un plan. Con este enfoque, se representa la política como una secuencia de acciones que van desde un estado inicial a un estado final. Dichas acciones son definidas en términos de precondiciones, donde el estado debe ser establecido antes que la acción pueda ser ejecutada, y postcondiciones, que es el estado resultante de la ejecución de las acciones. De manera distinta que en los demás enfoques del ADIR, las técnicas de planificación utilizan no solamente las demostraciones de estado-acción sino que también información adicional en forma de anotaciones o intenciones del maestro. Los algoritmos basados en demostraciones difieren en como las reglas, asociando pre y post condiciones con acciones, son aprendidas y si alguna información adicional es provista por el maestro.

De las primeras publicaciones respecto al aprendizaje de planes de ejecución basados en demostración, se aprende un plan para manipulación de objetos basado en las observaciones de los movimientos del maestro [KII94]. Otros enfoques utilizan diálogo hablado como una técnica para demostrar planes de tareas y también para permitir al robot verificar partes no especificadas a través del diálogo. En dicho trabajo, muchos maestros proveen instrucciones verbales de navegación con la intención de crear un vocabulario que mapea a primitivas sensimotoras [ACR04], y un maestro presenta un robot con una serie de declaraciones condicionales que son procesadas en el plan [RYSV07].

Otros métodos basados en planes requieren anotaciones del maestro. Intenciones de tarea provistas en respuesta a preguntas del aprendiz son utilizadas para codificar las post condiciones del plan [FD95], y con retroalimentación humana se dirige la atención a elementos particulares del dominio cuando se aprende un plan de alto nivel a través de la técnica de sombras [NM03].

2.2. Algoritmos genéticos

En el marco del control motor de robots, podemos definir al comportamiento de un robot como la secuencia de posiciones de sus motores en determinado rango de tiempo para llevar a cabo cierta tarea. Dicho comportamiento puede ser evaluado según distintos criterios, que pueden describirse con una función de evaluación del comportamiento (comparar con la métrica en el aprendizaje por imitación 2.1.3). Definamos a la función $\varphi : D \to C$, donde los elementos del dominio D son los posibles comportamientos de un robot y los elementos del codominio C son los valores de evaluación de cada comportamiento. En este marco, el aprendizaje por imitación tiene como objetivo encontrar un máximo de la función φ partiendo de la observación del maestro ya que dicho valor se corresponderá con el comportamiento que mejor se adapta (según la función φ) a la tarea que se quiere aprender. Por un lado, la función φ generalmente no es trivial ya que puede requerir la ejecución del comportamiento por parte del robot real (o de un simulador del mismo) o cálculos no lineales. Por otro, el dominio D puede ser amplio. En este marco, una posible solución para llevar a cabo dicho aprendizaje es la aplicación de algoritmos genéticos para encontrar un valor perteneciente a D que maximice φ partiendo de las demostraciones del maestro, ya que los algoritmos genéticos son heurísticas que permiten encontrar máximos o mínimos de diversas funciones, las cuales no tienen porque ser lineales y a su vez son eficientes en la búsqueda de soluciones en dominios grandes.

2.2.1. Representación de los individuos

Las representaciones más típicas de individuos son cadenas de bits ya que pueden ser manipuladas de forma fácil por operadores genéticos como el cruzamiento y mutación, aunque se admiten otras representaciones siempre y cuando se puedan definir los operadores genéticos utilizando dicha representación. Con esto en cuenta se pueden utilizar otros tipos de datos, ya sea cadena de enteros o reales, o una combinación de diferentes tipos de datos.

Para la aplicación de algoritmos genéticos en el aprendizaje de secuencias motoras, en la representación del individuo debe estar codificada la secuencia de alguna forma que se pueda reproducir a partir de su representación. Un posible enfoque es codificar las posturas del robot en determinados instantes del tiempo, donde cada postura y su instante del tiempo son un gen del individuo. Cada postura se determina por los datos de la posición de cada motor aunque también se pueden codificar otros datos del motor (como velocidad, torque, etc.). El individuo podría ser representado por una secuencia de posturas y un tiempo asociado, donde cada gen del individuo es un arreglo de enteros o reales y cada entero del arreglo refiere a la posición de determinado motor para esa postura en un instante del tiempo (ver figura 2.5).

2.2.2. Función de fitness

La función de fitness define el criterio para ordenar a potenciales individuos y luego seleccionarlos probabilísticamente para incluirlos en la próxima generación. Por ejemplo si la tarea es aprender a clasificar reglas, entonces la función de fitness tipicamente tendrá un componente que puntúa la precisión en la clasificación de la regla sobre un conjunto de ejemplos de entrenamiento. Más generalmente, cuando el individuo representa a un procedimiento complejo (por ejemplo si representa una sucesión de órdenes para controlar un robot como sucede en el aprendizaje de secuencias motoras), la función de fitness debe medir el rendimiento resultante de aplicar esa sucesión de órdenes en vez de medir el rendimiento de cada regla por separado. Esta función juega el papel de métrica de performance para cada individuo (ver subsección 2.1.3).

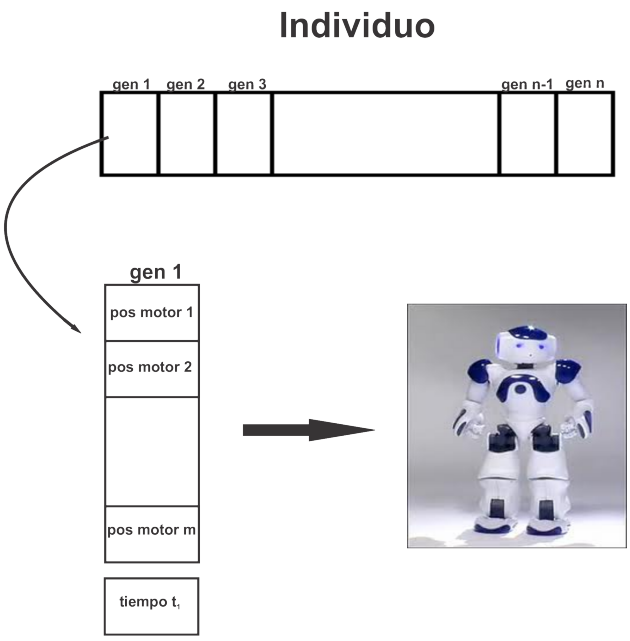


Figura 2.5: Representación del individuo donde cada gen del mismo es un arreglo de enteros o reales y los elementos de dicho arreglo corresponden a las posiciones de cada uno de los motores del robot. A su vez para cada gen se asocia un tiempo que corresponde al tiempo en el que los motores del robot adquieren esas posiciones

2.2.3. Operadores genéticos

La generación de sucesores en un algoritmo genético es determinada por un conjunto de operadores que recombinan y mutan a miembros de la población actual. Estos operadores corresponden a versiones influenciadas por operaciones genéticas encontradas en la evolución biológica, siendo el de selección, recombinación y mutación los más comunes. Dichos operadores son aplicados de acuerdo a cierta probabilidad que varía según el enfoque utilizado, donde el caso extremo sería que dicha probabilidad sea 1 por lo cual el operador es aplicado siempre.

2.2.3.1. Selección

Existen diversas maneras de seleccionar a los individuos de una próxima generación, las cuales se pueden categorizar según qué tipo de individuos se selecciona (nuevo o combinación entre nuevos y existentes) o según el fitness de los mismos.

- En base a los tipos de individuos que selecciona el operador, se puede tener un operador de selección que elija solamente de entre los individuos hijos recién generados (cada individuo será descartado luego de una generación) o que seleccione individuos tanto de los hijos como de los individuos que ya existían en la población.

- A su vez, teniendo en cuenta el fitness, se pueden seleccionar solamente a los mejores individuos de la población (enfoque elitista), o por otro lado seleccionarlos de forma aleatoria o una combinación entre elitista y aleatoria (enfoque probabilístico). Un enfoque puramente elitista garantiza mantener a los mejores individuos de la población pero a la vez puede provocar que el algoritmo no progrese correctamente cayendo en máximos locales. Un enfoque puramente probabilista permite lograr diversidad genética en la población pero a su vez podría no llegar a encontrar a los mejores individuos.

Ejemplos

Uno de los métodos de selección utilizados es conocido como Selección Proporcional al Fitness o Roulette Wheel Selection, donde la idea es seleccionar a los individuos de manera probabilística. Como se puede ver en la ecuación 2.4, la probabilidad de seleccionar un individuo es igual a su función de fitness dividido la sumatoria de la función de fitness de los demás individuos de la población, o sea que tomando en cuenta una población con n individuos la probabilidad que tiene un individuo de ser seleccionado es proporcional a su propio fitness e inversamente proporcional al fitness de los demás individuos de la población actual.

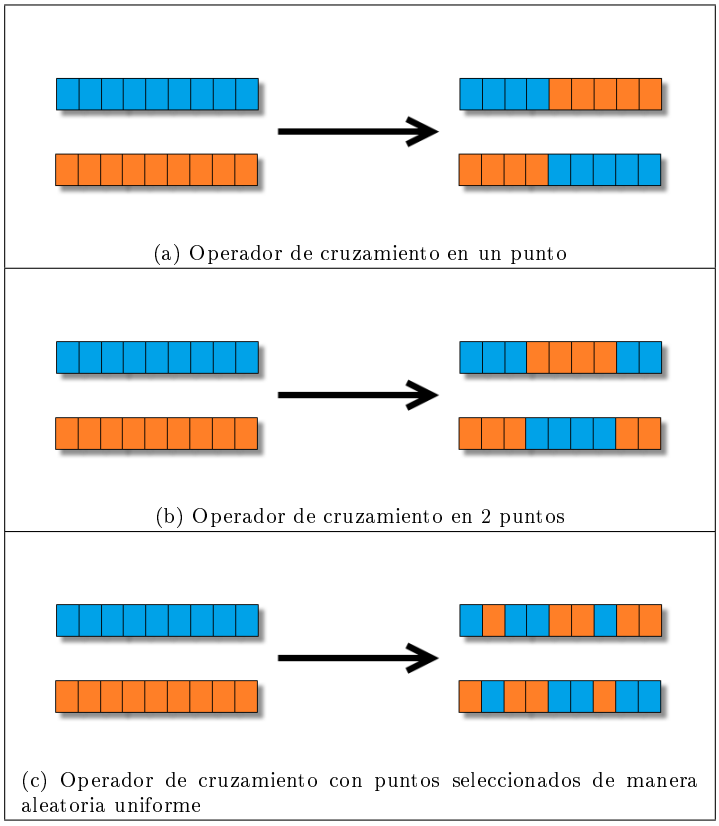
$$P(ind_x) = \frac{Fitness(ind_x)}{\sum_{i=1}^{n} Fitness(ind_i)} \tag{2.4}$$

Otro método que se utiliza es el de Tournament Selection donde dos individuos son seleccionados aleatoriamente de la población actual. Con una probabilidad predefinida p el individuo que se selecciona es el de mejor fitness y con probabilidad $(1 - p)$ el de menor fitness es elegido.

En el método Rank Selection, los individuos de la población actual primeramente son ordenados por su fitness. Luego la probabilidad que tiene un individuo de ser elegido es proporcional a la posición en esa lista ordenada en vez de ser proporcional a su fitness como en el método de Roulette Wheel Selection.

2.2.3.2. Cruzamiento

El operador de cruzamiento produce dos nuevos individuos - llamados hijos - a partir de dos padres combinando genes selectos de cada padre. El gen en la posición i en uno de los hijos es copiado del gen i de alguno de los padres y para el otro hijo ese gen es seleccionado del otro padre (ver figura 2.6). Para el caso de un operador de cruzamiento en un punto, se selecciona arbitrariamente un punto en la cadena de genes donde realizar el cruzamiento. Uno de los hijos creado tendrá los mismos genes que uno de los padres hasta el punto de cruzamiento inclusive y el resto de los genes seleccionados del otro padre. El otro hijo se genera seleccionando los primeros genes del padre distinto al cual se seleccionó primeramente para el primer hijo y el resto del padre restante. Para el cruzamiento en dos puntos se procede de manera similar pero ahora seleccionando genes del primer padre de los dos segmentos determinados desde el inicio al primer punto de cruzamiento y del segundo punto al final y los genes del segmento intermedio seleccionado del otro padre. Para el segundo hijo se hace lo mismo pero intercambiando los padres utilizados para el primero. Por último, en el cruzamiento con puntos de recombinación seleccionados de manera uniforme, cada gen es elegido de forma aleatoria uniforme de alguno de los padres. En la figura 2.6 se puede ver cada una de estas recombinaciones.



(a) Operador de cruzamiento en un punto

(b) Operador de cruzamiento en 2 puntos

(c) Operador de cruzamiento con puntos seleccionados de manera aleatoria uniforme

Figura 2.6: Operadores de cruzamiento

2.2.3.3. Mutación

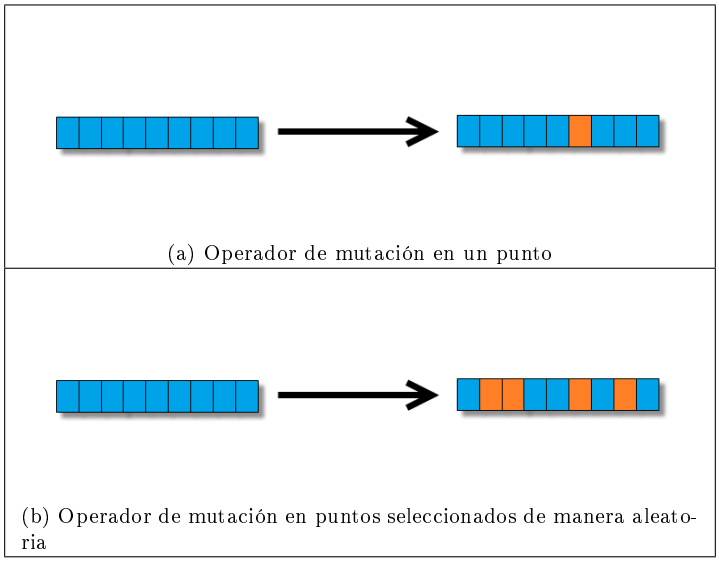
El operador de mutación en un punto (ver figura 2.7a) selecciona de manera aleatoria algún gen del individuo para cambiarlo. Este operador puede ser aplicado en múltiples genes seleccionados también de manera aleatoria (ver figura 2.7b). Generalmente este operador es aplicado luego de aplicarse el operador de cruzamiento sobre una determinada población.

2.2.3.4. Otros operadores

También existen otros operadores adicionales que son utilizados en algunos sistemas de algoritmos genéticos, especialmente operadores especializados en la representación de las soluciones, como por ejemplo para un sistema que aprende un conjunto de reglas para control robótico, se utiliza junto con el cruzamiento y mutación un operador especializado en reglas.

2.2.4. Búsqueda en espacio de soluciones

Los algoritmos genéticos emplean un método de búsqueda particular para encontrar a una solución o individuo que maximice el fitness. Dicha búsqueda se puede mover más abruptamente que algunos



(a) Operador de mutación en un punto

(b) Operador de mutación en puntos seleccionados de manera aleatoria

Figura 2.7: Operadores de mutación

otros métodos (por ejemplo regresión lineal), reemplazando a un individuo padre por su descendiente que puede ser radicalmente distinto, evitando caer en mínimos locales y a su vez haciéndolos eficientes en la búsqueda de soluciones en dominios grandes.

2.3. Técnicas de representación y manipulación de los datos

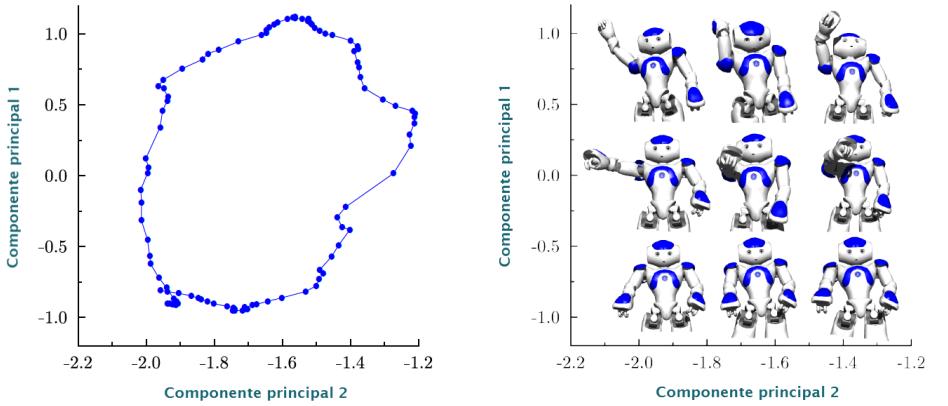
En un proceso de optimización el tiempo asociado a dicho proceso es proporcional a la cantidad de datos que se procesan. Debido a eso, se pueden utilizar técnicas de reducción de dimensiones de forma de manipular una cantidad de datos menor asegurando una pérdida total de información pequeña. Por ejemplo, en algunos proyectos utilizando PCA (Principal Component Analysis, ver subsección 2.3.1) lograron bajar desde 18 hasta 2 dimensiones con pérdidas de información aproximadas al 5 % (ver [BABVJ09]).

Las técnicas de reducción de dimensiones se pueden dividir en 2 grupos: por un lado las técnicas de reducción de dimensiones lineales y por otro las técnicas no lineales.

En [Ber11], donde se aplica un proceso de aprendizaje al que le llaman Visual Bootstrapping, que es el aprendizaje de una tarea por un robot humanoide a partir de datos observados de una persona realizando dicha tarea, realizaron un estudio sobre qué método se adaptaba mejor al problema dado. Para el método lineal utilizaron PCA (ver figura 2.8), mientras que para el método no lineal utilizaron una versión no lineal de PCA llamada neuro PCA (NPCA), también el algoritmo Locally Linear Embedding (LLE) y el algoritmo Breadth First Unfolding (BFU). Demuestran que para el problema que quieren resolver estas técnicas no son aplicables.

Para el caso lineal en el que aplican PCA, logran reducir los datos de las posiciones de cuerpo de 18 dimensiones a 2 dimensiones casi sin perdida de información. Por ejemplo, para un comportamiento de mover la mano en forma circular, se logró una compresión del 89 por ciento de la información y como para ese movimiento solamente los ángulos que se formaban en los hombros producían alta variabilidad

se logra mantener casi la totalidad de información.



(a) Puntos en el espacio bidimensional y la trayectoria cíclica obtenida a partir de esos puntos.

(b) Reproyección de nueve puntos de la trayectoria calculada en el espacio de dos dimensiones a posiciones del robot real.

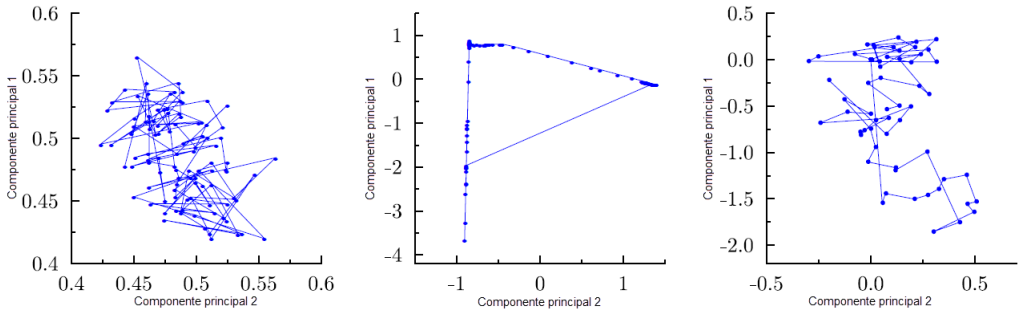
Figura 2.8: Ilustración de puntos en un espacio de dos dimensiones creado utilizando PCA a partir del movimiento observado y posturas del robot correspondientes a reproyecciones de algunos de esos puntos. Imagenes extraídas y modificadas de [Ber11].

Enfoques populares de procesamiento de movimientos [Amo10] muestran que métodos no lineales de reducción de dimensiones producen menos errores de reproyección que PCA, pero también que PCA representa a los datos en una mejor forma. Especialmente la representación cerrada de los datos es un factor importante para los últimos pasos del enfoque que se utiliza en ese trabajo, entre otras cosas en los métodos de optimización y de interpolación. Las trayectorias resultantes que se obtienen con los métodos no lineales no reconstruyen el estilo cíclico del movimiento en la forma en que lo hace PCA, tal como se puede ver en la figura 2.9. Debido a la característica anteriormente mencionada los autores del trabajo seleccionaron PCA.

Dada la semejanza de la naturaleza del problema del proyecto anteriormente mencionado con el proyecto de grado actual y al éxito que se obtuvo utilizando el algoritmo de reducción lineal PCA (durante el relevamiento del estado del arte también se observaron casos de éxito con otros proyectos similares, ver sección 5.3 del estado del arte [San12]), se decidió utilizar dicho algoritmo como técnica de reducción de dimensiones. Igualmente antes de aplicarlo, se aplicó el algoritmo reduciendo los datos demostrados a distintas dimensiones, de forma de comprobar que para dichos datos el algoritmo permitía realizar el pasaje de dimensiones y luego volver a la dimensión de datos inicial con pérdidas de información irrelevantes.

2.3.1. PCA

La técnica de PCA es una técnica estadística que permite reducir la dimensionalidad de un conjunto de datos. Sirve para hallar las causas de la variabilidad de determinado conjunto de datos y ordenarlas por importancia.



(a) NeuroPCA (b) LLE (c) BFU

Figura 2.9: Comparación de trayectorias generadas a partir de puntos determinados por las coordenadas correspondientes a las 2 componentes principales de los valores obtenidos al aplicarse tres métodos de reducción de dimensiones no lineales sobre un conjunto de datos. Como se observa en las figuras no se logra representar la naturaleza cíclica del movimiento que se está observando. Imagen extraída de [Ber11].

El PCA construye una transformación lineal que toma un nuevo sistema de coordenadas para el conjunto original de datos en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje (llamado el primer componente principal), la segunda varianza más grande es capturada en el segundo eje y así sucesivamente. Para construir esta transformación lineal debe construirse primero la matriz de covarianza o matriz de coeficientes de correlación. Debido a la simetría de esta matriz existe una base completa de vectores de la misma. La transformación que lleva de las antiguas coordenadas a las coordenadas de la nueva base es precisamente la transformación lineal necesaria para reducir la dimensionalidad de los datos. Además las coordenadas en la nueva base dan la composición en factores subyacentes de los datos iniciales.

Una de las ventajas del PCA aplicado en la reducción de dimensión de un grupo de datos, es que retiene aquellas características del conjunto de datos que contribuyen más a su varianza, manteniendo un orden de bajo nivel de los componentes principales e ignorando los de alto nivel. El objetivo es que esos componentes de bajo orden contengan el aspecto "más importante" de esa información.

En el documento de estado del arte elaborado al comienzo del proyecto se puede ver en detalle cómo se realiza la aplicación de esta técnica (ver apéndice C del documento [San12]).

2.3.2. Splines

Se puede definir a un spline como una curva diferenciable definida en porciones mediante polinomios. Utilizando splines se puede realizar interpolación, con resultados similares al aproximar con polinomios de alto grado, pero evitando oscilaciones. A su vez también son utilizados para el ajuste de curvas, lo que permiten la aproximación a formas complicadas. Debido a la simplicidad de su representación y facilidad de cómputo son muy utilizados para la representación de curvas en informática.

En el marco del proyecto de grado inicialmente se pensó la utilización de splines para la generación de la trayectoria de la tarea en una dimensión baja a partir de puntos de control, que definen dicha trayectoria y aproximan a la trayectoria real. Esta representación hace que la cantidad de información que se manipula sea mínima, produciendo mejores resultados en la computación de dichos datos.

En la práctica, debido a restricciones del simulador que no permiten ejecutar todas las posturas del comportamiento, no se utilizó esta representación de la trayectoria. Igualmente se utilizaron splines para suavizar la trayectoria definida por la tarea en R^3, luego de aplicarse el operador de mutación.

Con los splines en vez de usar un solo polinomio para interpolar los datos o aproximar una curva, se pueden usar segmentos de polinomios y unirlos adecuadamente para formar la curva. Se puede decir que una función spline está formada por varios polinomios, cada uno definido en un intervalo y que se unen entre si bajo ciertas condiciones de continuidad. Según el grado k que tengan esos polinomios queda definido ese spline, que puede ser lineal, cuadrático, cúbico o de algún grado mayor, aunque generalmente se utilizan hasta splines cúbicos. La aproximación por splines cúbicos es muy utilizada ya que interpola, no solo por valor de la función sino también por derivadas primera y segunda en los puntos donde se interpola, lográndose un muy buen resultado.

Definición splines grado k

Dado un conjunto de puntos $\{(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)\}$,donde $x_0 < x_1 < ... < x_n$ y dado un entero positivo k, una función de interpolación spline de grado k, $S(x)$ para ese conjunto de puntos cumple lo siguiente:

1. $S(x_i)$=y_i, para $i = 0, 1, ..., n$.
2. $S(x)$ es un polinomio de grado k en cada intervalo $[x_i, x_{i+1}]$, $i = 0, 1, ..., n-1$. En dicho intervalo llamaremos a $S(x)$ como $S_i(x)$.
3. $S(x)$ tiene derivada continua hasta orden $k-1$ en $[x_0, x_n]$.

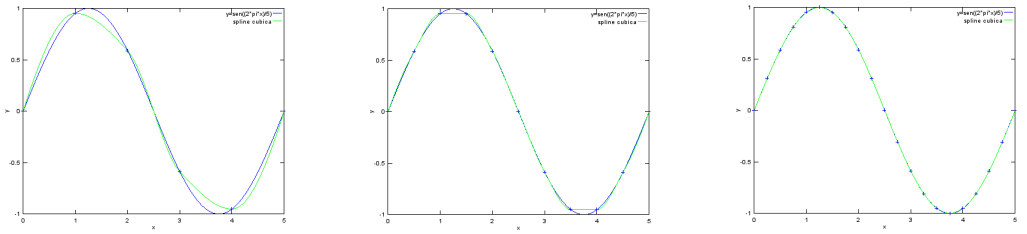
Splines cúbicos

Para $k = 3$ se tiene un caso particular de splines llamados splines cúbicos. Según la definición previa del caso genérico, en este tipo de splines cada $S_i(x)$ es un polinomio cúbico tal que $S(x_i) = y_i$ para $i = 0, 1, ..., n$; $S(x)$ es un polinomio de grado 3 en cada intervalo $[x_i, x_{i+1}]$ y $S(x)$ tiene derivadas primera y segunda continuas en el entorno $[x_0, x_n]$.

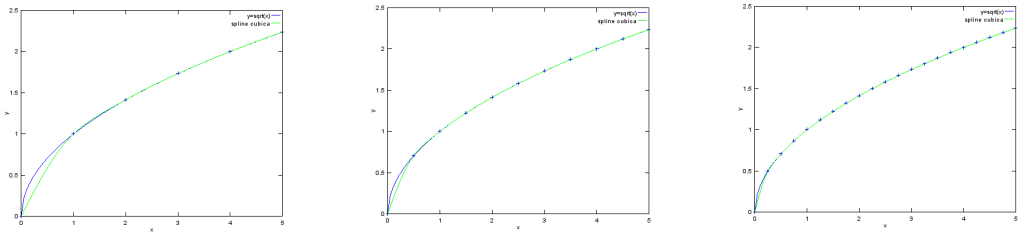
Imponiendo las condiciones previamente mencionadas, para el caso de splines cúbicos se obtiene un sistema de $4n-2$ ecuaciones con $4n$ incógnitas. La cantidad de ecuaciones se obtiene igualando polinomios cúbicos para los datos obtenidos, $2n$ ecuaciones (debido a la continuidad del polinomio establecido en el punto 1), luego igualando las derivadas primera y segunda en los puntos correspondientes se obtienen $2(n-1)$ ecuaciones más (debido a la continuidad de las derivadas establecido en el punto 3), o sea que en total son $2n + 2(n-1) = 4n-2$ ecuaciones. La cantidad de incógnitas se debe a que cada subpolinomio que define al spline, tiene 4 incógnitas y en total son n subpolinomios, o sea que el total de incógnitas es $4n$. Visto lo anterior, se necesitan dos ecuaciones más para obtener un sistema lineal de ecuaciones con una solución.

Dos soluciones pueden ser agregar $S''(x_0) = S''(x_n) = 0$ (el caso de fronteras independientes de la función o curva aproximada, si hubiere dicha función o curva) o agregar $S'(x_0) = f'(x_0)$ y $S'(x_n) = f'(x_n)$ (frontera sujeta al valor de la función o curva que se aproxima, si se estuviere aproximando a una función o curva con splines).

En la figura 2.10 se puede ver un ejemplo de aplicación de splines cúbicas sobre la función seno y raíz cuadrada.



(a) Gráficas de la función de seno y su aproximación con splines sobre 6, 11 y 21 puntos respectivamente.



(b) Gráficas de la función raíz cuadrada y su aproximación con splines sobre 6, 11 y 21 puntos respectivamente.

Figura 2.10: Aplicación de splines a la función seno y raíz cuadrada.

Capítulo 3

Problema y solución propuesta

3.1. Introducción

En el presente trabajo se pretende estudiar la aplicación del ADIR para enseñarle a un robot (que participa en el rol de aprendiz) comportamientos a partir de la demostración de los mismos por parte de una persona (que participa en el rol de maestro). El robot debe aprender una secuencia motora llevada a cabo en cierto tiempo de forma que lo lleve a realizar los movimientos demostrados por el maestro. En este trabajo el comportamiento demostrado es el de un gesto de atajada, similar al gesto que hacen los goleros en el fútbol para defender el arco.

Se puede dividir al proceso de aprendizaje aplicado en dos grandes etapas: una donde se realiza el mapeo maestro-aprendiz (ver definición en la subsección 2.1.4) y otra donde a partir de los datos mapeados se realiza la adaptación de la tarea al robot y su entorno utilizando algoritmos genéticos (ver sección 2.2). En la figura 3.1 se pueden observar las distintas actividades que se realizan y sus salidas para llevar a cabo el mapeo maestro-aprendiz así como las distintas actividades que se realizan por el algoritmo genético para lograr la optimización de la secuencia mapeada.

En una primer etapa se realiza el mapeo maestro-aprendiz (descrito en la sección 3.2). Como se observa en la parte superior de la figura 3.1 el mapeo maestro-aprendiz está compuesto por el mapeo desde el maestro y el mapeo hacia el aprendiz. En el mapeo desde el maestro se destaca la actividad de grabación y procesamiento de la tarea demostrada para obtener datos de posiciones de algunos puntos del maestro y luego la actividad de cálculo de ángulos donde a partir de los datos de las posiciones de los puntos del maestro se calculan los ángulos de sus articulaciones durante la demostración de la tarea. En el mapeo hacia el aprendiz los ángulos de las articulaciones de la persona son mapeados a posiciones de los motores del robot. Según la caracterización definida en la subsección 2.1.4, el mapeo es una imitación ya que existe mapeo hacia el aprendiz distinto a la función de identidad (ver subsección 2.1.4.2), más particularmente el mapeo correspondiente es sensores en el maestro (ver definición en la subsección 2.1.4.2).

En una segunda etapa del proceso de aprendizaje, se realiza la adaptación de la tarea al robot y su entorno utilizando algoritmos genéticos (descrito en la sección 3.3). Primeramente, como se observa en la parte inferior de la figura 3.1, a partir de los datos de la secuencia de posiciones de motores del robot (obtenidos en la etapa del mapeo maestro-aprendiz), se aplica PCA (ver definición en la subsección 2.3.1) para reducir la dimensión de dichos datos y se obtiene la secuencia de posturas en R^3. A partir de dicha secuencia se genera la población inicial del algoritmo genético. Se genera un

proceso evolutivo aplicando distintos operadores genéticos sobre los individuos de la población parcial para generar nuevas poblaciones. Para evaluar a cada individuo se realiza una reproyección de PCA sobre la secuencia que representa el individuo -que está en R^3- obteniéndose una secuencia en R^{18}, que representa las posiciones de los motores del robot en toda la secuencia. Luego se ejecuta dicha secuencia en el simulador y se evalúa su comportamiento.

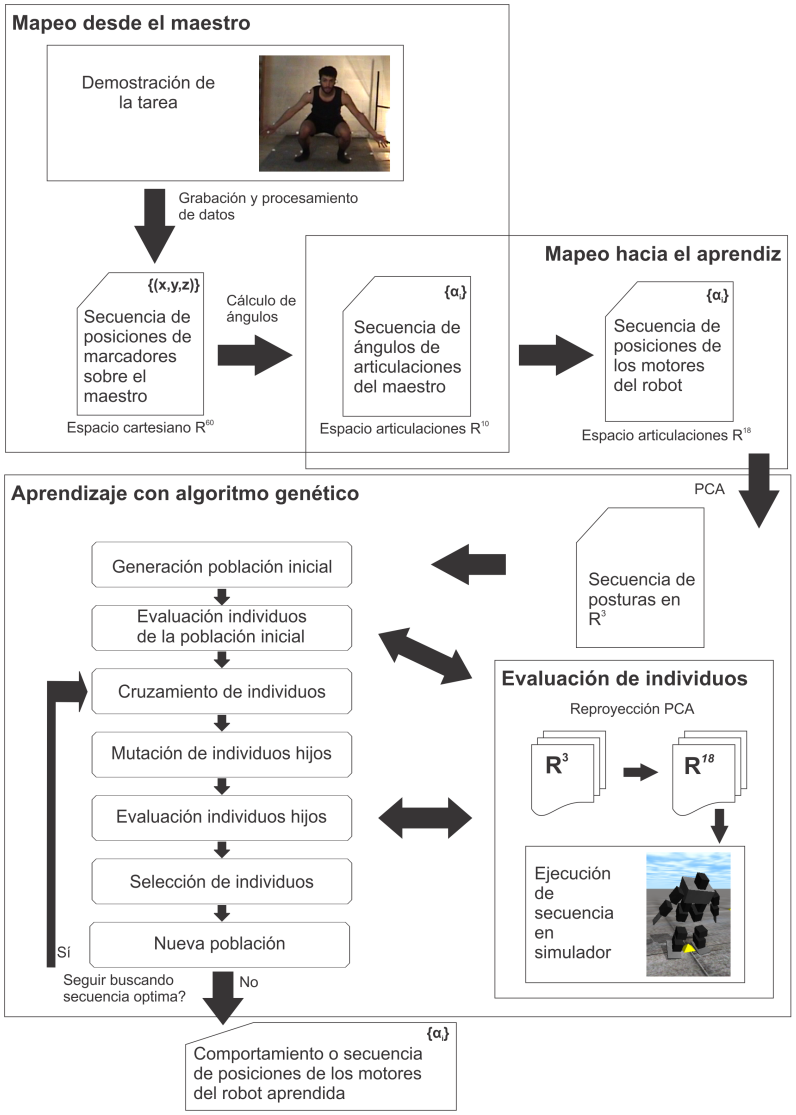


Figura 3.1: Aprendizaje por imitación utilizando algoritmos genéticos. Diagrama que describe las actividades y procesos que se aplican en el trabajo realizado para llevar a cabo el aprendizaje por imitación

El proceso de aprendizaje aplicado es similar al aprendizaje con proceso de automejora, el cual es

implementado por el algoritmo genético (ver subsección 2.1.2.1). La métrica utilizada para comparar las políticas en dicho proceso es parecida a la del tipo similaridad de caminos (ver subsección 2.1.3.1), donde la característica que se utiliza para compararlas es la estabilidad, pero con la diferencia que en la implementación de la métrica no se comparó directamente la característica de estabilidad sino que se determinó que el aprendiz debería tener esa característica sabiendo que el maestro la posee. En el algoritmo genético la métrica es implementada por la función de fitness (ver subsección 2.2.2).

