

Prototipado de una aplicación interactiva mediante Kinect

Motor de captura y reconocimiento de
posturas y gestos.

Javier Robledo Zarco

Índice

| | | |
|-------|--|----|
| 1. | Introducción | 8 |
| a. | Motivación | 8 |
| b. | Objetivos | 9 |
| 2. | Estado del arte | 10 |
| a. | Análisis de sistemas similares | 10 |
| i. | Reconocimiento basado en coordenadas | 10 |
| ii. | Reconocimiento basado en modelos ocultos de Markov..... | 11 |
| iii. | Dynamic Time Warping | 12 |
| b. | Sistemas poco explorados..... | 13 |
| c. | Kinect DTW Gesture Recognition | 14 |
| 3. | Planificación y presupuesto | 15 |
| a. | Planificación | 15 |
| b. | Presupuesto | 17 |
| 4. | Ánalisis..... | 17 |
| a. | Definición del alcance del sistema | 17 |
| b. | Definición de casos de uso | 18 |
| i. | UC-01-Capturar una postura:..... | 19 |
| ii. | UC-02-Ver posturas guardadas | 20 |
| iii. | UC-03-Cambiar el nombre de una postura | 21 |
| iv. | UC-04-Eliminar una postura..... | 23 |
| v. | UC-05-Asignar un gesto a una postura | 24 |
| vi. | UC-06-Eliminar el gesto de una postura | 26 |
| vii. | UC-07-Cambiar el tiempo de espera del motor | 27 |
| viii. | UC-08-Activar reconocimiento de posturas y gestos..... | 28 |
| ix. | UC-09-Exportar posturas..... | 29 |
| x. | UC-10-Importar posturas | 30 |
| c. | Extracción de requisitos de usuario | 31 |
| i. | Requisitos de capacidad..... | 32 |
| ii. | Requisitos de restricción | 34 |
| d. | Identificación de subsistemas | 35 |
| e. | Identificación de clases principales y responsabilidades | 36 |
| i. | Interfaz de comunicación | 36 |
| ii. | Interfaz del sensor..... | 36 |
| iii. | Interfaz de almacenamiento/recuperación | 37 |
| iv. | Núcleo del motor | 37 |
| f. | Definición de pruebas de aceptación del sistema..... | 37 |
| i. | GP-01 – Grabar una postura..... | 38 |
| ii. | GP-02 – Gestionar gestos asociados a una postura | 39 |
| iii. | GP-03 Reconocer una postura | 40 |
| iv. | GP-04 Reconocer un gesto | 40 |
| v. | GP-05 Gestionar posturas | 41 |
| vi. | GP-06 Exportar posturas | 42 |
| vii. | GP-07 Importar posturas..... | 43 |
| viii. | GP-08 Opciones..... | 43 |

| | | |
|-------|---|----|
| 5. | Diseño..... | 44 |
| a. | Definición de la arquitectura..... | 44 |
| b. | Diseño de clases y responsabilidades | 45 |
| i. | Interfaz de comunicación..... | 47 |
| ii. | Interfaz del sensor..... | 48 |
| iii. | Núcleo del motor | 49 |
| iv. | Interfaz de almacenamiento/recuperación | 52 |
| c. | Diseño de componentes del sistema | 52 |
| i. | Interfaz de comunicación..... | 52 |
| ii. | Interfaz del sensor..... | 62 |
| iii. | Núcleo del motor | 63 |
| iv. | Interfaz de Almacenamiento/Recuperación | 73 |
| d. | Diagramas de secuencia de los principales casos de uso..... | 75 |
| i. | UC-01: Capturar una postura | 75 |
| ii. | UC-02: Ver posturas guardadas..... | 78 |
| iii. | UC-03: Cambiar el nombre de una postura | 79 |
| iv. | UC-04: Eliminar una postura | 80 |
| v. | UC-05: Asignar un gesto a una postura..... | 81 |
| vi. | UC-06: Eliminar el gesto de una postura..... | 82 |
| vii. | UC-07: Cambiar tiempos de espera | 83 |
| viii. | UC-08: Activar reconocimiento de posturas y gestos | 84 |
| ix. | UC-09: Exportar posturas | 86 |
| x. | UC-10: Importar posturas | 87 |
| e. | Diseño de los ficheros de datos | 88 |
| f. | Mockups de interfaz de usuario..... | 88 |
| i. | Pantalla principal..... | 89 |
| ii. | Menú de administrar posturas..... | 89 |
| iii. | Menú de opciones avanzadas | 90 |
| iv. | Capturar una postura | 90 |
| v. | Agregar gesto | 91 |
| vi. | Importar ó exportar posturas..... | 91 |
| vii. | Mensajes de error/información | 91 |
| 6. | Resultado de las pruebas de aceptación..... | 92 |
| 7. | Conclusiones..... | 92 |
| a. | Conclusiones de los resultados | 92 |
| b. | Líneas futuras del sistema..... | 93 |
| 8. | Anexo I: Posturas comunes | 94 |
| 9. | Anexo II: Maquina de estados del reconocimiento de gestos | 95 |
| 10. | Anexo III: Manual de usuario | 95 |
| 11. | Anexo IIII: Bibliografía | 95 |

Tablas

| | |
|---|----|
| Tabla 1: Análisis D.A.F.O..... | 15 |
| Tabla 2: Gastos de amortizaciones..... | 17 |
| Tabla 3: Gastos de personal | 17 |
| Tabla 4: UC-01-Capturar una postura | 19 |
| Tabla 5: UC-02-Ver posturas guardadas..... | 20 |
| Tabla 6: UC-03-Cambiar nombre de una postura | 21 |
| Tabla 7: UC-04-Eliminar una postura | 23 |
| Tabla 8: UC-05-Asignar un gesto a una postura..... | 24 |
| Tabla 9: UC-06-Eliminar el gesto de una postura..... | 26 |
| Tabla 10: UC-07-Cambiar el tiempo de espera del motor | 28 |
| Tabla 11: UC-08-Activar reconocimiento de posturas y gestos | 28 |
| Tabla 12: UC-09-Exportar posturas | 29 |
| Tabla 13: UC-10-Importar posturas | 31 |
| Tabla 14: RUC-01..... | 32 |
| Tabla 15: RUC-02..... | 32 |
| Tabla 16: RUC-03..... | 32 |
| Tabla 17: RUC-04..... | 32 |
| Tabla 18: RUC-05..... | 32 |
| Tabla 19: RUC-06..... | 33 |
| Tabla 20: RUC-07..... | 33 |
| Tabla 21: RUC- 08..... | 33 |
| Tabla 22: RUC- 09..... | 33 |
| Tabla 23: RUC- 10..... | 33 |
| Tabla 24: RUC- 11..... | 33 |
| Tabla 25: RUC- 12..... | 34 |
| Tabla 26: RUC- 13..... | 34 |
| Tabla 27: RUC-14..... | 34 |
| Tabla 28: RUC-16..... | 34 |
| Tabla 29: RUR-01..... | 34 |
| Tabla 30: RUR-02..... | 35 |
| Tabla 31: RUR-03..... | 35 |
| Tabla 32: Responsabilidades de la clase Notificador | 36 |
| Tabla 33: Responsabilidades de la clase Entrada..... | 36 |
| Tabla 34: Responsabilidades de la clase Sensor..... | 37 |
| Tabla 35: Responsabilidades de la clase ManejadorFicheros | 37 |
| Tabla 36: Responsabilidades de la clase Capturador | 37 |
| Tabla 37: Responsabilidades de la clase Reconocedor | 37 |
| Tabla 38: Responsabilidades de la clase Postura | 37 |
| Tabla 39: Resumen del plan de pruebas | 38 |
| Tabla 40: Diseño de clase IObservador | 47 |
| Tabla 41: Diseño de clase Notificador..... | 47 |

| | |
|--|----|
| Tabla 42: Diseño de clase TipoError | 48 |
| Tabla 43: Diseño de clase Motor..... | 48 |
| Tabla 44:Diseño de la clase Sensor | 49 |
| Tabla 45: Diseño de la clase postura..... | 49 |
| Tabla 46: Diseño de la clase InterpreteDatos | 49 |
| Tabla 47: Diseño de la clase Capturador | 50 |
| Tabla 48: Diseño de la clase ModoCapturador | 50 |
| Tabla 49: Diseño de la clase Reconocedor | 50 |
| Tabla 50: Diseño de la clase ModoReconocedor | 51 |
| Tabla 51: Diseño de la clase RedNeuronas | 51 |
| Tabla 52: Diseño de la clase EstadoRed | 52 |
| Tabla 53: IObservador.ErrorProducido | 53 |
| Tabla 54: IObservador.GestoReconocido..... | 53 |
| Tabla 55: IObservador.MotorListo | 53 |
| Tabla 56:IObservador.nuevoFrameColor | 53 |
| Tabla 57: IObservador.nuevoFrameEsqueleto..... | 54 |
| Tabla 58: IObservador.PosturaCapturada..... | 54 |
| Tabla 59: IObservador.PosturaReconocida..... | 54 |
| Tabla 60: IObservador.PosturasExportadas | 54 |
| Tabla 61: Notificador.AgregarObservador | 55 |
| Tabla 62: Notificador.NotificarError..... | 55 |
| Tabla 63: Notificador.NotificarGestoReconocido | 55 |
| Tabla 64: Notificador.NotificarNuevoFrameColor | 55 |
| Tabla 65: Notificador.NotificarNuevoFrameEsqueleto | 55 |
| Tabla 66: Notificador.NotificarPosturaCapturada | 56 |
| Tabla 67: Notificador.NotificarPosturaReconocida..... | 56 |
| Tabla 68: NotificarPosturasExportadas | 56 |
| Tabla 69: Notificador.NotificarReconocedorLista..... | 56 |
| Tabla 70: Motor.ActivarReconocimiento | 56 |
| Tabla 71: Motor.AgregarObservador | 57 |
| Tabla 72: Motor.AgregarPosturaCapturada..... | 57 |
| Tabla 73: Motor.AnadirPosturaAGesto..... | 57 |
| Tabla 74: Motor.CambiarAlpha | 57 |
| Tabla 75: Motor.CambiarNeuronasIntermedias | 58 |
| Tabla 76: Motor.CambiarNombrePostura | 58 |
| Tabla 77: Motor.CambiarTasaAprendizaje..... | 58 |
| Tabla 78: Motor.CambiarTiempolnicio | 58 |
| Tabla 79: Motor.CambiarTiempoLimite | 58 |
| Tabla 80: Motor.CapturarPostura | 59 |
| Tabla 81: Motor.DesactivarCapturador | 59 |
| Tabla 82: Motor.DesactivarReconocedor | 59 |
| Tabla 83: Motor.EliminarPostura | 59 |
| Tabla 84: Motor.ExportarPosturas..... | 59 |
| Tabla 85: Motor.GetAlpha..... | 59 |
| Tabla 86: Motor.GetNeuronasIntermedias..... | 60 |

| | |
|---|----|
| Tabla 87: Motor.GetTasaAprendizaje | 60 |
| Tabla 88: Motor.GetTiempoFin | 60 |
| Tabla 89: Motor.GetTiempolInicio | 60 |
| Tabla 90: Motor.ImportarPosturas | 60 |
| Tabla 91: Motor.NotificarError | 61 |
| Tabla 92: Motor.NotificarGestoReconocido | 61 |
| Tabla 93: Motor.NotificarPosturaCapturada | 61 |
| Tabla 94: Motor.NotificarPosturaReconocida | 61 |
| Tabla 95: Motor.NotificarPosturasExportadas..... | 61 |
| Tabla 96: Motor.NotificarReconocedorListo..... | 62 |
| Tabla 97: Motor.NuevoFrameColor | 62 |
| Tabla 98: Motor.NuevoFrameEsqueleto..... | 62 |
| Tabla 99: Motor.RecuperarPosturas | 62 |
| Tabla 100: Sensor.InicializarSensor..... | 62 |
| Tabla 101: Sensor.NuevosFrames | 63 |
| Tabla 102: Sensor.NuevoFrameEsqueleto | 63 |
| Tabla 103: Sensor.NuevoFrameColor..... | 63 |
| Tabla 104: InterpreteDatos.Joints2Double | 64 |
| Tabla 105: Capturador.ActivarGuardadoPostura..... | 64 |
| Tabla 106: Capturador.ActivarTemporizador..... | 64 |
| Tabla 107: Capturador.CambiarTiempolInicio | 64 |
| Tabla 108: Capturador.CambiarTiempoLimite..... | 64 |
| Tabla 109: Capturador.CapturarPostura | 65 |
| Tabla 110: Capturador.DesactivarCapturador | 65 |
| Tabla 111: Capturador.DesactivarTemporizador | 65 |
| Tabla 112: Capturador.GetTiempolInicio | 65 |
| Tabla 113: Capturador.GetTiempoFin..... | 65 |
| Tabla 114: Capturador.NuevoFrameEsqueleto..... | 66 |
| Tabla 115: Capturador.NuevoSegundo | 66 |
| Tabla 116: Reconocedor.ActivarReconocedor | 66 |
| Tabla 117: Reconocedor.ActivarReconocimiento | 67 |
| Tabla 118: Reconocedor.AgregarPosturaCapturada..... | 67 |
| Tabla 119: Reconocedor.AnadirPosturaAGesto | 67 |
| Tabla 120: Reconocedor.CambiarAlpha | 67 |
| Tabla 121: Reconocedor.CambiarNeuronasIntermedias | 68 |
| Tabla 122: Reconocedor.CambiarNombrePostura | 68 |
| Tabla 123: Reconocedor.CambiarTasaAprendizaje | 68 |
| Tabla 124: Reconocedor.ComprobarFinGesto | 68 |
| Tabla 125: Reconocedor.ComprobarInicioGesto | 69 |
| Tabla 126: Reconocedor.ComprobarPosturaEsperada | 69 |
| Tabla 127: Reconocedor.DesactivarReconocedor | 69 |
| Tabla 128: Reconocedor.EliminarPostura | 69 |
| Tabla 129: Reconocedor.GetAlpha | 69 |
| Tabla 130: Reconocedor.GetNeuronasIntermedias..... | 70 |
| Tabla 131: Reconocedor.GetTasaAprendizaje | 70 |