3.2. Mapeo maestro-aprendiz

En el proceso de aprendizaje por imitación participan el maestro y el aprendiz. En el trabajo que se describe, una persona participó en el rol de maestro demostrando el gesto de atajada y el robot humanoide Bioloid participó en el rol de aprendiz. En la sección 2.1.4 se categorizó las distintas formas de obtención de datos de acuerdo a los tipos de mapeos desde el maestro y hacia el aprendiz. En las siguientes subsecciones se explicará como se realizó cada uno de estos mapeos y la categoría del mapeo utilizado.

3.2.1. Mapeo desde el maestro

Para obtener los datos demostrados por el maestro correspondientes a la ejecución de la tarea se utilizó una técnica de seguimiento de ciertos puntos específicos que determinan las articulaciones de la persona. En la figura 3.2 se puede observar la demostración de la tarea por parte del maestro, junto con los marcadores que se le pegaron al cuerpo para luego obtener utilizando el software adecuado las posiciones (en un espacio de coordenadas) de cada uno de esos puntos. Debido a que los datos se obtienen de forma directa desde el maestro, el mapeo desde el maestro $g_M(z, a)$ es igual a la función identidad $I(z, a)$ que opera sobre los estados z y las acciones a, determinados por los ángulos de las articulaciones en los instantes de tiempo. Aunque en la realidad, como se verá más adelante, se aplica una función para calcular los ángulos de las articulaciones a partir de las posiciones de los marcadores, dicha función es directa y no introduce errores, por lo tanto se considera que el mapeo desde el maestro es directo.

3.2.1.1. Técnica y sistema utilizado para obtención de datos[1]

Sistema de seguimiento

El sistema de seguimiento está formado por los siguientes elementos:

- Tres cámaras de grabación de video y sonido con una frecuencia de grabación de 60 imágenes por segundo.
- Plataforma física donde la persona realiza la tarea.
- Marcadores de referencia que sirven para crear el marco espacial de referencia necesario.
- Marcadores de referencia de puntos a seguir que se le pegan al cuerpo de la persona que realiza la tarea.

[1] Las grabaciones de las demostraciones y el procesamiento de las mismas se realizaron en el Hospital de Clínicas y fué un trabajo en colaboración con la Unidad de Investigación de Biomecánica de la Locomoción Humana (UIBLH), perteneciente a la Facultad de Medicina de la UdelaR. De dicha unidad participaron Patricia Polero, Gabriel Fábrica y Darío Santos.

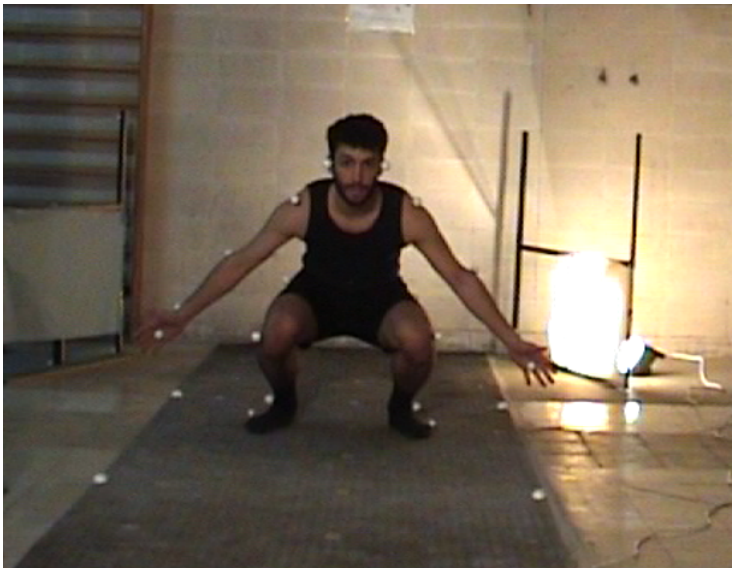


Figura 3.2: Maestro realizando la tarea que luego será aprendida por el aprendiz.

- Software que permite calibrar un sistema de referencia y puntos de seguimiento dados, y que a partir de una secuencia de imágenes grabadas desde distintas posiciones, en las que aparecen esos puntos calcula la posición en el espacio de cada punto respecto al marco de referencia calibrado.

Técnica de obtención de datos

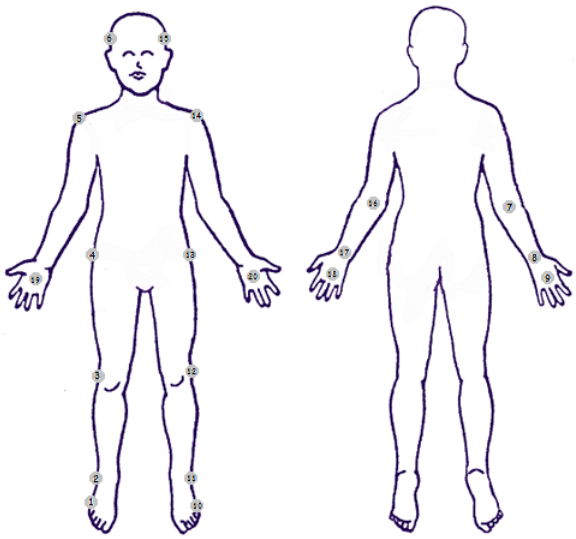
Primeramente, desde cada cámara se graba la imagen de los marcadores de referencia, los cuales se encuentran a una distancia conocida entre sí. Estas imágenes sirven para calibrar el sistema de referencia del seguimiento. Al momento de calibrar, el usuario debe indicar la distancia real entre los marcadores de referencia. El programa utilizará esa información para generar un marco de referencia que permitirá luego calcular las posiciones en el espacio de puntos específicos de la persona, según dicha referencia, mientras realiza la demostración de la tarea.

Luego, a la persona que realiza la demostración de la tarea se le colocan pequeñas esferas o marcadores, que sirven como guía para realizar el seguimiento de esos puntos del cuerpo de la persona en la imagen grabada.

Por último, se graban las demostraciones de las tareas realizadas desde 3 cámaras distintas, que graban distintos ángulos de la demostración. Al comenzar la grabación de las tareas se emite un sonido que luego servirá para sincronizar la imagen de inicio correspondiente a la secuencia de imágenes grabadas con cada cámara.

Marcadores Como se describió en la anterior sección, a la persona se le colocaron marcadores en puntos seleccionados de su cuerpo, de forma de poder captar las posiciones y hacer un seguimiento de dichos puntos mientras realiza la demostración. En la figura 3.3 se puede observar la posición que tiene cada marcador en el cuerpo de la persona. Utilizando un software apropiado, se puede realizar el seguimiento de la posición en el espacio de cada marcador, obteniéndose de esa forma los datos

respectivos a la tarea realizada.



Número Marcador	Parte del cuerpo
1	5to. metatarso derecho
2	tobillo derecho
3	rodilla derecha
4	cadera derecha
5	hombro derecho
6	cabeza derecha
7	codo derecho
8	muñeca derecha
9	mano derecha
10	5to. metatarso izquierdo
11	tobillo izquierdo
12	rodilla izquierda
13	cadera izquierda
14	hombro izquierdo
15	cabeza izquierda
16	codo izquierdo
17	muñeca izquierda
18	mano izquierda
19	palma derecha
20	palma izquierda

Figura 3.3: Posición de los marcadores colocados sobre el maestro

Obtención de datos utilizando el software de seguimiento

El software recibe como entrada las secuencias grabadas con cada cámara.

Calibración Primeramente se debe realizar la calibración de esa grabación. Para esto, dada una secuencia de imágenes (correspondiente a lo captado por una cámara), nos posicionamos en una imagen donde se pueda observar las esferas de referencia del marco de referencia, y se selecciona cada una indicándole al programa que esa es una esfera de referencia. Luego se debe repetir este paso por cada esfera de referencia. Por último se repite este procedimiento para las demás secuencias de imágenes (grabadas con las demás cámaras) y se le indica al programa la distancia real que existe entre cada esfera.

Parte de la calibración requiere indicar en qué imagen se detecta el sonido para la sincronización entre las secuencias grabadas.

Seguimiento de marcadores Se debe indicar en cada secuencia de imágenes a qué punto corresponde cada esfera de referencia. El sistema hace un seguimiento de las mismas a lo largo de la ejecución de la secuencia. En caso que el sistema no pueda realizar el seguimiento (porque el punto quedó oculto durante algunas imágenes o el software lo confundió con puntos a su alrededor), se deberá volver a indicar desde la imagen el punto correspondiente a la esfera de referencia.

Luego de realizado dicho proceso, a partir de los datos de calibración de cada cámara, de sincronización y de los datos de posiciones de las esferas de referencia de los puntos, el programa calcula la posición de cada marcador en la imagen correspondiente y da como resultado un archivo en formato

texto plano, que contiene una matriz de datos que corresponden a la posición de cada marcador en cada instante de tiempo, correspondiendo cada instante de tiempo a una imagen en particular.

Resultado del proceso de obtención de datos

Como resultado del proceso de obtención de datos demostrados se obtienen los datos completos correspondientes a una secuencia de ejecución de una tarea. Este conjunto de datos se puede representar con una matriz donde cada fila contiene la información correspondiente a la posición de una postura dada en un instante de tiempo dado mientras se ejecuta la tarea. La distancia en el tiempo entre dos posiciones consecutivas se redondeó a 0.02 segundos ya que la frecuencia de grabación de las cámaras es de 60 imágenes por segundo. En el cuadro 3.1 se puede observar la matriz de posiciones de marcadores genérica. La misma es de dimensión $n \times m$, obtenida a partir del proceso de n imágenes, donde en cada imagen se detectan a los m marcadores correspondientes. El marcador detectado i de la imagen j es representado por sus coordenadas x, y, z correspondientes a x_i^j,y_i^j,z_i^j respectivamente.

Para ejemplificar, al obtener datos de la ejecución de una tarea que tarda 3 segundos en ser ejecutada, habiendo realizado el seguimiento de 18 marcadores, se obtiene una matriz de 150 x 54 datos. Los 54 valores correspondientes de cada fila de la matriz corresponden a las coordenadas de cada marcador en un instante dado, ya que cada marcador seguido es representado por 3 valores correspondientes a las coordenadas x, y, z de un sistema cartesiano de coordenadas creado utilizando el sistema de referencia previamente mencionado. Cada una de las 150 filas corresponde a los datos respectivos de una imagen.

Según el ejemplo recién presentado para una secuencia de 3 segundos, siguiendo 18 marcadores a una frecuencia de 60 imágenes por segundo, se obtiene una cantidad de 150 x 54 = 8100 datos.

$$\begin{array}{cccccccccc} x_0^0 & y_0^0 & z_0^0 & x_1^0 & y_1^0 & z_1^0 & \cdots & x_m^0 & y_m^0 & z_m^0 \\ x_0^1 & y_0^1 & z_0^1 & x_1^1 & y_1^1 & z_1^1 & \cdots & x_m^1 & y_m^1 & z_m^1 \\ & \vdots & & & \vdots & & \ddots & & \vdots & \\ x_0^n & y_0^n & z_0^n & x_1^n & y_1^n & z_1^n & \cdots & x_m^n & y_m^n & z_m^n \end{array}$$

Cuadro 3.1: Matriz de posiciones de marcadores de dimensión $n \times m$

3.2.1.2. Cálculo de ángulos de las articulaciones

Los datos obtenidos de la demostración representan puntos en el espacio y tiempo (utilizando el sistema de coordenadas previamente descripto) que corresponden a puntos de la persona mientras ejecuta la tarea. Por ejemplo, un marcador es colocado en el hombro derecho de la persona. En los datos se puede verificar las posiciones x, y, z de ese punto en cada instante de tiempo grabado por las cámaras de la demostración de la tarea realizada por la persona.

Para realizar el mapeo de estos datos al robot, no se utilizaron los puntos del espacio cartesiano de coordenadas, ya que el robot y la persona tienen distinto tamaño de las partes de su cuerpo y la relación entre dichas partes también difiere. Por ejemplo la relación del tamaño de brazos y piernas es distinta en el robot y en la persona.

Debido a lo mencionado previamente, se decidió hacer el mapeo desde el espacio de articulaciones de la persona al espacio de articulaciones del robot.

Para obtener los ángulos de las articulaciones de la persona, se generan vectores convenientes para cada postura utilizando las posiciones de los marcadores. Para algunos pares de vectores, se puede

aplicar la definición de producto escalar entre dos vectores para calcular los ángulos determinados por ellos. El producto escalar entre dos vectores $\cdot$ se calcula según la ecuación 3.1, donde se multiplica el módulo de ambos vectores por el coseno de α, siendo α el ángulo formado entre los dos vectores u y v. Por otro lado el producto escalar entre dos vectores $u = [u_1, u_2, u_3]$ y $v = [v_1, v_2, v_3]$ también puede ser calculado según la ecuación 3.2, donde las componentes respectivas del vector se multiplican entre sí y luego se suman. El coseno de α queda determinado por el cociente del producto escalar y el producto del módulo de ambos vectores (ver la ecuación 3.3). Teniendo el valor del coseno del ángulo, se aplica la función arco coseno y se obtiene el ángulo deseado.

$$u \cdot v = |u||v| \cos(\alpha) \tag{3.1}$$

$$u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3 \tag{3.2}$$

$$\cos(\alpha) = \frac{u_1 v_1 + u_2 v_2 + u_3 v_3}{|u||v|} \tag{3.3}$$

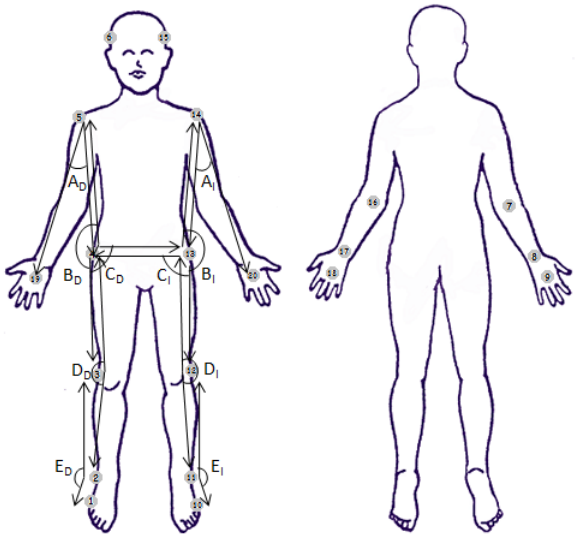
Debido a que la función arco coseno no es biyectiva en el entorno $[-\pi, \pi]$ y que los ángulos de las articulaciones pueden tomar cualesquiera valores de ese entorno, no es posible obtener el ángulo a partir de dicha función. Por ejemplo si calculamos $\arccos(0{,}7071)$ da como resultado aproximadamente tanto $\frac{\pi}{4}$ como $-\frac{\pi}{4}$. Para solucionar esto se aplico la función $atan2$ utilizando los vectores u y v según la ecuación 3.4. La misma computa la función arcotangente recibiendo dos argumentos como parámetros, computando el cuadrante correcto del ángulo calculado de acuerdo a los signos de sus argumentos. El rango de valores de retorno de dicha función pertenece al entorno $[-\pi, \pi]$. La función $\alpha{=}atan2(y, x)$ se define de la siguiente forma:

- α está en radianes y cumple $-\pi \leq \alpha < \pi$ y $\tan(\alpha) = \frac{y}{x}$
- si (x, y) está en el 1er cuadrante ($x > 0$, $y > 0$) entonces $0 < \alpha < \frac{\pi}{2}$
- si (x, y) está en el 2do cuadrante ($x < 0$, $y > 0$ entonces $\frac{\pi}{2} < \alpha \leq \pi$
- si (x, y) está en el 3er cuadrante ($x < 0$, $y < 0$) entonces $-\pi < \alpha < -\frac{\pi}{2}$
- si (x, y) está en el 4to cuadrante ($x > 0$, $y < 0$) entonces $-\frac{\pi}{2} < \alpha < 0$
- si (x, y) está sobre el eje y (x, $y{=}0$) entonces si $x \geq 0 \rightarrow \alpha = 0$ y si $x < 0 \rightarrow \alpha = \pi$
- si (x, y) está sobre el eje x ($x = 0$, y) entonces si $y > 0 \rightarrow \alpha = \frac{\pi}{2}$ y si $y < 0 \rightarrow \alpha = -\frac{\pi}{2}$
- si $(x, y) = (0, 0)$ entonces $\alpha = 0$

$$\alpha{=}atan2(\sqrt{(u \times v) \cdot (u \times v)}, u \cdot v) \tag{3.4}$$

En la figura 3.4 se puede observar tanto los vectores que se utilizaron a partir de los datos de posiciones de los marcadores, así como los ángulos que se calcularon a partir de dichos vectores. En los ángulos generados para la tarea aprendida no se utilizaron los marcadores 6, 7, 8, 9, 15, 16, 17 y 18, ya que se estimó que el utilizarlos no le agregaría información relevante para el aprendizaje de la atajada.

Como resultado se obtiene una matriz de puntos que representa en cada fila una postura de la secuencia demostrada, donde se puede ver para esa postura los ángulos que la definen.



Marcadores		Ángulo definido
Números	Descripción	
4 - 5 - 19	cadera - hombro - palma derechos	A_D
13 - 14 - 20	cadera - hombro - palma izquierdos	A_I
3 - 4 -5	rodilla - cadera - hombro derechos	B_D
12 - 13 - 14	rodilla - cadera - hombro izquierdos	B_I
3 - 4 - 13	rodilla derecha - cadera derecha - cadera izquierda	C_D
12 - 13 - 4	rodilla izquierda - cadera izquierda - cadera derecha	C_I
2 - 3 - 4	tobillo - rodilla - cadera derechos	D_D
11 - 12 - 13	tobillo - rodilla - cadera izquierdos	D_I
1 - 2 - 3	5to. metatarso - tobillo - rodilla derechos	E_D
10 - 11 - 12	5to. metatarso - tobillo - rodilla izquierdos	E_I

Figura 3.4: Ángulos generados a partir de la observación del maestro

3.2.2. Mapeo hacia el aprendiz

Luego de tener los datos registrados de la demostración de la tarea, se debe mapearlos hacia el aprendiz para que se produzca la transferencia de información desde el maestro hacia el aprendiz.

El robot humanoide Bioloid participó en el rol de aprendiz. Debido a que la demostración de la tarea no se realizó sobre el robot o una plataforma idéntica, el mapeo hacia el aprendiz $g_A(z, a)$ es distinto a la función identidad $I(z, a)$. El tamaño, así como forma y peso tanto de los miembros como del tronco del robot difieren con los de la persona. También difieren los grados de libertad de algunas de sus articulaciones (por ejemplo los hombros del robot poseen 2 grados de libertad a diferencia de los 3 grados de libertad que poseen los hombros de la persona). Según lo visto en la sección 2.1.4 al cumplirse que $g_A(z, a) \neq I(z, a)$ el proceso de tranferencia de datos desde el maestro hacia el aprendiz se denomina imitación. A su vez el mapeo desde el maestro es la función identidad $g_M(z, a) \equiv I(z, a)$ como se describió en la subsección 3.2.1. Debido a ambos mapeos el proceso de mapeo maestro-aprendiz entra en la categoría de sensores en el maestro 2.1.4.2.

En esta subsección primeramente se describirán las características del aprendiz y luego se describirá

cómo se realizó el mapeo de los datos registrados hacia el aprendiz.

3.2.2.1. Plataforma Robótica

La plataforma robótica utilizada fue la de Bioloid Expert Kit [wdB14] de Robotis. Según instrucciones del fabricante [duB], Bioloid es un kit robótico donde el usuario puede construir variedad de robots utilizando bloques con motores. El nombre Bioloid proviene de las palabras "Bio" + "all" + "oid" que significa que cualquier ser viviente puede ser construido en forma de robot.

Hay 3 tipos de hardware que provee Bioloid (ver figura 3.5):

- Dynamixel: Es la unidad básica de Bioloid que actúa como una articulación o como sensor. Los servos Dynamixel AX-12 son actuadores que son utilizados como articulaciones. A su vez los sensores AX-S1 Dynamixel son unidades que sensan tanto distancia como sonido.
- CM-5: Este es el controlador principal del robot Bioloid. Las baterías que se encuentran dentro del CM-5 le proveen energía a los Dynamixel conectados.
- Piezas conectoras: Conectan a las unidades del robot. Los Dynamixel pueden ser conectados juntos con el uso de una de estas piezas. También conectan al Dynamixel con el CM-5.

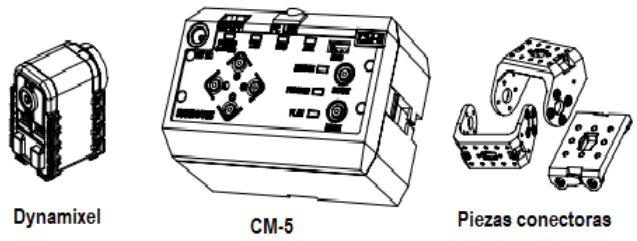


Figura 3.5: Hardware del Bioloid Expert Kit. Imagen extraída y modificada de [duB].

A su vez Bioloid provee el siguiente software:

- Programador de control de comportamiento: Es utilizado para crear un programa que controla el comportamiento del robot. El programa es utilizado para implementar el movimiento que el robot realiza de acuerdo a la información recibida a través de los dispositivos de entrada como son los sensores.
- Editor de movimiento: Es un sofware que ayuda en la creación de movimientos del robot.
- Terminal del robot: Es un tipo de programa de comunicación serial donde usuarios avanzados pueden ver información enviada por el robot y a su vez enviar caracteres tipeados en el teclado al robot.

Requerimientos de PC

- PC: IBM compatible
- SO: Windows 2000 o Windows XP

- CPU: Intel Pentium III 1 GHz o AMD Athlon XP o superior
- RAM: 256 MB o superior
- Tarjeta gráfica: función de aceleración 3D (Direct 3D soportado)
- Espacio libre de disco duro: como mínimo 300 MB
- Direct X 8.0 o superior

Bioloid humanoide

La versión humanoide del Expert Kit consta de 18 servos Dynamixel AX-12 que implementan las articulaciones del robot, 1 Dynamixel AX-S1 para sensado de sonido y distancia, el CM-5 y las piezas conectoras correspondientes para armar al robot (ver subsección 2.4.4 del manual Quick Start [rB]). En la figura 3.6 se puede visualizar una imagen del robot armado.

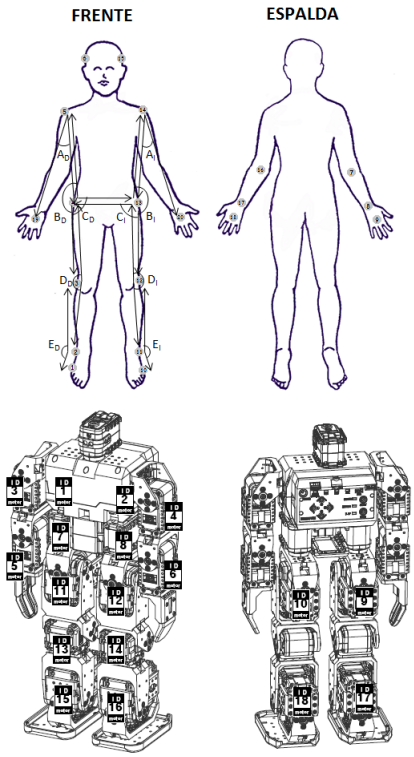
Figura 3.6: Robot Bioloid Expert humanoide ensamblado. Imágenes extraídas de [rB].

3.2.2.2. Mapeo de datos registrados hacia el aprendiz

Para poder aplicar los datos registrados al robot, se realizó un mapeo de los ángulos anteriormente descriptos a las articulaciones correspondientes del robot, las cuales son implementadas por sus motores. En la figura 3.7 se puede observar el mapeo de articulaciones desde el maestro hacia el aprendiz. Como se puede ver, algunos motores del robot no pudieron ser mapeados desde los ángulos de la persona debido a la diferencia que existe entre ambos cuerpos. En el cuadro 3.2 se puede observar junto con el mapeo el rango de posiciones (en grados) que tiene cada motor.

Brazos

Los motores 1 y 2 del robot implementan parte de la articulación del hombro derecho e izquierdo respectivamente. La articulación del hombro de la persona tiene 3 grados de libertad, sin embargo el hombro del robot posee solamente 2 grados de libertad, implementado también por los motores 3 y 4 para hombros derecho e izquierdo respectivamente aparte de los motores ya mencionados. Al realizar el gesto de atajada, la persona baja sus brazos con las palmas hacia adelante, un movimiento que en



Mapeo	
Ángulo persona	Motor robot
A_D	3
A_I	4
B_D	9
B_I	10
C_D	11
C_I	12
D_D	13
D_I	14
E_D	15
E_I	16

Figura 3.7: Mapeo de ángulos desde el maestro al aprendiz. En la figura se puede observar el mapeo de ángulos desde las articulaciones del maestro a los motores del robot. Los motores que no tienen un mapeo correspondiente tendrán una posición fija durante la ejecución de la tarea

el robot no es posible. Se decidió mapear cada articulación del hombro de la persona solamente con un motor, que permite el movimiento de apertura y cierre del brazo y de forma que las palmas del robot miran hacia el piso a medida que sus brazos bajan. El otro motor que implementa la articulación del hombro tiene un valor fijo durante la ejecución (-180 y 180 grados los motores de los hombros derecho e izquierdo respectivamente) lo que hace que las palmas apunten hacia abajo. Por otro lado tampoco se mapearon los motores 5 y 6 que implementan la articulación del codo derecho e izquierdo respectivamente, ya que en los datos demostrados, debido a las posiciones de los marcadores 7, 8, 9 y 16, 17, 18 (que corresponden a los marcadores del codo, muñeca y mano derechos e izquierdos respectivamente) respecto a las cámaras que grabaron la secuencia no pudieron ser detectados. De todas formas se observa que durante la ejecución de la tarea el brazo del maestro está extendido, o sea que la articulación del codo prácticamente tiene un valor fijo durante la misma. Se dejó el valor inicial fijo (0 grados) en los motores 5 y 6.

Marcadores que definen al ángulo	Ángulo	Motor robot	Valores motor (grados)
-	-	1	-180
	-	2	180
cadera - hombro - palma derechos	A_D	3	[-124,161]
cadera - hombro - palma izquierdos	A_I	4	[-124,161]
-	-	5	0
-	-	6	0
-	-	7	-11
-	-	8	11
rodilla - cadera - hombro derechos	B_D	9	[-4,110]
rodilla - cadera - hombro izquierdos	B_I	10	[-110,4]
rodilla der- cadera der - cadera izq	C_D	11	[-180,180]
rodilla izq - cadera izq - cadera der	C_I	12	[-180,180]
tobillo - rodilla - cadera derechas	D_D	13	[-95,89]
tobillo - rodilla - cadera izquierdas	D_I	14	[-89,95]
5to. metatarso - tobillo - rodilla der	E_D	15	[-180,90]
5to. metatarso - tobillo - rodilla izq	E_I	16	[-90,180]
-	-	17	0
-	-	18	0

Cuadro 3.2: Mapeo de ángulos desde el maestro al aprendiz

Piernas

La articulación que une la pierna con el tronco del cuerpo de la persona tiene 3 grados de libertad. En el robot esta articulación se implementa con los motores 7, 8, 9, 10, 11 y 12 para ambas piernas. Los motores 7 y 8 del robot permiten que las piernas derecha e izquierda giren sobre su eje transversal. En la persona, es dificil realizar el mapeo descripto anteriormente con los marcadores que se utilizaron. Algo que se podría hacer es calcular el ángulo que forma el vector que va del marcador del tobillo hasta el 5to. metatarso con un plano que se encuentra paralelo al tronco de la persona. De igual forma ese ángulo no sería del todo preciso ya que el pie puede girar de forma independiente a la pierna y por lo tanto no se estaría calculando el giro de la pierna sobre su eje transversal. Al observar la ejecución de la tarea, se ve que las piernas realizan un leve giro respecto a su eje transversal, moviendo los pies hacia afuera. En el robot se dejaron los motores 7 y 8 fijos (-7 y 7 grados para los motores derecho e izquierdo respectivamente) con un valor que hace que tengan un pequeño giro hacia afuera del cuerpo del mismo. Los motores 9 y 10 permiten al robot mover sus piernas hacia su cuerpo y se mapean con los ángulos formados por los marcadores del hombro, cadera y rodilla. A su vez los motores 11 y 12 permiten mover las piernas de forma lateral y se mapean con el ángulo que forman los marcadores de cadera con el marcador de cada rodilla.

Por último los motores 13, 14, 15, 16, 17 y 18 modelan el resto de las articulaciones de la pierna: rodilla y tobillo. Los motores 13 y 14 se mapean con el ángulo de la rodilla calculado con los marcadores de cadera, rodilla y tobillo de cada pierna respectivamente. La articulación del tobillo en la persona es una articulación de 3 grados de libertad que en el robot se implementa con 2 motores en cada pierna. El movimiento del pie hacia arriba se detecta con el ángulo formado por la rodilla, tobillo y metatarso en ambas piernas respectivamente. En el robot se mapea esos ángulos con los motores 15 y 16 para

cada pierna respectivamente. El movimiento que hace que la base del pie se incline hacia las laterales en el robot se implementa con los motores 17 y 18. Se calculó una aproximación de ese ángulo en la persona a través del ángulo formado por la proyección del marcador de rodilla en el piso, el marcador de tobillo y el de rodilla. Se observó que dicho ángulo variaba muy poco durante la ejecución de la tarea y su valor era próximo a 0 grados y por tanto se decidió dejar los motores 17 y 18 con valores fijos y nulos.

Se espera que en el proceso de aprendizaje se encuentre una secuencia de posiciones de motores que contrarreste la pérdida de información ocurrida durante el proceso de mapeo.

3.3. Implementación del algoritmo genético

En esta sección se describe la solución implementada utilizando algoritmos genéticos para realizar el aprendizaje por imitación. Primeramente, en la subsección 3.3.1 se describe la metahuerística genética utilizada. Luego, en la siguiente subsección 3.3.2 se habla de cómo se logró reducir el dominio de búsqueda de soluciones a través de la técnica PCA (ver subsección 2.3.1). En la subsección 3.3.3 se realiza una descripción de la representación de los individuos del algoritmo. En la siguiente subsección 3.3.4 se describe la generación de los individuos de la población inicial. En la subsección 3.3.5 se define la función de fitness y se describe brevemente el simulador utiilizado para poder evaluar a los individuos y de esa forma determinar su fitness. Para finalizar en las 2 últimas subsecciones se describen a los operadores genéticos implementados (subsección 3.3.6) y los criterios de terminación del algoritmo (subsección 3.3.7).

3.3.1. Metaheurística utilizada

En la implementación del proyecto se aplicaron dos metaheurísticas, gGA y una modificación de la misma.

Primeramente se aplicó la metaheurística Generational Genetic Algorithm (gGA), ya implementada en el framework utilizado[2]. Esta metaheurística, selecciona en cada generación los dos padres con mayor fitness y luego selecciona el resto de la población de mejores hijos. Al aplicar dicha metaheurística se renueva toda la población con la población de hijos, pero manteniendo a los dos mejores individuos previamente encontrados. Debido a la poca diversidad genética que hay en la población inicial y al elegir un tipo de selección elitista, no se logró que las poblaciones adquirieran diversidad y por lo tanto el algoritmo dejaba de encontrar mejores individuos al ejecutarse después de algunas generaciones.

Luego se modificó la metaheurística previamente mencionada, seleccionando de forma elitista un porcentaje dado de la población, de los padres e hijos que se generaron, y el resto de la población se selecciona de forma aleatoria. Con esta nueva metaheurística se mantiene a los mejores individuos ya generados, pero a su vez se logra que las generaciones tengan cierta diversidad genética al elegir de forma aleatoria al resto de los individuos, lo que aumenta la posibilidad de encontrar mejores individuos a medida que avanza el algoritmo.

En los algoritmos 3.1 y 3.2 se puede ver en código de alto nivel tanto el resumen de la metaheurística gGA como la implementación de los cambios sobre dicha metaheurística.

[2]Para la implementación del algoritmo genético se utilizó el framework JMetal [wdJ14]

Algoritmo 3.1 Resumen de metaheurística gGA

```
1  solucion gGA(int tamPoblacion)
2  {
3    conjuntoSoluciones padres = iniciarPoblacion(tamPoblacion);
4    evaluarFitness(padres);
5    int maxGeneraciones = MAX_GENERACIONES;
6    int generaciones = 1;
7    bool otrosCriteriosTerminacion = actualizarCriteriosTerminacion();
8    while((generaciones<maxGeneraciones)&&(!otrosCriteriosTerminación))
9    {
10     conjuntoParejas parejas = seleccionarParejas(padres);
11     conjuntoSoluciones hijos = cruzarParejas(parejas);
12     hijos = mutar(hijos);
13     evaluarFitness(hijos);
14     padres = seleccionarXMejores(padres,2) + seleccionarXMejores(hijos,tamPoblacion-2)
           ;
15     generaciones++;
16     otroscriteriosTerminacion = actualizarCriteriosTerminacion();
17   }
18   return padres->obtenerMejorIndividuo();
19 }
```

Algoritmo 3.2 Implementación de modificaciones sobre la metaheurística gGA

```
1  solucion gGAModificada(int tamPoblacion)
2  {
3    conjuntoSoluciones padres = iniciarPoblacion(tamPoblacion);
4    evaluarFitness(padres);
5    int maxGeneraciones = MAX_GENERACIONES;
6    bool otrosCriteriosTerminacion = actualizarCriteriosTerminacion();
7    while((generaciones<maxGeneraciones)&&(!otrosCriteriosTerminación))
8    {
9      conjuntoParejas parejas = seleccionarParejas(padres);
10     conjuntoSoluciones hijos = cruzarParejas(parejas);
11     hijos = mutar(hijos);
12     evaluarFitness(hijos);
13     conjuntoSoluciones poblacionElitista = seleccionarXMejores(padres+hijos,
           FRACC_ELITISTA*tamPoblacion);
14     padres = poblacionElitista + seleccionarXAleatoriamente(padres+hijos-
           poblacionElitista,(1-FRACC_ELITISTA)*tamPoblacion);
15     generaciones++;
16     otroscriteriosTerminacion = actualizarCriteriosTerminacion();
17   }
18   return padres->obtenerMejorIndividuo();
19 }
```

3.3.2. Reducción del dominio de búsqueda

En un proceso de optimización el tiempo asociado a dicho proceso es proporcional a la cantidad de datos que se procesan. Teniendo en cuenta la anterior afirmación, se aplicó una técnica de reducción de dimensión a los datos obtenidos y mapeados de forma de reducir la cantidad de datos a procesar en

la optimización sin perder demasiada información. Con dicha técnica se logra manipular una cantidad de datos menor asegurando una pérdida total de información pequeña.

Para reducir la cantidad de datos se aplicó la técnica lineal de PCA (ver 2.3.1) a los datos obtenidos de la persona que demostró la tarea y se realizó un estudio de los mismos para comprobar que la misma era aplicable a ese conjunto de datos en particular y poder determinar cuál es la pérdida de información que se produce al reducir a cada dimensión.

En las tablas del cuadro 3.3 se pueden observar tanto la media como la desviación estándar de los errores que se producen al aplicar la técnica de PCA. Dichos errores se calculan para cada ángulo en toda la secuencia de posturas de acuerdo a la fórmulas 3.5 y 3.6 donde $\vec{\alpha}$ es el vector con los valores del ángulo sobre toda la secuencia, n es igual a la cantidad de posturas en la secuencia, $PCA(\alpha_i)$ es el valor de el ángulo α en la postura i luego de aplicarse PCA sobre los datos para reducir los mismos a una dimensión menor y luego volver a la dimensión inicial y $PCA(\vec{\alpha})$ es el vector con los datos del ángulo α sobre toda la secuencia luego de aplicarse PCA para reducir y volver a la dimensión original.

$$Error\bar{P}CA(\bar{\alpha}) = \sum_{i=1}^{n} \left(\frac{\mid \alpha_i - PCA(\alpha_i) \mid}{n} \right) \tag{3.5}$$

$$DsvStdErrorPCA(\vec{\alpha}) = DsvStd(\mid \vec{\alpha} - PCA(\vec{\alpha}) \mid) \tag{3.6}$$

A su vez en la figura 3.8 se pueden observar las gráficas para cada articulación de los datos originales y los obtenidos al aplicar la reducción de dimensión y luego transformación de dichos datos a la dimensión inicial.

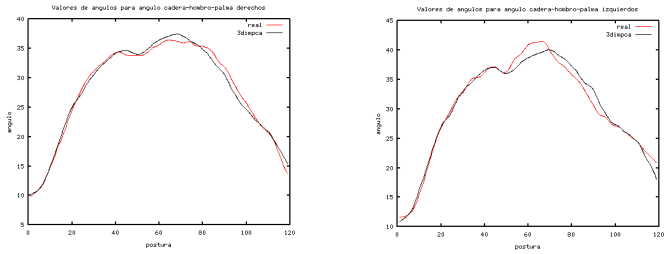
Media de errores

	ángulo (grados)									
dimensión	A_D	A_I	B_D	B_I	C_D	C_I	D_D	D_I	E_D	E_I
1	1.32895	1.75333	1.40310	1.84793	1.10365	0.81415	1.77166	1.53532	1.89309	2.73737
2	1.28670	1.75118	0.77133	0.77295	1.09755	0.77380	1.46401	0.59129	1.77330	1.33752
3	0.64519	0.83251	0.42797	0.53262	1.04765	0.73317	1.08216	0.49246	0.67302	1.05081
4	0.40012	0.51045	0.42110	0.38824	0.55764	0.73653	0.43252	0.47730	0.49765	0.36365
5	0.36554	0.34283	0.29630	0.34108	0.21826	0.73539	0.20731	0.44575	0.34025	0.29769
6	0.22653	0.23756	0.29249	0.18870	0.18470	0.22432	0.20343	0.44658	0.21127	0.29936
7	0.22792	0.20648	0.18927	0.14889	0.18475	0.21459	0.10096	0.08571	0.10982	0.07737
8	0.15800	0.11198	0.15476	0.14935	0.14913	0.04889	0.09312	0.08546	0.02652	0.03346
9	0.14632	0.10513	0.06533	0.03644	0.11965	0.01054	0.05610	0.05004	0.00899	0.02056
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

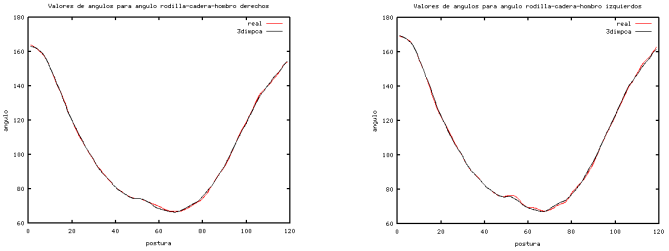
Desviación estándar de los errores

	ángulo (grados)									
dimensión	A_D	A_I	B_D	B_I	C_D	C_I	D_D	D_I	E_D	E_I
1	0.75326	1.11798	1.00162	1.44428	0.87811	0.61314	1.22083	1.11030	2.12200	1.59610
2	0.74005	1.12131	0.63548	0.55234	0.87494	0.51243	0.76087	0.46458	1.79663	0.79212
3	0.45918	0.74504	0.32838	0.41507	0.91577	0.54126	0.80226	0.33485	0.52615	0.82446
4	0.34780	0.33874	0.32219	0.26238	0.52396	0.53530	0.33509	0.32565	0.37933	0.34986
5	0.29304	0.21613	0.23686	0.22260	0.17255	0.53492	0.14864	0.32555	0.22782	0.21223
6	0.23255	0.13881	0.22750	0.16245	0.15589	0.17021	0.15129	0.31997	0.17814	0.20821
7	0.21972	0.13713	0.14797	0.13274	0.15251	0.16155	0.08839	0.08054	0.08179	0.05810
8	0.14375	0.10222	0.13718	0.12212	0.13482	0.03952	0.08662	0.07917	0.02258	0.03123
9	0.13414	0.09638	0.05990	0.03341	0.10969	0.00967	0.05143	0.04587	0.00824	0.01884
10	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

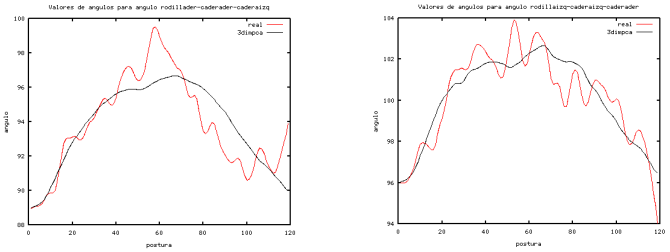
Cuadro 3.3: Tablas de media y desviación estándar de errores en los ángulos al aplicar PCA. Ver referencia de los ángulos en la figura 3.7.



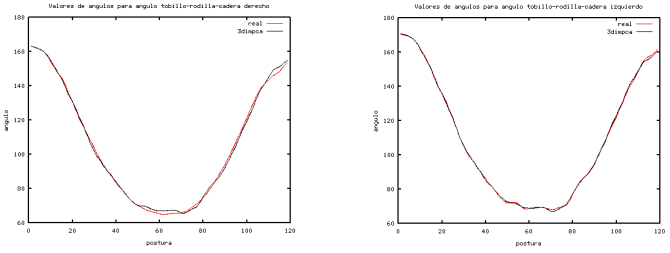
(a) Gráficas para los ángulos cadera-hombro-palma derecho e izquierdo.



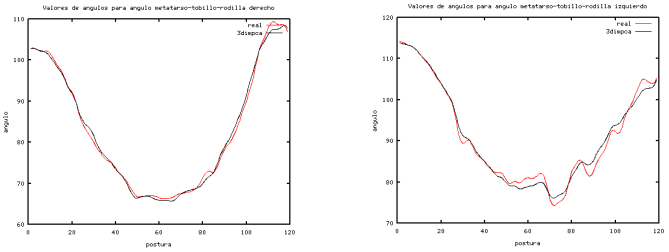
(b) Gráficas para los ángulos rodilla-cadera-hombro derecho e izquierdo.



(c) Gráficas para los ángulos rodilla-cadera-cadera derecho e izquierdo.



(d) Gráficas para ángulos tobillo-rodilla-cadera derecho e izquierdo.



(e) Gráficas para ángulos metatarso-tobillo-rodilla derecho e izquierdo.

Figura 3.8: Comparativo de los ángulos reales con los obtenidos al aplicar PCA para reducir dimensión y volver a aplicar la técnica para volver a la dimensión mayor. En la imagen se puede observar que el error al aplicar la técnica es relativamente pequeño.

3.3.2.1. Cálculo del número de componentes principales

Para realizar el cálculo del número de componentes principales que se utilizará con PCA, o sea el número de las dimensiones a las que se reduce, se utilizó la comparación de varianza retenida al reproyectar los puntos obtenidos con PCA. Se estableció que se queria retener el 98 % de la varianza de los datos. Para realizar dicha comparación se realizan los pasos del algoritmo 3.3. Como se observa en el algoritmo se utilizó la ecuación 3.8 para calcular la varianza retenida, dando como resultado el valor $k = 3$ (ver cuadro 3.4). A su vez para ese valor, se observa en las tablas del cuadro 3.3 que se obtiene aproximadamente 1 o menos de 1 grado de media de error en la reproyección de los datos para cada ángulo y menos de 1 de desviación estándar para cada ángulo respecto a esos datos. Para ese valor de componentes principales la reducción es del 70 % de la información ya que se reduce desde 10 dimensiones a 3 dimensiones.

Cabe destacar que para este conjunto de datos en particular se podría haber realizado la reducción a 1 o 2 componentes principales obteniéndose buenos resultados (retención de 94.7 % y 97 % de la varianza respectivamente).

$$p_i^j = \frac{p_i^j - \mu_j}{\gamma_j} \tag{3.7}$$

$$\frac{\frac{1}{m}\sum_{i=1}^{m} \|p_i - PCA(p_i)\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|p_i\|} \leq \epsilon \tag{3.8}$$

Algoritmo 3.3 Cálculo del número de componentes principales

1. Primeramente se nomalizan los datos y se los escala para que puedan ser comparables, según la ecuación 3.7, donde a cada valor de cada ángulo de la secuencia, p_i^j, se le resta μ_j y se lo divide por γ_j, donde μ_j es la media de los valores en toda la secuencia del ángulo j de las posturas, γ_j es la escala que se calcula realizando la resta entre el mayor valor y el menor valor del ángulo j en toda la secuencia y p_i^j es el ángulo j en la secuencia i, o sea el ángulo j de la postura p_i.

2. Se inicializa k en 1 y se inicializa ϵ con el valor de $1 - \frac{x}{100}$, donde x corresponde al valor del porcentaje de varianza que se quiere retener.

3. Se aplica la técnica de PCA con k componentes principales. Luego se divide el promedio de los errores de proyección al cuadrado sobre la variación total de los datos según describe la ecuación 3.8, donde p_i es la postura de la secuencia i compuesta por todas las posiciones de los motores del robot y m el total de posturas en la secuencia, y de esa forma se obtiene el porcentaje de la varianza no retenida.

4. Si la varianza no retenida es menor o igual que ϵ el algoritmo devuelve k. Sino se actualiza k con el valor $k + 1$ y se vuelve a ejecutar el paso 3.

k	1	2	3	4	5
% varianza	94.70	97.01	98.43	99.51	99.87

Cuadro 3.4: Porcentaje de varianza retenida al aplicar PCA utilizando los primeros k componentes principales, o sea reduciendo los datos a dimensión k

3.3.3. Representación de los individuos

Cada individuo solución del problema está compuesto por una secuencia de posturas en R^3, que son las posturas que determinan su comportamiento. Como se describirá en la subsección 3.3.5 para poder evaluar a cada individuo se necesita ejecutar en un simulador su secuencia de posturas. Se detectó que el simulador seleccionado no permite la ejecución de posturas que están a menos de 0.2 segundos de distancia en el tiempo. Debido a que 2 posturas contiguas de la secuencia de posturas distan 0.02 segundos en el tiempo (ver subsección 3.2.1.1) es claro que no se pueden ejecutar en el simulador todas las posturas que definen el comportamiento del aprendiz. Debido a esa restricción se necesitó agregarle a la representación del individuo una secuencia de índices que referencian al subconjunto de posturas que se ejecutan en el simulador. En la figura 3.9 se puede observar una representación gráfica de tal individuo. El tipo de solución implementado que modela las soluciones se llama ArrayIntPointSolutionType. Contiene una variable del tipo arreglo de enteros (que indica qué posturas ejecutar) y otra variable del tipo arreglo de puntos en R^3 que puede verse como una matriz de reales de dimensiones $LARGOSECUENCIA \times 3$ (la que contiene la secuencia de posturas).

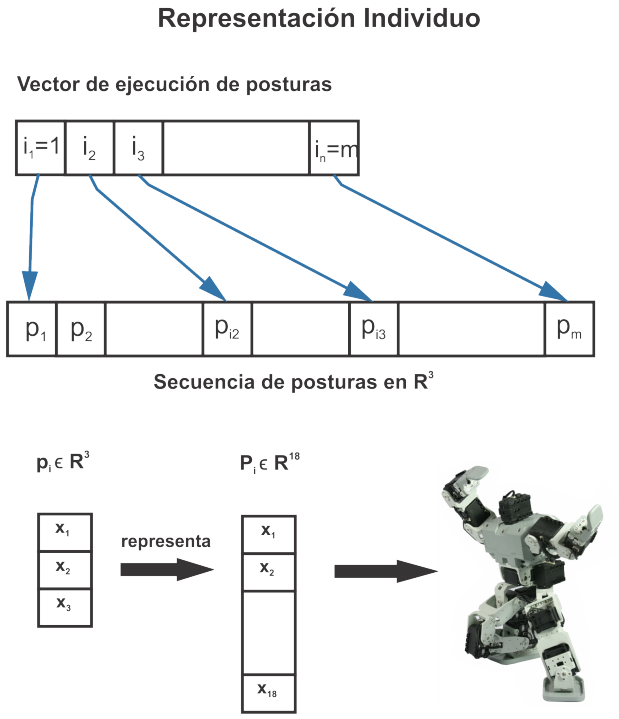


Figura 3.9: Representación del individuo solución del algoritmo genético. Como se puede apreciar en la imagen, los indices del vector de ejecución de posturas de largo n, indican posturas del vector de posturas de largo m. Cada postura pertenece al espacio R^3 que a su vez representa una postura del robot del espacio R^{18}

Tiempo entre ejecución de posturas

Debido a la restricción del simulador anteriormente mencionada y que a su vez se espera que entre dos posturas no haya más de cierta distancia en el tiempo (ya que se dejaría la ejecución librada a la implementación de movimientos de motores del controlador y se perdería la información implicita que tiene la secuencia de posturas), se definieron dos constantes MINDIST y MAXDIST, que indican el mínimo de distancia que debe haber entre la ejecución de una postura y otra contigua en el simulador y el máximo de distancia entre dos ejecuciones de posturas contiguas en el simulador respectivamente.

Por un lado, como 2 posturas contiguas distan 0.02 segundos en el tiempo MINDIST debe ser mayor o igual que 10 para que se cumpla con la restricción que tiene el simulador de que el tiempo de ejecución entre 2 posturas debe ser mayor o igual a 0.2 segundos. Por el otro, debido a que la secuencia más larga de entre las 3 demostradas tiene 162 posturas y exigiendo que entre la ejecución de una postura y otra no hayan más de $\frac{1}{4}$ de las posturas de la secuencia, o sea no haya más de 40 posturas de diferencia entre 2 ejecutadas para la secuencia más larga (esta exigencia se realizó para que como se mencionó anteriormente no se deje librada la ejecución de la secuencia a la implementación de movimientos de motores del controlador y para no perder demasiada información respecto al individuo), se determinó que el valor de MINDIST y MAXDIST deben cumplir que $MINDIST < MAXDIST$ y que MINDIST y MAXDIST pertenezcan al rango $[10, 40]$.

Se le setearon a MINDIST y MAXDIST los valores 15 y 30 respectivamente. Tomando en cuenta los valores de MINDIST y MAXDIST previamente mencionados, entre la ejecución de dos posturas en el simulador como mínimo habrá 0.3 segundos y como máximo 0.6 segundos. Esto permite por un lado cumplir con la restricción impuesta por la herramienta utilizada y por el otro con la exigencia que se determinó de que entre la ejecución de una postura y la siguiente no hayan más de un cuarto de las posturas del individuo.

3.3.4. Generación de individuos de la población inicial

Para generar la población inicial se utilizaron como insumo los datos obtenidos y procesados de la demostración de la tarea por parte del maestro. De esos datos se utilizó una demostración para la generación de la población inicial al realizar el aprendizaje de los parámetros del algoritmo genético (ver subsección 4.2.1). A su vez se utilizaron las otras dos demostraciones restantes para la generación de la población incial en el testeo de parámetros aprendidos (ver subsección 4.2.2).

Según lo descripto en la subsección 3.3.3 la estructura de las soluciones tiene dos partes, una que refiere a los ángulos que el robot debería tener en cierto instante de tiempo, que los llamamos posturas, y la otra que refiere a cuáles de esas posturas se deben ejecutar en el simulador para poder evaluar la solución. A continuación se describe cómo se realiza la inicialización de individuos de cada parte de las soluciones.

3.3.4.1. Inicialización de la matriz de posturas

A partir de los datos obtenidos de las demostraciones y aplicando las técnicas descriptas en las subsecciones 3.2.2 y 2.3 para mapear esos datos y reducirlos a dimensión 3, se inicializa cada individuo con los datos correspondientes. En la población inicial del algoritmo genético cada individuo tendrá la misma matriz de posturas, debido a que dicha matriz fue creada a partir de los datos de ángulos de una sola demostración del maestro. A pesar de la poca diversidad genética inicial, se espera que la aplicación de los operadores genéticos de mutación y cruzamiento generen la diversidad necesaria para que el algoritmo genético encuentre buenos individuos (con fitness alto).

3.3.4.2. Inicialización del vector de ejecución de posturas

Para inicializar el vector de ejecución de posturas se realizó un algoritmo que inicializa dicho vector con igual probabilidad entre los demás posibles vectores de ejecución de posturas que cumplen con las restricciones impuestas en la codificación de las soluciones. Cabe recordar, como se describió en la subsección 3.3.3, que los datos que conforman dicho vector deben respetar distancias máximas y mínimas de forma de cumplir con los requerimientos que tiene el simulador en la ejecución de posturas. En el algoritmo 3.4 se describe cómo se inicializa cada individuo de la población inicial del algoritmo genético.

3.3.4.3. Mapa de estabilidad

Debido a que la estructura corporal de la persona que cumplió el rol de maestro difiere en gran medida respecto a la estructura corporal del robot (en grados de libertad de las articulaciones, distribución de peso, etc.) - ver problema de correspondencia, sección 2.2 del documento de Estado del Arte [San12] -, al aplicar sobre el robot la secuencia de datos obtenida de la demostración se generan posturas en las cuales el robot probablemente no esté en equilibrio estático. Esto causa dificultades en la estabilidad del robot sobre todo en las posturas inicial y final de la secuencia. En la postura final es clara la dificultad que causa el problema ya que el robot al finalizar de ejecutar toda la secuencia termina con una postura que no está en equilibrio estático y por lo tanto cae al suelo. En la postura incial el problema de estabilidad se debe a que la secuencia a ejecutar parte de una postura en la que el robot está desequilibrado lo que causa que caiga al suelo rapidamente. El desequilibrio estático en las posturas intermedias no tiene porque afectar la estabilidad del robot al realizar la secuencia, ya que el mismo está en movimiento y puede lograr un equilibrio dinámico.

En la ejecución del algoritmo genético el hecho de que la primer postura de algún individuo no este en equilibrio estático, trae la dificultad de que al evaluar dicho individuo en el simulador el robot caiga al suelo de forma rápida, no pudiéndose distinguir otras buenas características del individuo respecto a otros individuos. Esto hace que el aprendizaje sea mucho más lento, ya que el algoritmo debe encontrar primeramente individuos estables al principio de la secuencia para recién empezar a buscar individuos que tengan equilibrio dinámico. En cambio el hecho de que la postura final no esté en equilibrio estático es un problema que el algoritmo genético puede resolver de forma más fácil, ya que habiendo aprendido secuencias casi estables es posible distinguir en el simulador fácilmente cuáles terminan con equilibrio estático y cuáles no.

Para intentar resolver el problema se utilizó la técnica de mapa de estabilidad (que se puede ver en [Ber11]) tratando de corregir el hecho que la primer postura del robot no está en equilibrio estático. La misma consiste en generar en el espacio de posturas de R^3, un mapa que pueda indicar en todo el espacio cuáles posturas de ejecución son estables según el algoritmo y de esa forma, utilizando dicho mapa, el algoritmo genético puede partir de posturas que son estables y llegar a aprender una secuencia estable.

Generación de la matriz de estabilidad

Para aplicar la técnica se divide al espacio R^3 con una grilla de $X * Y * Z$ puntos. Para cada vértice de dicha grilla, se evalúa el mismo de la siguiente forma: se ejecuta en el simulador la secuencia de posturas p_0, $p_1$donde p_0 representa la postura inicial en la que el robot está en reposo y en equilibrio y desde la que parten todas las secuencias de ejecución que representa cada individuo del algoritmo

Algoritmo 3.4 Inicialización del vector de ejecución de posturas

```
1  solucion inicializacionVectorEjecucionPosturas(solucion sol)
2  {
3    // primeramente se genera el vector de distancias, que indica la
4    // distancia o diferencia entre los valores de indices contiguos
5    // en el vector de ejecucion de posturas
6    vector vectorDistancias;
7    // minima y maxima distancia entre valores de indices contiguos
8    int min=MINIMA_DISTANCIA;
9    int max=MAXIMA_DISTANCIA;
10   int sumaDistancias = 0;
11   int n = largoVectorEjPosturas;
12   int p = totalPosturas;
13   // el vector de ej de posturas es de largo n, el de dist. es largo n-1
14   for(int i=0; i<n-1; i++)
15   {
16     // la distancia se sortea aleatoriamente de forma que todos los
17     // vectores posibles de ejecucion de posturas tengan igual
18     // probabilidad de ser creados
19       int j = sortearSinRepeticion(0, n-2);
20     // para cualquier vector de ejecucion de posturas, la suma de
21     // distancias entre indices contiguos sera p-1. el valor minimo
22     // posible se determina restando a la sumas de distancias de
23     // indices contiguos (p-1) la distancia ya sumada parcialmente
24     // (sumaDistancias) y el valor (n-1-i)*MAXIMA_DISTANCIA.
25       int minPosible = (p-1-sumaDistancias)-(n-1-i)*MAXIMA_DISTANCIA;
26     // el valor minimo se determina entre el maximo de min y minPosible
27       min = maximo(min, minPosible);
28     // el valor maximo posible se calcula en forma similar al minPosible
29     // pero usando la constante MINIMA_DISTANCIA
30       int maxPosible = (p-1-sumaDistancias)-(n-1-i)*MINIMA_DISTANCIA;
31       max = minimo(max, maxPosible);
32       int distancia = sortear(min, max);
33       vectorDistancias->set(j, distancia);
34       sumaDistancias += distancia;
35   }
36   //en base al vector de distancias se inicializa el vector de posturas
37   vector vectorEjPosturas;
38   //la primer postura en ejecutar sera la postura 0 y la ultima la p
39   vectorEjPosturas->set(0, 1);
40   vectorEjPosturas->set(n-1, p);
41   // habiendo generado el vector de distancias, para crear el vector
42   // de ejecucion de posturas solamente se suma a cada indice el valor
43   // del indice anterior mas la distancia del indice anterior
44   for(int i=1; i < n-1; i++)
45   {
46     vectorEjPosturas->set(i, vectorEjPosturas->get(i-1) + vectorDistancias->get
         (i-1));
47   }
48   sol->vectorEjecucionPosturas = vectorEjPosturas;
49   return sol;
50 }
```

genético y p_1 representa la postura correspondiente del vértice que se está evaluando y se determina si en cada una de esas ejecuciones el robot se cae. Si el mismo se cae, para ese vértice de la grilla la posición correspondiente del robot se la toma como inestable y viceversa. Como resultado de dichas evaluaciones se obtiene una matriz de 3 dimensiones la cual contiene la información de estabilidad. Para cada coordenada (x, y, z) correspondiente a un punto del espacio R^3 indica si dicho punto es estable de acuerdo al criterio establecido anteriormente.

Cabe destacar que al aplicar el criterio de estabilidad establecido, cuando un punto es evaluado como estable, no significa que el robot con la postura correspondiente a ese punto tenga equilibrio estático o dinámico desde cualquier otra postura. El criterio solamente da una noción aproximada de la estabilidad de determinada postura.

Utilización de los datos de estabilidad en el algoritmo genético

Dentro del algoritmo genético, el algoritmo de estabilidad se aplica solamente a la primer postura de la secuencia que representan los individuos que integran la población inicial, para que en la ejecución de dicha secuencia el robot parta de una postura estable según el algoritmo de estabilidad, lo que puede llegar a brindar estabilidad al robot durante el resto de la ejecución de la secuencia. Al aplicar dicha técnica se espera lograr una mayor eficiencia en el algoritmo genético.

Operador de estabilidad

El operador de estabilidad implementado utiliza la información que brinda la matriz de estabilidad para encontrar al punto estable más cercano al punto que se quiere estabilizar y da como salida ese punto.

Para cada punto a estabilizar se aplica el algoritmo 3.5.

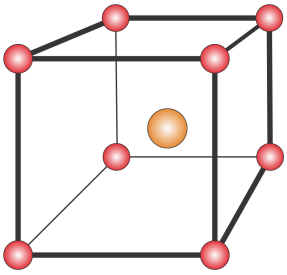
Algoritmo 3.5 Algoritmo de estabilidad para un punto de R^3

1. Primeramente, se verifican los puntos envolventes al punto a ser estabilizado. En el caso aplicado con R^3, los puntos envolventes forman un cubo (si la distancia entre los puntos para cada dimensión es igual) que contiene al punto a ser estabilizado. La matriz de estabilidad contiene datos que indican si cada uno de los vértices de ese cubo son estables o no.

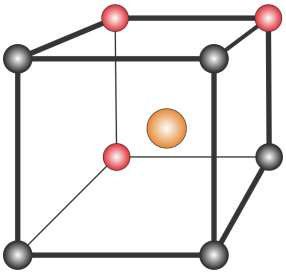
2. Si los puntos envolventes al punto a estabilizar son todos estables, se considera al punto en cuestión como un punto estable. Cabe destacar que este criterio no garantiza que el punto a estabilizar sea estable (ver figura 3.10a).

3. Si no todos los puntos envolventes son estables pero hay por lo menos uno que sí lo sea, entre los puntos estables encontrados en el cubo envolvente se selecciona uno de esos puntos como el nuevo punto estabilizado. El algoritmo devuelve ese punto como punto estabilizado. (ver figura 3.10b)

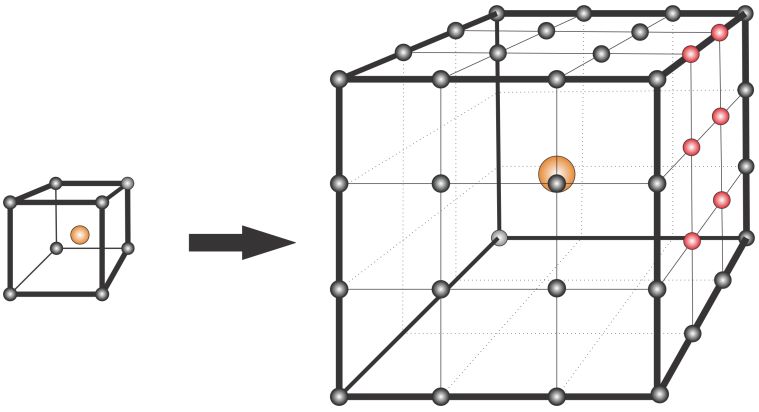
4. Si ningún punto envolvente es estable, se pasa a buscar puntos estables sobre el cubo próximo mayor que envuelve al punto a estabilizar. En este caso, la búsqueda se hace tanto sobre los puntos que forman los vértices de dicho cubo como sobre los puntos que se encuentran sobre las caras de dicho cubo. Si se encuentra algún punto estable, el algoritmo devuelve dicho punto como el nuevo punto estabilizado. Si no se encuentra ningún punto estable, se pasa a buscar los puntos estables sobre el cubo próximo mayor al actual hasta encontrar algún punto estable (ver figura 3.10c).



(a) El punto a estabilizar (color naranja) tiene todos los puntos vértices de la grilla que forman un cubo a su alrededor estables (puntos de color rojo). En este caso el algoritmo de estabilidad devuelve al mismo punto a estabilizar como punto estable



(b) El punto a estabilizar (color naranja) tiene algunos puntos de los vértices de la grilla que forman un cubo a su alrededor estables (puntos de color rojo) y otros no (puntos de color negro). En este caso el algoritmo devuelve uno de los puntos rojos como punto estable

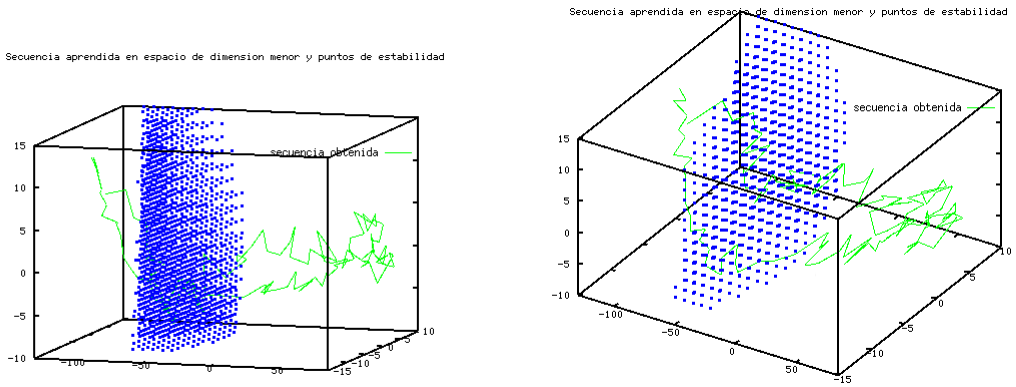


(c) El punto a estabilizar (color naranja) no tiene niguno de los puntos de los vértices de la grilla que forman un cubo a su alrededor estables (todos los puntos son negros). En este caso se procede a buscar algún punto estable en el cubo envolvente al punto. Como se observa en la imagen en ese siguiente cubo se encuentran algunos puntos de estabilidad (puntos de color rojo). En este caso el algoritmo devuelve uno de los puntos rojos como punto estable

Figura 3.10: Algoritmo de estabilidad. En las distinas figuras se puede observar las situaciones descriptas en los pasos 2, 3 y 4 del algoritmo de estabilidad (ver algoritmo 3.5)

Como se puede observar en cada paso del algoritmo se busca el punto estable más cercano al punto a estabilizar. El algoritmo de búsqueda utilizado es más eficiente que realizar la comparación de distancias de cada punto estable de la grilla al punto a estabilizar (ya que realiza la búsqueda sobre un subconjunto de los puntos estables) y a su vez brinda una función de distancia próxima a la real.

En la figura 3.11 se puede observar un ejemplo de puntos de estabilidad generados y la trayectoria de una secuencia de posturas en R^3.



(a) (b)

Figura 3.11: Matriz de estabilidad y trayectoria de la secuencia de posturas donde se pueden observar desde distintos ángulos los puntos de estabilidad generados (en azul) y la trayectoria de la secuencia de posturas en R^3 (en verde)

3.3.5. Función de fitness

La función de fitness del algoritmo genético mide la adaptación del individuo al problema que se quiere resolver (ver subsección 2.1.3). En el caso particular del trabajo realizado un individuo que tiene mayor fitness respecto a otro es un individuo mejor adaptado en la realización del gesto de atajada. A partir del fitness del individuo el algoritmo realiza distintas acciones respecto al mismo (lo cruza con otro individuo o no, o lo descarta o no).

Para poder medir el fitness del individuo se utiliza un simulador físico -ver subsección 3.3.5.1- del robot (partes del robot con su peso, medidas y articulaciones) y su entorno (piso, gravedad, etc.), el cual permite ejecutar la secuencia de posiciones de los motores en el tiempo indicado y obtener datos de la ejecución del robot para poder medir el desempeño de ese individuo en la realización de la tarea.