| | |
|---|----|
| Tabla 132: Reconocedor.ImportarPosturas | 70 |
| Tabla 133: Reconocedor.InvalidarRed | 70 |
| Tabla 134: Reconocedor.NuevoFrameEsqueleto..... | 71 |
| Tabla 135: Reconocedor.RecuperarPosturas | 71 |
| Tabla 136: RedNeuronas.CambiarAlpha | 71 |
| Tabla 137: RedNeuronas.CambiarNeuronasIntermedias | 71 |
| Tabla 138: RedNeuronas.CambiarTasaAprendizaje | 71 |
| Tabla 139: RedNeuronas.EntrenarRed..... | 72 |
| Tabla 140: RedNeuronas.GetAlpha | 72 |
| Tabla 141: RedNeuronas.GetNeuronasIntermedias | 72 |
| Tabla 142: RedNeuronas.GetTasaAprendizaje..... | 72 |
| Tabla 143: RedNeuronas.InvalidarRed..... | 73 |
| Tabla 144: RedNeuronas.ReconocerPostura | 73 |
| Tabla 145: RedNeuronas.ReconocerPostura | 73 |
| Tabla 146: ManejadorFicheros.AbrirFichero..... | 73 |
| Tabla 147: ManejadorFicheros.escribirPostura | 74 |
| Tabla 148: ManejadorFicheros.exportarPosturas..... | 74 |
| Tabla 149: ManejadorFicheros.importarPosturas | 74 |
| Tabla 150: ManejadorFicheros.leerPostura | 74 |
| Tabla 151: Resumen del resultado de las pruebas | 92 |
| Tabla 152: Posturas comunes | 94 |

Diagramas

| | |
|---|----|
| Diagrama 1: Gantt | 16 |
| Diagrama 2: Casos de uso | 19 |
| Diagrama 3: Actividades UC-01..... | 20 |
| Diagrama 4: Actividades UC-02 | 21 |
| Diagrama 5: Actividades UC-03 | 22 |
| Diagrama 6: Actividades UC-04..... | 23 |
| Diagrama 7: Actividades UC-05 | 25 |
| Diagrama 8: Actividades UC-06 | 27 |
| Diagrama 9: Actividades UC-07 | 28 |
| Diagrama 10: Actividades UC-08..... | 29 |
| Diagrama 11: Actividades UC-09..... | 30 |
| Diagrama 12: Actividades UC-10..... | 31 |
| Diagrama 13: Subsistemas | 36 |
| Diagrama 14: Arquitectura del sistema | 45 |
| Diagrama 15: Clases del sistema | 46 |
| Diagrama 16: Clases de la interfaz de comunicación..... | 47 |
| Diagrama 17: Clases de la interfaz del sensor..... | 48 |
| Diagrama 18: Clases del núcleo del motor..... | 49 |
| Diagrama 19: Clases de la Interfaz de almacenamiento/recuperación | 52 |
| Diagrama 20: Diseño de la clase ManejadorFicheros | 52 |

| | |
|---|----|
| Diagrama 21: Diagrama de secuencia de UC-01, Parte 1..... | 75 |
| Diagrama 22: Diagrama de secuencia de UC-01, Parte 2..... | 76 |
| Diagrama 23: Diagrama de secuencia de UC-01, Parte 3..... | 77 |
| Diagrama 24: Diagrama de secuencia de UC-02 | 78 |
| Diagrama 25: Diagrama de secuencia de UC-03 | 79 |
| Diagrama 26: Diagrama de secuencia de UC-04 | 80 |
| Diagrama 27: Diagrama de secuencia de UC-05 | 81 |
| Diagrama 28: Diagrama de secuencia de UC-06 | 82 |
| Diagrama 29: Diagramas de secuencia de UC-07..... | 83 |
| Diagrama 30: Diagrama de secuencia de UC-08, Parte 1..... | 84 |
| Diagrama 31: Diagrama de secuencia de UC-08, Parte 2..... | 85 |
| Diagrama 32: Diagrama de secuencia de UC-09 | 86 |
| Diagrama 33: Diagrama de secuencia de UC-10 | 87 |

Ilustraciones

| | |
|---|----|
| Ilustración 1: Deslizamiento a la derecha | 11 |
| Ilustración 2: Golpeo de tenis | 12 |
| Ilustración 3: Esquema de un perceptrón multicapa | 13 |

1. Introducción

a. Motivación

En la actualidad se está viviendo una evolución en la forma en la que los seres humanos interactuamos con los computadores. Las interfaces que se basaban en teclado y ratón como los PC'S, las que se basaban en únicamente teclados físicos como los teléfonos móviles y las interfaces que se basaban en mandos como las usadas por reproductores de video, televisiones y consolas, están comenzando a cambiar.

En los últimos años, debido al auge de los Tablet PC como el Ipad de Apple, o los distintos modelos con Android en su interior, a los Smartphone como el Samsung Galaxy SIII o el HTC One X y consolas como la Nintendo Wii, las interfaces anteriormente nombradas empiezan a perder adeptos en favor de nuevas formas de interactuar con sus dispositivos. Interfaces que involucran una participación mucho mas activa del usuario (como el mando WiiMote de Wii ó el sensor Kinect de Xbox 360 y PC) ó que no requieren dispositivos de entrada externos (como los teclados táctiles de los Smartphone y Tablets actuales).

De este nuevo tipo de interfaces, las más interesantes son las que involucran al cuerpo del usuario, puesto que abre muchas posibilidades inexplorables con dispositivos de entrada externos al mismo. Ahora el usuario puede usar su propia mano para dirigir el puntero por la pantalla o simplemente tocar una parte de esta, lo cual hace mucho más rápido e intuitivo el uso de un cursor en pantalla, pero también crea nuevos retos y problemas.

En el caso que nos ocupa, el sensor Kinect provee una interfaz de reconocimiento de 20 puntos distintos del cuerpo. El dispositivo es capaz por si solo de dar a una frecuencia de 30 hercios una posición aproximada de estos 20 puntos con un ligero margen de error.

Como salta a la vista esto permitiría a los usuarios una nueva forma de interactuar con sus programas, podrían abrir el reproductor de video simplemente levantando el brazo derecho, podrían “desplazar” literalmente el contenido que muestra su pantalla con un simple gesto de sus manos, etc.

Además, como veremos a continuación, una interfaz que permita reconocer los gestos del cuerpo, puede tener muchas aplicaciones ajenas a las interfaces de usuario de aplicaciones. Puede ser útil en campos como el tratamiento médico o el entrenamiento de deportistas. Incluso si el sensor Kinect mejorara su precisión algo más, podría abordarse el desarrollo de un interprete de lenguaje de signos a lenguaje escrito, lo cual facilitaría la vida de personas con discapacidades que les impida comunicarse de forma hablada.

El mayor problema que presenta la interfaz de Kinect es que actualmente no existe ningún motor que permita al usuario definir sus propios movimientos en tiempo de ejecución, personalizando así los gestos que podrá reconocer el software. Este impedimento hace que cada vez que se desarrolla una aplicación que utilice como entrada los datos recabados por el sensor Kinect, tengan que estudiarse primero los movimientos que tendrá que reconocer el software, para después codificar a mano los mecanismos necesarios para reconocerlos durante la ejecución del mismo.

Por este motivo, en el presente Trabajo de Fin de Grado trataré de abordar dicho problema para buscar una solución basada en el uso de las redes de neuronas, que son una rama de la inteligencia artificial que ya ha sido probada con éxito anteriormente en el reconocimiento de patrones.

b. Objetivos

El principal objetivo del presente Trabajo de Fin de Grado es realizar un sistema de captura y reconocimiento de gestos que permita al usuario definir sus propias posturas, grabarlas en tiempo de ejecución y luego ser reconocidas por el sistema en tiempo real.

-El sistema permitirá al usuario definir sus propios gestos y posturas en tiempo de ejecución.

-El sistema reconocerá las posturas y gestos que el usuario haya definido.

-El sistema contará con mecanismos que permitan personalizar la interfaz de usuario así como incluir el sistema completo como un componente en aplicaciones mayores.

Este sistema podría ser útil en muchas y diversas situaciones como:

-Deportes:

- **Iniciación y mejora en técnica de deportistas:** Un deportista experimentado (un entrenador por ejemplo) podría grabarse utilizando el sistema realizando el movimiento técnico en el que quiere entrenar a sus jugadores (por ejemplo una volea de tenis ó un “slapshot” en hockey) y podría utilizar el sistema como apoyo a la hora de decidir si un jugador está realizando correctamente el gesto si tiene que entrenar a muchos jugadores a la vez o de manera que los jugadores puedan entrenar la técnica sin la presencia del entrenador.

-Desarrollo de software

- **Modulo básico de aplicaciones mayores:** El sistema podría servir como un módulo más de un proyecto mayor, provocando las siguientes ventajas:
 - **Personalización de interfaces de usuario:** Utilizando la captura en tiempo de ejecución el usuario podría cambiar los gestos definidos para interactuar con el programa, lo cual sería muy útil en aplicaciones como los videojuegos o aplicaciones que puedan proveer una interfaz de usuario basada en Kinect.
 - **Facilita la captura de gestos:** Con el motor se podrían capturar los gestos realizándolos directamente frente al sensor, en vez de tener que realizar un estudio sobre cada gesto en concreto y después codificar un patrón de reconocimiento para el mismo.

-Medicina:

- **Tratamiento fisioterapéutico:** Un fisioterapeuta, sin ningún tipo de conocimiento sobre programación podría capturarse a si mismo realizando un ejercicio de fisioterapia (estiramientos o ejercicios sencillos de fuerza) y un

paciente suyo podría utilizar el sistema para saber si está realizando el ejercicio correctamente sin necesidad de que el fisioterapeuta esté delante.

- **Sistema básico de comunicación para personas discapacitadas:** Si una persona muda tiene parálisis en los brazos, no puede comunicarse mediante el lenguaje de signos, el sistema podría dotar de significado a gestos que la persona pueda realizar con otras partes del cuerpo para comunicarse.

2. Estado del arte

Actualmente, existen pocos sistemas que permitan al usuario definir sus propias posturas y gestos en tiempo de ejecución, por tanto es necesario analizar las líneas de desarrollo existentes para tener una orientación básica sobre los caminos que han tomado otros antes y así saber que líneas de investigación pueden no estar explotadas y tener potencial así como extraer las principales ventajas e inconvenientes de los diferentes sistemas de reconocimiento de gestos que se usan actualmente.

a. Análisis de sistemas similares

Actualmente se encuentran tres corrientes principales en los sistemas de reconocimiento de gestos y posturas mediante Kinect, el reconocimiento basado en coordenadas, los basados en modelos ocultos de Markov y los que utilizan un algoritmo de “Dynamic Time Warping”:

i. Reconocimiento basado en coordenadas

El primer tipo de reconocimiento del cual he podido obtener información es también el más básico. Este se basa en analizar los gestos que se van a usar en el sistema y codificarlos directamente realizando comparaciones entre las coordenadas del cuerpo para verificar que están cumpliendo las condiciones necesarias para cada gesto. Así este tipo de sistemas pueden reconocer con un nivel de efectividad muy alto gestos sencillos, pero son totalmente ineficaces si el gesto/postura tiene una complejidad mayor, que impide definirlo basándose en coordenadas.

Por ejemplo, este sistema permite reconocer bien movimientos como “Desplazar la mano derecha desde el hombro hasta extender el brazo”. En dicho ejemplo empezarían comprobando que en el inicio del gesto la coordenada X e Y de la mano derecha se encuentran muy próximas al hombro derecho y que finalmente la coordenada Y de mano y hombro derechos serían la misma, lo que significaría que están alineadas de forma horizontal, la coordenada X de la mano derecha debería ser mayor a la que inicial.

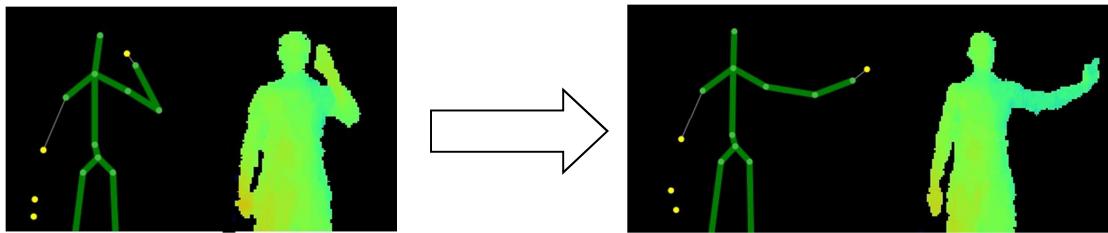


Ilustración 1: Deslizamiento a la derecha

Como se puede ver en ese ejemplo definir el gesto por coordenadas ha sido fácil, pero si utilizamos un gesto como “Realizar un golpe de revés de tenis con las dos manos” la definición del gesto en coordenadas se vuelve imposible ya que hay que sumar a la complejidad intrínseca del gesto que cada persona realiza dicho gesto con pequeñas variaciones.