Como indicador de adaptación en la realización de la tarea por simplicidad se tomó en cuenta solamente la estabilidad del robot durante la realización de la misma. Debido a que los individuos son creados a partir de datos de cómo un humano realiza el gesto de atajada, los mismos ya nacen realizando ese gesto, pero por diferencias en la estructuras corporales del maestro y aprendiz en principio no están adaptados para realizarlo sin caerse. Otros indicadores de adaptación que podrían ser útiles pero que no se utilizaron son el tiempo en que se realiza la tarea, la posición relativa de los brazos del robot al cuerpo al momento de hacer el gesto de atajada de forma de maximizar el área que cubre el cuerpo del robot y el torque mínimo al realizar la tarea si se ejecutara la misma con torque variable.

A continuación se describe el simulador utilizado para poder ejecutar en el robot la secuencia de ángulos que representa cada individuo (ver subsección 3.3.5.1) y luego se describe la función que se aplica sobre los datos de la ejecución de la tarea en el simulador para poder medir la estabilidad (ver subsecciones 3.3.5.2 y 3.3.5.3). Con el fin de evaluar la estabilidad del robot durante la ejecución de la tarea se desarrollaron dos funciones diferentes aunque en la práctica se aplico la función de fitness por posturas. Primeramente se desarrolló una función de evaluación que mide el tiempo que el robot estuvo sin caerse. Finalmente se desarrolló otra función que determina la postura que el robot tiene al

momento de caerse y calcula el fitness en base a ese dato (ver discusión en subsección 4.1.1.2).

3.3.5.1. Simulador Bioloid Control

Bioloid Control fue el simulador utilizado en el desarrollo del proyecto. El mismo es de código libre, disponible en la página del proyecto Bioloid Control (ver [wdBC14]). Posee los archivos de configuración para el robot humanoide de Bioloid.

El proyecto Bioloid Control consiste de:

- Controlador de robot. Es el programa que se le carga al CM-5 del robot. Realiza la lectura y escritura desde y hacia los servos y sensores del robot.
- Visualización del robot con OpenGL. El programa tiene un visualizador del estado del robot que lee información poligonal desde memoria compartida y la despliega en pantalla.
- Programa principal. El mismo controla al robot en un ciclo cerrado enviando y recibiendo mensajes desde el robot (el robot y el PC donde se ejecuta el programa deben estar conectados por un cable serial).

Junto con el proyecto de Bioloid Control se encuentra también un simulador físico del robot, que a su vez también viene con archivos de configuración que implementan el robot humanoide (ver figura 3.12). Este simulador físico utiliza el motor físico ODE. Posee las siguientes características:

- Escrito en C++ con orientación a objetos.
- Simulador del bus bioloid implementado por TCP.
- Física realística: dimensiones, masas, inercias tomadas del grupo de investigación Portugeuse.
- Parámetros del robot creados desde un archivo Excel (vía traductor Perl a XML).
- Componentes bioloid (servos, IMUs, etc) reaccionan a paquetes de mensaje de manera realista.
- Código entendible y extensible (código libre).

Como se describió anteriormente el proyecto Bioloid Control podría separarse en 2 partes:

Controlador - implementa un controlador del robot, resolviendo la comunicación con el mismo, ejecución de movimientos con distintos tipos de interpolación, etc.

Simulador - simulador físico, que recibe paquetes con instrucciones para los servos del robot, tal como si fuese el robot real y simula la ejecución de dichas instrucciones utilizando el motor físico ODE.

En el marco del proyecto se decidió utilizar tanto el controlador como el simulador físico para realizar las evaluaciones de las secuencias generadas por el algoritmo genético.

El simulador físico es necesario para poder evaluar de una forma cercana a la realidad, la bondad de una secuencia de posturas al ejecutarla en el robot. Se decidió utilizar el controlador, ya que el mismo resuelve el control motor y también la comunicación con el robot real. El controlador permite, a partir de una posición de un motor y el tiempo que se espera que tarde el motor en alcanzarla, realizar el movimiento con esos parámetros en ese motor en particular.

Para lograr que se pudiera evaluar una secuencia de posturas dadas, se tuvieron que hacer cambios sobre el controlador y el simulador, de forma de poder sincronizarlos y obtener dicha evaluación. A priori el simulador no implementa ninguna funcionalidad que permita reiniciar el robot a una posición inicial, ni tampoco obtener posiciones en el mundo simulado de las partes que forman el robot, de manera de obtener los datos que permitan evaluar una secuencia ejecutada. En la documentación de herramientas [San14], se detallan los cambios realizados tanto en el simulador como en el controlador para lograr estas funcionalidades.

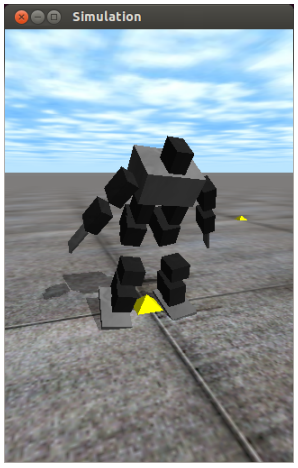


Figura 3.12: Entorno gráfico que brinda el simulador Bioloid Control. En la misma se puede se observar al robot realizando la atajada.

3.3.5.2. Función de fitness por tiempo

En un principio la función de fitness se realizó tomando en cuenta los tiempos de caída del robot. Dicha función devuelve, en caso que la altura del pecho del robot pase por debajo de cierto umbral (que llamaremos umbral inferior), un valor entre 0 y 0.9, dependiendo del tiempo en que el mismo pasó dicho umbral. Se considera que si la altura del pecho del robot es inferior a ese umbral el robot perdió la estabilidad y cayó o está cayendo al suelo.

En caso que la altura del pecho del robot haya pasado por debajo de un umbral más alto (que llamaremos umbral superior) pero no haya pasado por el umbral inferior, se calcula la evaluación del robot entre 0.9 y 1, también dependiendo del momento en que el mismo pasó por debajo del umbral superior. La distinción que se realizó entre umbral superior e inferior, para los casos en que el mismo pasa por el umbral superior pero no por el inferior, sirve como indicativo de que el robot esta desestabilizado y se está por caer aunque durante el tiempo de evaluación de la postura no llego a caerse todavía.

Para el caso de que la altura del pecho del robot cae por debajo del umbral inferior, se devuelve el fitness de dicha ejecución de la tarea de acuerdo a la ecuación 3.9 donde $tiempoCaídaUmbraInferior$ es el tiempo en que se detectó que el robot pasó por la altura correspondiente al umbral inferior, $tiempoInicioSecuencia$ es el tiempo en que se detectó el robot empezó a moverse (que puede ser distinto del tiempo que de comienzo de simulación) y $tiempoSimulación$ es el tiempo que llevó en

realizarse toda la simulación. En cambio, si el robot no cayó por debajo del umbral inferior pero sí por debajo del umbral superior la evaluación queda determinada según la ecuación 3.10 donde *tiempoCaídaUmbralSuperior* es el tiempo en que se detectó que el robot pasó por la altura correspondiente al umbral superior y el resto de las expresiones son similares a la anterior ecuación. En caso que el robot no haya caído por debajo de ninguno de los umbrales, se considera que la ejecución de la secuencia fue estable y el fitness tiene valor 1.

$$fitness = \left(\frac{tiempoCaídaUmbralInferior - tiempoInicioSecuencia}{tiempoSimulación} \right) * 0{,}9 \tag{3.9}$$

$$fitness = 0{,}9 + \left(\frac{tiempoCaídaUmbralSuperior - tiempoInicioSecuencia}{tiempoSimulación} \right) * 0{,}1 \tag{3.10}$$

3.3.5.3. Función de fitness por posturas

Como se mencionó anteriormente la función de evaluación finalmente fue calculada tomando en cuenta la postura que ejecuta el robot en lugar del tiempo de ejecución al momento de caerse. La evaluación por posturas permite evaluar con precisión el porcentaje ejecutado de la secuencia al momento de caída del robot, lográndose una evaluación más precisa comparada con la evaluación por tiempo.

Al igual que en la función de fitness por tiempo se utilizaron los umbrales inferior y superior de forma similar.

La evaluación para el caso que la altura del pecho del robot esté alguna vez por debajo del umbral inferior se realiza según la ecuación 3.11 donde *posturaCaídaUmbralInferior* es el índice de la postura que tiene el robot al momento de pasar por el umbral inferior y *totalPosturasSecuencia* es el número de posturas que tiene la secuencia que esta siendo evaluada. Si la altura del pecho nunca estuvo por debajo del umbral inferior pero si por debajo del superior el fitness se calcula según la ecuación 3.12 donde *posturaCaídaUmbralSuperior* es el índice de la postura que tiene el robot al momento de pasar por el umbral superior y *totalPosturasSecuencia* es el total de posturas de la secuencia que está siendo evaluada. Como en el caso anterior si la altura del pecho del robot no estuvo por debajo de ninguno de los umbrales, se considera que la ejecución de la secuencia fue estable y el fitness se evalúa en 1.

$$fitness = \left(\frac{posturaCaídaUmbralInferior}{totalPosturasSecuencia} \right) * 0{,}9 \tag{3.11}$$

$$fitness = 0{,}9 + \left(\frac{posturaCaídaUmbralSuperior}{totalPosturasSecuencia} \right) * 0{,}1 \tag{3.12}$$

3.3.6. Operadores genéticos

3.3.6.1. Operador de cruzamiento

El operador de cruzamiento implementado realiza el cruzamiento en un punto sobre la matriz de posturas (ver subsección 2.2.3.2). Luego de seleccionar a dos individuos de la población de acuerdo a cierto criterio, a los que llamaremos padres, se aplica el operador de cruzamiento sobre ambos, dando como resultado dos nuevos individuos, a los que llamaremos hijos.

Selección de padres para realizar cruzamiento

Para poder preservar a los mejores individuos de cada población y a su vez mantener diversidad genética, se aplicó un operador de selección que devuelve un padre de entre los mejores individuos y el otro sorteado aleatoriamente de la población. Esta selección de padres representa un buen compromiso entre buen fitness y variabilidad.

Implementación del operador

La aplicación del operador consiste primeramente en seleccionar de manera aleatoria una postura, que será el punto de cruce para generar ambos hijos. Uno de los hijos tendrá como secuencia de posturas la secuencia de uno de los padres hasta la postura sorteada excluyéndola y el resto de la secuencia de posturas se tomará de las posturas de la secuencia del otro padre a partir de la postura sorteada hasta el final. Para generar el otro hijo se procede en forma similar pero tomando los padres de forma inversa. Por otro lado el primer hijo generado tendrá como vector de ejecución de posturas el mismo vector de ejecución de posturas del primer padre y el segundo hijo tendrá como vector de ejecución de posturas el mismo vector que el segundo padre. En la figura 3.13 se puede observar la aplicación del operador de cruzamiento implementado.

En la construcción de este operador se buscó un desarrollo sencillo que transmitiera la información contenida en ambos padres a sus descendientes.

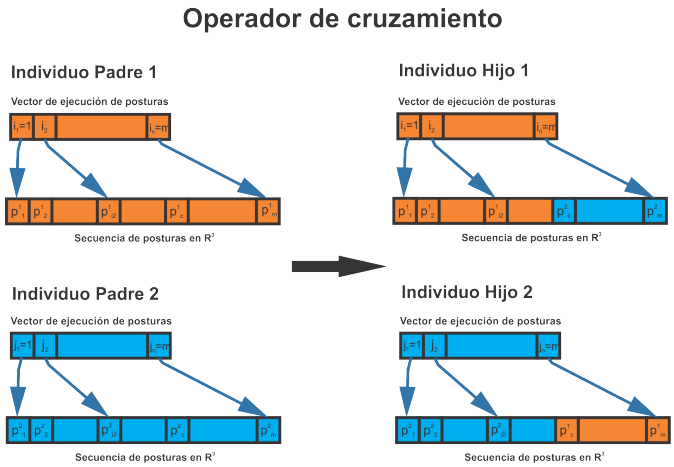


Figura 3.13: Operador de cruzamiento implementado. Como se observa se aplica el cruzamiento en un punto, donde cada hijo tendrá una parte de la secuencia de posturas de uno de los padres y la restante del otro padre. Por otro lado el hijo 1 y el hijo 2 tendrán vectores de ejecución de posturas iguales a los padres 1 y 2 respectivamente

3.3.6.2. Operador de mutación

El operador de mutación implementado opera de forma distinta sobre la matriz de posturas y el vector de ejecución de posturas. A su vez la probabilidad de que se aplique la mutación sobre una u otra estructura es independiente. Cabe destacar que durante el aprendizaje y testeo del algoritmo

genético el parámetro de mutación sobre el vector de ejecución de posturas fue 0, por lo tanto no se aplicó dicho operador a ningún individuo.

Mutación sobre matriz de posturas

El operador de mutación sobre la matriz de posturas fue implementado con la filosofía de mutación sobre un punto. El mismo se implementó de dos formas distintas, una primera forma que realiza mutación polinómica y otra en la que se sortea de forma aleatoria el nuevo gen dentro de un entorno dado pero que luego se aplica un alisado sobre la curva de la trayectoria del comportamiento en R^3. Cabe destacar que en el aprendizaje y testeo del algoritmo genético se utilizó solamente la segunda implementación mencionada anteriormente ya que se vieron mejores resultados que aplicando la primera.

La primer forma aplica mutación en un punto pero con mutación polinomial. Inicialmente se selecciona de forma aleatoria un gen del individuo, que corresponde a una de las posturas de la matriz de posturas. Para esa postura seleccionada en R^3, se aplica mutación polinomial, que permite que la postura mutada se modifique de forma que sea más probable que la posición de la nueva postura esté más cerca de donde estaba inicialmente y con menos probabilidad da como resultado posturas más alejadas. El entorno sobre el cual se puede modificar el punto, queda determinado en base a los mínimos y máximos en cada dimensión de los puntos pertenecientes a una vecindad del punto a ser mutado. En este caso se definió la vecindad como los 5 puntos anteriores y los 5 puntos posteriores al punto mutado. Utiliza una distribución de probabilidad polinomial según la ecuación 3.13 donde x_i^t es el valor del punto a mutar, x_i^L y x_i^U son los valores mínimos y máximos en la vecindad respectivamente, $u_i \sim U[0,1]$ y η es un parámetro que controla la variabilidad de la perturbación y $x_i^{(t+1)}$ es el nuevo valor calculado (ver [Mor11]). En la ecuación se observa que si el valor sorteado u_i cumple que $u_i < 0{,}5$ el nuevo valor calculado estará entre x_i^L y x_i^t y que a su vez si $u_i \geq 0{,}5$ entonces el nuevo valor calculado estará entre x_i^t y x_i^U. En la figura 3.14 se puede observar el comportamiento de este operador para distintos valores de η.

$$x_i^{(t+1)} = x_i^t + (x_i^U + x_i^L).\delta_i \; donde \; \delta_i = \begin{cases} (2.u_i)^{\frac{1}{(\eta+1)}} - 1, & si \; u_i < 0{,}5 \\ 1 - [2.(1-u_i)]^{\frac{1}{(\eta+1)}}, & si \; u_i \geq 0{,}5 \end{cases} \tag{3.13}$$

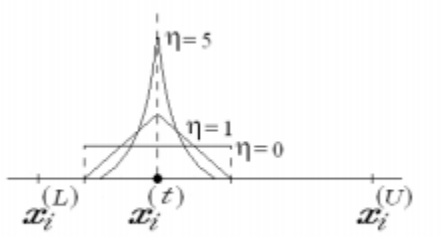


Figura 3.14: Mutación polinomial. Imagen extraída de [Mor11].

El segundo operador de mutación sobre la matriz de posturas desarrollado opera de la siguiente forma. Primeramente se selecciona de forma aleatoria un índice del vector de ejecución de posturas

del individuo, al que llamaremos i_M. Dicho índice indica el punto de R^3 de la matriz de posturas al que se aplicará la mutación p_{i_M}. A su vez en el vector de ejecución de posturas se selecciona el índice anterior i_{M-1} y el posterior i_{M+1} al índice que define la postura o punto que será mutado. Dichos índices indican las posturas o puntos de R^3 que llamaremos $p_{i_{M-1}}$ y $p_{i_{M+1}}$. Para el punto p_{i_M} se sortea aleatoriamente un nuevo valor en un entorno de ese punto. El entorno donde se sortea el nuevo punto al igual que en el caso del operador de mutación polinómica esta definido por el mínimo y máximo de los puntos vecinos a ese punto. La vecindad de ese punto está definida por un valor fijo, en el caso implementado fue 5. Luego, utilizando splines cúbicas se calcula una curva en R^3 que pasa sobre los puntos $p_{i_{M-1}}$, p_{i_M} y $p_{i_{M+1}}$. Todos los genes que se encuentran entre las posturas $p_{i_{M-1}}$ y $p_{i_{M+1}}$ son recalculados de forma de pertencer a esa curva (ver figura 3.15).

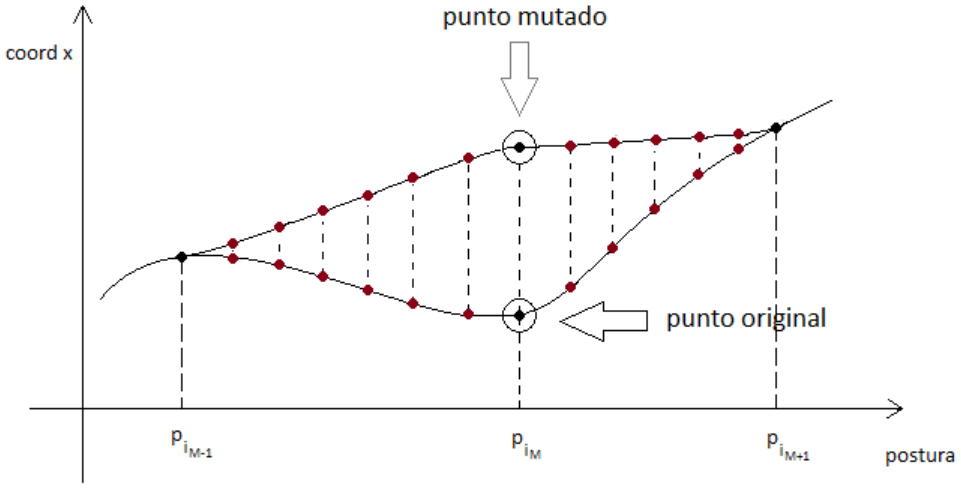


Figura 3.15: Mutación de un punto y alisado de la curva utilizando splines. En la imagen se puede observar la gráfica de coordenada x de las posturas respecto a las posturas de la secuencia, así como la trayectoria generada por las mismas y su coordenada x entre la postura $p_{i_{M-1}}$ y la $p_{i_{M+1}}$ incluidas. A su vez se puede observar la nueva trayectoria definida por el valor de la mutación de la coordenada x de la postura p_{i_M} y los valores de las coordenadas de las posturas $p_{i_{M-1}}$ y la $p_{i_{M+1}}$. También se observan los valores corregidos de la coordenada x de las posturas que están entre $p_{i_{M-1}}$ y la $p_{i_{M+1}}$ para que estén sobre la nueva trayectoria generada.

Mutación sobre vector de ejecución de posturas

El operador de mutación sobre el vector de ejecución de posturas fue implementado como mutación sobre un punto y opera de la siguiente forma (ver subsección 2.2.3.3). Se selecciona de forma aleatoria uno de los índices del vector de ejecución de posturas (que indica la ejecución en el simulador de una postura en particular), distinto del primer y último índice ya que dichos valores son siempre fijos e indican la ejecución de la primer y última postura de la matriz de posturas. Para el índice seleccionado, se sortea un valor en un rango de valores de forma que cumpla con las restricciones de distancia MINDIST y MAXDIST con los índices vecinos.

3.3.6.3. Operador de selección

La selección de la nueva generación se realiza de forma elitista para el 10 % de la nueva población y aleatoria para el restante 90 % de individuos. Estos individuos son seleccionados del conjunto de individuos de la generación anterior más los hijos generados en la generación actual. La selección elitista permite no perder a los mejores, y la parte aleatoria permite mantener cierta diversidad genética y no perder a los parcialmente buenos.

3.3.7. Condiciones de terminación

Para la correcta finalización del algoritmo genético se determinaron condiciones de terminación del mismo. Dichas condiciones se pueden separar en dos tipos distintos, finalización por cantidad de generaciones ejecutadas en el algoritmo y finalización por no mejora de valores de promedio y no encontrar mejores individuos, lo que da un indicio de que la relación esfuerzo-ganancia no vale la pena.

3.3.7.1. Cantidad de generaciones

Esta condición de terminación se aplica luego de que se ejecutan 200 generaciones del algoritmo. Ese valor corresponde con el menor valor que hace que estadísticamente en la mayoría de los casos el algoritmo termine por los otros criterios de terminación. Esa cantidad de generaciones da un marco para que el algoritmo pueda converger a una solución deseable, pero a su vez interrumpe la ejecución en casos que no se produzca mejora en esa cantidad de generaciones. También cabe destacar que al igual que hay un límite máximo de generaciones, se estableció un límite mínimo de generaciones del algoritmo antes de pararlo, dicho límite es 10. Ese valor corresponde al menor valor de generaciones necesarias estadísticamente para encontrar individuos mejores que los de la población inicial (fitness 10 % mejores que los iniciales).

3.3.7.2. Fitness promedio y mejor individuo

Estas dos condiciones de terminación detectan los casos en que el algoritmo no produce mejoras y por tanto no es viable seguir invirtiendo esfuerzo en buscar la solución.

Una de las condiciones que se determinaron es si el fitness promedio de la población no es mejor que el mejor promedio de población encontrado hasta el momento más un epsilon por 20 generaciones. Este valor corresponde con el menor valor que hace que estadísticamente la población promedio sea mejor que el mejor promedio encontrado más un epsilon. Esta condición es tomada en cuenta luego de que el algoritmo encuentra al menos un individuo que haya evaluado de forma estable.

La otra condición de terminación se aplica cuando no se mejora al mejor individuo por más de 10 generaciones seguidas. Este valor es el mínimo valor en el que estadísticamente en la mayoría de los casos el algoritmo encuentra un nuevo individuo mejor.

Capítulo 4

Entrenamiento, testeo y resultados

4.1. Puesta a punto del algoritmo

Antes de poder realizar el entrenamiento y testeo del algoritmo genético implementado se realizó una puesta a punto del mismo. En la misma se hizo la evaluación y ajuste de los operadores genéticos y de la función de evaluación, de modo de poder llegar a obtener un resultado deseable, el cual es que a través de aplicar dicho algoritmo genético se logre la adaptación de la tarea al robot. También se realizó una búsqueda de posibles valores de los parámetros del algoritmo (tamaño de población, probabilidad de cruzamiento y mutación) de forma de poder definir los entornos a probar en una etapa de calibración del algoritmo.

Como resultado de esta actividad se logró obtener un versión del algoritmo genético definiendo a los operadores genéticos, función de evaluación, etc. que serían usados y un entorno de valores para cada uno de sus parámetros, donde se pudo observar que el algoritmo converge a una solución, la cual logra mejorar el fitness promedio de las poblaciones y a su vez permite encontrar soluciones que evalúan de forma satisfactoria.

4.1.1. Ajustes y modificaciones realizados

4.1.1.1. Testeo y modificación de operadores

Se implementaron y testearon los operadores descriptos en el cuadro 4.1, donde también se puede ver cuáles fueron finalmente utilizados en el algoritmo genético.

Cruzamiento

Selección de padres para cruzamiento Se implementaron tres operadores distintos:

- Primeramente se aplicó el operador de selección llamado NTournament. Dicho operador sortea aleatoriamente n individuos de toda la población y entre los individuos sorteados selecciona el de mayor fitness.
- Luego se aplicó un operador de selección aleatorio, que sortea un individuo del conjunto de individuos de la población con la misma probabilidad.

Tipo de Operador	Implementaciones	Aplicado en algoritmo
Selección para cruzamiento	Selección NTorunament	No
	Selección aleatoria	No
	Un padre de entre los mejores individuos y el otro seleccionado aleatoriamente	Si
Cruzamiento	En un punto sobre vector de ejecución de posturas y vector de posturas	No
	En un punto sobre vector de posturas y copia del vector de ejecución de posturas	Si
Mutación	Mutación polinómica en un punto sobre vector de posturas y mutación aleatoria en un punto sobre vector de ejecución de posturas	No
	Mutación aleatoria en un punto sobre posturas con alisamiento de curva. Sin mutación sobre vector de ejecución de posturas	Si
Selección	2 mejores padres y el resto de la población de los mejores hijos	No
	10 % mejores individuos entre padres e hijos y el resto de la población de forma aleatoria	Si

Cuadro 4.1: Implementaciones de operadores testeadas

- Finalmente, con la intención de preservar los mejores genes de cada población, pero a su vez mantener diversidad genética, se aplicó un operador de selección, el cual retorna un padre de entre los mejores individuos y el otro lo sortea aleatoriamente del resto de la población.

En la aplicación de los distintos operadores de selección de padres no se observaron cambios importantes en la eficiencia y eficacia del algoritmo.

Implementación del operador El operador de cruzamiento desarrollado realiza el cruzamiento en un punto de ambos individuos (ver subsección 3.3.6.1) . Dada la codificación de cada individuo la implementación de este operador presenta ciertas dificultades. Como se describe en la subsección 3.3.3, cada individuo se define por una matriz de posturas y a su vez por un vector de ejecución de posturas que indica cuáles posturas de la matriz de posturas se deben ejecutar en el simulador al evaluar dicho individuo.

Se implementaron las siguientes dos variantes:

- Primeramente se implementó un operador de cruzamiento que realiza el cruzamiento en un punto sobre el vector de ejecución de posturas y a partir de la postura indicada por el punto de cruzamiento realiza el cruzamiento sobre la matriz de posturas. Por las restricciones de la codificación, no fue posible realizar un cruzamiento sobre el vector de ejecución de posturas que fuera simétrico, o sea en un cruzamiento donde se obtuvieran dos vectores de ejecuciones de postura hijos en el que ambos tuvieran genes de ambos padres, ya que al realizar el cruzamiento se pueden violar algunas de las restricciones impuestas a la codificación del individuo. Por lo tanto el operador primeramente implementado devuelve un vector de ejecución de posturas hijo, en el cual uno de los padres aporta la primera parte del vector y el otro la parte final de dicho

vector, y en el cual se ajustan los valores de los genes donde se da el cruce (y si fuere necesario otros genes también) de forma de seguir respetando la estructura del vector de ejecución de posturas, donde cada gen dista de sus genes vecinos, más y menos que ciertos valores mínimos y máximos respectivamente.

- Finalmente, debido a que la parte del individuo que mejor define su comportamiento es la matriz de posturas y no el vector de ejecución de posturas se implementó el operador de cruzamiento descripto en la subsección 3.3.6.1. El mismo realiza un cruzamiento en un punto sobre la matriz de posturas y devuelve el vector de ejecución de posturas de uno de los padres para generar uno de los hijos y el vector de ejecución de posturas del otro padre para generar el otro hijo.

Mutación

Dado la codificación de las soluciones, el operador de mutación puede realizar la mutación tanto en el vector de ejecución de posturas como en la matriz de posturas. Se implementaron variantes del operador sobre ambas estructuras del individuo (ver subsección 3.3.6.2).

Mutación sobre vector de ejecución de posturas

Se implementó un operador:

- Sobre el vector de ejecución de posturas, primeramente se implementó la mutación aplicandola sobre un gen del mismo que es sorteado aleatoriamente. A su vez para modificar el valor de ese gen se sortea, de forma equiprobable, algún valor en el rango de valores que respetan las restricciones impuestas a la codificación del vector de ejecución de posturas.

Finalmente se descartó realizar la mutación sobre dicho vector ya que según lo observado en algunas pruebas la inserción de dicho operador no produce mejoras significativas al algoritmo.

Mutación sobre matriz de posturas

Se implementaron dos operadores:

- Inicialmente se aplicó la mutación polinómica en algún gen sorteado de forma aleatoria. Debido a la característica de dicho operador, que modifica el punto (gen) mutado a un punto cercano con mayor probabilidad, la implementación del mismo no producía los cambios necesarios sobre los individuos, de forma de llegar en un tiempo razonable (200 generaciones) a una solución que fuera estable. Debido a lo mencionado previamente se probó la implementación de un operador de mutación diferente.

- El operador implementado finalmente utilizado en el algoritmo fue el de mutación de un gen del individuo elegido de manera aleatoria. El valor del nuevo gen es sorteado en un entorno definido por valores máximos y mínimos de un entorno de puntos vecinos al mismo. A su vez, también se modifican los puntos vecinos al punto mutado, que causa el efecto que se puede describir como alisamiento de la curva alrededor del punto mutado (ver figura 3.15).

Como resultado de la aplicación del segundo operador de mutación al algoritmo genético se observó que el algoritmo encuentra soluciones estables.

Selección

Se implementaron dos operadores:

- Selección de los 2 mejores padres de la población y el restante n-2 mejores hijos.
- Selección del 10 % de los mejores individuos de la población y el resto de la población de forma aleatoria. Este operador pretende preservar a los mejores individuos de la población con el 10 % de selección elitista y a su vez mantener diversidad genética de la población seleccionando individuos de forma aleatoria

4.1.1.2. Testeo y modificación de la función de evaluación

Como se mencionó previamente en la subsección 3.3.5, se evalúa la estabilidad de la secuencia reproducida. O sea que una secuencia que es realizada de forma que el robot no se caiga se evalúa con el mayor fitness posible, o sea con valor 1. No se toma en cuenta en la evaluación de la misma aspectos de calidad de la imitación sino simplemente la estabilidad del robot al imitar. Primeramente se realizó la evaluación tomando los tiempos de caída del robot. Debido al efectos de la aplicación de los operadores de cruzamiento y mutación sobre las soluciones del algoritmo, alguna solución podría llegar a cambiar de forma de ejecutar la misma secuencia de posturas pero en más tiempo, evaluándose de forma distinta según la función de evaluación definida. Por ejemplo, el robot podría realizar casi toda una secuencia en la mitad de tiempo que otra secuencia similar, y en ese momento caer y esto resultaría en una evaluación bastante inferior a que si el robot ejecutara la misma secuencia de forma más lenta. Tomando en cuenta esos posibles comportamientos, se decidió realizar el cambio de evaluación para evaluar el comportamiento en base a la postura que tiene el robot al momento de caer.

4.1.1.3. Búsqueda de entornos de valores de parámetros para calibración

En la puesta a punto del algoritmo se realizaron distintas ejecuciones modificando los valores de los parámetros de modo de encontrar algún rango de valores donde fuera factible realizar la calibración. En cada corrida del algoritmo se buscó que tanto el fitness promedio como el mejor individuo de cada generación fuera creciente con relación a la cantidad de generaciones que se ejecutan en el algoritmo.

En esta etapa según las ejecuciones realizadas se pudo observar que el tamaño de la población del algoritmo era medianamente influyente en los resultados del algoritmo, logrando mejores resultados (llegar a individuos estables en menos generaciones), para poblaciones entre 50 y 70 individuos, pero también con poblaciones menores. A su vez se observó que la tasa de mutación necesaria para lograr resultados en relativamente pocas generaciones (máximo 200), debía ser bastante grande (igual o superior a 5 por ciento). También se observaron distintos resultados en el algoritmo variando la tasa de cruzamiento del mismo, donde siempre el algoritmo convergía más rápidamente con una tasa de cruzamiento igual o superior al 70 por ciento.

4.1.1.4. No determinismo de las soluciones

Pese a que luego de realizar la puesta a punto del algoritmo el mismo lograba encontrar soluciones de mayor fitness, inclusive soluciones estables, se observa que a medida que el algortimo encuentra soluciones cercanas a ser estables, las mismas varian mucho al volver a ser ejecutadas en el simulador, con la consecuente variación de su evaluación.

Pruebas

Para soluciones puntuales del algoritmo, se observó que al volver a ejecutar muchas veces una solución que había evaluado como estable, la misma volvía a ser evaluada como estable aproximadamente entre el 5 y 10 por ciento de las evaluaciones.

Modificaciones al algoritmo genético

Para poder solucionar el problema observado se pensaron dos posibles soluciones. Una es modificar los operadores del algoritmo genético de forma que los individuos que el mismo encuentra tengan una naturaleza distinta, pero de modo que la ejecución de los mismos en el simulador sea estable. Esta solución se descartó porque no se encontró la forma de poder realizarla. La otra solución es la de crear un mecanismo que detecte individuos que varian mucho en su evaluación entre distintas evaluaciones y descartarlas del algoritmo.

Se modificó el comportamiento del algoritmo genético realizando la reevaluación en cada generación del 10 por ciento mejor de la población. De esa forma si un individuo que dentro de la población esta evaluado como muy bueno (ya que pertenece al 10 % mejor de la población), tiene como característica que su evaluación varía mucho al reevaluarla, probablemente será descartado.

4.1.2. Conclusiones

Luego de la etapa de puesta a punto del algoritmo, en la que se realizaron algunas modificaciones a los operadores genéticos y se encontró un rango de valores de cada parámetro en el que se observó el correcto funcionamiento del algoritmo, se puede estimar que el algoritmo puede resolver el problema de encontrar un conjunto de individuos considerablemente mejores (en término de fitness y consecuentemente de estabilidad al realizar la tarea observada), y a su vez encontrar algunos individuos en los que su ejecución en el simulador es estable.

A su vez se pudo observar que los individuos encontrados no son deterministas al ejecutarlos en el simulador. Esto significa que al volver a ejecutar en el simulador las soluciones que el algoritmo calificó como estables no siempre producen un comportamiento en el que el robot realiza la tarea sin caerse. Pese a que las soluciones encontradas no son deterministas, pensando en la aplicación de las mismas al robot real, igualmente sirven como base para encontrar una solución en el robot real que ejecute de forma correcta luego de la correspondiente adaptación de la misma al robot real.

4.2. Aprendizaje del algoritmo

Luego de concluída la etapa de puesta a punto del algoritmo, primeramente se realizó la etapa de calibración de los parámetros y luego el testeo según los resultados obtenidos. Para realizar la calibración se utilizaron los datos obtenidos de la grabación de una persona al realizar el gesto de atajada y para testear el algoritmo se utilizaron los datos obtenidos de otras dos secuencias grabadas. De esta forma se puede verificar el comportamiento del algoritmo con un conjunto de datos diferente al utilizado en la calibración del mismo.