Este tipo de reconocimiento además plantea otro grave problema, y es que todos los gestos han de ser codificados al desarrollar la aplicación, impidiendo al usuario definir sus propios gestos y posturas, con lo que se impide personalizar la aplicación.

ii. Reconocimiento basado en modelos ocultos de Markov.

Por otra parte, para solventar dichos problemas se utiliza otra rama principal en el reconocimiento de gestos con Kinect, el reconocimiento basado en modelos ocultos de Markov. Estos sistemas basan el reconocimiento en un modelo estadístico que dada la información de entrada trata de extraer los parámetros ocultos que podrían componer el patrón de entrada. Esto los hace muy eficientes para realizar después análisis de patrones lo que los hace idóneos para el reconocimiento de gestos y posturas, ya que estos no son más que sucesiones de coordenadas siguiendo unos determinados patrones, diferentes para cada gesto.

Este tipo de sistemas, como ha quedado claro dejado claro en el párrafo anterior, tienen claras ventajas sobre el reconocimiento basado en coordenadas. El realizar el reconocimiento mediante un modelo estadístico implica que se podría realizar una generalización del modelo que debería funcionar para cualquier usuario que realice el gesto, aunque cometiera un pequeño error si se cuenta con un conjunto de entrada de datos representativo del gesto a reconocer.

En el otro lado de la balanza tenemos que para la codificación de estos modelos son necesarios conocimientos estadísticos. Además es necesario un conjunto de datos de entrada, de los cuales depende por completo la efectividad del modelo, ya que un conjunto de datos mal elegido podría hacer que los parámetros extraídos por el modelo no fueran los reales. Y por último seguimos teniendo heredado uno de los problemas del reconocimiento basado en coordenadas, estos modelos no pueden generalizarse para cualquier gesto, si no que deben ser codificados de forma independiente para gesto o postura a reconocer.

Para finalizar con el análisis de este sistema volvemos a explicar los dos gestos nombrados en el reconocimiento basado en coordenadas, en el caso de desplazar la mano desde el hombro hasta extender el brazo habría que definir un modelo oculto de Markov el cual recibiría como entradas las posiciones X e Y del recorrido de la mano y del codo en todo

momento, después se entrenaría dicho modelo con varias personas distintas y finalmente al volver a insertar dicha serie de datos en un nuevo caso el sistema debería reconocer con éxito el gesto. En el caso del revés del tenis se crearía otro modelo oculto de Markov que recibiría un patrón de entrada distinto para entrenarlo, ahora se le pasarían las coordenadas X, Y e Z de las manos, hombros y codos, las cuales evidentemente serán muy diferentes a las recibidas en el modelo para reconocer el deslizamiento de la mano hacia la derecha.



Ilustración 2: Golpeo de tenis

iii. Dynamic Time Warping

Esta técnica permite reconocer patrones lineales en el tiempo que se produzcan a velocidades diferentes a la velocidad en la que fueron capturados. Estos sistemas tienen la principal ventaja de que permiten un porcentaje de acierto muy alto aun variando la velocidad de realización del gesto. A diferencia del modelo anterior, usando esta técnica se pueden definir en tiempo de ejecución las posturas y los gestos, grabándose a uno mismo y después realizando la validación al realizar el gesto directamente sobre el software. El principal problema que tiene este modelo es que aunque abstrae muy bien la velocidad de ejecución del gesto, tiene mas problemas con las desviaciones sobre el mismo, y dado que lo interesante es conseguir un reconocimiento de patrones lo mas universal posible, esta alternativa pierde algo de interés.

b. Sistemas poco explorados

Por otra parte hay un camino que no ha sido demasiado investigado en el reconocimiento de posturas y gestos mediante Kinect, que es el reconocimiento basado en redes de neuronas artificiales. Las redes de neuronas artificiales son una rama de la inteligencia artificial que como su nombre indica trata de imitar el funcionamiento de las neuronas. Esta rama ha sido ampliamente usada en la inteligencia artificial en los campos de clasificación automática de objetos y detección de patrones, lo cual las convierte en un sistema idóneo para realizar un sistema de reconocimiento de gestos.

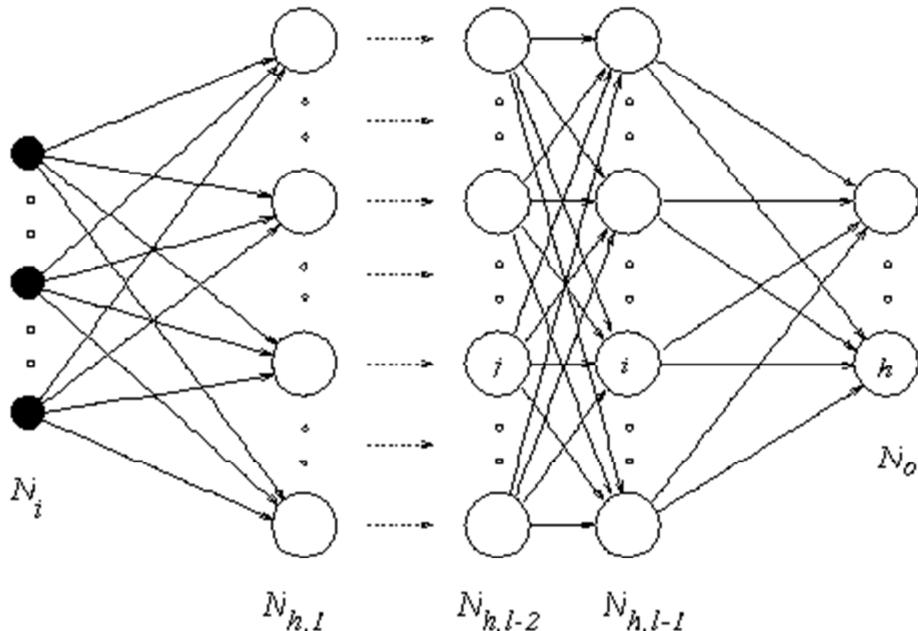


Ilustración 3: Esquema de un perceptrón multicapa

Dentro de las redes de neuronas artificiales hay varios sistemas muy conocidos, siendo el más interesante para el propósito que nos ocupa el perceptrón multicapa, considerado un clasificador universal. Este tipo de red de neuronas es el más interesante porque se trata de un algoritmo de aprendizaje supervisado, esto implica que para realizar el aprendizaje hay que definir en los datos de entrada tanto los parámetros conocidos como el resultado esperado.

Dicho sistema funciona de la siguiente manera: Recibe un conjunto de datos numéricos de entrada. Estos datos contendrán las características que se han observado del evento a clasificar (en el caso del reconocimiento de posturas, las coordenadas de cada articulación que nos proporciona Kinect) y asociado a cada conjunto de características, la clase a la que pertenece el objeto. Si se pretende por ejemplo distinguir entre dos posturas, el patrón de entrada que recibirá la red de neuronas contendrá las coordenadas de cada articulación en cada postura, y asociado a cada conjunto de coordenadas, un conjunto de datos entre -1 y 1 que indicarán a qué clase pertenece. Después la red va propagando las entradas, multiplicándola por los pesos de cada conexión hacia las neuronas de la capa siguiente y al final en las neuronas de salida se obtiene un resultado que está entre -1 y 1. Si el resultado es el esperado (coincide con el patrón de entrada) se pasa al siguiente conjunto de entrada. Si no es el esperado se reajustan los pesos de la red de manera automática desde la

última capa hasta la primera y se prueba de nuevo con el conjunto de entrada siguiente. Este proceso se repite hasta que el total de errores cometidos en el entrenamiento está por debajo de un límite pre establecido al empezar a entrenar la red. Una vez entrenada para realizar el reconocimiento se vuelven a pasar un conjunto nuevo de datos de entrada, pero esta vez sin salida asociada, puesto que la misma la calculará la red de manera similar a la que lo hacía en el entrenamiento.

Dentro del campo de reconocimiento de patrones, este tipo de red de neuronas lleva utilizándose con éxito desde los años 60, pero su uso no está muy extendido en los sistemas de detección de gestos con Kinect.

La principal ventaja del perceptrón multicapa es su capacidad para clasificar objetos en base a los patrones de entrada dados con lo que podría ser una opción muy a tener en cuenta en el reconocimiento de gestos con Kinect, ya que con unos datos de entrada lo suficientemente representativos debería ser capaz de clasificar las posturas que el usuario defina sin necesidad de tener que codificarlos a mano.

c. Kinect DTW Gesture Recognition

El principal motor disponible en código abierto de manera que cualquiera pueda usarlo es el Kinect DTW Gesture Recognition alojado en codeplex.com. El funcionamiento es similar al del presente TFG, el motor permite al usuario grabar gestos y luego es capaz de reconocer los gestos grabados por el usuario.

Para ello se basa en el algoritmo de Dynamic Time Warping mencionado anteriormente lo que le permite reconocer gestos aunque sea a una velocidad diferente a la de la grabación original.

La principal ventaja de este sistema es la facilidad para grabar los gestos, simplemente se pulsa el botón de capturar gesto y se realiza el gesto delante del sensor Kinect. Por otra parte no está exento de inconvenientes. Aunque el motor lleva existiendo desde la aparición de la beta del SDK de Kinect para Windows, lleva sin actualizarse más de un año, lo cual hace pensar que el proyecto esté descontinuado ó que no haya despertado demasiado interés. Esto también puede deberse a que el motor solo es capaz de reconocer gestos en dos dimensiones, y aun así no lo hace con un porcentaje de acierto demasiado elevado.

Para analizar si el sistema a desarrollar es interesante frente a dicho competidor he realizado el siguiente análisis D.A.F.O. (Debilidades, Amenazas, Fortalezas, Oportunidades)

| Debilidades: | Fortalezas: |
|---|--|
| <ul style="list-style-type: none">- No puede reconocer gestos ni posturas realizadas con un ángulo respecto a la cámara diferente al que fueron grabados.- La captura de gestos no se realiza de forma automática, si no que se ha de hacer mediante la captura de posturas sucesivas. | <ul style="list-style-type: none">- Puede reconocer gestos en 3 dimensiones- El reconocimiento de posturas y de gestos es casi universal, permite que un usuario sea el que capture el gesto y lo reconocería aun en la ejecución de otro usuario.- El prototipo es fácil de modificar y se puede personalizar la interfaz del mismo |

| Amenazas: | Oportunidades: |
|---|--|
| <ul style="list-style-type: none"> - El sistema de codeplex existe desde la publicación de la beta del SDK de Kinect para Windows, lo cual lo ha dotado de bastante fama | <ul style="list-style-type: none"> - No hay alternativas de código abierto que permitan capturar y reconocer gestos en tres dimensiones con un porcentaje de acierto alto. - El uso de Kinect en Windows todavía no está muy extendido con lo que se puede mejorar el prototipo tras lanzarlo aprovechando los pocos competidores que hay. |

Tabla 1: Análisis D.A.F.O.

3. Planificación y presupuesto

a. Planificación

Para la realización del proyecto contaré con los meses desde junio hasta agosto, ambos incluidos, con una capacidad de trabajo de tres horas diarias durante 5 días a la semana. Esto da en total una capacidad de trabajo de 15 horas semanales.

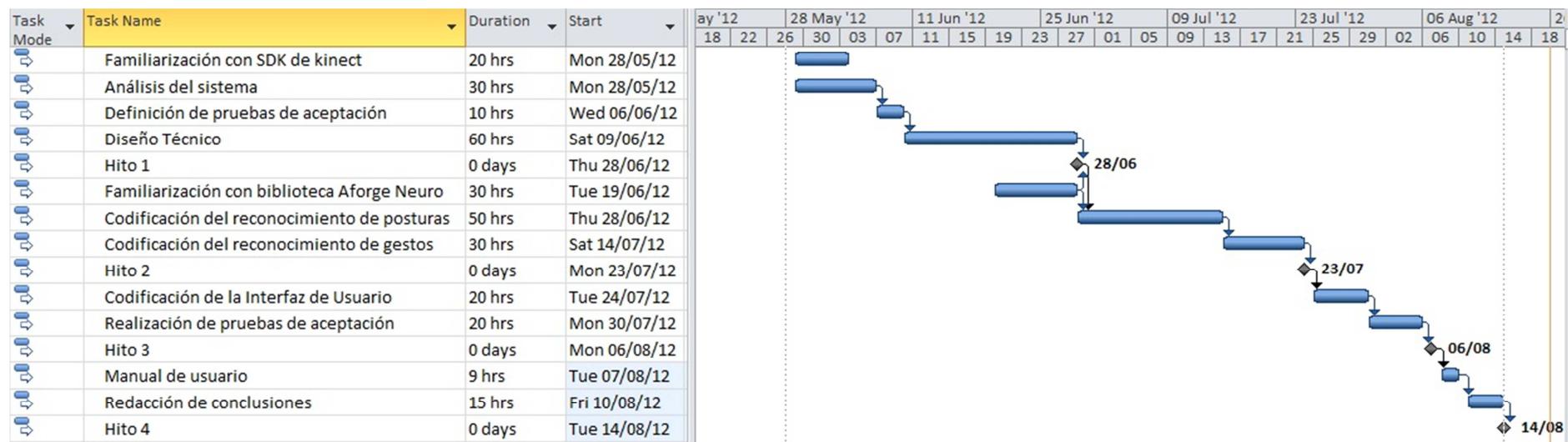


Diagrama 1: Gantt

b. Presupuesto

Para la realización del sistema se contará principalmente con dos tipos de gastos, los gastos en amortizaciones y los gastos en personal. En el primer grupo, como se detalla en la tabla siguiente, los gastos son la adquisición del material necesario para la realización del TFG, esto implica un equipo portátil con Windows instalado y un sensor Kinect.

| Concepto | Precio | Tiempo de vida útil | Tiempo de uso en el TFG | Precio para el TFG |
|--|--------|---------------------|-------------------------|--------------------|
| Ordenador portátil Samsung RV511 (Licencia Windows 7 home incluida) | 450€ | 60 meses | 4 meses | 30€ |
| Sensor Kinect | 150€ | 60 meses | 4 meses | 10€ |

Tabla 2: Gastos de amortizaciones

En cuanto al gasto en personal se incluyen los siguientes participantes en el proyecto junto al papel que desempeñan en el mismo y el coste asociado:

| Personal | Rol en el TFG / Categoría | €/Hombre mes | Dedicación al TFG (hombres mes) | Precio para el TFG |
|----------------------|-------------------------------------|--------------|---------------------------------|--------------------|
| Javier Robledo Zarco | Desarrollador principal / Ingeniero | 2.694,39€ | 2.24 (294 horas) | 6035,43€ |
| Ángel García Crespo | Tutor / Ingeniero Senior | 4.289,54 € | 0.06 (8 horas) | 257,37€ |

Tabla 3: Gastos de personal

Nota: Hombre mes = 131.25 horas.