4.2.1. Calibración

Se calibraron los siguientes parámetros en la ejecución del algoritmo genético: población inicial (p_i), probabilidad de aplicación del operador de cruzamiento (p_c) y probabilidad de aplicación del operador

de mutación (p_m) (ver aplicación similar en [Ben12]).

4.2.1.1. Aproximación

En la etapa de aproximación se eligieron entornos de valores para cada parámetro a calibrar según lo obtenido en las ejecuciones en la etapa de puesta a punto del algoritmo (ver sección 4.1). Luego de esta etapa se obtuvieron valores aproximados de los parámetros del algoritmo de acuerdo al criterio de selección de los valores de los parámetros. En el cuadro 4.2 se pueden observar los valores utilizados en la etapa de aproximación.

	Mínimo	Máximo	Incremento
pi	30	70	20
p_c	0.7	0.9	0.1
p_m	0.05	0.15	0.05

Cuadro 4.2: Valores de parámetros utilizados en la etapa de aproximación.

El criterio utilizado para la selección de valores tanto en la etapa de aproximación como en la de refinamiento fue el de relevancia estadística, donde dados dos algoritmos A y B (en este caso determinados por una configuración de parámetros de población inicial, probabilidad de cruzamiento y mutación), se considera que el algoritmo A es mejor que el B si la diferencia entre los fitness promedio de las poblaciones finales alcanzadas por cada algoritmo es estrictamente mayor que la mayor de las desviaciones estándar de cada población. En la ecuación 4.1 se expresa dicha desigualdad.

$$f_{AVG}(A) - f_{AVG}(B) > max(std(A), std(B)) \tag{4.1}$$

Para determinar el algoritmo seleccionado en cada etapa se siguen los pasos del algoritmo 4.1, donde primeramente se testean todos los algoritmos entre sí de acuerdo al criterio de relevancia estadística, luego si hubiere más de un algoritmo solución se desempata según la cantidad de individuos estables que generó dicho algoritmo en la población final y por último según el fitness promedio de la población final.

Algoritmo 4.1 Criterio de selección de algoritmo (con una cierta configuración de parámetros) del algoritmo genético

1. Se testean todos los algoritmos (cada uno con cierta configuración de parámetros) entre sí. Como resultado se obtiene un conjunto de algoritmos solución S de la forma $S = \{A \,/\, \nexists B,\, B > A\}$, o sea que los algoritmos de S son tales que no tienen ningún otro algoritmo mejor de acuerdo al criterio de relevancia estadística.

2. Del conjunto S se selecciona los algoritmos que generaron mayor cantidad de individuos estables en sus poblaciones finales, obteniéndose el conjunto de algoritmos S'.

3. Por último, si fuere necesario del conjunto S' se selecciona el algoritmo con mayor promedio de fitness en sus poblaciones finales.

Como resultado los parámetros que devolvió el algoritmo en la etapa de aproximación fueron: población inicial: 70, probabilidad de cruzamiento: 0.70 y probabilidad de mutación: 0.10.

4.2.1.2. Refinamiento

En la etapa de refinamiento se eligieron nuevos entornos de valores para cada parámetro a calibrar, de acuerdo a los valores obtenidos en la etapa de aproximación (ver subsección 4.2.1.1). En el cuadro 4.3 se pueden observar los valores utilizados en la etapa de refinamiento de la calibración.

	Mínimo	Máximo	Incremento
pi	60	80	10
p_c	0.65	0.75	0.05
p_m	0.075	0.125	0.025

Cuadro 4.3: Valores de parámetros utilizados en la etapa de refinamiento.

El criterio utilizado para la selección de valores fue similar al utilizado en la etapa de calibración. Como resultado los parámetros que devolvió el algoritmo fueron: población inicial: 70, probabilidad de cruzamiento: 0.70 y probabilidad de mutación: 0.075.

4.2.2. Testeo de parámetros aprendidos

Para poder verificar el resultado de la calibración del algoritmo genético, se utilizaron datos de dos demostraciones de gestos de ajada de similares características a las que tiene la demostración con la que se realizó el entrenamiento del algoritmo. Se realizaron 10 corridas del algoritmo genético con los mismos parámetros que retornó la calibración (5 veces con cada una de las demostraciones). En el cuadro 4.4 se puede observar los valores del promedio de fitness del mejor individuo de la población final de cada corrida y el fitness promedio de las poblaciones finales de cada corrida. A su vez en las figuras 4.1, 4.2 y 4.3 se pueden observar el mejor, peor y fitness promedio con desviación estándar de cada generación para cada corrida en el aprendizaje y testeo con la secuencia 1 y 2 respectivamente.

Etapa	Promedio fitness mejor individuo	Promedio fitness población
Aprendizaje	0.844	0.678
Testeo-Secuencia1	0.785	0.734
Testeo-Secuencia2	0.622	0.495

Cuadro 4.4: Promedio de fitness de mejor individuo y promedios de la población final en las corridas del aprendizaje y testeo

En los resultados se observa que tanto en la calibración como en el testeo utilizando las dos secuencias se mejora el mejor individuo y el promedio de la población.

En los resultados de la calibración (realizada con la secuencia demostrada 3), en tres de las cinco corridas el algoritmo encuentra individuos estables, aunque solamente en una de esas tres corridas se encuentra algún individuo que evalúa de forma estable en la última generación.

En el testeo utilizando la secuencia 1 solamente en una de las cinco corridas el algoritmo encuentra a un individuo que evalúa de forma estable, aunque el promedio del mejor fitness se acerca al mismo promedio de la calibración y el promedio de fitness de las poblaciones finales incluso lo supera.

En el testeo utilizando la secuencia 2, se observan valores de fitness del mejor individuo y del promedio de la población bastante menores que en la calibración y en el testeo de la secuencia 1. Esto

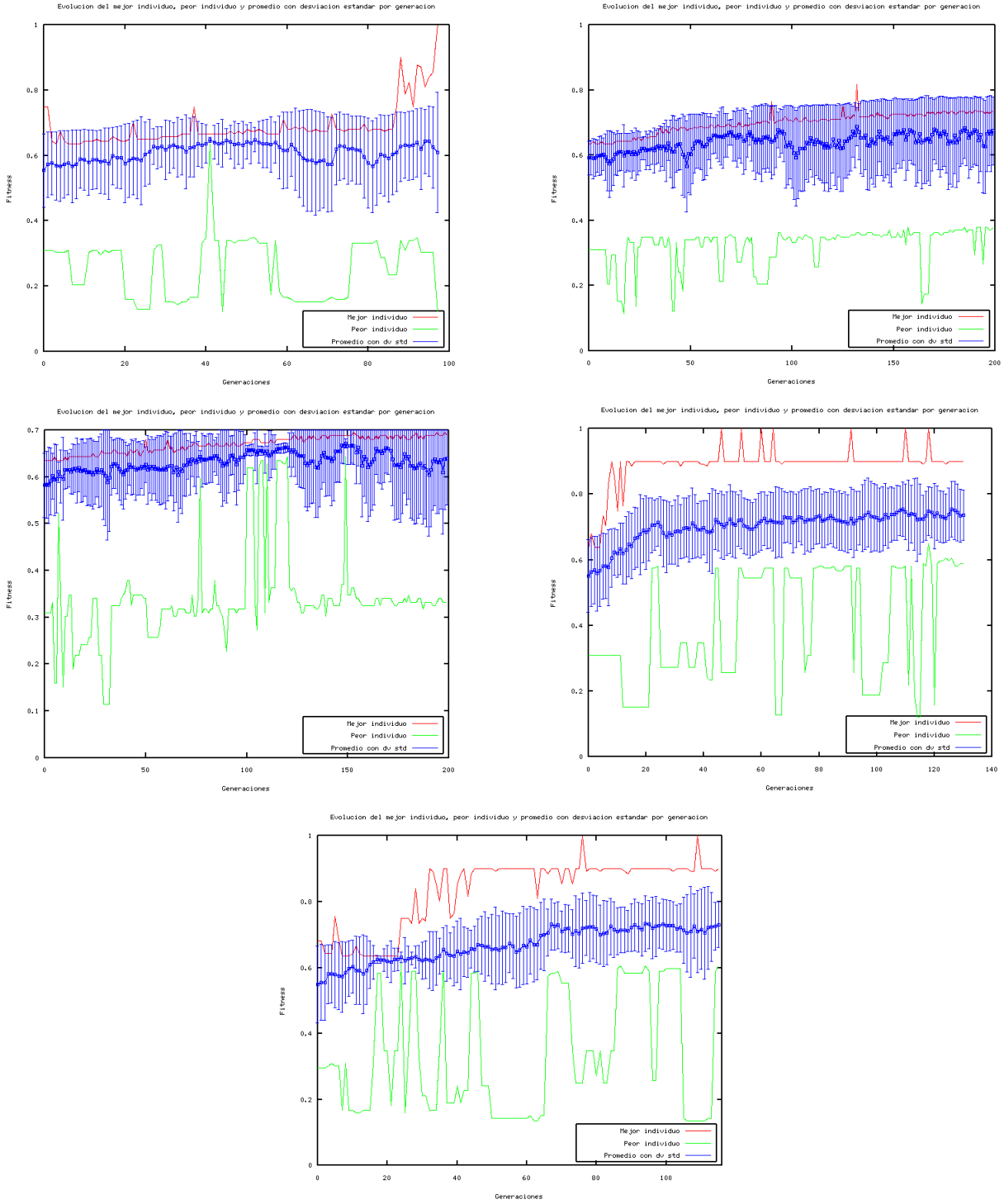


Figura 4.1: Gráficas de mejor, peor y fitness promedio con desviación estándar de cada generación en las 5 corridas al realizar la calibración con la secuencia 3.

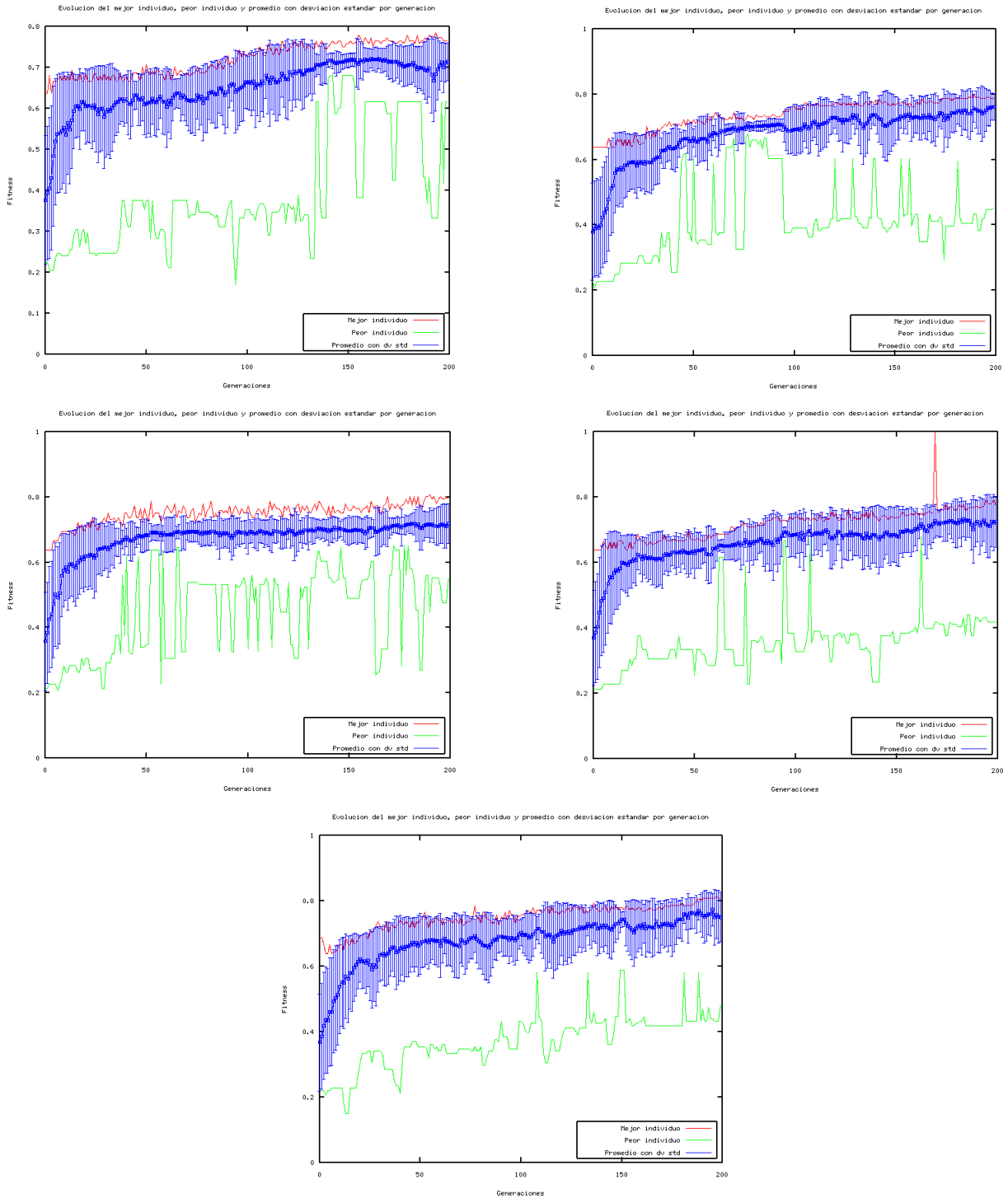


Figura 4.2: Gráficas de mejor, peor y fitness promedio con desviación estándar de cada generación en las 5 corridas del testeo de aprendizaje utilizando la secuencia 1 de demostración de la atajada.

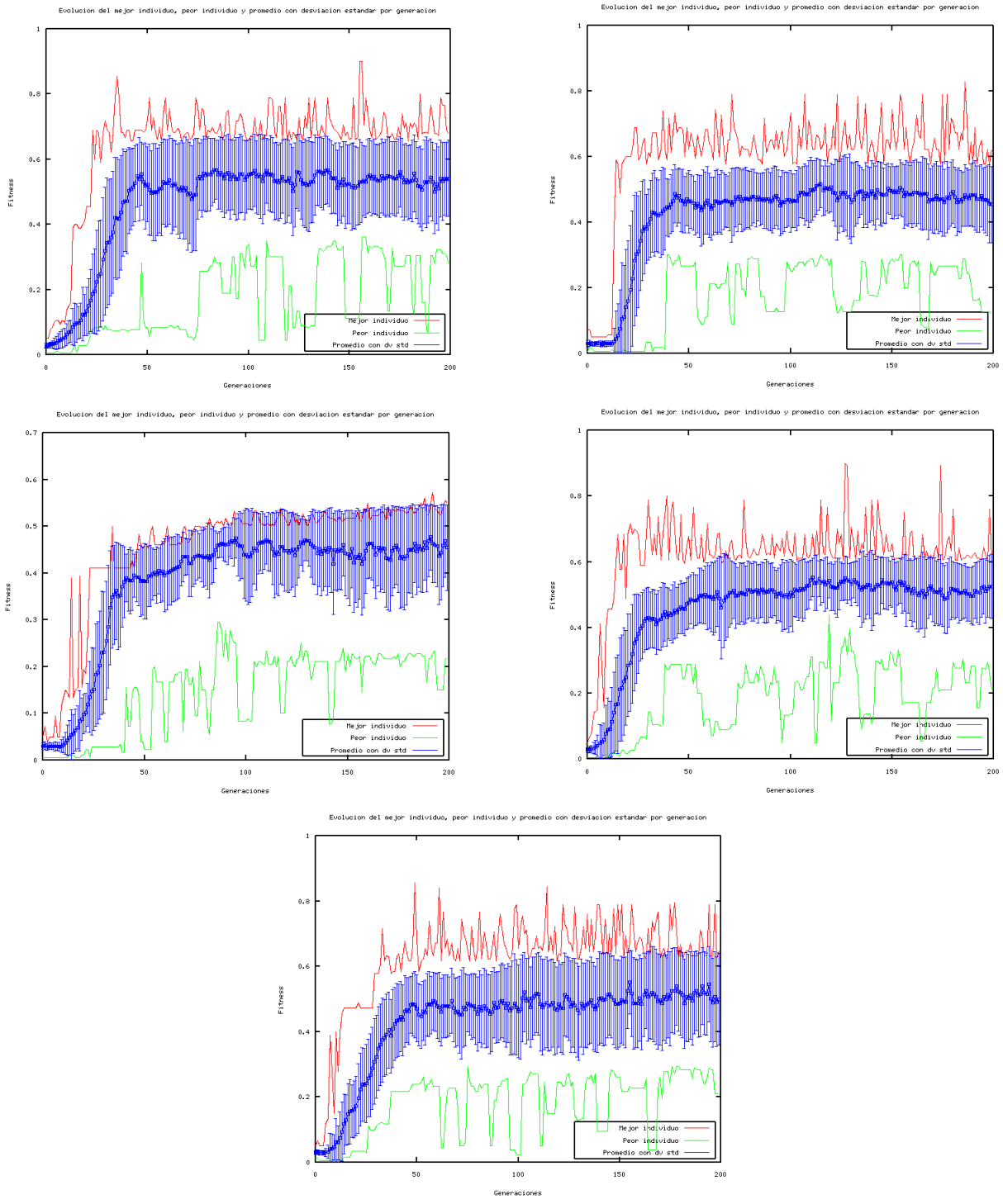


Figura 4.3: Gráficas de mejor, peor y fitness promedio con desviación estándar para cada generación para las 5 corridas del testeo de aprendizaje utilizando la secuencia 2 de demostración de la atajada.

se puede explicar debido a que la secuencia 2 (162 posturas en 3.62 segundos) es bastante más larga que la secuencia 3 (119 posturas en 2.36 segundos) y la secuencia 1 (127 posturas en 2.52 segundos) utilizadas para la calibración y testeo respectivamente. En la demostración de la secuencia 2 el maestro se mantiene por mucho tiempo agachado al realizar el gesto de atajada mientras que en las otras dos secuencias ese tiempo es bastante menor. Debido a la forma de los pies del robot, en la posición en cuclillas queda en desequilibrio estático. En la ejecución del algoritmo genético utilizando las secuencias 1 y 3, se observa que el robot se equilibra sobre la punta de sus pies para mantener la estabilidad en esa posición y luego toma impulso hacia atrás y hacia adelante para volver a la posición vertical. Como en la secuencia 2 el tiempo en que está agachado es demasiado grande, el algoritmo genético no logra encontrar a partir de esa secuencia un comportamiento que equilibre al robot mientras está agachado.

En el testeo de ambas secuencias, el menor promedio de fitness observado en los individuos inciales en cada corrida de los algoritmos (0.2 menos de fitness aproximadamente para la secuencia 1 y 0.5 menos de fitness aproximadamente para la secuencia 2), se puede explicar por el hecho de que la matriz de estabilidad (que se utiliza en el algoritmo de estabilidad y produce una mejor evaluación de fitness de los individuos inciales) fue calculada utilizando las transformaciónes de la secuencia 3 (que es utilizada en la calibración). Como todas las secuencias son diferentes (aunque con características similares), la transformación que se genera al aplicar PCA y por tanto las reproyecciones de los puntos que están en la dimensión R^3 difieren. Esto hace que un punto de R^3 que según la matriz de estabilidad (calculada con la transformación de PCA de la secuencia 3) es estable, no lo sea para las secuencias 1 y 2.

4.2.3. Conclusiones

En la etapa de calibración del algoritmo se observa que el algoritmo genético encuentra individuos estables y logra mejorar el promedio de fitness de las poblaciones para distintos conjuntos de parámetros (población inicial, probabilidad de cruzamiento y mutación).

En la etapa de testeo de las secuencias se observa que los parámetros óptimos obtenidos en la etapa de calibración permiten al algoritmo lograr un buen rendimiento con una de las secuencias y un rendimiento no tan bueno con la otra. Esto probablemente se deba a que una de las secuencias es bastante parecida a la utilizada en la calibración y la otra difiere en el tiempo y la forma en cómo es llevada a cabo. Por otro lado, sería razonable que si se realiza el cálculo de la matriz de estabilidad para cada secuencia que se va a testear, el algoritmo logre un mejor promedio de fitness de población inicial y esto llevaría probablemente a obtener un mejor promedio de fitness de la población final y mejores individuos.

4.3. Testeo con otras tareas

Para probar más ampliamente el algoritmo implementado y calibrado con la tarea de atajada, se aplicó el mismo utilizando los valores calibrados para llevar a cabo el aprendizaje de tareas distintas a la atajada. Estas son un saludo y un gesto de pateo de pelota.

En el caso del saludo, al realizar la secuencia el maestro está parado sin movimiento en sus piernas, por lo que se estimó que la ejecución de la secuencia sobre el robot podría ser estable o muy cercana a la estabilidad. Al realizar la imitación del saludo (esto es replicar los ángulos mapeados desde el maestro sin aplicar el algoritmo de aprendizaje) se observó que algunas ejecuciones son estables. Sin embargo, debido al problema de correspondencia y al movimiento que realiza con los brazos para saludar, el robot se cayó en algunas ejecuciones de la secuencia.

Por otro lado, en el caso del gesto de pateo de pelota, durante casi toda la secuencia el maestro está en equilibrio en solo una pierna y moviendo la otra como para realizar el pateo. En este caso, en todas las pruebas de ejecución de la imitación de la tarea se observó que el robot se caía.

4.3.1. Resultados

Se realizaron tres corridas del algoritmo genético para el saludo y una corrida para el gesto de pateo (utilizando los mismos parámetros que retornó la calibración con datos de la atajada). Los resultados obtenidos al ejecutar el saludo y el gesto de pateada de pelota se pueden observar en el cuadro 4.5, donde para cada tarea se ven los valores del promedio de fitness del mejor individuo de la población final de cada corrida y el fitness promedio de las poblaciones finales de cada corrida. A su vez en las figuras 4.4 y 4.5 se pueden observar el fitness mejor, peor y promedio con desviación estándar de cada generación para cada corrida en el testeo de las secuencias de saludo y gesto de pateo respectivamente.

Tarea	Promedio fitness mejor individuo	Promedio fitness población
Saludo	1	1
Gesto de pateada	0.594	0.467

Cuadro 4.5: Promedio de fitness de mejor individuo y fitness promedios en corridas de las tareas de saludo y gesto de pateo

4.3.2. Conclusiones

En el caso de la secuencia de saludo, aunque inicialmente algunos individuos evaluaron de forma estable, se observa que en todas las corridas el algoritmo genético logró encontrar poblaciones con todos los individuos estables. Se puede decir que los parámetros calibrados para el caso de la tarea de atajada ayudan a que el algoritmo converja a buenas soluciones.

En el caso de la secuencia del gesto de pateo, aunque no se logró poblaciones con promedio de fitness alto, en algunas generaciones se encontraron secuencias que evaluaron de forma estable. Para esta tarea quizás los parámetros calibrados para la atajada no sean los que dan las mejores soluciones y habría que volver a calibrar el algoritmo genético para obtener mejores resultados.

4.4. Pruebas en robot

Debido a que el robot es una herramienta sensible, para minimizar la probabilidad de daños al mismo se utilizó uno de los programas que viene por defecto con el kit de Bioloid, llamado Motion Editor (ver descripción del mismo en [duB], ver figura 4.6) en las ejecuciones de las secuencias. El mismo permite cargar una lista de posturas, a su vez para cada postura la velocidad con que se ejecutará junto con un parámetro de tiempo de espera luego de la ejecución de cada una. En las pruebas en el robot, se decidió ejecutar únicamente las secuencias de atajada que evaluaron en el simulador de forma estable o casi estable.

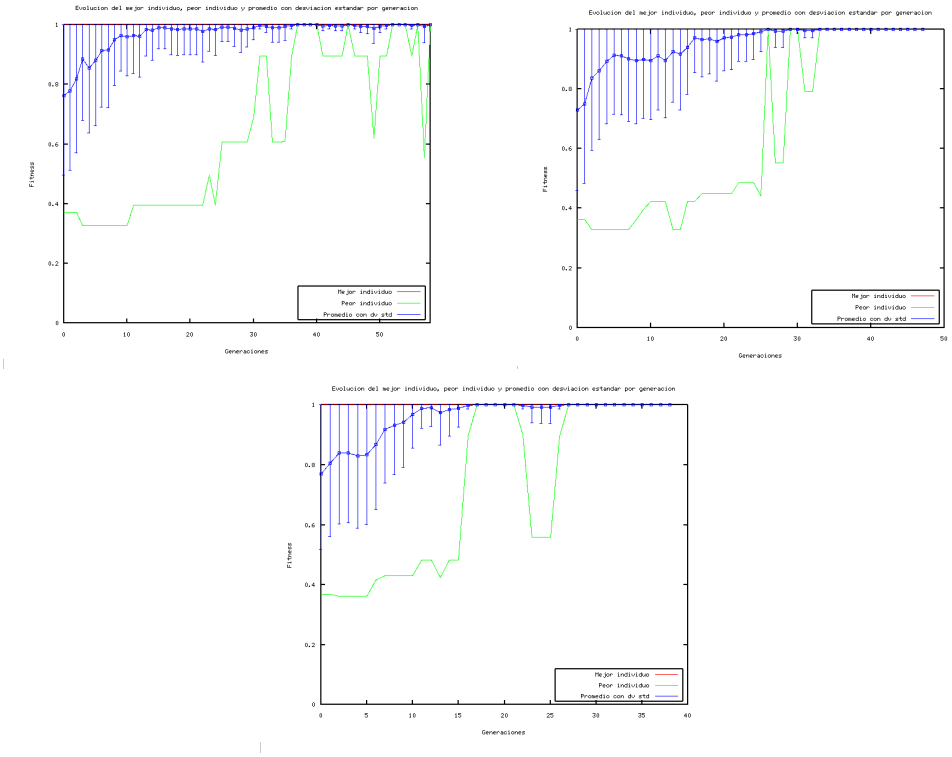


Figura 4.4: Gráficas de mejor, peor y fitness promedio con desviación estándar de cada generación en las 3 corridas del testeo de aprendizaje utilizando la secuencia de demostración del saludo.

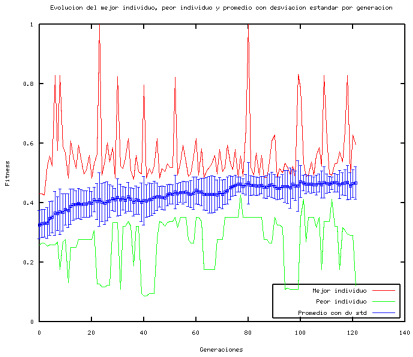


Figura 4.5: Gráficas de mejor, peor y fitness promedio con desviación estándar de cada generación en la corrida del testeo de aprendizaje utilizando la secuencia de demostración del gesto de pateo.

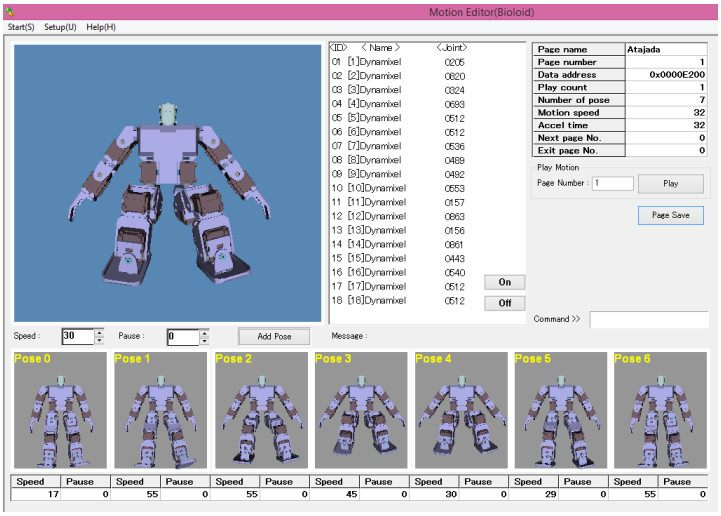


Figura 4.6: Programa Motion Editor de Bioloid

4.4.1. Determinación del parámetro de velocidad

Una secuencia solución del algoritmo genético implementado está compuesta por una secuencia de posturas, las cuales se ejecutan en determinado tiempo. Debido a que el programa Motion Editor no permite indicarle que ejecute una postura en un tiempo determinado, pero si a determinada velocidad, se realizaron medidas de la ejecución de posturas en el robot, con distintas velocidades, para determinar el tiempo de ejecución de una postura (ver cuadro 4.6 y figura 4.7). A partir de esos datos se aproximó una función $f(x)$, utilizando splines cúbicos, que determina para cada velocidad el tiempo de ejecución de la postura. De esa forma calculando la raíz de la expresión $f(x) = t$ (donde $f(x)$ es un polinomio cúbico) se puede determinar la velocidad para que se realice la postura en el tiempo t determinado. En el cuadro 4.7 se puede observar las velocidades calculadas para las secuencias que se ejecutaron.

Velocidad	Tiempo promedio de ejecución de postura (segs)
10	1.65
20	0.8125
30	0.55
40	0.4125
50	0.325
55	0.3

Cuadro 4.6: Velocidades y tiempo de ejecución de posturas en el robot de acuerdo a los datos medidos

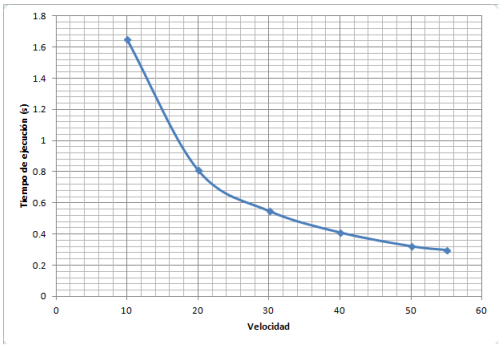


Figura 4.7: Gráfica de velocidades y tiempos de ejecución de posturas

Tiempo de ejecución de postura (segs)	Velocidad calculada
0.30	55
0.36	45
0.54	30
0.56	29

Cuadro 4.7: Velocidades calculadas para los tiempos de ejecución de posturas de las secuencias de atajada ejecutadas en el robot

4.4.2. Resultados y conclusiones

Se ejecutaron las secuencias de dos formas distintas: una teniendo al robot suspendido en el aire para evaluar la forma del movimiento del mismo y la otra con el robot en el piso para evaluar la estabilidad.

En la ejecución de ambas secuencias con el robot suspendido en el aire, se observó que el mismo realizaba los movimientos propios de la atajada. Sin embargo, en ninguna de las pruebas estando sobre el piso el robot ejecutó de forma estable.

El resultado anterior respecto a la estabilidad se puede explicar por los siguientes factores:

- El no determinismo de las secuencias aprendidas por el algoritmo genético en el simulador. Respecto a este punto, quizás se debería haber probado ejecutar más veces la secuencia que evaluó de forma estable en el simulador, para que alguna evalúe de forma estable en el robot real.

- Las diferencias que hay entre el entorno simulado y el real, lo que hace que la secuencia que fue adaptada en el simulador para que evalúe de forma estable, no esté adaptada en la realidad. De todos modos, probablemente dichas secuencias sean una aproximación de una secuencia que evalúa en forma estable en la realidad.

Capítulo 5

Discusión, conclusiones y trabajo a futuro

5.1. Discusión

5.1.1. Proceso de aprendizaje por demostración-imitación

En el marco del proyecto de grado descripto en este documento, se realizó un estudio del estado del arte de las técnicas utilizadas en el ADIR. A su vez se llevó a cabo la aplicación del aprendizaje por imitación para solucionar un problema complejo como es el aprendizaje del control motor de un robot humanoide para realizar cierta tarea a partir de la observación de la misma. En vista de los resultados obtenidos, se pudo observar cómo la aplicación del aprendizaje por imitación en conjunción con otra técnica de aprendizaje como algoritmos genéticos permitió resolver (al menos parcialmente) el problema inicialmente planteado.

El proceso completo de aprendizaje utilizado consiste de:

1. Mapeo desde el maestro y hacia el aprendiz.
2. Adaptación de la tarea al aprendiz
    a) Entrenamiento del algoritmo genético para búsqueda de soluciones.
    b) Verificación del aprendizaje utilizando un conjunto de datos diferente al utilizado para el entrenamiento del algoritmo.
    c) Prueba de la secuencia aprendida en el robot real.

Aunque el proceso utilizado permitió resolver el aprendizaje para una tarea particular, el mismo proceso podría ser utilizado para resolver el aprendizaje motor de distintas tareas, en principio siendo necesario solamente realizar pequeños cambios a las herramientas ya utilizadas. A su vez se podría automatizar todo el proceso anteriormente mencionado de forma de que no sea necesario que una persona experta participe en la aplicación del mismo, sino que cualquier persona con conocimientos de cómo llevar a cabo la tarea pueda transmitirlos al robot.

5.1.2. Herramientas utilizadas

Por otra parte, se observaron varias dificultades en las herramientas seleccionadas que es necesario destacar.

Primeramente, el simulador utilizado en la solución posee la desventaja de que sólo permite realizar simulaciones en tiempo real. Esto hace que las tareas de puesta a punto del algoritmo, calibración y testeo del algoritmo (ver sección 4.1) insuman demasiado tiempo. Consecuente a lo expresado anteriormente, se observaron dificultades sobretodo en la etapa de puesta a punto del algoritmo, ya que con cada modificación que se hace sobre el mismo, la verificación del impacto de la misma lleva demasiado tiempo.

Otro problema que trae acarreado el simulador utilizado es que no posee una API (interfaz de programación de la aplicación) apropiada que permita evaluar los comportamientos. Se realizaron las modificaciones necesarias para disponer de esa funcionalidad en el simulador, tratando de modificar al minímo el funcionamiento del mismo. Se utilizaron funciones ya implementadas en el simulador, como por ejemplo el movimiento del control motor a través de scripts que se le pasan al controlador, que utiliza internamente la función de ir desde una posición del motor a otra en determinado tiempo. Esas funcionalidades dentro del controlador utilizado tienen algunas restricciones como por ejemplo el no poder mover el motor de una posición a otra en menos de determinado tiempo, lo que llevó a realizar una codificación de las soluciones del genético bastante más compleja de lo que en principio debería ser. Sería interesante, probar el mismo proceso realizado en este proyecto, utilizando algún simulador más apropiado, que no posea las carencias anteriormente mencionadas.