Con lo cual, el presupuesto total será:

| Concepto | Precio |
|----------------|---------------|
| Personal | 6.293€ |
| Amortizaciones | 40€ |
| Total | 7.599€ |

Tabla 4: Resumen de costes

4. Análisis

a. Definición del alcance del sistema

Tras considerar las líneas de investigación tomadas con anterioridad por otros desarrolladores y teniendo en cuenta que el objetivo principal del presente proyecto era el prototipado de una biblioteca de clases que utilizaran Kinect, las principales funcionalidades que he decidido tener en cuenta para la aplicación son las siguientes:

-**Reconocimiento universal de gestos** mediante un sistema basado en un **perceptrón multicapa**.

-Capacidad para **capturar gestos del usuario**, lo que permitirá **personalizar** las aplicaciones que utilicen la librería.

-Capacidad para notificar cuando se reconoce una postura o un gesto, lo que permitiría desarrollar interfaces de usuario que utilicen la biblioteca sin necesidad de editar el código fuente de la misma.

-Capacidad para **guardar y cargar el estado del motor**, con objeto de poder salvar las posturas y gestos con los que ha sido entrenado para poder seguir usándolas después y de manera que los usuarios puedan intercambiar sus posturas guardadas.

b. Definición de casos de uso

Al ser el principal objetivo de la librería el servir de componente para proyectos mayores, la extracción de requisitos se dificulta, puesto que el interesado en desarrollar el proyecto es el mismo desarrollador y no hay un cliente del cual extraer los requisitos, por tanto en este caso serán de gran ayuda los casos de uso.

Se han identificado diez casos de uso principales, los cuales se explicarán detalladamente, indicando los pasos que realizará el usuario y adjunto a la descripción de los mismos se proporcionan los diagramas de actividad que contemplan tanto los pasos que deberá dar el sistema como los del usuario en cada caso de uso.

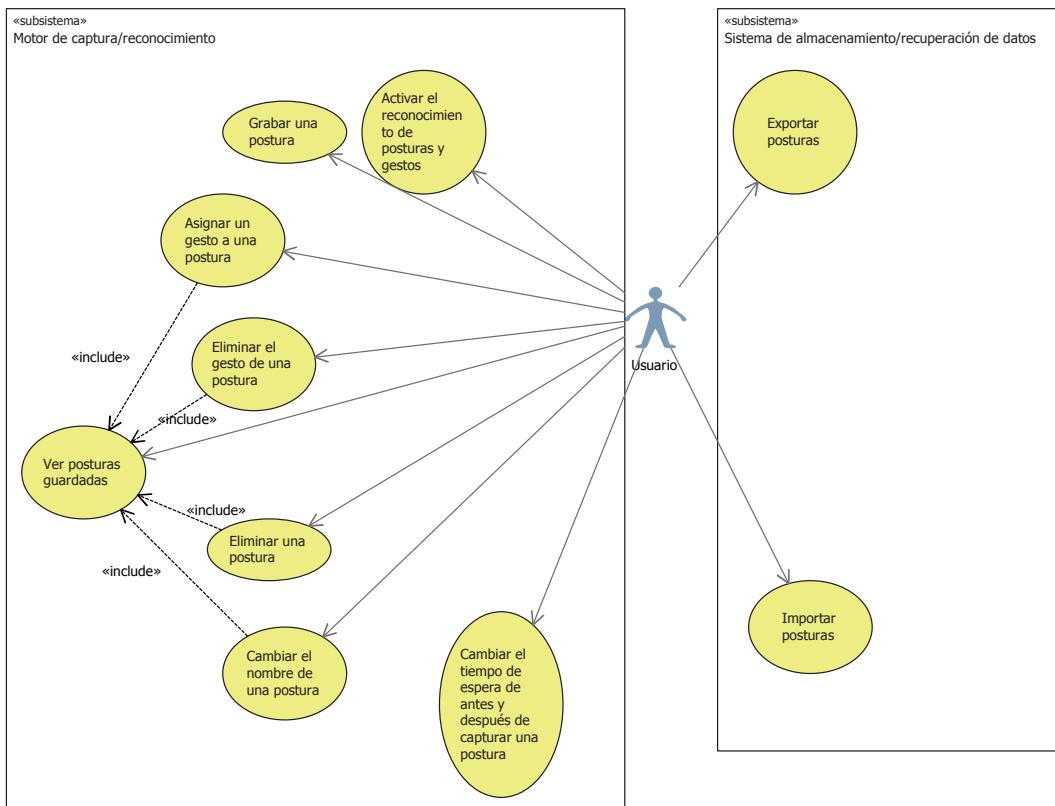


Diagrama 2: Casos de uso

i. **UC-01-Capturar una postura:**

Especificación

| | |
|---|--|
| Identificador: | UC-01 |
| Nombre: | Capturar una postura |
| Precondiciones: | -N/A |
| Descripción del escenario: | 1.- Iniciar la captura de posturas 2.- Colocarse en una posición en la que se reconozca el esqueleto 3.- Colocarse en la postura deseada |
| Postcondiciones: | - La postura se almacena y el sistema muestra un mensaje de confirmación |
| Errores posibles: | 1.- El sistema no es capaz de reconocer al usuario antes de 30 segundos |
| Escenario alternativo (Error 1): | El sistema mostrará un mensaje de error |

Tabla 5: UC-01-Capturar una postura

Diagrama de actividades

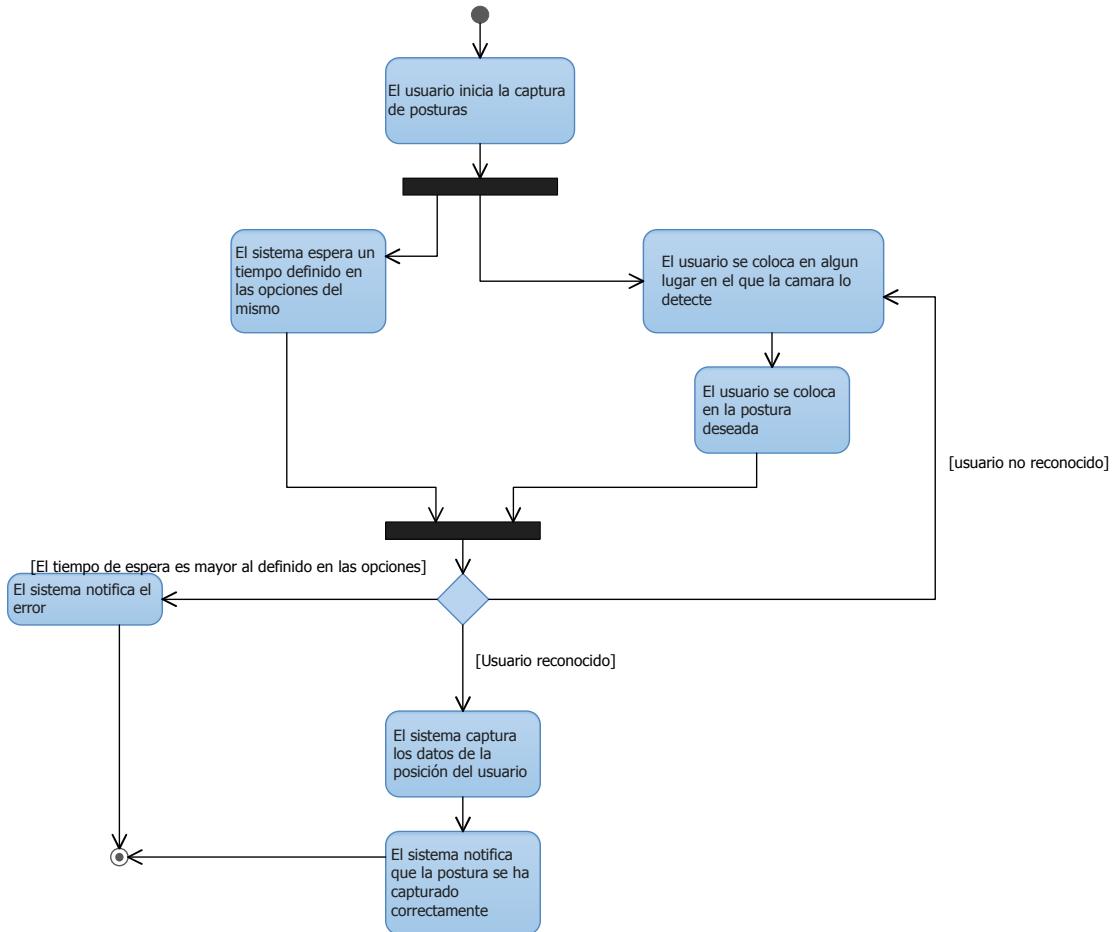


Diagrama 3: Actividades UC-01

ii. UC-02-Ver posturas guardadas

Especificación

| | |
|---|--|
| Identificador: | UC-02 |
| Nombre: | Reconocer una postura |
| Precondiciones: | N/A |
| Descripción del escenario: | 1.- El usuario pulsa el botón de ver posturas guardadas |
| Postcondiciones: | -El sistema muestra una lista con los nombres de las posturas que ha capturado |
| Errores posibles: | 1.- El sistema no ha capturado ninguna postura con anterioridad -El sistema muestra una lista vacía |
| Escenario alternativo (Error 1): | |

Tabla 6: UC-02-Ver posturas guardadas

Diagrama de actividades

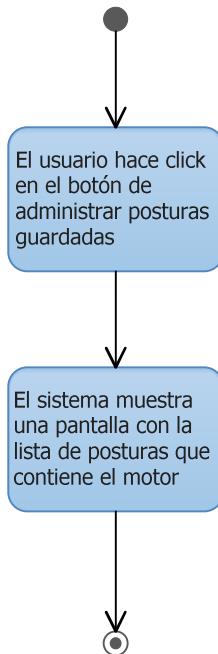


Diagrama 4: Actividades UC-02

iii. UC-03-Cambiar el nombre de una postura

Especificación

| | |
|---|---|
| Identificador: | UC-03 |
| Nombre: | Cambiar el nombre de una postura |
| Precondiciones: | -El motor tiene al menos una postura almacenada |
| Descripción del escenario: | 1.- Caso de uso UC-02 2.- Seleccionar una postura de la lista 3.- Pulsar el botón de cambiar nombre 4.- Escribir el nuevo nombre de la postura 5.- Pulsar continuar |
| Postcondiciones: | -El nombre de la postura seleccionada se sustituye por el nuevo |
| Errores posibles: | 1.- El nuevo nombre que introduce el usuario ya está asignado a otra postura. |
| Escenario alternativo (Error 1): | 4b.- El sistema notifica que ya hay una postura con ese nombre. |

Tabla 7: UC-03-Cambiar nombre de una postura

Diagrama de actividades

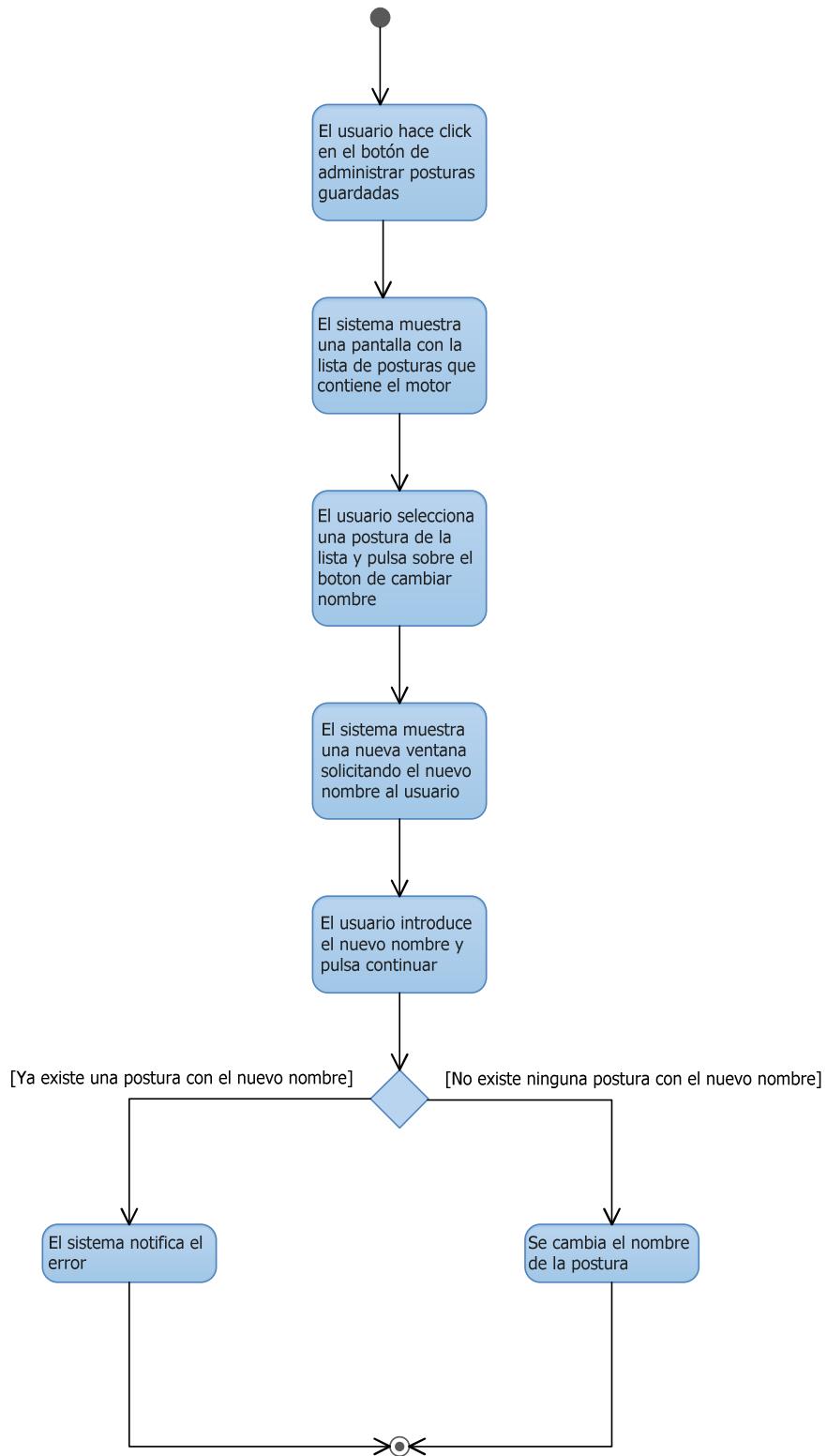


Diagrama 5: Actividades UC-03

iv. **UC-04-Eliminar una postura**

Especificación:

| | |
|-----------------------------------|---|
| Identificador: | UC-04 |
| Nombre: | Eliminar una postura |
| Precondiciones: | -El motor tiene al menos una postura almacenada |
| Descripción del escenario: | 1.- Caso de uso UC-02 2.- Seleccionar una postura de la lista 3.- Pulsar sobre el botón de eliminar postura |
| Postcondiciones: | - Se elimina la postura de la lista de posturas del motor |
| Errores posibles: | N/A |

Tabla 8: UC-04-Eliminar una postura

Diagrama de actividades

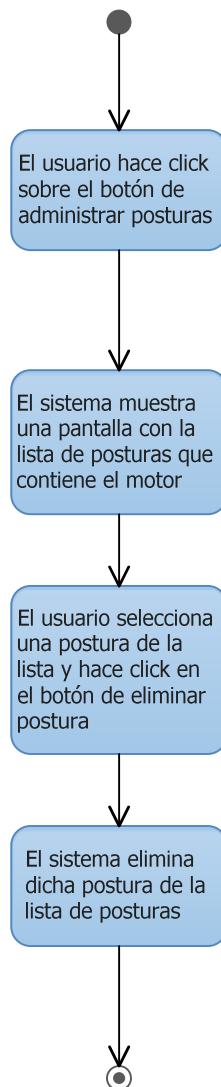


Diagrama 6: Actividades UC-04

v. **UC-05-Asignar un gesto a una postura**

Especificación:

| | |
|-----------------------------------|---|
| Identificador: | UC-05 |
| Nombre: | Asignar un gesto a una postura |
| Precondiciones: | -El motor tiene almacenada al menos una postura |
| Descripción del escenario: | 1.- Caso de uso UC-02 2.- Seleccionar una postura de la lista 3.- Pulsar el botón de asignar gesto 4.- Introducir la posición de la postura dentro del gesto, el nombre del gesto y si la postura es la última o no. 5.- Pulsar el botón de continuar |
| Postcondiciones: | - Se asocian los datos introducidos a la postura seleccionada |
| Errores posibles: | N/A |

Tabla 9: UC-05-Asignar un gesto a una postura

Diagrama de actividades:

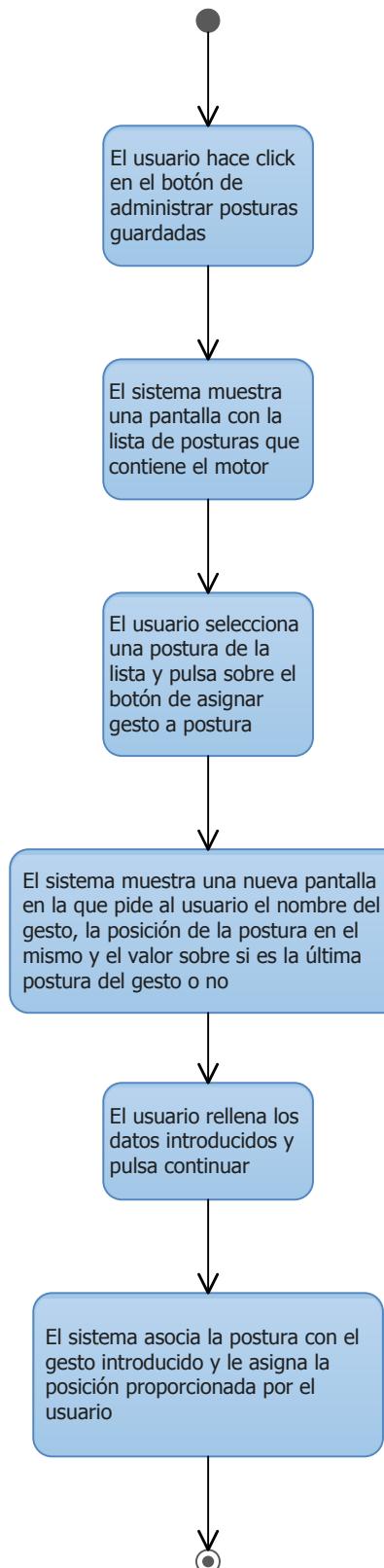


Diagrama 7: Actividades UC-05

vi. **UC-06-Eliminar el gesto de una postura**

Especificación:

| | |
|---|---|
| Identificador: | UC-06 |
| Nombre: | Eliminar el gesto de una postura |
| Precondiciones: | -N/A |
| Descripción del escenario: | 1.- Caso de uso UC-02 2.- Seleccionar una postura de la lista 3.- Pulsar el botón de eliminar gesto |
| Postcondiciones: | - El motor elimina los datos de gesto asociados a la postura |
| Errores posibles: | 1.- La postura no tiene datos de gesto asociados |
| Escenario alternativo (Error 1): | -El sistema no hace nada |

Tabla 10: UC-06-Eliminar el gesto de una postura

Diagrama de actividades:

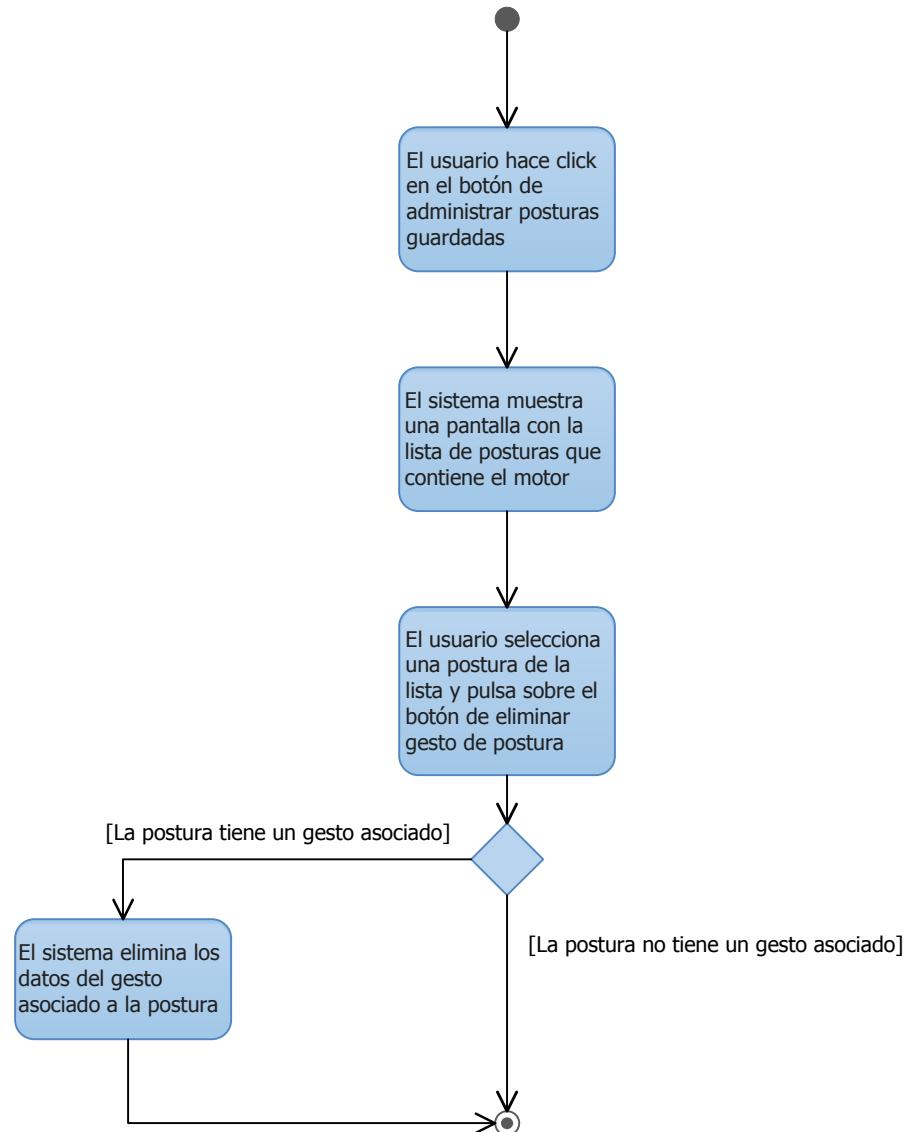


Diagrama 8: Actividades UC-06

vii. UC-07-Cambiar el tiempo de espera del motor

Especificación:

| | |
|-----------------------------------|--|
| Identificador: | UC-07 |
| Nombre: | Cambiar el tiempo de espera del motor |
| Precondiciones: | N/A |
| Descripción del escenario: | 1.- Pulsar el botón de opciones 2.- Introducir los nuevos tiempos de espera de antes y después de capturar posturas 3.- Pulsar el botón de continuar |
| Postcondiciones: | -El sistema almacena los nuevos valores para los tiempos de espera |

| | |
|--------------------------|-----|
| Errores posibles: | N/A |
|--------------------------|-----|

Tabla 11: UC-07-Cambiar el tiempo de espera del motor

Diagrama de actividades:

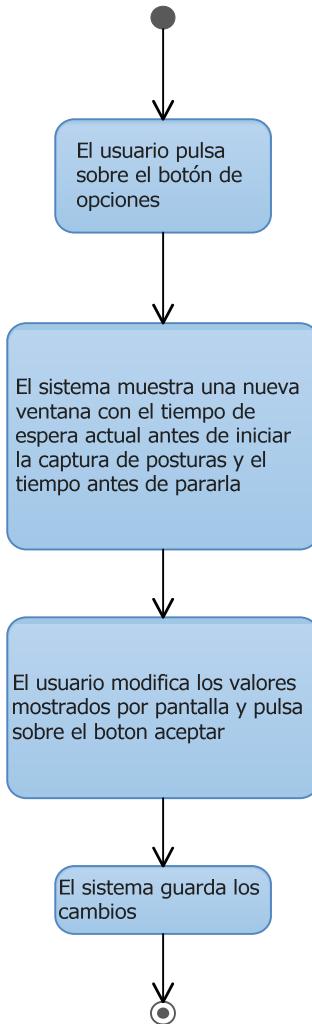


Diagrama 9: Actividades UC-07

viii. UC-08-Activar reconocimiento de posturas y gestos

Especificación:

| | |
|---|---|
| Identificador: | UC-08 |
| Nombre: | Activar reconocimiento de posturas y gestos |
| Precondiciones: | N/A |
| Descripción del escenario: | 1.- Pulsar el botón de activar reconocimiento |
| Postcondiciones: | -El sistema entrena al motor con los datos de las posturas capturadas y lo pasa a modo reconocimiento |
| Errores posibles: | 1.-El sistema no tiene posturas capturadas |
| Escenario alternativo (Error 1): | -El sistema muestra un mensaje de error |