La plataforma robótica utilizada tiene algunas desventajas. Una es que la cantidad de articulaciones implementada por el robot no permite emular correctamente todas las articulaciones humanas. En la imitación de la atajada se vió esta dificultad ya que al realizar el gesto de atajada, el humano puede mover su brazo desde arriba hacia abajo con la palma de la mano apuntando hacia el frente debido a que la articulación de cada uno de sus hombros posee 3 grados de libertad, pero en el robot ese movimiento no es posible ya que la articulación de cada uno de sus hombros tiene solamente 2 grados de libertad. La solución en este caso fue mapear esta articulación de forma que la palma de la mano mirase hacia el piso. Otra dificultad que influyó en la realización del gesto de atajada de forma estable es la forma de los pies del robot. Al realizar la tarea la forma de los pies del robot no permitía que el mismo se pusiera en cuclillas teniendo cierta estabilidad. En el humano, las articulaciones del pie juegan un papel importante en el equilibrio al realizar el gesto de atajada.

Hoy en día existen robots humanoides que tienen una forma más parecida a los humanos (grados de libertad de las articulaciones, relación entre las partes del cuerpo, etc., ver por ejemplo el robot Nao de Aldebaran [wdNA14]), lo que les permitiría realizar la imitación de la tarea de una mejor forma.

5.2. Conclusiones

El ADIR posee varias características deseables de un proceso de aprendizaje que invitan a aplicarlo. Una de ellas es la no necesidad de que un experto programe el comportamiento de una máquina, sino que cualquier persona con conocimientos de cómo realizar ese comportamiento pueda enseñarle solo a una máquina. Gracias a esta característica es de esperar que se utilicen cada vez más procesos de aprendizaje por imitación en robots de uso doméstico, empresarial, industrial, etc.

En el campo del aprendizaje de control motor, se pudo observar que es posible aplicar el aprendizaje por imitación para lograr que un robot aprenda a realizar cierta secuencia motora ejecutando una tarea.

5.3. Trabajo a futuro

Respecto al trabajo a futuro que se puede hacer se destaca el aprendizaje de otras tareas, la aplicación de otro simulador en el proceso de aprendizaje, la utilización de una forma de obtención de datos distinta a la utilizada en el proyecto y la automatización de todo el proceso de aprendizaje.

5.3.1. Aprendizaje de otras tareas

Según lo descripto en la sección 5.1, se puede aplicar el mismo proceso utilizado con la tarea de atajada (y testeado con el saludo y gesto de pateo) para realizar el aprendizaje motor de otras tareas, como por ejemplo levantarse, caminar, etc.

5.3.2. Aplicación de otro simulador/plataforma robótica

Uno de los puntos negativos respecto al simulador, discutidos en la sección 5.1, es que sólo permite la simulación en tiempo real, y esto hace que se insuma mucho tiempo en las corridas del algoritmo genético. Otro punto negativo es que el simulador no tiene una API que permita saber las posiciones del robot en cada momento, y mover los motores del robot de una posición a otra en tiempos cortos (la limitante observada del sistema es de aprox. 200 ms entre una posición y otra del robot). Debido a lo resaltado anteriormente, sería interesante realizar todo el proceso de aprendizaje pero utilizando un simulador del robot que tenga la posibilidad de realizar la simulación en tiempos menores al real, y a su vez que tenga la API apropiada para ejecutar los movimientos en el robot y poder evaluar las secuencias de posturas del mismo. La posibilidad de simular en tiempo menor al real, permite reducir en gran manera el tiempo de puesta a punto del algoritmo, logrando una mejor calidad en el desarrollo y testeo de los operadores genéticos, y por consecuencia en la solución desarrollada. También permite reducir el tiempo tanto de calibración como de testeo de la solución. Por otro lado, una API con las características mencionadas ayudaría al momento de programar la solución, sin necesidad de modificar la herramienta.

También se podría utilizar una plataforma robótica diferente, lo que conllevaría también a la utilización de un simulador adecuado para dicha plataforma.

5.3.3. Utilización de medio distinto para la obtención de datos

Como trabajo a futuro sería interesante trabajar con otras herramientas distintas a la utilizada en este proyecto para la obtención de los datos demostrados. Una de las herramientas puede ser Kinect (ver [wdK14]). Kinect es un controlador de video juegos desarrollado por Microsoft para la consola XBox 360, aunque también se han desarrollado librerías de uso libre para los sistemas operativos más populares y que permite su utilización en PC´s con Windows, Linux o Mac (ver por ejemplo [wdOK14]). El uso de dicha herramienta permitiría captar demostraciones de distintas tareas de forma más fácil, y de esa forma utilizar esos datos en nuevos procesos de aprendizaje. Como el sistema de obtención de datos se puede integrar a una PC, permite automatizar la obtención de datos en el proceso de aprendizaje.

5.3.4. Automatización de todo el proceso de aprendizaje

En la sección 5.1 se describió el proceso de aprendizaje utilizado en el proyecto. Una perspectiva a futuro podría ser realizar la automatización de dicho proceso. Esto implica automatizar cada uno de los

puntos del proceso de aprendizaje para que al aplicarlo no haya la necesidad de realizar modificaciones a las herramientas ni análisis por parte de algún experto.

Bibliografía

[ACR04] J. Aleotti, S. Caselli, and M. Reggiani. *Leveraging on a virtual environment for robot programming by demonstration, Robotics and Autonomous Systems, vol. 47, no. 2-3, pp. 153.161.* 2004.

[ACVB09] B.D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. *A survey of robot learning from demostration, Robotics and Autonomous Systems 57(5): 469-483.* 2009.

[Amo10] Heni Ben Amor. *Imitation Learning of Motor Skills for Synthetic Humanoids. PhD thesis, Freiberg University of Mining and Technology.* 2010.

[BABVJ09] Heni Ben Amor, Erik Berger, David Vogt, and Bernhard Jung. *Kinesthetic Bootstrapping: Teaching Motor Skills to Humanoid Robots through Physical Interaction, KI-09 Proceedings of the 32nd annual German conference on Advances in artificial intelligence, Pages 492-499.* 2009.

[Ben12] Facundo Benavides. *Planificación de movimientos aplicada en robótica autónoma móvil, Tesis de Maestría, Facultad de Ingeniería, UdelaR.* 2012.

[Ber11] Erik Berger. *Master Thesis: Visual Bootstrapping A Technique for Motion Capture Based Imitation Learning, Freiberg University of Mining and Technology.* 2011.

[BG12] Aude Billard and Daniel Grollman. *Robot Learning By Demonstration, Scholarpedia.* 2012.

[duB] Guía de uso Bioloid. *Bioloid User's Guide, ver. 1.00.*

[FD95] H Friedrich and R. Dillman. *Robot programming based on a single demmonstration and user intentions, In: 3rd European Workshop on Learning Robots at ECML-95.* 1995.

[KII94] Y. Kuniyoshi, M. Inaba, and H. Inoue. *Learning by watching: Extracting reusable task knowledge from visual observation of human performance, Robotics and Automation, IEEE Transactions on, pp. 799-822.* 1994.

[Mor11] Isolina Alberto Moralejo. *Herramientas para algoritmos evolutivos multiobjetivo, Tesis Doctoral, Facultad de Ciencias, Universidad Zaragoza.* 2011.

[NM03] M. N. Nicolescu and M. J. Mataric. *Methods for robot task learning: Demonstrations, generalization and practice, In Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems.* 2003.

[rB] Guía rápida Bioloid. *Bioloid Quick Start, ver. 1.0.*

[RFFG96] G Rizzolatti, L Fadiga, L Fogassi, and V. Gallese. *Premotor cortex and the recognition of motor actions, Brain Res Cogn Brain Res. 1996 Mar;3(2):131-41.* 1996.

[RM03] Rajesh P. N. Rao and Andrew N. Meltzoff. *Imitation Learning in Infants and Robots: Towards probabilistic Computational Models.* 2003.

[RYSV07] P. E. Rybski, K. Yoon, J. Stolarz, and M. Veloso. *Interactive robot task training through dialog and demonstration, HRI-07, Proceedings of the ACM/IEEE international conference on Human-robot interaction, Pages 49-56.* 2007.

[San12] Marcos Sander. *Documento del Estado del Arte - PGILearn, Facultad de Ingeniería, UdelaR.* 2012.

[San14] Marcos Sander. *Documentación de herramientas - PGILearn, Facultad de Ingeniería, UdelaR.* 2014.

[wdB14] Sitio web de Bioloid. http://www.robotis.com/xe/bioloid_main_en, último acceso: Nov 2014.

[wdBC14] Sitio web de Bioloid Control. http://bioloidcontrol.sourceforge.net/, último acceso: Nov 2014.

[wdJ14] Sitio web de JMetalCPP. *http://jmetalcpp.sourceforge.net/.* último acceso: Nov 2014.

[wdK14] Sitio web de Kinect. http://www.xbox.com/en-us/kinect, último acceso: Nov 2014.

[wdNA14] Sitio web de Nao Aldebaran. http://www.aldebaran.com/en/humanoid-robot/nao-robot, último acceso: Nov 2014.

[wdOK14] Sitio web de Open Kinect. http://openkinect.org/wiki/main_page, último acceso: Nov 2014.

[ZLC11] Abel Javier Zamora, Martha E. López, and Rosalva Cabrera. *Imitación en grupos animales, evaluación de una respuesta novedosa para obtener alimento en las palomas, Universidad Nacional Autónoma de México, México.* 2011.

Documentación de herramientas utilizadas
Proyecto PGILearn

Marcos Sander

Tutor
Facundo Benavides

Instituto de Computación
Facultad de Ingeniería
Universidad de la República
Montevideo-Uruguay

2014

Resumen

El siguiente documento especifíca las herramientas utilizadas en el desarrollo del proyecto de grado PGILearn. Para cada herramienta se explica su instalación, configuración de parámetros y ejecución de la misma, así como detalles de su funcionamiento.

Índice general

1. Introducción **1**
1.1. Organización del documento . . . 1

2. Software de obtención de datos **3**
2.1. Introducción . . . 3
2.2. Instalación de DVideow . . . 4
2.3. Digitalización del movimiento . . . 4
2.3.1. Sincronización de cámaras . . . 4
2.3.2. Digitalización de puntos . . . 5
2.4. Calibración del sistema . . . 7
2.5. Reconstrucción de la secuencia . . . 10

3. Análisis de datos con Octave **11**
3.1. Introducción . . . 11
3.2. Instalación . . . 11
3.3. Scripts generados . . . 11
3.3.1. Script atajada_pca_splines . . . 12
3.3.2. Script grafica_estabilidad . . . 12
3.3.3. Script grafica_datos_genetico . . . 12

4. Simulador físico incluído con Bioloid Control **15**
4.1. Introducción . . . 15
4.2. Instalación . . . 15
4.2.1. Instalación/compilación del Firmware . . . 16
4.2.2. Instalación del viewer . . . 16
4.2.3. Instalación del main (controlador) . . . 16
4.2.3.1. Instalaciones y modificaciones previas . . . 16
4.2.3.2. Corrección de errores de código . . . 16
4.2.3.3. Instalación . . . 17
4.2.4. Instalación del physics (simulador) . . . 17
4.2.4.1. Instalaciones y modificaciones previas . . . 17
4.2.4.2. Instalación del simulador . . . 18
4.3. Modificaciones realizadas . . . 18
4.3.1. Corrección de errores y adaptaciones . . . 18
4.3.1.1. Agregado de lock en physics . . . 19

4.3.1.2. Modificación de referencias en physics . . . 19
4.3.1.3. Error por límite de file descriptors en main y physics . . . 19
4.3.1.4. Leaks de memoria en physics . . . 20
4.3.2. Modificaciones realizadas al controlador y simulador . . . 20
4.3.2.1. Comunicación del algoritmo genético y el simulador . . . 21
4.3.2.2. Cambios en el main (controlador) . . . 23
4.3.2.3. Cambios en el physics (simulador) . . . 24
4.4. Ejecución del sistema . . . 25
4.4.1. Archivos de configuración y entrada . . . 25
4.4.1.1. Main . . . 25
4.4.1.2. Physics . . . 25
4.4.2. Comandos para ejecución del sistema . . . 25
4.4.2.1. Ejecución del viewer . . . 25
4.4.2.2. Ejecución del physics (simulador) . . . 25
4.4.2.3. Ejecución del main (controlador) . . . 26
4.4.2.4. Ejecución de más de una instancia del sistema . . . 26
4.4.3. Archivos de salida generados . . . 26

5. Algoritmo genético - JMetal 27
5.1. Introducción . . . 27
5.2. Modificaciones a JMetal y clases auxiliares . . . 28
5.3. Instalación . . . 30
5.4. Ejecución del sistema aprendizaje . . . 30
5.4.1. Configuración del entorno . . . 31
5.4.1.1. Puertos de comunicación . . . 31
5.4.1.2. Lectura del archivo de evaluación . . . 31
5.4.1.3. Configuración de parámetros por corrida . . . 31
5.4.1.4. Configuración de corridas del genético . . . 32
5.4.2. Archivos de entrada . . . 32
5.4.3. Ejecución . . . 32
5.4.3.1. Simple . . . 32
5.4.3.2. Con más de un entorno . . . 33
5.4.4. Archivos de salida de la ejecución . . . 33
5.4.5. Conjunto óptimo de parámetros . . . 35

6. Ejecutables auxiliares 37
6.1. Introducción . . . 37
6.2. Aplicaciones . . . 37
6.2.1. stability_Matrix_Generator . . . 37
6.2.2. initial_Population_Fitness . . . 37
6.2.3. execute_Best_Behavior . . . 38

Índice de algoritmos

4.1. Ejemplo de script que permite el movimiento de motores de Bioloid Control 22

Índice de figuras

2.1. Sincronización con DVideow 4
2.2. Marcadores utilizados en la digitalización del movimiento 6
2.3. Configuraciones y seguimiento en la digitalización de la secuencia 7
2.4. Soporte con marcadores guía utilizados en el proyecto 8
2.5. Configuración de las referencias e ilustración del marcado de puntos para calibración . . 9
2.6. Configuración de la reconstrucción de la secuencia 10

3.1. Comparación de la aplicación de PCA y splines cúbicos a los datos demostrados 13
3.2. Aplicación de splines a datos de dimensión menor 13
3.3. Errores promedio de reproyección al aplicar PCA 14
3.4. Mapa de estabilidad y trayectoria sobre dimensión menor 14
3.5. Estadísticas del algoritmo genético 14

4.1. Máquina de estados que implementa el simulador (physics) 21
4.2. Máquina de estados que implementa el controlador (main) 21
4.3. Diagrama de despliegue del sistema de aprendizaje 22

5.1. Estructura de clases de JMetal 27
5.2. Representación de soluciones en JMetal 28
5.3. Estructura de nuevas clases agregadas y modificadas en JMetal 28
5.4. Representación de soluciones implementadas en JMetal 29
5.5. Diagrama de clases del paquete behavior 30

Índice de cuadros

2.1. Ejemplo de sincronización de cámaras 5

4.1. Puertos de comunicación del main (controlador) 24
4.2. Puertos de comunicación del physics (simulador) 24
4.3. Ejemplo de configuración de puertos para dos instancias del sistema 26

5.1. Ejemplo de configuración de puertos para dos instancias del sistema de aprendizaje . . . 33
5.2. Configuración de ruta de archivo de evaluación 33
5.3. Configuración de ejecuciones en instancias del sistema de aprendizaje 34

Capítulo 1

Introducción

El siguiente documento pretende brindar una descripción de las herramientas utilizadas en el marco del proyecto de grado [San14]. En los siguientes capítulos se describen las herramientas utilizadas, instalación y uso de las mismas. A su vez se describen los archivos generados para su posterior utilización.

1.1. Organización del documento

Este documento se organiza de la siguiente forma: en el capítulo 1 se hace una introducción del mismo. En el capítulo 2 se describe el software utilizado para la obtención de datos. En el capítulo 3 se describe el análisis de datos realizados utilizando Octave. En el capítulo 4 se describe el simulador físico utilizado, instalación, configuración y cambios realizados para utilizarlo en el proyecto. En el capítulo 5 se describe el framework genético utilizado (JMetal), instalación y configuración. Por último el capítulo 6 describe otras aplicaciones utilizadas en el desarrollo del proyecto.

Capítulo 2

Software de obtención de datos

2.1. Introducción

Para observar la ejecución de las tareas demostradas, esto es, obtener los datos de esas demostraciones para utilizarlos en el sistema de aprendizaje, las mismas se grabaron y luego se procesaron utilizando un software. La demostración fue grabada con 3 cámaras en distintas posiciones y sobre la persona se colocaron marcadores de forma de identificar puntos clave para obtener los ángulos de sus articulaciones al realizar la tarea. Luego utilizando un software adecuado se extrajo de dichas grabaciones la información de la posición de esos marcadores en el espacio (en determinado sistema de coordenadas).

En la subsección 3.2.1.1 del documento final del proyecto de grado [San14], se describen otros detalles de los elementos utilizados para realizar la grabación de las secuencias.

Para extraer información de los marcadores que se colocaron sobre la persona y de esa forma obtener los datos de la ejecución de la tarea se utilizó el sistema DVideow, brindado por el laboratorio de Biomecánica de la Universidad de la República.

En los archivos entregados del proyecto se encuentra la versión de dicho software con el programa instalador del mismo, utilizado para realizar el análisis correspondiente en el proyecto. El mismo funciona sobre el sistema operativo Windows XP.

Luego de haber grabado en las 3 cámaras los marcadores guía para generar el sistema de coordenadas de referencia así como las secuencias de la persona con los marcadores colocados sobre sus articulaciones se debe primeramente digitalizar el movimiento de las secuencias, luego realizar la calibración dado que el sistema la utiliza como referencia de distancias para generar la secuencia y por último con datos de la calibración y de la secuencia digitalizada realizar la reconstrucción de la secuencia obteniéndose en un archivo de texto plano .3D las posiciones x,y,z de cada marcador que está sobre la persona a lo largo de toda la ejecución de la tarea. En las siguientes secciones se explicará cómo se realizó la digitalización del movimiento (ver sección 2.3), la calibración (ver sección 2.4), y por último la reconstrucción de la secuencia de forma de poder obtener información de esa ejecución (ver sección 2.5).

2.2. Instalación de DVideow

La instalación de DVideow es sencilla, solamente ejecutar el archivo de instalación y seguir los pasos indicados por el programa. Como se mencionó anteriormente, el programa ejecuta sobre Windows XP. En ese sentido, se probó instalarlo sobre Windows 7 sin éxito.

2.3. Digitalización del movimiento

2.3.1. Sincronización de cámaras

Para realizar la sincronización entre las 3 cámaras al grabar las secuencias inicialmente se debe emitir un sonido fuerte (en nuestro caso se utilizó un pitido), para que luego se utilice como referencia la primer imagen de cada cámara donde se identifica ese sonido (ver figura 2.1 donde se visualiza en DVideow la identificación del sonido). Como ejemplo supongamos que se quiere digitalizar una secuencia (dentro de una grabación con 1000 imagenes) y en la cámara 1 se identificó que la misma ocurre desde la imagen 100 hasta la 200. Para este ejemplo, en el cuadro 2.1 se muestra el número de imagen de inicio y fin de la secuencia en las demás cámaras calculado utilizando el número de la imagen en cada cámara donde se identificó que ocurre la señal de sincronización y el número de imagen de inicio y fin de la secuencia para la cámara 1.

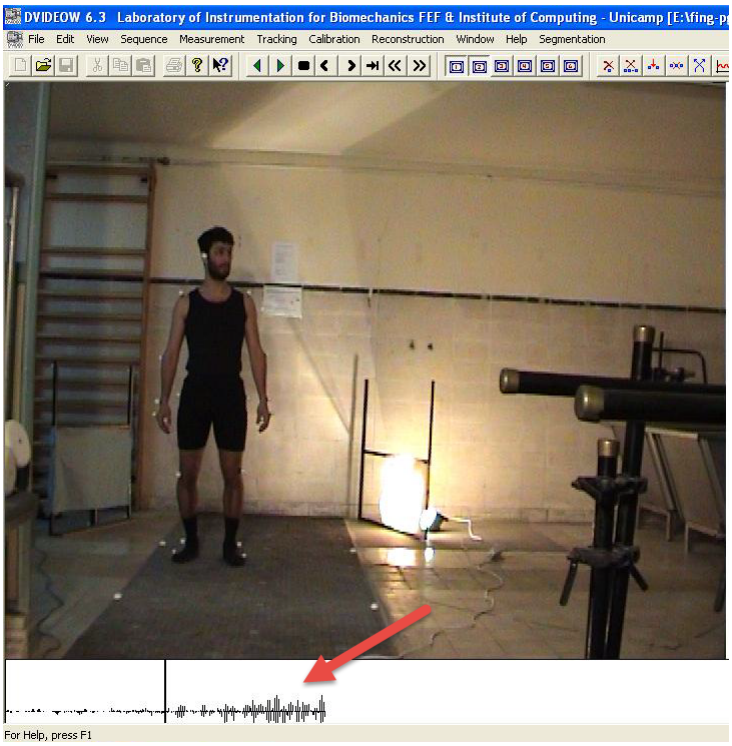


Figura 2.1: Sincronización con DVideow

	Imagen sonido	Imagen inicio secuencia	Imagen fin secuencia
Cámara 1	20	100	200
Cámara 2	80	100 - 20 + 80 = 160	200 - 20 + 80 = 260
Cámara 3	60	100 - 20 + 60 = 140	200 - 20 + 60 = 240

Cuadro 2.1: Ejemplo de sincronización de cámaras

2.3.2. Digitalización de puntos

Primeramente antes de realizar la digitalización de los puntos se debe determinar los marcadores que se digitalizarán y asignarle un número correspondiente. Para un marcador dado que se digitalizará luego de asignarle un número, en cada cámara se seleccionará a ese marcador con el mismo número. En el marco del proyecto se utilizaron 20 marcadores sobre la persona con los cuales se definieron 10 ángulos para la tarea demostrada de atajada. En la figura 2.2 se pueden visualizar los marcadores utilizados.

Para realizar la digitalización de los puntos o marcadores se deben seguir los siguientes pasos:

1. En DVideow seleccionar la opción New Measurement y en la ventana que se despliega seleccionar la cantidad de marcadores que se digitalizará, la frecuencia de procesamiento -que refiere a la frecuencia con que se grabaron las imagenes (en el caso de ejemplo 60 ips)- y por último seleccionar el valor 0 para el número de frames de manera que tome la cantidad de forma automática (ver figura 2.3a).

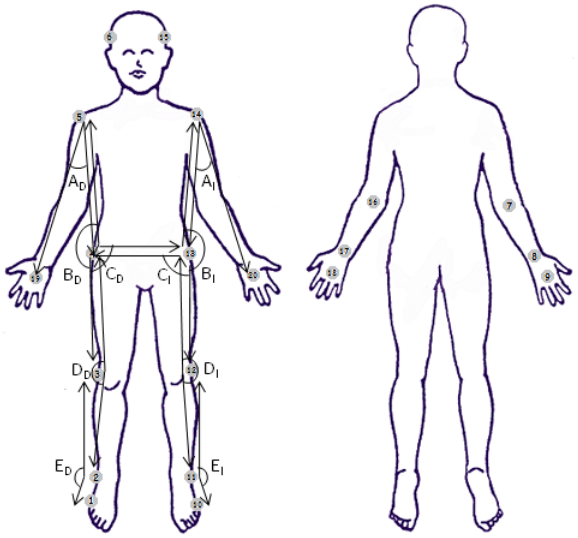
2. Seleccionar el número de cámara y luego ir al menú Sequence, opción Load y cargar el video de la cámara correspondiente.

3. Ir al menú Measurement, opción Enable Marking para activar la herramienta de marcado.

4. Ir al menú Tracking, opción Setup y seleccionar en Markers los puntos a digitalizar, por ejemplo 1 to 20 si son todos o sino seleccionar Select y marcar cuáles se digitalizarán. En Frames seleccionar la imagen de incio y fin de la secuencia en la cámara correspondiente. En Segmentation agregar las herramientas que se desean utilice el software para reconocimiento automático de los marcadores, en el ejemplo se utilizaron Inverse, Erosion y GetMarkers (ver figura 2.3b).

5. Ir a la primer imagen de la secuencia a digitalizar y seleccionar todos los puntos/marcadores correspondientes. Si en la cámara que se está digitalizando los puntos un marcador no se viere durante toda la secuencia dejarlo sin seleccionar.

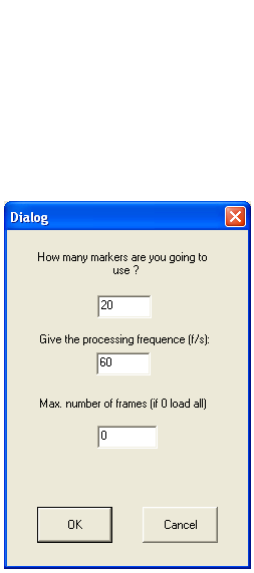
6. Luego de tener marcados todos los puntos para la primer imagen de la secuencia seleccionar menú Tracking, opción TrackMarkers para hacer el marcado automático en el restante de las imagenes de la secuencia. Si el programa falla en el seguimiento de algún marcador, ir hasta la primer imagen donde falló ese seguimiento, corregir la posición de ese marcador manualmente (y de los demás marcadores que hubieren fallado también) y luego volver a realizar el marcado automático con la opción TrackMarkers (ver figura 2.3c).



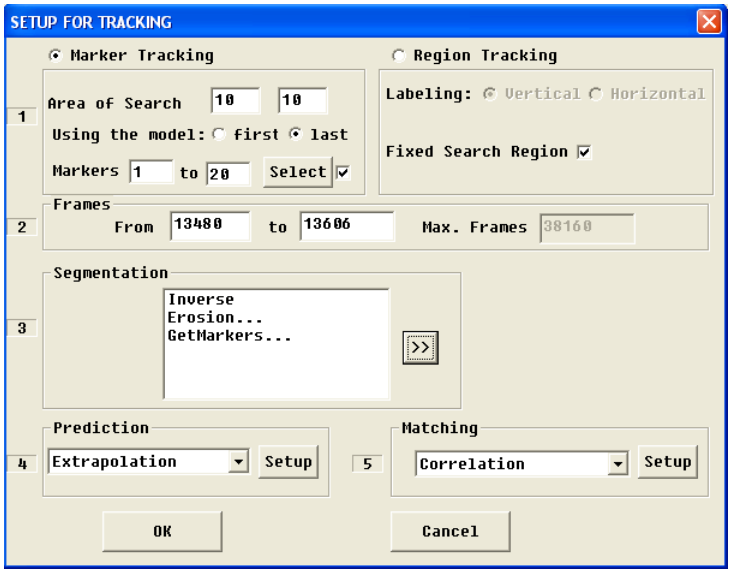
Número Marcador	Parte del cuerpo
1	5to. metatarso derecho
2	tobillo derecho
3	rodilla derecha
4	cadera derecha
5	hombro derecho
6	cabeza derecha
7	codo derecho
8	muñeca derecha
9	mano derecha
10	5to. metatarso izquierdo
11	tobillo izquierdo
12	rodilla izquierda
13	cadera izquierda
14	hombro izquierdo
15	cabeza izquierda
16	codo izquierdo
17	muñeca izquierda
18	mano izquierda
19	palma derecha
20	palma izquierda

Figura 2.2: Marcadores utilizados en la digitalización del movimiento

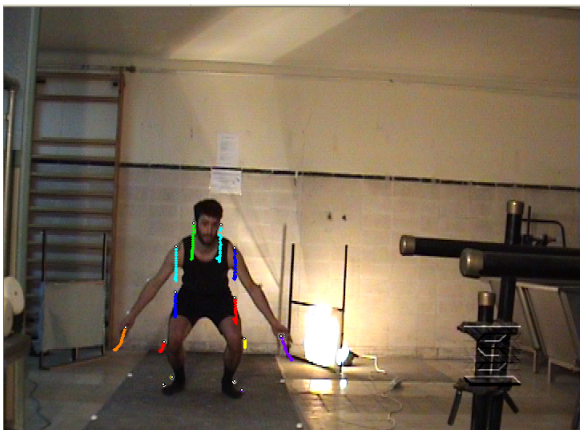
7. Grabar el archivo de digitalización yendo al menú Measurement, opción Save. En el proyecto los archivos digitalizados se nombraron Acción_SecuenciaX_CamaraY.DAT, como por ejemplo Atajada_Secuencia1_Camara1.DAT.



(a) Configuración de marcadores y frecuencia



(b) Configuración del seguimiento de marcadores



(c) Seguimiento de marcadores en la ejecución de secuencia

Figura 2.3: Configuraciones y seguimiento en la digitalización de la secuencia

2.4. Calibración del sistema

Primeramente antes de calibrar el sistema se deben determinar los marcadores guía de calibración del mismo. Como parte de la calibración en cada cámara se deberá seleccionar cada uno de esos marcadores. En total se utilizaron 36 marcadores guía, de los cuáles se puede ver un ejemplo en la figura 2.4.

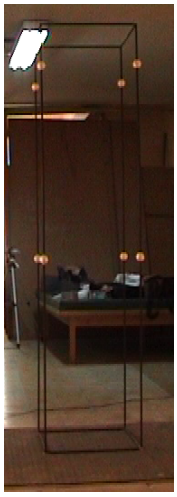


Figura 2.4: Soporte con marcadores guía utilizados en el proyecto

Para calibrar el sistema se siguen los siguientes pasos:

1. En DVideow seleccionar la opción New Measurement y en la ventana que se despliega seleccionar la cantidad de marcadores guía utilizados para calibrar, la frecuencia de procesamiento (en el caso de ejemplo 60 fps) y por último seleccionar el valor 0 como número de frames de manera que lo tome de forma automática.

2. Seleccionar el número de cámara y para la misma cargar el video seleccionando menú Sequence, opción Load y luego elegir el archivo.

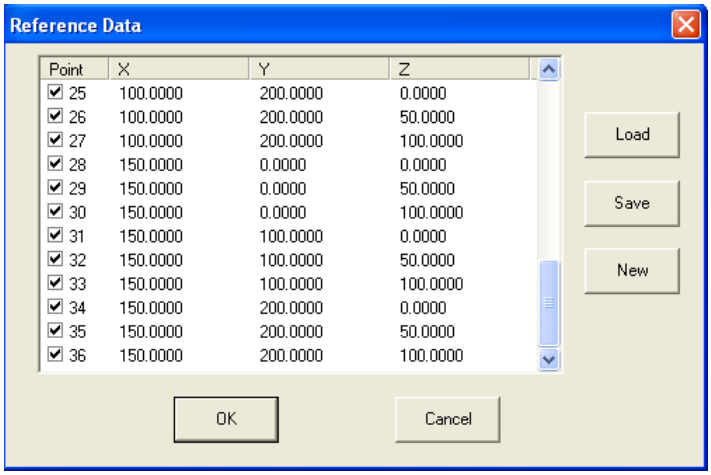
3. Cargar la digitalización del movimiento realizada en la cámara correspondiente contenida en un archivo .DAT. Cargarla yendo al menú Measurement, opción Load y eligiendo el archivo .DAT correspondiente.

4. Ir al menú Calibration, submenú Compute, opción References, estando en la imagen con todos los puntos de calibración marcados. En esa ventana seleccionar la opción Load y cargar el archivo de referencias .REF correspondiente. Ese archivo de referencias contiene para cada marcador de la calibración su posición en el espacio x,y,z representada en metros. Luego de haber cargado las referencias seleccionar cada punto de calibración correspondiente (ver figura 2.5a).

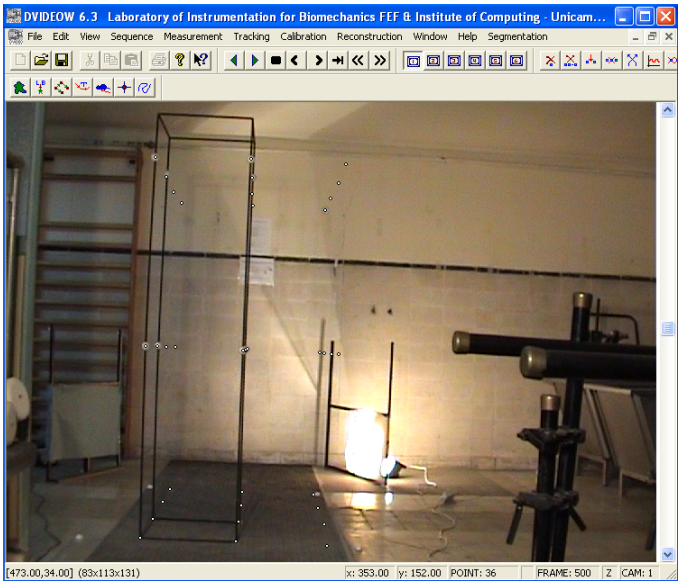
5. Para marcar los puntos de calibración, primeramente activar la herramienta de marcado en menú Measurement opción Enable Marking. Luego ir a alguna imagen en particular y marcar con la herramienta correspondiente cada uno de los puntos o marcadores de calibración (del 1 al 36). Durante la ejecución del proyecto sucedió que no se veían todos los puntos de calibración a la vez en ninguna imagen (porque la guía de calibración utlizada solamente tiene 12 marcadores, y se la coloca en distintas posiciones para simular una guía de 36 marcadores), por lo que se anotaron las posiciones x,y de cada punto de calibración previamente y luego estando en una imagen en particular se marcaron todos los puntos o marcadores de calibración (ver figura 2.5b).

6. Ir al menú Calibration, submenú Compute, opción DLT.

7. Guardar la calibración en menú Calibracion, opción Save y eligiendo el archivo .CAL correspondiente. En el proyecto los archivos digitalizados se nombraron Calibracion_CamaraY.CAL, como por ejemplo Calibracion_Camara1.CAL.