Tabla 12: UC-08-Activar reconocimiento de posturas y gestos

Diagrama de actividades:

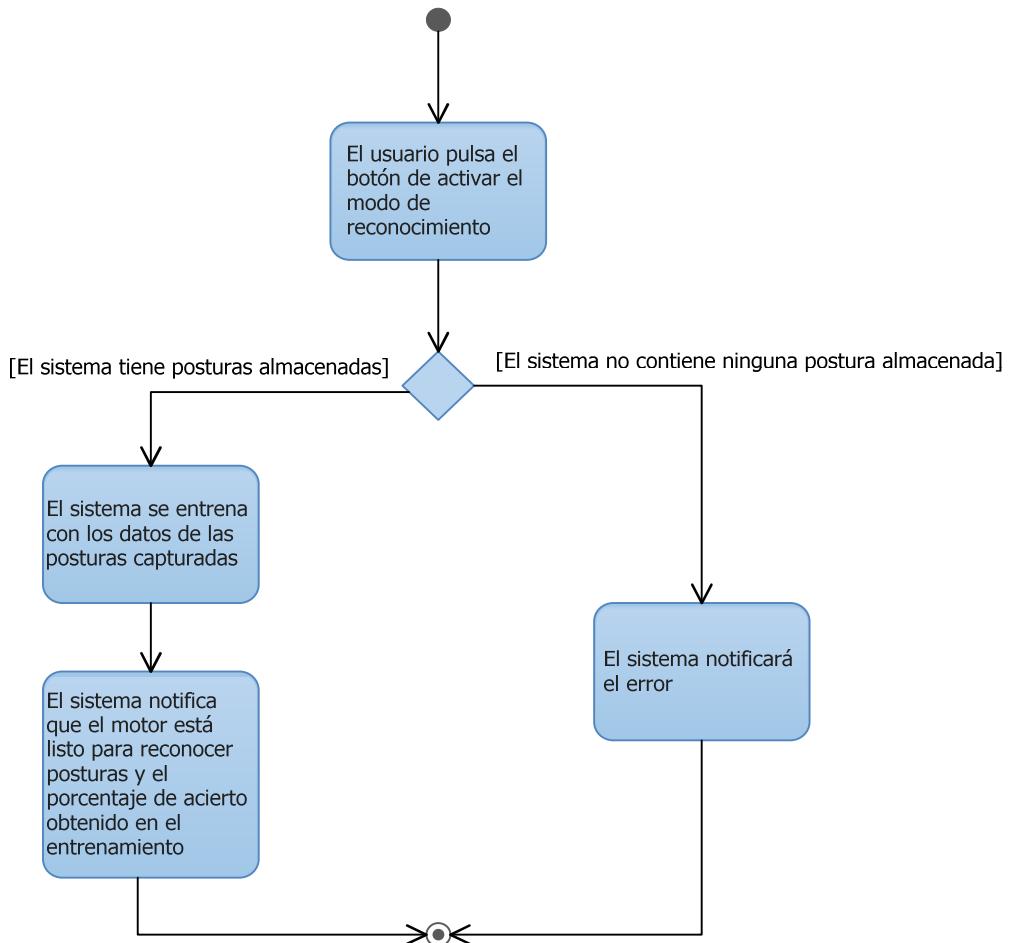


Diagrama 10: Actividades UC-08

ix. UC-09-Exportar posturas

Especificación:

| | |
|---|---|
| Identificador: | UC-09 |
| Nombre: | Exportar posturas |
| Precondiciones: | -El motor tiene almacenada al menos una postura |
| Descripción del escenario: | 1.- Pulsar el botón de exportar posturas 2.- Escribir la ruta del fichero que contendrá las posturas 3.- Pulsar el botón de continuar |
| Postcondiciones: | -El sistema escribe los datos de las posturas en el fichero seleccionado |
| Errores posibles: | 1.-El usuario no dispone de permisos de escritura en la ruta seleccionada |
| Escenario alternativo (Error 1): | -El sistema muestra un mensaje de error |

Tabla 13: UC-09-Exportar posturas

Diagrama de actividades:

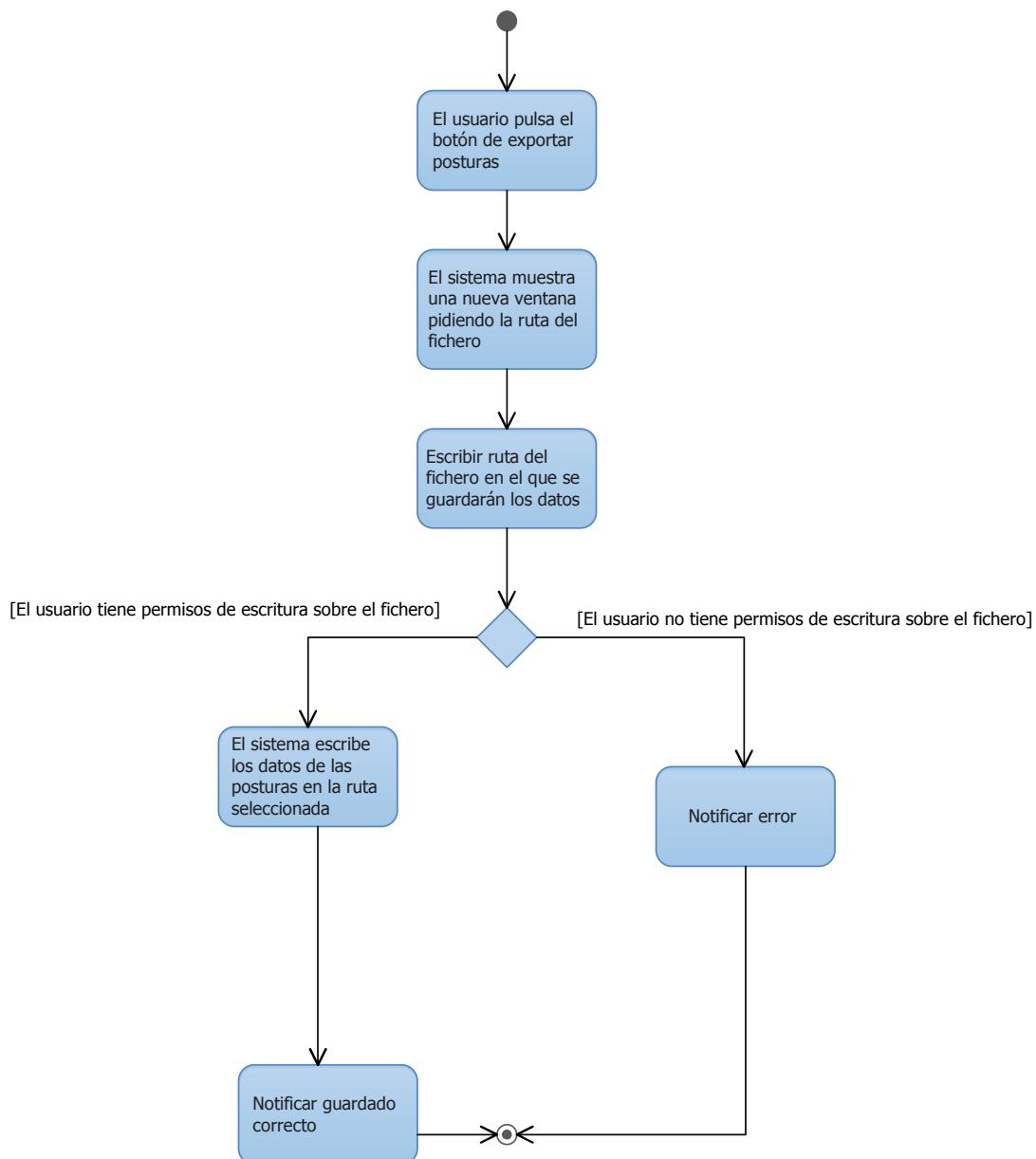


Diagrama 11: Actividades UC-09

x. UC-10-Importar posturas

Especificación:

| | |
|-----------------------------------|---|
| Identificador: | UC-10 |
| Nombre: | Importar posturas |
| Precondiciones: | N/A |
| Descripción del escenario: | 1.- Pulsar el botón de importar posturas 2.- Escribir la ruta del fichero que contendrá las posturas 3.- Pulsar el botón de continuar |
| Postcondiciones: | -El sistema recupera los datos de las posturas almacenadas en el fichero |

| | |
|---|---|
| Errores posibles: | 1.-El usuario no dispone de permisos de lectura en la ruta seleccionada 2.-La ruta introducida no existe |
| Escenario alternativo (Todos los errores): | -El sistema muestra un mensaje de error |

Tabla 14: UC-10-Importar posturas

Diagrama de actividades:

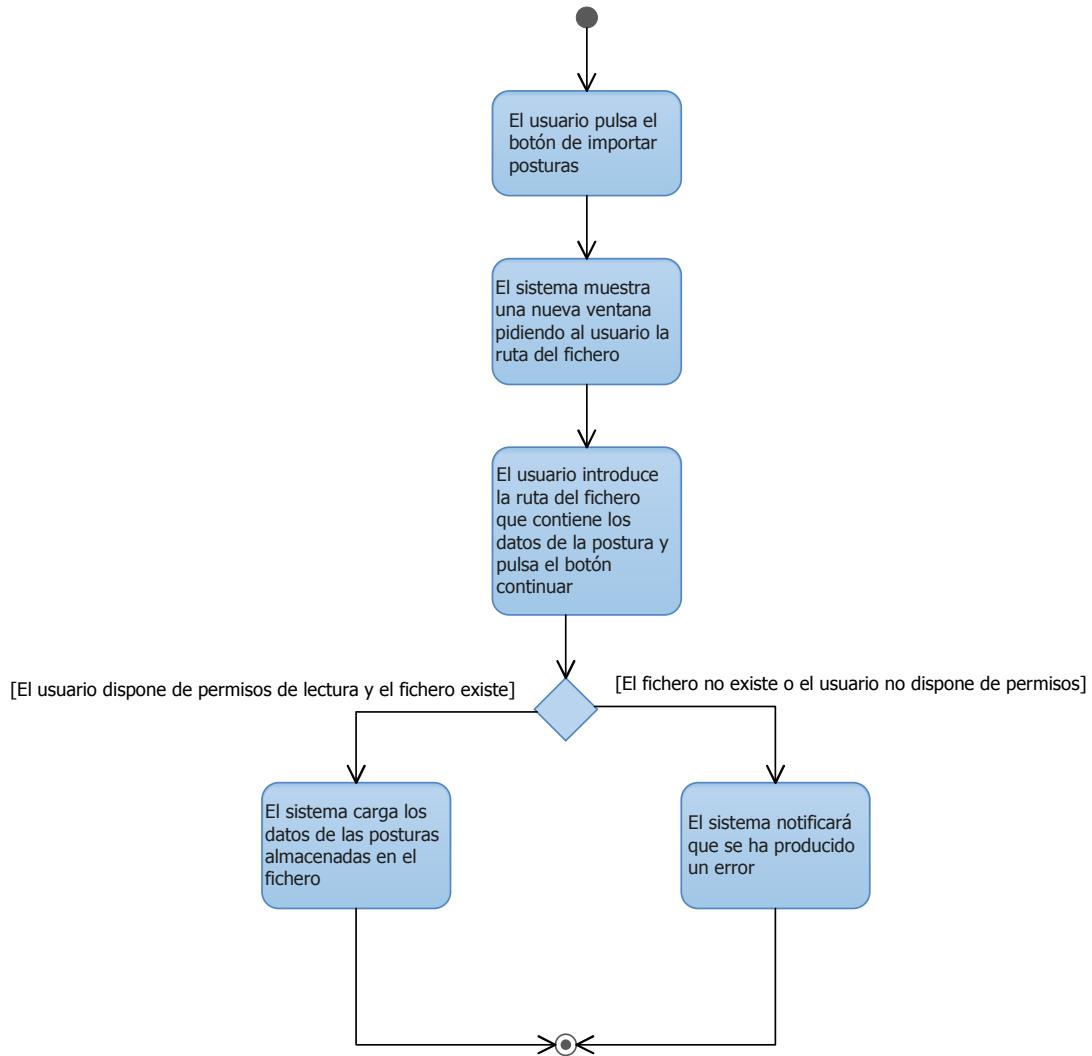


Diagrama 12: Actividades UC-10

c. Extracción de requisitos de usuario

Basándome tanto en el alcance del sistema como en la definición de los casos de uso he realizado la extracción de requisitos de usuario. Los resultados de dicho proceso se muestran en las tablas siguientes, que contienen el identificador, nombre, una breve descripción, el tipo, la necesidad y la prioridad de cada requisito identificado.

i. Requisitos de capacidad

| Identificador | RUC-01 | | | | |
|---------------|--|---|-------|-------------|------|
| Nombre | Captura de posturas | | | | |
| Descripción | Habrá mecanismos que permitan al sistema capturar una postura y almacenarla en memoria principal | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | X | Media | | Baja |

Tabla 15: RUC-01

| Identificador | RUC-02 | | | | |
|---------------|--|---|-------|-------------|------|
| Nombre | Reconocimiento de posturas | | | | |
| Descripción | Una vez que una postura esté almacenada en memoria principal el sistema deberá ser capaz de reconocerla cuando el usuario la realice. | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | X | Media | | Baja |

Tabla 16: RUC-02

| Identificador | RUC-03 | | | | |
|---------------|---|--|-------|-------------|------|
| Nombre | Definir de gestos | | | | |
| Descripción | Habrá mecanismos que permitan al sistema definir un gesto y almacenarlo en memoria principal | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | | Media | X | Baja |

Tabla 17: RUC-03

| Identificador | RUC-04 | | | | |
|---------------|---|---|-------|-------------|------|
| Nombre | Reconocimiento de gestos | | | | |
| Descripción | Tras definir un gesto este deberá ser reconocido por el sistema si el usuario realiza dicho gesto frente al sensor. | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | X | Media | | Baja |

Tabla 18: RUC-04

| Identificador | RUC-05 | | | | |
|---------------|--|--|-------|-------------|------|
| Nombre | Exportado de los datos de entrada del motor | | | | |
| Descripción | Se podrán exportar los datos de entrada del motor a un fichero almacenado en memoria secundaria | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | | Opcional | |
| Prioridad | Alta | | Media | | Baja |

Tabla 19: RUC-05

| Identificador | RUC-06 | | | | |
|---------------|--|--|---|-------------|--|
| Nombre | Importado de los datos de entrada del motor | | | | |
| Descripción | Se podrán importar los datos de entrada del motor desde un fichero almacenado en memoria secundaria | | | | |
| Tipo | Capacidad | | X | Restricción | |

| | | | | | |
|------------------|--------------------|--|-------|-----------------|---|
| Necesidad | Obligatorio | | | Opcional | X |
| Prioridad | Alta | | Media | Baja | X |

Tabla 20: RUC-06

| Identificador | RUC-07 | | | | |
|--------------------|--|---|--------------------|------|---|
| Nombre | Distinción entre posturas parecidas | | | | |
| Descripción | El motor deberá distinguir entre posturas que tengan algunos rasgos similares (ver posturas 1 y 2 en Tabla 153). | | | | |
| Tipo | Capacidad | X | Restricción | | |
| Necesidad | Obligatorio | | Opcional | | X |
| Prioridad | Alta | | Media | Baja | X |

Tabla 21: RUC-07

| Identificador | RUC-08 | | | | |
|--------------------|---|---|--------------------|---|------|
| Nombre | Mostrar las posturas capturadas | | | | |
| Descripción | El sistema contará con mecanismos que permitan al usuario consultar el nombre de las posturas que tiene insertadas | | | | |
| Tipo | Capacidad | X | Restricción | | |
| Necesidad | Obligatorio | X | Opcional | | |
| Prioridad | Alta | | Media | X | Baja |

Tabla 22: RUC- 08

| Identificador | RUC-09 | | | | |
|--------------------|---|---|--------------------|---|------|
| Nombre | Modificar nombre de posturas | | | | |
| Descripción | El sistema permitirá cambiar el nombre de una postura después de capturarla | | | | |
| Tipo | Capacidad | X | Restricción | | |
| Necesidad | Obligatorio | X | Opcional | | |
| Prioridad | Alta | | Media | X | Baja |

Tabla 23: RUC- 09

| Identificador | RUC-10 | | | | |
|--------------------|--|---|--------------------|---|------|
| Nombre | Autoguardado de posturas | | | | |
| Descripción | El sistema guardará automáticamente los datos de las posturas en un fichero de memoria secundaria al activar el reconocimiento. | | | | |
| Tipo | Capacidad | X | Restricción | | |
| Necesidad | Obligatorio | | Opcional | | X |
| Prioridad | Alta | | Media | X | Baja |

Tabla 24: RUC- 10

| Identificador | RUC-11 | | | | |
|--------------------|---|---|--------------------|---|------|
| Nombre | Carga automática de posturas | | | | |
| Descripción | El sistema cargará automáticamente los datos de posturas del fichero en el que se guardan automáticamente si este existe. | | | | |
| Tipo | Capacidad | X | Restricción | | |
| Necesidad | Obligatorio | | Opcional | | X |
| Prioridad | Alta | | Media | X | Baja |

Tabla 25: RUC- 11

| Identificador | RUC-12 | | | | |
|---------------|----------------------|--|--|--|--|
| Nombre | Opciones del sistema | | | | |

| | | | | | |
|--------------------|---|--|-------|--------------------|------|
| Descripción | El sistema permitirá personalizar los valores del tiempo de espera de antes de capturar una postura y antes de parar de capturar. | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | | Media | | Baja |

Tabla 26: RUC- 12

| Identificador | RUC-13 | | | | |
|--------------------|--|---|-------|--------------------|------|
| Nombre | Eliminar posturas | | | | |
| Descripción | El sistema permitirá al usuario eliminar posturas capturadas con anterioridad | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | X | Media | | Baja |

Tabla 27: RUC- 13

| Identificador | RUC-14 | | | | |
|--------------------|---|---|-------|--------------------|------|
| Nombre | Eliminar gesto asociado a una postura | | | | |
| Descripción | El sistema permitirá al usuario desasociar un gesto de una postura. | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | X | Media | | Baja |

Tabla 28: RUC-14

| Identificador | RUC-14 | | | | |
|--------------------|--|---|-------|--------------------|------|
| Nombre | Cambiar gesto asociado a una postura | | | | |
| Descripción | El sistema permitirá al usuario cambiar el gesto asignado a una postura. | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | X | Media | | Baja |

Tabla 29: RUC-16

ii. Requisitos de restricción

| Identificador | RUR-01 | | | | |
|--------------------|--|--|-------|--------------------|------|
| Nombre | Formato de los ficheros de estado del motor | | | | |
| Descripción | El estado del motor se guardará en ficheros de formato XML | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | | Media | X | Baja |

Tabla 30: RUR-01

| Identificador | RUR-02 | | | | |
|--------------------|--|---|-------|--------------------|------|
| Nombre | Número mínimo de posturas reconocibles | | | | |
| Descripción | El motor deberá ser capaz de distinguir entre al menos dos posturas almacenadas. | | | | |
| Tipo | Capacidad | | X | Restricción | |
| Necesidad | Obligatorio | | X | Opcional | |
| Prioridad | Alta | X | Media | | Baja |

Tabla 31: RUR-02

| Identificador | RUR-03 | | | | | |
|---------------|---|--|---|-------------|---|------|
| Nombre | Número mínimo de posturas almacenadas | | | | | |
| Descripción | El motor deberá ser capaz de almacenar al menos 2 posturas. | | | | | |
| Tipo | Capacidad | | X | Restricción | | |
| Necesidad | Obligatorio | | X | Opcional | | |
| Prioridad | Alta | | | Media | X | Baja |

Tabla 32: RUR-03

d. Identificación de subsistemas

Dado que los principales objetivos de la librería consisten en el reconocimiento de gestos y en conseguir la mayor versatilidad posible tanto a la hora de guardar los datos del motor como al comunicarse con los proyectos en los que se decida integrar la librería, se han identificado los siguientes subsistemas principales junto a sus responsabilidades asociadas:

-**Interfaz de comunicación:** Se encargará de proveer las funcionalidades necesarias para comunicarse con el proyecto en el que se quiera integrar la biblioteca. Servirá principalmente como intermediario entre la interfaz de usuario y el motor.

-**Interfaz del sensor:** Se encargará de hacer de intermediario entre el sensor Kinect, el motor y la interfaz de comunicación.

-**Interfaz de almacenamiento/recuperación:** Se encargará de almacenar y recuperar los datos referentes al motor en memoria secundaria.

-**Núcleo del motor:** Proveerá la funcionalidad principal del motor, la captura y el reconocimiento de los gestos y posturas basándose en los datos que reciba de la interfaz del sensor.

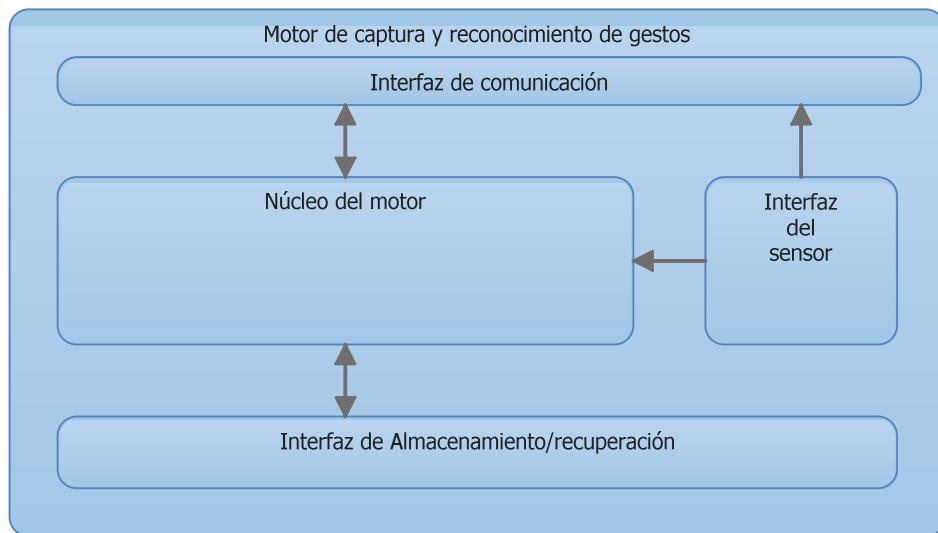


Diagrama 13: Subsistemas

e. Identificación de clases principales y responsabilidades

A estas alturas es posible empezar a identificar las clases principales de cada subsistema, así como las responsabilidades de cada una. Las clases identificadas en este paso son:

i. Interfaz de comunicación

| | |
|--------------------------|--|
| Clase | Notificador |
| Subsistema | Interfaz de comunicación |
| Responsabilidades | Notificar a las interfaces de módulos externos los eventos producidos tanto por el núcleo del motor (reconocimiento de una postura, captura de una postura...) como por la interfaz del sensor (Frame de color listo...) |

Tabla 33: Responsabilidades de la clase Notificador

| | |
|--------------------------|---|
| Clase | Motor |
| Subsistema | Interfaz de comunicación |
| Responsabilidades | Servir como interfaz de entrada al núcleo del motor. Deberá contar con componentes que permitan a un módulo externo interactuar con el núcleo del motor y además actuará como nexo de unión entre los diferentes subsistemas del motor. |

Tabla 34: Responsabilidades de la clase Entrada

ii. Interfaz del sensor

| | |
|--------------------------|--|
| Clase | Sensor |
| Subsistema | Interfaz del sensor |
| Responsabilidades | Deberá notificar de nuevos eventos del sensor Kinect a la interfaz de comunicación. Además esta clase será responsable de restringir el acceso directo al sensor Kinect. |

Tabla 35: Responsabilidades de la clase Sensor

iii. Interfaz de almacenamiento/recuperación

| | |
|--------------------------|--|
| Clase | ManejadorFicheros |
| Subsistema | Interfaz de almacenamiento/recuperación |
| Responsabilidades | Esta clase será encargada de interactuar con los ficheros de memoria secundaria. Permitirá exportar e importar los datos de entrada del motor. |

Tabla 36: Responsabilidades de la clase ManejadorFicheros

iv. Núcleo del motor

| | |
|--------------------------|---|
| Clase | Capturador |
| Subsistema | Núcleo del motor |
| Responsabilidades | Esta clase se encargará de recoger los datos sobre la posición del cuerpo que recibirá desde la interfaz del sensor. Tendrá componentes que permitan capturar una as. |

Tabla 37: Responsabilidades de la clase Capturador

| | |
|--------------------------|---|
| Clase | Reconocedor |
| Subsistema | Núcleo del motor |
| Responsabilidades | Esta clase será la encargada de reconocer las posturas y gestos en base a los datos obtenidos a través de la interfaz del sensor. Además contará con la lista de posturas almacenadas en memoria principal. |

Tabla 38: Responsabilidades de la clase Reconocedor

| | |
|--------------------------|---|
| Clase | Postura |
| Subsistema | Núcleo del motor |
| Responsabilidades | Representará una postura, tanto el nombre como el conjunto de datos de entrada que servirán como entrenamiento para el motor así como los datos del gesto asociados a la misma. |

Tabla 39: Responsabilidades de la clase Postura

f. Definición de pruebas de aceptación del sistema

Para la validación de que el proyecto cumple con los requisitos mínimos establecidos en este apartado se define el siguiente plan de pruebas, que se ejecutarán tras la codificación. Se muestra el identificador de cada grupo de pruebas, el nombre del grupo, el objetivo que tiene y los requisitos que validarán:

| ID | Nombre | Objetivo | Requisitos validados |
|----|--------|----------|----------------------|
| | | | |

| | | | |
|--------------|---------------------------------------|---|--|
| GP-01 | Grabar una postura | Se probará que el sistema es capaz de almacenar correctamente en memoria principal los datos sobre la posición de las diferentes partes del cuerpo en un instante determinado | RUC-01 |
| GP-02 | Gestionar gestos asociados a posturas | Se probará que se pueden asignar y eliminar correctamente gestos a una postura. | RUC-03, RUC-14, RUC-15 |
| GP-03 | Reconocer una postura | Se probará que el sistema es capaz de reconocer una postura que ha sido capturada previamente por el sistema | RUC-01, RUC-02, RUC-07, RUR-02, RUR-03 |
| GP-04 | Reconocer un gesto | Se probará que el sistema es capaz de reconocer un gesto previamente definido por el usuario | RUC-03 RUC-04 |
| GP-05 | Gestionar posturas | Se probará que el sistema puede cambiar el nombre de posturas o eliminarlas | RUC-08, RUC-09, RUC-13 |
| GP-06 | Exportar posturas | Se probará el guardado de la información de las posturas a un fichero | RUC-05, RUR-01 |
| GP-07 | Importar posturas | Se probará el guardado y la recuperación del estado del motor. | RUC-06, RUR-01 |
| GP-08 | Opciones | Se probará que el motor puede guardar y cargar automáticamente las posturas capturadas y que se puede modificar el tiempo de espera para la captura | RUC-10, RUC-11, RUC-12 |

Tabla 40: Resumen del plan de pruebas

La especificación detallada de cada prueba junto a los pasos para su correcta realización se muestra a continuación. La información sobre las diferentes posturas que se utilizarán se especifica en la Tabla 153:

i. GP-01 - Grabar una postura

El objetivo de este grupo de pruebas es comprobar que el sistema es capaz de almacenar correctamente en memoria principal los datos de la posición del cuerpo que extraerá mediante el sensor Kinect. Dada la naturaleza del objetivo a validar se realizarán dos pruebas, una para comprobar que se puede capturar una postura correctamente y otra para validar que se muestra un mensaje de error después de esperar el tiempo estipulado.