(a) Configuración de referencias de marcadores guía para la calibración



(b) Marcadores seleccionados para la calibración de una cámara

Figura 2.5: Configuración de las referencias e ilustración del marcado de puntos para calibración

2.5. Reconstrucción de la secuencia

Luego de tener realizada la digitalización y la calibración de cada cámara para poder reconstruir la secuencia realizar los siguientes pasos:

1. Cargar en cada cámara los videos correspondientes con menú Sequence opción Load y seleccionando el archivo correspondiente. Luego cargar la secuencia digitalizada con el menú Measurement opción Load y cargando el archivo .DAT correspondiente. Por último cargar la calibración correspondiente con menú Calibration, opción Load y seleccionando el archivo .CAL correspondiente de esa cámara.

2. En menú Reconstruction marcar la opción Ignore Unmarked si en alguna cámara hubieron marcadores sin marcar porque no se veían en esa cámara (es lo más probable).

3. En menú Reconstruction seleccionar la opción 3D. Dentro de la ventana seleccionar las imagenes de inicio y fin de la secuencia en cada cámara y tildar las cámaras utilizadas, y por último seleccionar los puntos a reconstruir (en el ejemplo del 1 al 20), presionar Ok y guardar el archivo .3D (ver figura 2.6). Dicho archivo contiene la secuencia de la tarea ejecutada, esto es, los puntos x, y ,z de cada marcador en el sistema de coordenadas cartesiano brindado (ver sección 2.4) en cada frame durante toda la secuencia. En el proyecto los archivos reconstruidos se nombraron Acción_SecuenciaX.3D, como por ejemplo Atajada_Secuencia1.3D.

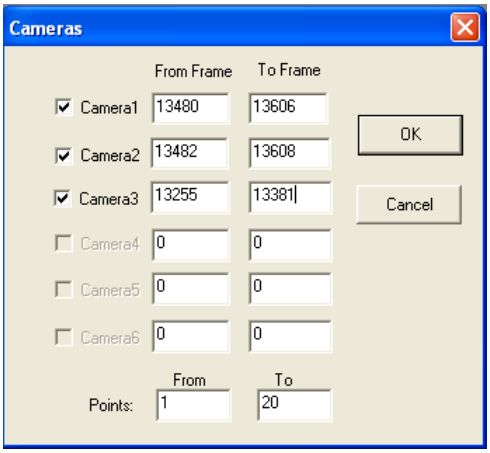


Figura 2.6: Configuración de la reconstrucción de la secuencia

Capítulo 3

Análisis de datos con Octave

3.1. Introducción

En el proyecto de grado se utilizó la metaheurística de los algoritmos genéticos para realizar la tarea de aprendizaje de una secuencia de posiciones de los motores de un robot para realizar cierta tarea a partir de la demostración de la misma por una persona. Previamente a implementar la solución se realizó el análisis de los datos registrados de la demostración de forma de poder aplicar la técnica de reducción de dimensiones (PCA) y representación de datos (splines) que mejoran la eficiencia en el aprendizaje y a su vez poder determinar a partir de ese análisis las dimensiones a reducir de manera de mantener un gran porcentaje de la información obtenida de la demostración y a su vez lograr la eficiencia en el aprendizaje antes mencionada. Otra técnica también utilizada en el aprendizaje de la tarea fue la aplicación de mapas de estabilidad, que brindan al algoritmo una idea de zonas en la dimensión menor donde el robot posee posturas que son estables.

Octave fue la herramienta utilizada para realizar el análisis de los datos demostrados en una primera instancia y a su vez el análisis de resultados obtenidos luego de la ejecución del algoritmo. GNU Octave es una lenguage interpretado de alto nivel, inicialmente pensado para computaciones numéricas que provee herramientas gráficas para visualización y manipulación de datos. El lenguage Octave es similar a Matlab y permite portabilidad de la mayoría de los programas. Es distribuido bajo los términos de Licencia General Pública GNU (ver [wdGO14]). La versión utilizada fue la 3.6.4.

3.2. Instalación

Para la instalación de Octave tanto en el sistema operativo Windows 7 y 8 como en Linux Ubuntu se siguieron las instrucciones que se detallan en la página oficial de la aplicación (ver [dO14] para información de instalación en distintos sistemas y [dGOeW14] para información de instalación específica sobre sistemas Windows).

3.3. Scripts generados

La tarea aprendida en el proyecto de grado fue la de un gesto de atajada. Para realizar el análisis de las distintas técnicas aplicadas a los datos y del resultado de la ejecución del algoritmo gené-

tico se crearon y utilizaron los siguientes scripts: *atajada_pca_splines, grafica_estabilidad* y *grafica_datos_genetico*.

3.3.1. Script atajada_pca_splines

Para realizar el análisis de los datos demostrados se generó el script *atajada_pca_splines* en Octave en el cual primeramente se realiza el cálculo de ángulos de la persona obteniéndose una secuencia de valores de ángulos de cada articulación y luego se implementan técnicas de reducción e incremento de dimensiones utilizando PCA y representación compacta de trayectoria utilizando splines. Se grafican los ángulos obtenidos luego de aplicar la reducción de dimensiones reduciendo a distintos valores, proyectando a la dimensión original y comparando con los datos originales. El script utiliza también los scripts *levideow* y *matfiltfilt_oct* que implementan la obtención de datos registrados y suavizan la curva de datos al graficarlos respectivamente. Por otro lado también es necesario que Octave tenga instalado el paquete Signal para poder ejecutar este script.

Los datos obtenidos de la demostración de la tarea se encuentran y leen del archivo de texto Atajada3.txt previamente generado por el software descripto en el capítulo 2.

En la gráfica 3.1 y 3.2 se puede ver la comparación de los resultados al aplicar solo PCA y PCA junto con splines tanto en la dimensión mayor como en la menor. A su vez en la gráfica 3.3 se puede observar los distintos errores obtenidos al aplicar PCA reduciendo a distintas dimensiones.

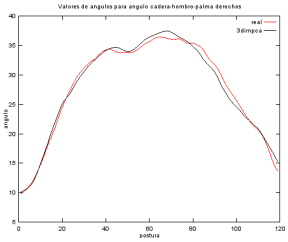
En resumen el script *atajada_pca_splines* aplica el algoritmo PCA a los datos obtenidos con el software anteriormente descripto, aproxima esos datos con una trayectoria utilizando splines y por último grafíca los datos resultantes.

3.3.2. Script grafica_estabilidad

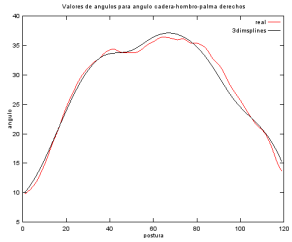
A su vez en el proyecto se aplicó la técnica de mapas de estabilidad. El script *grafica_estabilidad* permite graficar los puntos de estabilidad en la dimensión menor obtenidos luego de crear el mapa de estabilidad. A su vez se puede visualizar la trayectoria en la dimensión menor de una secuencia sin estabilizar (obtenida de los datos originales) y una secuencia estabilizada (utilizando el mapa de estabilidad). Los datos de la matriz de estabilidad se leen del archivo graficaEstabilidad.txt generado por el algoritmo de creación de matriz de estabilidad y a su vez las trayectorias se leen de los archivos de texto plano VAR y VAR2 obtenidos de ejecuciones del genético que describe cada uno una trayectoria sobre la dimensión menor. En la figura 3.4 se puede ver un ejemplo de la gráfica del mapa de estabilidad y de una trayectoria en la dimensión menor.

3.3.3. Script grafica_datos_genetico

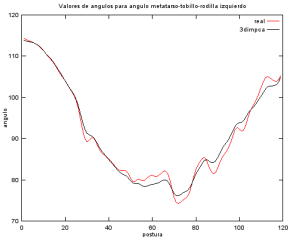
El script *grafica_datos_genetico* toma los datos de la salida de una corrida del genético (contenidos en el archivo estadisticasGenetico.txt) y los grafica de modo de poder visualizar por generación el individuo de fitness mayor, menor, promedio y desviación estándar (ver figura 3.5).



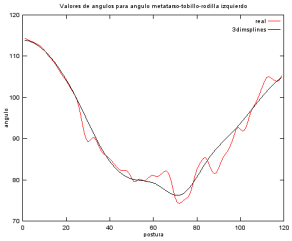
(a) Ángulos originales y reproyectados de la articulación cadera-hombro-palma derechos a lo largo de la secuencia de posturas luego de aplicarse PCA para reducir a 3 dimensiones



(b) Ángulos originales y reproyectados de la articulación cadera-hombro-palma derechos a lo largo de la secuencia de posturas luego de aplicarse PCA para reducir a 3 dimensiones y splines cúbicos con 11 puntos de control



(c) Ángulos originales y reproyectados de la articulación metatarso-tobillo-rodilla izquierdos a lo largo de la secuencia de posturas luego de aplicarse PCA para reducir los datos a 3 dimensiones



(d) Ángulos originales y reproyectados de la articulación metatarso-tobillo-rodilla izquierdos a lo largo de la secuencia de posturas luego de aplicarse PCA para reducir los datos a 3 dimensiones y splines cúbicos con 11 puntos de control

Figura 3.1: Comparación de los resultados al aplicar solamente PCA o en conjunto con splines cúbicos sobre las articulaciones cadera-hombro-palma derechos y metatarso-tobillo-rodilla izquierdos

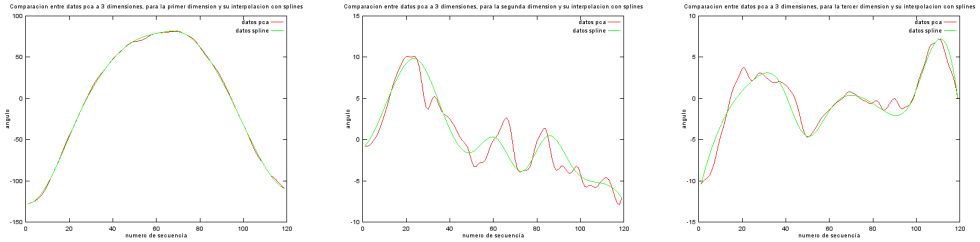


Figura 3.2: Comparación de los resultados al aplicar solamente PCA o en conjunto con splines cúbicos utilizando 11 puntos de control sobre la dimensión menor

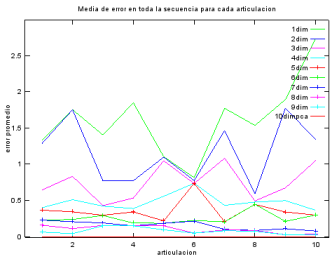


Figura 3.3: Gráfica de errores promedio al aplicar PCA para reducir a distintas dimensiones y luego volver a reproyectar esos datos sobre la dimensión original

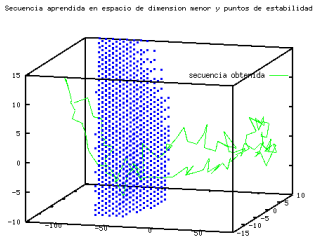


Figura 3.4: Mapa de estabilidad y trayectoria sobre dimensión menor

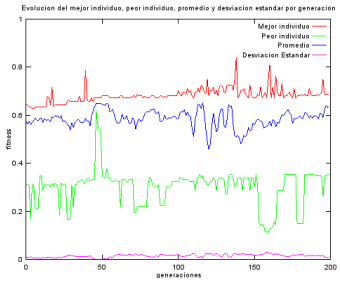


Figura 3.5: Gráfica de estadísticas del algoritmo genético

Capítulo 4

Simulador físico incluído con Bioloid Control

4.1. Introducción

El simulador físico utilizado en el proyecto de grado fue el que viene incluido con Bioloid Control. Bioloid Control es un proyecto de código de abierto realizado por la Universidad de Karlsruhe de Alemania, que permite trabajar con el robot Bioloid Premium. El proyecto podría separarse en dos partes principales, una parte que implementa un controlador del robot (que implementa la comunicación con el robot, ejecuta movimientos utilizando distintos tipos de interpolación, etc.) y la otra parte que es un simulador físico del robot (el mismo recibe instrucciones para control de los servos AX12 y simula la ejecución de dichas instrucciones utilizando el motor físico ODE). En el documento de proyecto de grado [San14] sección 3.3.6.1 se realiza un resumen de las funcionalidades tanto del proyecto Bioloid Control así como del simulador físico que contiene el mismo.

En la página oficial del proyecto [wdBC14] se pueden ver los detalles generales del mismo, características, instalación, ejecución, etc. así como descargar una versión actualizada del software.

4.2. Instalación

Según las especificaciones el sistema puede ser instalado tanto en el sistema operativo Windows como en Linux, aunque en el marco de proyecto de grado solamente se probó su ejecución sobre Linux.

La última versión de Bioloid Control puede ser descargada de su sitio (ver [dBC14a]). De todos modos conviene bajar la versión en código (con SVN, ver [dBC14b]) ya que la misma trae integrado el código del simulador físico. La versión utilizada en el proyecto es la 0.2.1 y el sistema operativo sobre el cual se instaló Bioloid Control fue Ubuntu 12.04.

El directorio principal de archivos de Bioloid Control contiene los siguientes sub-directorios:

- parser: Archivos de configuración de lexx e yacc y binarios windows flex (gnu) y bison (gnu). Internamente Bioloid Control implementa un lenguage script el cual es interpretado por el parser.
- main: Código fuente del programa principal.

- controller: Código fuente y archivos .hex precompilados del controlador del robot (programa μC).
- viewer: Código fuente del visor 3D OpenGL.
- physics: Código fuente del simulador físico.

4.2.1. Instalación/compilación del Firmware

Esta instalación no es necesaria si solamente se va a utilizar el simulador y no el robot real. En el proyecto no se realizó esta instalación por ese motivo.

Realizar los siguientes pasos:

- Cambiarse al directorio controller y abrir el archivo: include/constants.h.
- Cambiar AX12_COUNT del valor por defecto al número de servos AX12 conectados al cm-5 y guardar el archivo.
- Recompilar el programa: En Linux: todo. En Windows: Abrir RobotWrapper.pnproj con Programmers Notepad 2 y ejecutar Tools->Make Clean seguido de Tools->Make All. Si todo funciona como debería el archivo controller/RobotWrapper.hex se debería crear. Seguir los pasos del Manual Bioloid User para escribir el firmware controller/RobotWrapper.hex en el robot.

4.2.2. Instalación del viewer

En la carpeta viewer del proyecto compilarlo realizando los siguientes comandos:

```
make clean
make all
```

Se puede ejecutar el visor en una consola de comandos (en Linux) moviéndose al directorio viewer y ejecutando ./viewer o ejecutando el archivo viewer.exe (en Windows).

4.2.3. Instalación del main (controlador)

4.2.3.1. Instalaciones y modificaciones previas

- Instalar ncurses con comando sudo apt-get isntall libncurses5-dev
- En el archivo *include/wrapper.h* del directorio principal de main modificar AX12_COUNT desde su valor por defecto al número de servos AX12 conectados al CM-5 y cambiar AXS1_COUNT al número de sensores conectados CM-5 (1 por defecto). Guardar el archivo. En el proyecto no se realizó este paso ya que se utilizaron los 18 servos.

4.2.3.2. Corrección de errores de código

Se observó que la versión utilizada tenía errores en el código que no permitían la compilación correcta del main de Bioloid Control. Se realizaron los siguientes cambios:

- Debido a un error al compilar por redefinicion de una variable de entrada de función x en el archivo *interpolation.h*, comentar en ese archivo la siguiente definición (donde estaba definida la variable x dos veces) ya que no había implementación de esa operación en el .cpp: void linear_subsequent(IPOPRECISION *x, IPOPRECISION *y, IPOPRECISION *z, IPOPRECISION *h, int n, IPOPRECISION x, IPOPRECISION *y, int &oldindex);
- Debido al siguiente error al compilar: *src/platform.cpp:551:15: error: 'SSIZE_MAX' was not declared in this scope*, agregarle al principio del archivo src/platform.cpp la siguiente línea: #include <limits.h>. En limits.h esta definido el SSIZE_MAX.
- En *src/platform.cpp* modificar la línea 50 #define _SERIALDONTINIT por #define SERIALDONTINIT
- En *src/platform.cpp* modificar la línea 55 #define _INCLUDETCP por #define INCLUDETCP
- En *src/platform.cpp* modificar en la línea 972 #ifdef INCLUDE_TCP por #ifdef INCLUDETCP
- Debido a los errores de compilación *src/practicalsocket.cpp:50:38: error: 'strerror' was not declared in this scope* y *src/practicalsocket.cpp:136:42: error: 'memset' was not declared in this scope* agregar al principio del archivo las lineas #include <stdlib.h> y #include <string.h> a src/practicalsocket.cpp en la que se encuentran esas funciones (al parecer en versiones anteriores del compilador C++ estaban definidas en otra librería distinta a la utilizada en el proyecto).

4.2.3.3. Instalación

En la carpeta main del proyecto compilarlo realizando los siguientes comandos:

```
make clean
make all
```

Se puede ejecutar el programa main en Linux estando en una consola de comandos, parados en el directorio main y ejecutando ./bioloid. Lo mismo en Windows ejecutando el archivo análogo correspondiente.

4.2.4. Instalación del physics (simulador)

4.2.4.1. Instalaciones y modificaciones previas

El simulador físico implementado tiene dependencia con las siguientes librerías las cuales deben estar previamente instaladas en el sistema:

- ODE - motor físico que utiliza physics
- STL - Standard Template Libary
- Boost - librería para C++ (se la utiliza en physics para parseo de comandos en línea, hilos, etc.)
- asio - para sockets TCP (incluído en Boost para las versiones más actuales)

ODE es una librería que permite realizar simulación física de la dinámica de cuerpos. En su sitio oficial [dODE14] se puede observar una descripción más detallada de ODE, así como descargar la versión actualizada con guías para su instalación.

Boost es un conjunto de librerías gratuitas para C++ que implementan una amplia gama de funcionalidades, entre ellas en el simulador se utiliza boost para el parseo de comandos en línea, manejo de hilos y procesos, manejo de comunicación entre procesos, etc.. En el sitio oficial de Boost [wdBCL14] se puede ver una guía de uso e instalación de dicha librería.

Primeramente instalar las librerías que tienen dependencia con el physics:

- Descargar e instalar compilador C++ al sistema (por ejemplo ejecutando en línea de comandos: sudo apt-get install build-essential).
- Descargar cmake (por ejemplo ejecutando en línea de comandos: sudo apt-get install cmake).
- Descargar e instalar ODE en el sistema operativo. Previamente a compilar ODE agregar la línea #define dSINGLE al archivo */usr/local/include/ode/common.h* para utilizar la representación de números reales float en ODE.
- Descargar e instalar Boost en el sistema operativo.
- Instalar libxi y libxmu (en el proyecto se instalaron todos los paquetes que sugiere Ubuntu). Estos paquetes son librerías gráficas para visualización de la simulación.

Luego realizar las siguientes modificaciones:

- En *CMakeLists.txt* modificar la siguiente línea del archivo para que apunte a la carpeta donde se instaló ODE: SET(ode_dir ode) por SET(ode_dir /home/usuario/Escritorio/ODE/ode-0.12), en el hipotético caso que ODE estuviera instalado en /home/usuario/Escritorio/ODE/ode-0.12.
- En *CMLists.txt* modificar la línea TARGET_LINK_LIBRARIES(${PROJECT_NAME} boost_thread-mt boost_program_options) agregándole las opciones: boost_system boost_filesystem, de modo que quede TARGET_LINK_LIBRARIES(${PROJECT_NAME} boost_thread-mt boost_program_options boost_system boost_filesystem). Esto es para que se referencie a librerías boost utilizadas para la comunicación que se programó para sincronización entre el handler y el simulador.

4.2.4.2. Instalación del simulador

En la carpeta physics del proyecto realizar los siguientes pasos:

- Estando en la carpeta principal de physics, ejecutar en línea de comandos cmake que utiliza CMakeLists.txt para generar el Makefile.
- Estando en la carpeta principal del physics, ejecutar en línea de comandos make para compilar el simulador.

Se puede ejecutar el simulador en una consola de comandos (en Linux) moviéndose al directorio physics y ejecutando ./physics.

4.3. Modificaciones realizadas

4.3.1. Corrección de errores y adaptaciones

En las siguientes subsecciones se detallan algunas correcciones de errores y modificaciones realizadas en el sistema para que pudiera funcionar correctamente.

Listado 4.1: Cambio realizado en *bioloid.cpp*

```
1 BMessage BMessageQueue::next_message()
2 {
3   boost::mutex::scoped_lock scoped_lock(mutex_for_queue);
4   BMessage m=q.top();
5   q.pop();
6   return m;
7 }
```

4.3.1.1. Agregado de lock en physics

Debido a un error que provocaba una caída en physics con segmentation fault, debugueando se detectó que el problema estaba en el acceso que se hacía a una estructura sin un lock para no permitir el cambio de la misma por más de un proceso a la vez. Al querer acceder otro proceso a la estructura cuando otro más había borrado el elemento de la misma en paralelo se producía la caída. El cambio que se hizo fue agregar un lock en la misma. En el archivo *bioloid.cpp* en la función BMessageQueue::next_message() se agregó la línea boost::mutex::scoped_lock scoped_lock(mutex_for_queue); para realizar el lock (ver listado 4.1).

4.3.1.2. Modificación de referencias en physics

Al ejecutar physics se producía un error al no encontrarse el archivo de configuración del robot referenciado debido a la posición del archivo en la estructura utilizada. Se cambió la referencia que se hace en al archivo main.cpp dentro de la carpeta physics. Se modificó la siguiente línea:

("world.robot_file", po::value<string>(&robot_file)->default_value("../../physics/physics_humanoid.xml"), "robot definition file")

por ("world.robot_file", po::value<string>(&robot_file)->default_value("../physics/physics_humanoid.xml"), "robot definition file").

También se obtenía el siguiente error, Error: Can't open image file '../../physics/ode/drawstuff/textures/sky.ppm' Para corregir el error, en scene.cpp dentro de ../physics/source modificar la línea 194 de: fn.path_to_textures = (char *)"../../physics/ode/drawstuff/textures"; por: fn.path_to_textures = (char *)"../physics/drawstuff-obsolete/textures";

4.3.1.3. Error por límite de file descriptors en main y physics

Tanto en physics como en main luego de cierto tiempo de uso de la herramienta sin reiniciarla sucedía el siguiente error:

```
Exception: Too many files opened.
```

Este error se debe a la no liberación de recursos por parte del controlador y simulador. El límite de file descriptors por defecto que puede tener abierto un proceso al mismo tiempo en linux es de 1024. Al ejecutar las secuencias aproximadamente 500 veces se sobrepasa ese límite en el controlador (y casi en el simulador). Se optó por solucionar parcialmente el problema cambiando el límite por defecto en linux por 65536. Se estuvo viendo algunas causas pero nunca se corrigió por completo este error, como se mencionó antes la solución es modificar el límite de file descriptors del sistema operativo para eludir

el error. En las pruebas realizadas ejecutando miles de veces las secuencias no volvió a suceder dicho error.

Comandos útiles en Linux para esto son: lsof -p XXX |wc -l : donde XXX es el número id del proceso. Este comando permite visualizar la cantidad de file descriptors (fds) que tiene abierto el proceso. ulimit -n : devuelve el límite de fds por proceso que permite el sistema. cat /proc/sys/fs/file-nr : muestra los límites de file descriptors abiertos que se permiten en todo el sistema. El tercer número refiere a ese límite. ulimit -n número: establece parcialmente un nuevo valor límite de fds abiertos para cada proceso. La otra forma de cambiar el límite de fds es modificar el archivo /etc/security/limits.conf y agregarle las siguientes cuatro líneas: * hard nofile 65536 * soft nofile 65536 root hard nofile 65536 root soft nofile 65536. Las líneas con * establecen el límite para todos los procesos distintos de root.

4.3.1.4. Leaks de memoria en physics

Para el debug de mal uso de memoria que causan errores en el physics se utilizó el programa Valgrind (ver página oficial http://valgrind.org/). Utilizando la herramienta se modificaron algunas partes del código que acarreaban problemas con la memoria.

Compilación del Physics para utilizarlo con Valgrind

Compilar con la opción -Wall y -g para que se pueda ver la línea exacta donde se produce el error de memoria. En la consola de comandos estando en el directorio physics/src ejecutar:

```
g++ bioloid.cpp bioloid_devices.cpp body.cpp geom.cpp joint.cpp main.cpp scene.cpp sensor.cpp servo.cpp util.cpp world.cpp -Wall -g -c -I ../include/ -I ../../main/include/
```

Luego linkeditar ejecutando en línea de comandos:

```
g++ -Wall -g -o physicsdeb main.o bioloid.o bioloid_devices.o body.o geom.o joint.o scene.o sensor.o servo.o util.o world.o ../../main/src/configuration.o ../../main/src/tinyxml.o ../../main/src/tinyxmlerror.o ../../main/src/tinyxmlparser.o ../../main/src/tinystr.o -lode -lboost_program_options -lboost_system -lcurses -lpthread -lboost_thread /home/marcos/Escritorio/ODE/ode-0.12/drawstuff/src/.libs/libdrawstuff.a -lGL -lGLU -lXmu -lXi -lX11
```

En esas líneas se referencia a librerías de ODE, drawstuff, boost, curses, phtread, y las librerías gráficas GL, GLU, etc.

4.3.2. Modificaciones realizadas al controlador y simulador

Bioloid Control no implementa ninguna interfaz que permita a algún otro software interactuar con el mismo de forma de poder enviarle datos de secuencias a ejecutar en el robot para simularlas y luego obtener información de dicha simulación. En el caso particular del proyecto era necesario poder evaluar en el simulador las distintas secuencias generadas por el algoritmo genético implementado. En la siguiente sección se describen los cambios realizados tanto en el controlador como en el simulador de Bioloid Control de forma de que permitiera realizar la evaluación de secuencias. En el simulador se implementa la máquina de estados descripta en la figura 4.1, así como en el controlador la descripta en la figura 4.2.

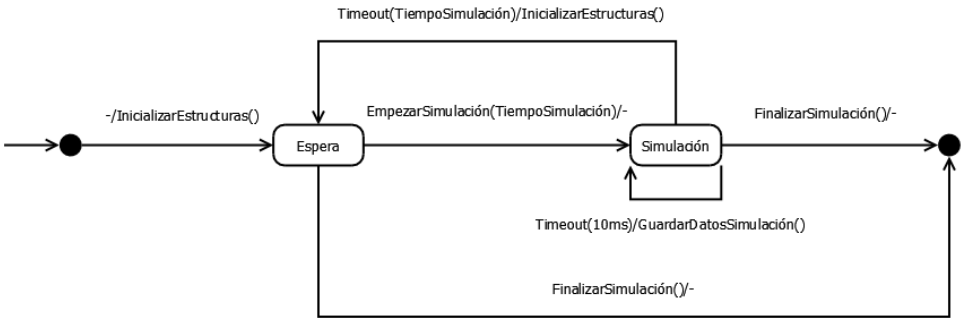


Figura 4.1: Máquina de estados que implementa el simulador (physics).

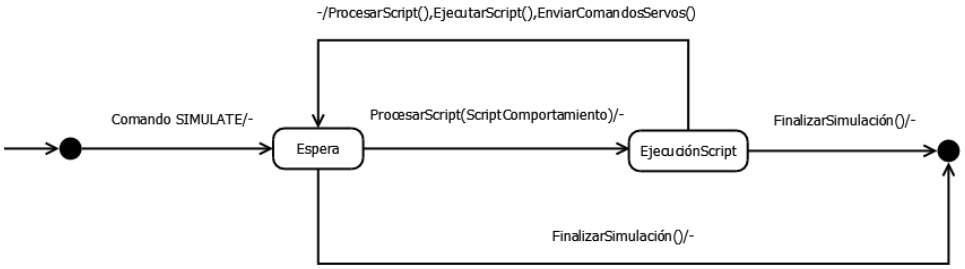


Figura 4.2: Máquina de estados que implementa el controlador (main).

4.3.2.1. Comunicación del algoritmo genético y el simulador

En la figura 4.3 se puede observar un diagrama de despliegue de todo el sistema. Como se observá en dicha figura tanto el algoritmo de aprendizaje, como el controlador y simulador del robot están en nodos distintos, o sea que su ejecución podria ser en máquinas distintas lograndose un mejor rendimiento del sistema. Para comunicar a los distintos nodos se utilizó TCP.

La clase simulatorHandler contenida en el paquete behavior del desarrollo realizado (ver capítulo 5) es la encargada de realizar la comunicación del sistema de aprendizaje con el robot, a la vez que realiza una sincronización entre el Controlador y el Simulador de Bioloid Control.

Una vez que una secuencia de posturas es generada, dicha secuencia es enviada a simulatorHandler de forma de poder evaluarla en el simulador. Para poder realizar dicha tarea primeramente el controlador procesa la secuencia de datos y envía al simulador en forma ordenada la secuencia de instrucciones que permiten ejecutar la secuencia de posturas en el tiempo dado.

Se tomó la decisión de interactuar con el simulador a través del controlador de Bioloid Control, ya que el mismo tiene implementado el control motor. La idea al utilizar tanto el controlador como el simulador Bioloid Control, era solamente realizar los cambios necesarios para poder disponer de la funcionalidad necesaria para realizar la simulación, de manera de no alterar de forma drástica la funcionalidad proporcionada por Bioloid Control. Una de las funcionalidades que se utilizaron del controlador, fue el control del robot a través de scripts de ejecución. El controlador permite el control del robot a través de scripts, con un lenguaje en el que se permite ejecutar funcionalidades similares a los comandos que se ejecutan en el controlador por el usuario. Dentro de la funcionalidad que brinda el controlador a través de scripts está el poder mover un motor cualquiera a una posición

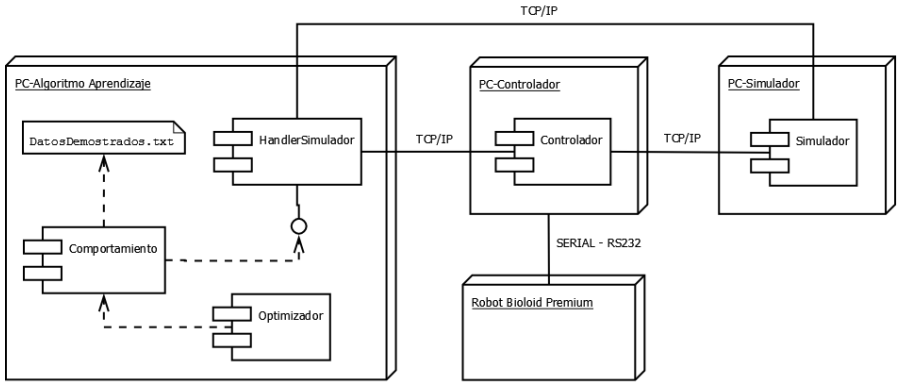


Figura 4.3: Diagrama de despliegue del sistema de aprendizaje

dada, en determinado tiempo con el comando SET(MOTOR, POSICIÓN, TIEMPO). También permite sincronizar el movimiento de más de un motor con el comando SYNC y ENDSYNC, de forma de poder realizar al mismo tiempo el movimiento de todos los motores. Con ambas funcionalidades es posible ir moviendo de una postura dada del robot (definida por la posición de cada motor del mismo) a otra postura deseada en el tiempo deseado. En el algoritmo 4.1 se puede observar un ejemplo de script de ejecución que permite mover al robot a una postura en que todos los motores tienen la posición inicial (posición igual a 0) en un tiempo de 2 segundos.

Algoritmo 4.1 Script de ejecución que permite mover todos los motores del robot desde determinada posición a la posición inicial de cada motor en un tiempo de 2 segundos.

```
1  tiempo=2000;
2  sync
3     set(1,0,tiempo);
4     set(2,0,tiempo);
5     set(3,0,tiempo);
6     set(4,0,tiempo);
7     set(5,0,tiempo);
8     set(6,0,tiempo);
9     set(7,0,tiempo);
10    set(8,0,tiempo);
11    set(9,0,tiempo);
12    set(10,0,tiempo);
13    set(11,0,tiempo);
14    set(12,0,tiempo);
15    set(13,0,tiempo);
16    set(14,0,tiempo);
17    set(15,0,tiempo);
18    set(16,0,tiempo);
19    set(17,0,tiempo);
20    set(18,0,tiempo);
21 endsync;
```

La clase behavior, utilizando los datos propios de comportamiento, genera el script que permite

que el controlador pueda ejecutar dicho comportamiento.

Para realizar la sincronización entre el controlador y el simulador, se utiliza conexión TCP y se envían los mensajes de sincronización a través de esa conexión. Esto permite que las clases que implementan el aprendizaje puedan ser ejecutadas en una máquina distinta a la que se ejecuta la simulación, en caso que se disponga de una máquina dedicada para realizar la simulación.

El algoritmo que realiza simulator handler para sincronizar la ejecución del comportamiento es el siguiente:

1. Se genera el script de ejecución dado.
2. Se avisa al simulador que pase a estado de simulación, o sea que empiece a realizar la simulación. El mismo queda a espera de instrucciones AX-12 para ser ejecutadas en el robot simulado.
3. Se avisa al controlador que puede procesar el script generado de forma de generar las instrucciones necesarias para realizar la tarea descripta en dicho script. Desde ese momento el controlador empieza a enviar al simulador las instrucciones necesarias para realizar la ejecución de las posturas en el robot.
4. Espera la ejecución de la tarea el tiempo necesario. Luego realiza la evaluación de la tarea ejecutada en el simulador.

4.3.2.2. Cambios en el main (controlador)

En el controlador se implementó un nuevo comando SIMULATE, que permite al mismo entrar en modo de espera de órdenes del handler de la simulación. En este modo el controlador implementa el siguiente comportamiento (ver figura 4.2):

1. Inicialmente se queda a la espera de que el handler le avise de que se debe empezar a procesar el script que contiene la secuencia de posturas a ejecutar.
2. Luego de recibir el aviso, el controlador lee el script, lo procesa y genera la secuencia de instrucciones que envía de forma ordenada y en el tiempo estipulado al simulador.
3. Por último luego de enviar todas las instrucciones vuelve al estado inicial de espera.

Comunicación

El main se comunica con otros dos programas por medio de TCP: con el physics (simulador), enviando las instrucciones a ejecutar en el simulador generadas por el script y recibiendo respuesta de dichas instrucciones; y con el handler, ya que el mismo es el encargado de coordinar el proceso de simulación. En el cuadro 4.1 se pueden ver los puertos utilizados en main para dichas comunicaciones. Para modificar uno de los puertos de comunicación con physics se puede cambiar el puerto en el archivo de configuración que se utilizará (en el caso del proyecto config_physics.xml). También se puede modificar el archivo por defecto en el fuente global.cpp donde se define el puerto. Para cambiar el puerto de comunicación con el handler se debe modificar en el fuente cmd.cpp el valor de puerto correspondiente (y volver a compilar todo el main).

Comunicación con:	Puerto	Definido por defecto en	Archivo de config	Parámetro
physics (entrada)	7777	global.cpp	config_physics.xml	Software - TcpPortIn
physics (salida)	7777	global.cpp	config_physics.xml	Software - TcpPortOut
handler	5431	cmd.cpp	-	-

Cuadro 4.1: Puertos de comunicación del physics (controlador)

4.3.2.3. Cambios en el physics (simulador)

Los cambios realizados en physics implementan el siguiente comportamiento (ver figura 4.1):

1. En el simulador físico se inicializan las estructuras de datos necesarias para realizar la simulación (cuerpos, articulaciones, geometrías, etc).
2. Se pasa al estado de espera hasta que el handler le avise que debe empezar la simulación.
3. Al recibir el aviso se empieza a simular y se recibe también el tiempo por el cual deberá continuar la simulación. Queda a la espera de mensajes que llegan por TCP desde el controlador en los que se reciben las instrucciones a ejecutar. A intervalos de 10 ms guarda en un archivo de texto las posiciones de cada parte del robot, así como la posición de cada motor en la simulación. Esta información es luego utilizada para evaluar la secuencia ejecutada.
4. Por último luego de simular por el tiempo previamente indicado por el handler, el simulador pasa al estado inicial donde se volverá a inicializar las estructuras de datos.

Comunicación

El physics se comunica con otros dos programas por medio de TCP: con el main (controlador), recibiendo las instrucciones a ejecutar y enviando respuesta de dichas instrucciones; y con el handler, ya que el mismo es el encargado de coordinar la simulación. En el cuadro 4.2 se pueden ver los puertos utilizados en physics para dichas comunicaciones. Para modificar el puerto de comunicación con main se puede cambiar el puerto en el archivo de configuración que se utilizará (en el caso del proyecto physics.conf). También se puede modificar el archivo por defecto en el fuente main.cpp donde se define el puerto. Para cambiar el puerto de comunicación con el handler se debe modificar en el fuente main.cpp el valor de puerto correspondiente (y volver a compilar todo el physics).

Comunicación con:	Puerto	Definido por defecto en	Archivo de configuración	Parámetro corresp.
main	7777	main.cpp	physics.conf	#port
handler	5432	main.cpp	-	-

Cuadro 4.2: Puertos de comunicación del physics (simulador)

4.4. Ejecución del sistema

4.4.1. Archivos de configuración y entrada

4.4.1.1. Main

El archivo *config_physics.xml* (utilizado en el proyecto) es el que contiene la configuración del controlador para que funcione junto con el simulador. En él se definen los parámetros de cada servo que implementan al humanoide (velocidad, torque, etc.), si se comunicará el programa con el robot real, las interfaces de comunicación habilitadas (en dicho archivo se habilita la interfaz TCP y se define el puerto para comunicarse con el simulador).

El archivo *dh_humanoid.xml* contiene los parámetros que definen al humanoide en el controlador (fue el archivo de configuración utilizado en el proyecto).

Al ejecutar el main en modo simulación (con comando SIMULATE), el mismo lee de la carpeta scripts el archivo con nombre *move_servos.s*, el cual describe alguna secuencia (con el formato de los scripts de Bioloid Control). Por lo tanto, para poder ejecutar las secuencias en el controlador se debe guardar un archivo con nombre move_servos.s describiendo con el formato de scripts adecuado la secuencia que se quiere ejecutar.

4.4.1.2. Physics

El archivo *physics_humanoid.xml* es el archivo con la configuración de las posiciones de los cuerpos, servos y articulaciones, que definen en el simulador al robot humanoide formado por 18 servos.

El archivo *physics.conf* contiene los parámetros que aplicarán al programa, entre ellos si está habilitada la pantalla gráfica de simulación, ancho y alto de dicha pantalla, habilitación y puerto de interfaz para la comunicación con el controlador, parámetros propios de la simulación como gravedad, tiempo, etc.

4.4.2. Comandos para ejecución del sistema

La ejecución del controlador junto con el simulador requiere que primeramente se ejecute el simulador. Al iniciar el controlador intentará conectarse al physics como cliente, por lo tanto este debe estar ya levantado. En las siguientes secciones se verá en orden los comandos de ejecución del sistema y por último se explica como ejecutar más de una instancia del mismo.

4.4.2.1. Ejecución del viewer

En una sesión de terminal, posicionarse en la carpeta viewer del proyecto Bioloid Control y ejecutar el siguiente comando para levantar el viewer:

```
cd main ../viewer/viewer -geometry 320x480+0+0 &
```

La opción geometry y su valor definen el tamaño de la pantalla a visualizar

4.4.2.2. Ejecución del physics (simulador)

En una sesión de terminal distinta a la que se ejecutó el viewer, posicionarse en la carpeta physics y ejecutar el siguiente comando:

```
./physics –simulation.time_end=0 –gui.width=320 –gui.height=480
```

La opción simulation.time_end indica el tiempo de simulación, con valor 0 indica simular infinitamente. La opción gui.width y gui.height definen el tamaño de la pantalla de visualización de la simulación.

4.4.2.3. Ejecución del main (controlador)

En la misma sesión donde se ejecutó el viewer, posicionarse en la carpeta main y ejecutar el siguiente comando:

```
./bioloid -config config_physics.xml
```

Con la opción config se le indica al programa principal que utilice cierto archivo de configuración (en el ejemplo config_physics).

Una vez levantado el programa, dentro del mismo ejecutar el comando SIMULATE para que el main entre en modo de espera de comunicación con el handler para empezar a realizar la ejecución de las secuencias simuladas.

4.4.2.4. Ejecución de más de una instancia del sistema

Para ejecutar más de una instancia del sistema se deben abrir distintas secciones de terminal de forma de ejecutar el simulador y controlador en dichas secciones. También configurar los puertos TCP por los cuales se comunicará cada instancia de forma que cada comunicación sea por un puerto distinto. Un ejemplo de configuración de puertos TCP para correr 2 instancias en paralelo sería modificar los archivos de configuración y los puertos en los fuentes correspondientes según lo indicado en el cuadro 4.3.

Programa	Se comunica con	Puerto utilizado
main1	physics1	7777
main1	handler1	5431
physics1	main1	7777
physics1	handler1	5432
main2	physics2	7787
main2	handler2	5433
physics2	main2	7787
physics2	handler2	5434

Cuadro 4.3: Ejemplo de configuración de puertos para la ejecución de dos instancias del sistema de Bioloid Control y el sistema de aprendizaje

4.4.3. Archivos de salida generados

Luego de la evaluación de una secuencia pasada al controlador y luego de ser ejecutada en el simulador, como resultado se obtiene un archivo de salida que contiene las posiciones x, y, z de cada motor en instantes de 0.1 segundos durante toda la simulación (en cada línea del archivo están las posiciones x, y, z para cada motor en el orden motor 1, motor 2,.., motor 18). Dicho archivo se nombró *evaluacion.txt* y se encuentra dentro de la carpeta *src* en *physics*.

Capítulo 5

Algoritmo genético - JMetal

5.1. Introducción

Para trabajar con los algoritmos genéticos se utilizó la biblioteca JMetal. Aunque la misma no es orientada a algoritmos genéticos mono-objetivos trae implementadas algunas metaheurísticas para ese tipo de algoritmos, así como distintas codificaciones de soluciones y operadores genéticos de cruzamiento, mutación, y selección de individuos. En la figura 5.1 se puede observar un diagrama de clases donde se puede ver la estructura de JMetal y en la figura 5.2 se observa un diagrama de clases de la representación de sus soluciones.

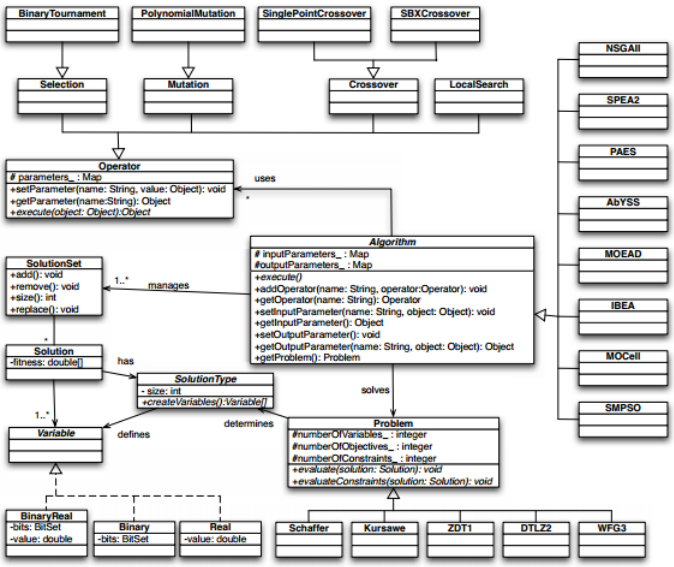


Figura 5.1: Estructura de clases de JMetal.

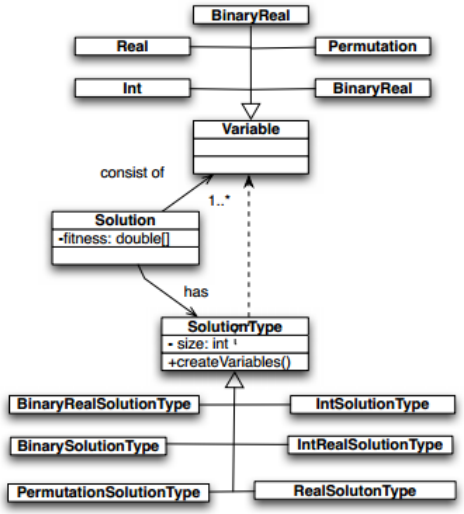


Figura 5.2: Representación de soluciones en JMetal.

5.2. Modificaciones a JMetal y clases auxiliares

A las implementaciones que trae JMetal se le agregaron nuevas clases que implementan la solución. En la figura 5.3 se observa las clases que definen a los operadores genéticos creados: RandomSelection y ArrayIntPointMutation, así como las clases que definen a operadores genéticos modificados: NTournament, SinglePointCrossover. También se observa la clase creada Atajada, que define el problema en cuestión. Por último también se puede observar la clase gGA modificada que implementa la metaheurística utilizada. A su vez en la figura 5.4 se puede observar las clases ArrayInt y ArrayPoint creadas, que son del tipo Variable, las que implementan un arreglo de enteros y arreglo de puntos respectivamente junto con los métodos para operar sobre esas representaciones. La clase ArrayIntPointSolutionType es un nuevo tipo de solución creado, que utiliza las variables del tipo ArrayInt y ArrayPoint, y modela la representación de los individuos en el algoritmo genético.

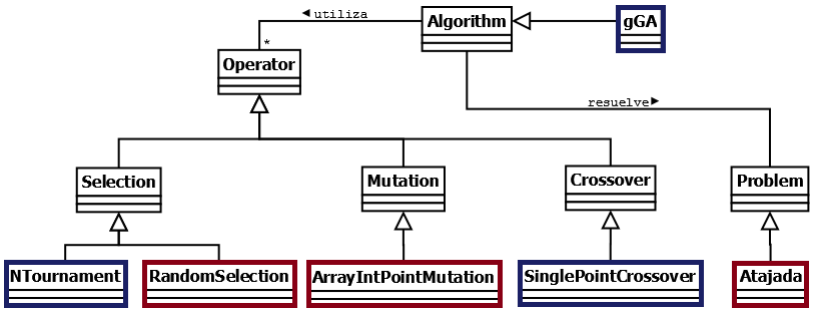


Figura 5.3: Estructura de clases agregadas (en color rojo) y modificadas (en color azul) a JMetal

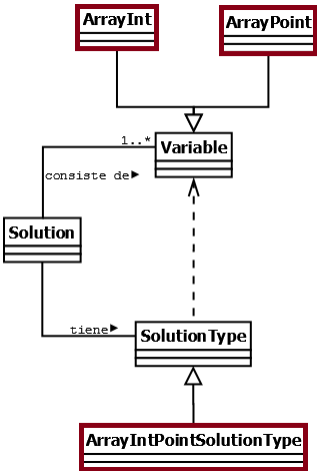


Figura 5.4: Representación de soluciones implementadas en JMetal (en color rojo)

Por otro lado también se crearon clases que implementan la abstracción de comportamiento y la comunicación del genético con Bioloid Control (ver figura 5.5). Dichas clases pertenecen al paquete de clases behavior. En la figura 5.5 se puede ver el diagrama de clases correspondiente a las clases contenidas en behavior.

La clase behavior representa un comportamiento del robot. El comportamiento se define por una secuencia de posturas, donde entre cada par de posturas existe un intervalo de tiempo definido también en dicho comportamiento. A su vez cada comportamiento se define también por una secuencia de ejecución de posturas, que determina de la secuencia de posturas, cuáles ejecutar en el simulador. Esta clase contiene el método que permite evaluar cada comportamiento en el espacio de dimensiones mayor.

La clase behaviorHandler maneja a los comportamientos y transformaciones entre comportamientos y las clases solución pertenecientes al genético. El método reduceDimensions crea un comportamiento en el espacio de dimensiones mayor, le aplica el método de reducción de dimensiones PCA, y devuelve los datos correspondientes a la dimensión menor. A su vez el método increaseDimensions implementa el incremento de dimensión desde datos que están en una dimensión menor hacia la dimensión mayor.

Por último la clase simulatorHandler implementa la comunicación y sincronización con el simulador de forma de poder realizar las evaluaciones de cada comportamiento. Cada comportamiento contenido en una instancia de clase behavior, puede ser evaluado mediante el método implementado en su clase llamado evaluate. Esta clase luego de generar el script necesario para que se pueda evaluar el comportamiento, le pide a la clase simulatorHandler que realice la ejecución del script tanto en el controlador como en el simulador. El resultado de ejecutar el comportamiento en el controlador y simulador es un archivo de texto que contiene datos de la ejecución realizada. Este archivo es luego utilizado por el método evaluate de la clase behavior para realizar la evaluación del comportamiento ejecutado en el simulador.

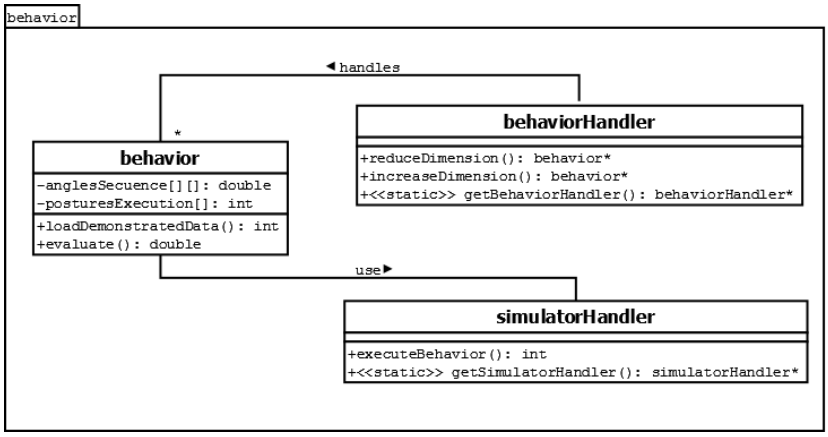


Figura 5.5: Diagrama de clases que implementan el comportamiento y comunicación con Bioloid Control (paquete behavior).

5.3. Instalación

En la página [wdJ14] se puede descargar la última versión de la librería JMetal. Como se mencionó anteriormente se realizaron modificaciones a la librería utilizada de JMetal y se agregaron clases que implementan la solución. En la entrega de los archivos del proyecto se incluye el código fuente de las clases desarrolladas. Dentro de JMetal se creó el paquete behavior, el cual contiene las clases que implementan el algoritmo de aprendizaje utilizando algoritmos genéticos. Se implementa el algoritmo genético junto con el handler que se comunica con el controlador/simulador de Bioloid Control para poder realizar las evaluaciones de las secuencias generadas por el algoritmo de aprendizaje. A su vez en las carpetas correspondientes se puede ver las implementaciones de los operadores genéticos realizadas, las codificaciones de soluciones, etc.

Para instalar tanto la librería de JMetal original como la modificada para el proyecto, simplemente abrir una línea de comandos, posicionarse en el directorio principal de JMetal y ejecutar el comando make. Esto hará que se compilen los fuentes y se generen los ejecutables. En el directorio *bin* se puede encontrar el ejecutable gGA_main, el cual implementa el algoritmo genético utilizado en el proyecto. GGA (generational genetic algorithm) es la metaheurística (con modificaciones) que se utilizó para la implementación del algoritmo de aprendizaje, por eso el archivo ejecutable de salida es gGA_main.

5.4. Ejecución del sistema aprendizaje

Para poder ejecutar el sistema de aprendizaje en conjunción con el sistema de simulación de forma de poder evaluar las secuencias generadas se debe primeramente configurar los puertos de comunicación entre el handler del sistema y el sistema de simulación, compilar el sistema y luego generar los archivos de entrada necesarios. Como salida de la ejecución del sistema se obtendrán un conjunto de archivos correspondientes a las salidas de cada corrida ejecutada. Por último se ejecutará un programa que tomando como insumo los archivos de salida de las corridas del genético realiza una comparación para determinar cuáles conjuntos de parámetros son los estadísticamente óptimos.

5.4.1. Configuración del entorno

Previo a la ejecución del sistema de aprendizaje se deben configurar los puertos de comunicación que se utilizarán, el lugar donde se encuentra el archivo de evaluación del simulador para poder evaluar las secuencias y cuáles serán las corridas del genético que se realizarán en la ejecución del programa.

5.4.1.1. Puertos de comunicación

Como se mencionó en el capítulo 4 el handler del algoritmo de aprendizaje se comunica tanto con el controlador (main) como con el simulador (physics) de Bioloid Control. Para que dicha comunicación sea correcta se deben configurar los puertos de comunicación con el controlador y el simulador.

En el fuente *simulatorHandler.cpp* se definen las constantes *host1, port1, host2* y *port2* donde *host1* y *port1* refieren al host y puerto de comunicación con el simulador respectivamente, y *host2* y *port2* refieren al host y puerto de comunicación con el controlador respectivamente. Se puede modificar dichas constantes y volver a compilar el proyecto para comunicarse mediante otros hosts/puertos con el controlador y simulador de Bioloid Control.

5.4.1.2. Lectura del archivo de evaluación

Cada vez que se realiza la ejecución en el simulador de una secuencia de posturas, los datos de esa ejecución quedan guardados en un archivo llamado *evaluacion.txt*, el cual se encuentra en el directorio src del physics de Bioloid Control. En *behavior.h* se debe configurar la constante EVALUATION_FILE de forma de que la misma sea la ruta relativa o absoluta hasta el archivo *evaluation.txt*. Tener en cuenta que dicha ruta podría inclusive ser una ruta de red, si la máquina que realizara las simulaciones físicas fuese una máquina distinta (dedicada) a la máquina que ejecuta el algoritmo genético. En ese caso se deben tener los permisos necesarios para poder acceder a ese archivo en la otra máquina.

5.4.1.3. Configuración de parámetros por corrida

En el archivo *gGA.cpp* se definen constantes correspondientes a parámetros de una corrida del genético. A continuación se listan algunas de esas constantes y se explica su significado:

ARCHIVO_ESTADÍSTICAS Prefijo del nombre del archivo que tiene los datos estadísticos de la corrida (mejor, peor, promedio y desviación estándar de fitness por generación). Por defecto tiene el valor "estadisticasGenetico_".

ARCHIVOS_FITNESS_PF Prefijo del nombre del archivo que tiene los fitness de la población final de la corrida. Por defecto tiene el valor "fitnessPoblacionFinal_".

ARCHIVO_INDIVIDUOS_PF Prefijo del nombre del archivo que tiene los genes de cada individuo de la población final. Por defecto tiene el valor "individuosPoblacionFinal_"

MAXIMO_GENERACIONES Cantidad máxima de generaciones por corrida. Por defecto son 200.

MINIMO_GENERACIONES Cantidad mínima de generaciones por corrida. Por defecto son 10.

MAXIMO_GENERACIONES_SIN_CAMBIO Cantidad máxima de generaciones sin mejora en el individuo máximo y promedio que deben ocurrir como criterio para detener el algoritmo antes de la cantidad máxima de generaciones. Para asegurar individuos estables se aplica este criterio recién después de que se encuentra un individuo con fitness 1. Por defecto tiene valor 20.

MINIMO_CAMBIO_PROMEDIO_POBLACION Valor mínimo que debe cambiar el promedio de población para considerarse que hay una mejora en dicha población. Por defecto tiene valor 0.001

PORCENTAJE_ELITISTA_POBLACION Tasa de población elitista que se seleccionará para cada siguiente generación del algoritmo genético.

MAX_BEING_BEST Cantidad de generaciones seguidas en las que un individuo tiene que ser el individuo con mejor fitness para parar el algoritmo por este criterio. Por defecto tiene valor 10.

5.4.1.4. Configuración de corridas del genético

Para configurar cuáles corridas se ejecutarán en el algoritmo se debe modificar en el fuente *gGA_main.cpp* la matriz *parametersExecution* y la variable *totalParameters*. La matriz parametersExecution tiene en cada fila los datos de una corrida en el siguiente orden: población inicial, tasa de cruzamiento, tasa de mutación y número de corrida. La variable totalParameters tiene el valor de la cantidad de corridas que se harán en la ejecución del genético.

5.4.2. Archivos de entrada

Los archivos de entrada que el algoritmo genético utiliza como insumo para su ejecución son los siguientes:

atajada.txt En el mismo directorio donde se encuentra el ejecutable *gGA_main* debe encontrarse el archivo *atajada.txt*, el cual contiene los datos de demostración de una secuencia de la tarea realizada por la persona. Es el archivo obtenido luego de procesar los datos de una secuencia demostrados por la persona (ver capítulo 2, donde se describe cómo se obtuvieron dichos datos). El algoritmo genético utiliza estos datos para generar la población inicial de individuos.

StabilityMatrix.txt También en el mismo directorio del ejecutable gGA_main debe estar el archivo *StabilityMatrix.txt*, el cual contiene la matriz de estabilidad que debe haber sido previamente generada. En subsección 6.2.1 se describe cómo se genera dicha matriz de estabilidad. El algoritmo genético utiliza estos datos para aplicar el algoritmo de estabilidad sobre la primer postura de la ejecución de cada secuencia. Esto se realizó para lograr una mayor efectividad en el aprendizaje, al tener la primer postura estabilizada.

5.4.3. Ejecución

5.4.3.1. Simple

Para ejecutar el sistema primeramente se debe haber ejecutado en otras ventanas de línea de comandos tanto el simulador (physics) como el controlador (main) de Bioloid Control (y su comando SIMULATE).

Luego de tener levantado el software de simulación, en una nueva ventana de línea de comandos, posicionarse en la carpeta bin de JMetal y ejecutar el siguiente comando:

```
./gGA_main > salida_genetico.txt
```

Como se aprecia en el comando se direcciona la salida estándar al archivo de texto salida_genetico.txt.

En las ventanas tanto del controlador como del simulador se podrá visualizar cómo se ejecutan las evaluaciones de las secuencias una tras otra.

5.4.3.2. Con más de un entorno

Debido a que las simulaciones se hacen en tiempo real, el tiempo de ejecución del algoritmo genético es muy grande. Para tratar de aminorar los tiempos se puede ejecutar en paralelo más de una instancia del algoritmo genético (y controlador y simulador por cada una de esas instancias). La cantidad de las mismas estará determinada por la capacidad de memoria y procesamiento del sistema donde se ejecutan dichas instancias. En el marco del proyecto se realizó la ejecución de dos instancias de todo el sistema (genético, simulador y controlador) en paralelo sobre la misma máquina, lográndose la reducción del tiempo de ejecución aproximadamente a la mitad del tiempo inicial .

Para configurar la ejecución de dos (o más) instancias sobre una misma máquina se debe tener en cuenta que se deben configurar distintos puertos y archivos de entrada para cada una de esas instancias. A su vez se deben configurar las corridas de forma que una instancia realice unas corridas y la otra las complementarias al total. En los siguientes cuadros se puede ver un ejemplo de configuración de puertos (cuadro 5.1), archivos (cuadro 5.2) y corridas (cuadro 5.3) para la ejecución de dos instancias (instancia 1 - bioloidControlPhysics1 y JMetal1, instancia 2 - bioloidControlPhysics2 y JMetal2).

Programa	Se comunica con	Puerto utilizado
main1	physics1	7777
main1	handler1 (gGA_main1)	5431
physics1	main1	7777
physics1	handler1 (gGA_main1)	5432
main2	physics2	7787
main2	handler2 (gGA_main2)	5433
physics2	main2	7787
physics2	handler2 (gGA_main2)	5434

Cuadro 5.1: Ejemplo de configuración de puertos para dos instancias del sistema de aprendizaje

Programa	Ruta del archivo de evaluación en behavior.h
gGA_main1	"../../../simuladorBioloidControl/bioloidControlPhysics1/bioloidcontrol/physics/evaluacion.txt"
gGA_main2	"../../../simuladorBioloidControl/bioloidControlPhysics2/bioloidcontrol/physics/evaluacion.txt"

Cuadro 5.2: Configuración de ruta de archivo de evaluación

5.4.4. Archivos de salida de la ejecución

Los archivos de salida que el algoritmo genético genera durante y luego de su ejecución son los siguientes:

estadisticasGenetico_POB_CRUZ_MUT_COR Por cada corrida configurada previamente a la ejecución del algoritmo de aprendizaje, se genera un archivo de nombre *estadisticasGeneti-*

Programa	Población inicial	Cruzamiento	Mutación	Corrida
gGA_main1	50	80	5	1
gGA_main1	50	80	5	2
gGA_main2	50	80	5	3
gGA_main2	50	80	5	4
gGA_main1	70	80	5	1
gGA_main1	70	80	5	2
gGA_main2	70	80	5	3
gGA_main2	70	80	5	4

Cuadro 5.3: Configuración de ejecuciones en instancias del sistema de aprendizaje

co_POB_CRUZ_MUT_COR, donde POB será el valor de cantidad de población, CRUZ la tasa de cruzamiento, MUT la tasa de mutación y COR el número de corrida de la corrida correspondiente. En dicho archivo en cada línea se podrá visualizar los datos de fitness máximo, mínimo, promedio y desviación estándar de la generación correspondiente al número de esa línea para la corrida correspondiente.

fitnessPoblacionFinal_POB_CRUZ_MUT_COR Por cada corrida configurada previamente a la ejecución del algoritmo de aprendizaje, se genera un archivo de nombre *fitnessPoblacionFinal_POB_CRUZ_MUT_COR*, donde POB será el valor de cantidad de población, CRUZ la tasa de cruzamiento, MUT la tasa de mutación y COR el número de corrida de la corrida correspondiente. En dicho archivo se muestran los fitness de todos los individuos de la población final de la corrida y en cada línea se podrá visualizar los datos de fitness de un individuo determinado de la población final. Dichos datos están ordenados de mayor a menor, de la primera línea a la última.

individuosPoblacionFinal_POB_CRUZ_MUT_COR Por cada corrida configurada previamente a la ejecución del algoritmo de aprendizaje, se genera un archivo de nombre *individuosPoblacionFinal_POB_CRUZ_MUT_COR*, donde POB será el valor de cantidad de población, CRUZ la tasa de cruzamiento, MUT la tasa de mutación y COR el número de corrida de la corrida correspondiente. En dicho archivo se imprimen (ordenados de mayor a menor según su fitness) los genes de cada individuo de la población final, así como su fitness y valor objetivo.

FUN_POB_CRUZ_MUT_COR/VAR_POB_CRUZ_MUT_COR Por cada corrida configurada previamente a la ejecución del algoritmo de aprendizaje, se genera un archivo de nombre *FUN_POB_CRUZ_MUT_COR* y otro de nombre *VAR_POB_CRUZ_MUT_COR*, donde POB será el valor de cantidad de población, CRUZ la tasa de cruzamiento, MUT la tasa de mutación y COR el número de corrida de la corrida correspondiente. Los archivos FUN_* y VAR_* contienen cada uno el valor objetivo del individuo de mayor fitness de la población final y los genes del individuo de mayor fitness de la población final para la corrida correspondiente respectivamente.

5.4.5. Conjunto óptimo de parámetros

Luego de ejecutar el algoritmo genético se obtiene como resultado los archivos previamente mencionados. El ejecutable *comparator* utiliza como insumo los archivos *fitnessPoblacionFinal_POB_CRUZ_MUT_COR* y utilizando un criterio programado selecciona un conjunto de parámetros dado. El criterio utilizado en el proyecto fue el de significancia estadística (ver sección 5.2 del documento final de proyecto de grado).

Para ejecutar el programa que aplica el criterio primeramente se debe ir al directorio bin del proyecto de JMetal. En esa carpeta deben estar los archivos generados por el algoritmo genético *fitnessPoblacionFinal_**. Ejecutar el comando ./comparator. El programa escribe en la salida estándar el conjunto de parámetros óptimo calculado: población inicial, tasa de cruzamiento y tasa de mutación.

Capítulo 6

Ejecutables auxiliares

6.1. Introducción

En conjunción con lo presentado en el capítulo 5 también se crearon y utilizaron otras aplicaciones de menor porte para realizar ciertas tareas necesarias en el marco del proyecto. A continuación se describen algunas de esas aplicaciones utilizadas. Al compilar el proyecto dichas aplicaciones se generan en la carpeta bin del JMetal.

6.2. Aplicaciones

6.2.1. stability_Matrix_Generator

La técnica de mapas de estabilidad se utilizó como ayuda para el algoritmo genético, modificando solamente la primer postura de la secuencia de modo que la misma sea estable de acuerdo a la técnica de mapas de estabilidad. En la subsección 3.3.3.3 del documento final de proyecto [San14] se describe la técnica de mapas de estabilidad utilizada en el proyecto de grado.

El ejecutable stabilityMatrixGenerator genera el mapa de estabilidad correspondiente utilizado en el proyecto. Las constantes MIN_* y MAX_* y n* del fuente stabilityMatrixGenerator.cpp definen los puntos (del espacio de dimensión menor) sobre los cuales se testeará la estabilidad e implícitamente definen un subespacio de ese espacio menor sobre el cual se hará el mapeo de estabilidad (ver figura 3.4).

Para poder ejecutar este programa primeramente se debe levantar tanto el simulador como el controlador de Bioloid Control y ejecutar el comando SIMULATE dentro del controlador. Luego al ejecutar stabilityMatrixGenerator, el programa empezará a evaluar en los puntos definidos (utilizando el simulador de Bioloid Control) cuáles posturas son estables y cuáles no según la definición de estabilidad de la técnica utilizada, e imprimirá ese resultado en la salida estándar. Conviene derivar la salida estándar a un archivo de texto ya que luego se utilizará este archivo como entrada del algoritmo genético.

6.2.2. initial_Population_Fitness

Esta aplicación evalúa el fitness de todos los posibles individuos inciales del algoritmo genético para una demostración dada y devuelve por salida estándar el fitness de cada individuo ejecutado,

el promedio de finess y el total de posibles individuos iniciales. Los individuos iniciales del algoritmo genético tienen como matriz de posturas las posturas demostradas por la persona. Lo que varía en cada individuo inicial es su vector de ejecución de posturas de acuerdo a su estructura definida (ver la sección 4.4.2 *Codificación de las soluciones del genético* del documento final [San14]).

La finalidad de esta aplicación era ver la variabilidad en fitness de los individuos iniciales del algoritmo genético. Se produce poca diversidad genética, pero que más adelante es lograda al aplicarse los operadores de cruzamiento y mutación.

Para ejecutar la aplicación, primeramente se debe levantar tanto el simulador como el controlador de Bioloid Control y ejecutar el comando SIMULATE dentro del controlador. Luego al ejecutar initialPopulationFitness, el programa empezará a evaluar a todos los posibles individuos iniciales.

6.2.3. execute_Best_Behavior

Esta aplicación evalúa n veces a determinado individuo, cuyos genes deben estar en el archivo de texto plano VAR. La constante NUMBER_OF_EVALUATIONS en el código fuente determina la cantidad de evaluaciones que se hará del mismo individuo. Devuelve por salida las evaluaciones realizadas y un promedio de las mismas.

Para ejecutar la aplicación, primeramente se debe levantar tanto el simulador como el controlador de Bioloid Control y ejecutar el comando SIMULATE dentro del controlador. Luego al ejecutar executeBestBehavior, el programa evaluará la cantidad de veces indicada al individuo cuyos genes están en el archivo VAR dentro del mismo directorio que se encuentra executeBestBehavior.

Bibliografía

[dBC14a] Archivos de Bioloid Control. *http://sourceforge.net/projects/bioloidcontrol/files/*. último acceso: Nov 2014.

[dBC14b] SVN de Bioloid Control. *http://sourceforge.net/p/bioloidcontrol/code/HEAD/tree/*. último acceso: Nov 2014.

[dGOeW14] Instalacion de GNU Octave en Windows. *http://wiki.octave.org/Octave_for_ Windows*. último acceso: Nov 2014.

[dO14] Instalación de Octave. *http://www.gnu.org/software/octave/download.html*. último acceso: Nov 2014.

[dODE14] Sitio Web de Open Dynamics Engine. *http://www.ode.org/*. último acceso: Nov 2014.

[San14] Marcos Sander. *Documento Final - PGILearn*. 2014.

[wdBC14] Sitio web de Bioloid Control. *http://bioloidcontrol.sourceforge.net/*. último acceso: Nov 2014.

[wdBCL14] Sitio web de Boost C++ Libraries. *http://www.boost.org/*. último acceso: Nov 2014.

[wdGO14] Sitio web de GNU Octave. *http://www.gnu.org/software/octave/*. último acceso: Nov 2014.

[wdJ14] Sitio web de JMetal. *http://jmetalcpp.sourceforge.net/*. último acceso: Nov 2014.