1. GP-01.1

Prueba para validar que se capture una postura correctamente.

Pasos:

1. Abrir el programa
2. Entrar en modo captura
3. Escribir el nombre de la postura
4. Activar la captura de posturas

5. Colocarse en una posición en la que el sistema reconozca al usuario
6. Colocarse en la postura 1.
7. Esperar a que se muestre el mensaje de confirmación

Para determinar que la postura ha sido insertada correctamente se deberá verificar que la misma está incluida en la lista de posturas capturadas.

2. GP-01.2

Prueba para verificar que se muestra el mensaje de error si no se encuentra al usuario en el tiempo estipulado.

Pasos:

1. Abrir el programa
2. Entrar en modo captura
3. Escribir el nombre de la postura
4. Activar la captura de posturas
5. Colocarse en una posición en la que el sistema **NO** reconozca al usuario
6. Esperar a que se muestre el mensaje de error

Se considerará la prueba pasada cuando el mensaje de error se muestre.

ii. GP-02 – Gestionar gestos asociados a una postura

El objetivo de este grupo de pruebas es verificar que el sistema puede asignar correctamente gestos a una postura y después eliminarlos, y que no saltan errores al intentar eliminar gestos de posturas que no tuvieran uno asignado.

Se realizarán cuatro pruebas dentro de este grupo, una para comprobar que se puede asignar correctamente un gesto a una captura, otra para cambiar el gesto de una postura con un gesto asignado, otra para eliminar el gesto asignado a una postura y la última para comprobar el correcto funcionamiento al intentar eliminar el gesto de una postura que no lo tuviera asignado.

1. GP-02.1

Esta prueba verificará que se puede asignar un gesto a una postura. Se da por supuesto que el sistema habrá capturado con anterioridad al menos una postura

Pasos:

1. Abrir el programa
2. Pulsar sobre el botón de ver posturas
3. Seleccionar una postura de la lista que no tenga un gesto asignado
4. Pulsar sobre el botón asignar gesto
5. Rellenar los datos pedidos
6. Pulsar el botón de aceptar

Se considerará que se ha superado la prueba si al seleccionar la postura de nuevo ya tiene asignados los datos introducidos con anterioridad.

2. GP-02.2

Esta prueba verificará que se puede cambiar el gesto asignado a una postura. Los pasos para la realización son los mismos que en GP-02.1 pero en este caso en el paso 3 se seleccionará una postura que ya tenga un gesto asignado.

3. GP-02.3

Se probará que se puede eliminar el gesto de una postura que tuviera alguno asignado.

Pasos:

1. Abrir el programa
2. Pulsar sobre el botón de ver posturas
3. Seleccionar una postura de la lista que tenga un gesto asignado
4. Pulsar sobre el botón de eliminar gesto

La prueba se considerará pasada si al seleccionar la postura de nuevo ya no tiene asignados los datos del gesto.

4. GP-02.4

Se probará que al eliminar el gesto de una postura que no tuviera un gesto asignado el sistema no lanza un error si no que deja la postura como está. Los pasos para la realización de esta prueba serán los mismos que en GP-02.3 pero en este caso en el paso 3 se seleccionará una postura que no tenga ningún gesto asignado. Se considerará pasada la prueba si tras pulsar el botón de eliminar gesto no salta ningún error.

iii. GP-03 Reconocer una postura

El objetivo de este grupo de pruebas es validar el reconocimiento de posturas, para ello se realizará una única prueba que consistirá en el reconocimiento de una postura sobre 4 capturadas con anterioridad o importadas desde un fichero XML.

1. GP-03.1

Pasos:

1. Abrir el programa
2. Capturar las posturas 1, 2, 3 y 4 (Ver pasos de prueba GP-01.1)
3. Pulsar sobre el botón que activa el reconocimiento
4. Colocarse en la posición en la que el sensor Kinect reconozca al usuario
5. Colocarse en la postura 1

Se considerará que se ha pasado la prueba si en el programa se muestra que se ha reconocido la postura 1.

iv. GP-04 Reconocer un gesto

En este grupo de pruebas se verificará que el sistema es capaz de reconocer un gesto previamente definido por el usuario. Para ello se realizarán dos pruebas sencillas. Primero se probará a realizar un gesto consistente en dos posturas y después se probará uno consistente en tres posturas.

1. GP-04.1 y GP-04.2

Los primeros pasos son comunes a las dos pruebas así que se describirán de forma conjunta.

Pasos:

1. Abrir el programa
2. Capturar las posturas 1, 2, 3, 4 y 5 (Ver pasos de prueba GP-01.1)
3. Pulsar sobre el botón que activa el reconocimiento
4. Asignar a la postura 3 el gesto “Deslizar derecha” con posición 0 y es final sin marcar
5. Asignar a la postura 4 el gesto “Deslizar derecha” con posición 1 y es final marcado
6. Asignar a la postura 5 el gesto “Levantar las manos” con posición 0 y es final sin marcar
7. Asignar a la postura 1 el gesto “Levantar las manos” con posición 1 y es final sin marcar
8. Asignar a la postura 2 el gesto “Levantar las manos” con posición 2 y es final marcado
9. Colocarse en la posición en la que el sensor Kinect reconozca al usuario

Pasos específicos de GP-04.1

10. Realizar postura 3
11. Realizar postura 4

Si el sistema marca como gesto realizado el gesto “Deslizar derecha” se considerará la prueba pasada

Pasos específicos de GP-04.1

10. Realizar postura 5
11. Realizar postura 1
12. Realizar postura 2

Si el sistema marca como gesto realizado el gesto levantar las manos se considerará la prueba pasada

v. GP-05 Gestionar posturas

El objetivo de este grupo de pruebas es validar la gestión de posturas. Se probará que se pueden eliminar posturas y que se puede cambiar el nombre a las mismas. También se probará que a una postura no se le puede asignar el nombre de otra.

1. GP-05.1

En esta prueba se comprobará que se puede eliminar correctamente una postura

Pasos:

1. Abrir el programa
2. Pulsar sobre el botón de ver posturas

3. Seleccionar una postura cualquiera de la lista
4. Pulsar sobre el botón de eliminar postura

Si al volver a pulsar sobre el botón de ver posturas la postura eliminada no se encuentra en la lista se considerará la prueba pasada.

2. GP-05.2

En esta prueba se comprobará que se puede cambiar correctamente el nombre de una postura por otro nombre sin asignar.

Pasos:

1. Abrir el programa
2. Pulsar sobre el botón de ver posturas
3. Seleccionar una postura cualquiera de la lista
4. Pulsar sobre el botón de cambiar nombre
5. Introducir el nuevo nombre de la postura, que no debe coincidir con el de ninguna otra postura
6. Pulsar el botón aceptar

Si al volver a pulsar sobre el botón ver posturas la postura ha cambiado de nombre, se considerará pasada la prueba.

3. GP-05.3

En esta prueba se comprobará que no se puede asignar a una postura existente el nombre de otra. Los pasos serán los mismos que en GP-05.2 pero en este caso en el paso 5 se introducirá un nombre ya asignado a otra postura. Se considerará que se ha pasado la prueba si el sistema muestra un mensaje de error y no cambia el nombre de ninguna postura.

vi. GP-06 Exportar posturas

Este grupo de pruebas intentará validar que se puede exportar la lista de posturas a ficheros almacenados en el disco duro. Para ello se realizarán dos pruebas. En la primera, tras capturar algunas posturas se intentará exportarlas a un fichero que no exista en el sistema, en la segunda se intentará exportar a un fichero existente en el sistema.

1. GP-06.1

Pasos:

1. Abrir el programa
2. Capturar las posturas 1, 2, 3, 4 y 5 (Ver pasos de prueba GP-01.1)
3. Pulsar el botón de exportar posturas
4. Escribir el nombre del fichero en el que se guardarán las posturas
5. Pulsar el botón de aceptar

Se considerará pasada la prueba si se crea un archivo con el nombre indicado y contiene la lista de posturas capturadas por el sistema.

2. GP-06.2

Esta prueba intentará sobrescribir un fichero con los datos nuevos de posturas. Los pasos de realización serán los mismos que en GP-06.1 pero ahora en el paso 4 se escribirá el nombre de un fichero existente en el sistema. Se considerará que se ha superado la prueba si el sistema sobrescribe el fichero antiguo con los datos nuevos.

vii. GP-07 Importar posturas

Este grupo de pruebas intentará validar que se puede importar la lista de posturas de un fichero almacenado en el disco duro. Se realizarán dos pruebas, una intentará importar desde un archivo que existe y contiene datos de las posturas y la otra intentará importar los datos desde un fichero inexistente.

1. GP-07.1

Esta prueba intentará cargar los datos de posturas almacenados en un fichero existente en el disco duro.

Pasos:

1. Abrir el programa
2. Pulsar el botón de importar posturas
3. Escribir la ruta del fichero que contiene los datos de las posturas
4. Pulsar el botón aceptar.

Se considerará que se ha pasado la prueba si al pulsar el botón de ver posturas la lista muestra las posturas que contenía el fichero introducido.

2. GP-07.2

Esta prueba verificará que el sistema muestra un mensaje de error al intentar cargar las posturas desde un fichero inexistente. Los pasos de realización serán los mismos que en GP-07.1, pero en este caso se escribirá la ruta de un fichero inexistente. Se considerará que se ha superado la prueba si se muestra un mensaje de error y no se carga ninguna postura.

viii. GP-08 Opciones

En este grupo de pruebas se probarán las funcionalidades referentes a las opciones del sistema. Concretamente se probará que el sistema puede guardar y cargar automáticamente las posturas almacenadas y que se pueden modificar los tiempos de captura de posturas.

1. GP-08.1

Esta prueba validará que se guardan y cargan automáticamente los datos al activar el reconocimiento de posturas.

Pasos:

1. Abrir el programa
2. Capturar posturas 1 y 2 (Ver pasos de GP-01.1)
3. Pulsar el botón de activar reconocimiento
4. Pulsar el botón de salir
5. Volver a abrir el programa

Si al pulsar sobre el botón de ver posturas aparecen las posturas 1 y 2 se considerará superada la prueba.

2. GP-08.2

Esta prueba validará que se pueden modificar los tiempos de espera antes y después de la captura.

Pasos:

1. Abrir el programa
2. Pulsar el botón opciones
3. Cambiar el tiempo de espera antes de capturar postura a 1 segundo y el tiempo de después a 2 segundos.
4. Pulsar el botón aceptar
5. Intentar capturar postura y verificar que se cumplen los tiempos marcados.

Si los tiempos coinciden se considerará superada la prueba.

5. Diseño

a. Definición de la arquitectura

Para el presente TFG, dados los diferentes requisitos de usuario y casos de uso comunes, definí una primera arquitectura para el sistema que se basaba en los subsistemas especificados en el análisis y una interfaz gráfica de usuario unida a la biblioteca de clases. El problema que tenía este diseño es que permitía poca personalización e integración de la biblioteca en proyectos mayores. Si se deseaba implementar otra interfaz había que modificar la interfaz proporcionada o la biblioteca de clases, no se podía implementar una interfaz nueva de otra forma. Además dicho diseño contaba con una clase por cada subsistema (exceptuando el núcleo del motor, que contaba con la clase “núcleo” y la clase “postura”), lo cual lo hacía difícil de mantener si encontraba algún fallo.

Por eso elegí el presente diseño, que continúa con la arquitectura de 4 subsistemas, pero que a diferencia del anterior implementa un patrón de diseño “observer”, lo que permite a proyectos mayores implementar su propia interfaz sin necesidad de cambiar el código de la librería.

Por otra parte como se puede observar en el diagrama siguiente, aunque el sistema continúa dividido en 4 subsistemas, decidí cambiar la forma en la que se comunican entre sí, dejando una sola clase encargada de conocer a la clase principal de cada subsistema, de manera que actuara como controlador y redirigiera las llamadas entre diferentes subsistemas. Tomé dicha decisión para evitar errores que pudieran ser provocados porque diferentes subsistemas no tuvieran la misma instancia de la clase principal de otro subsistema.

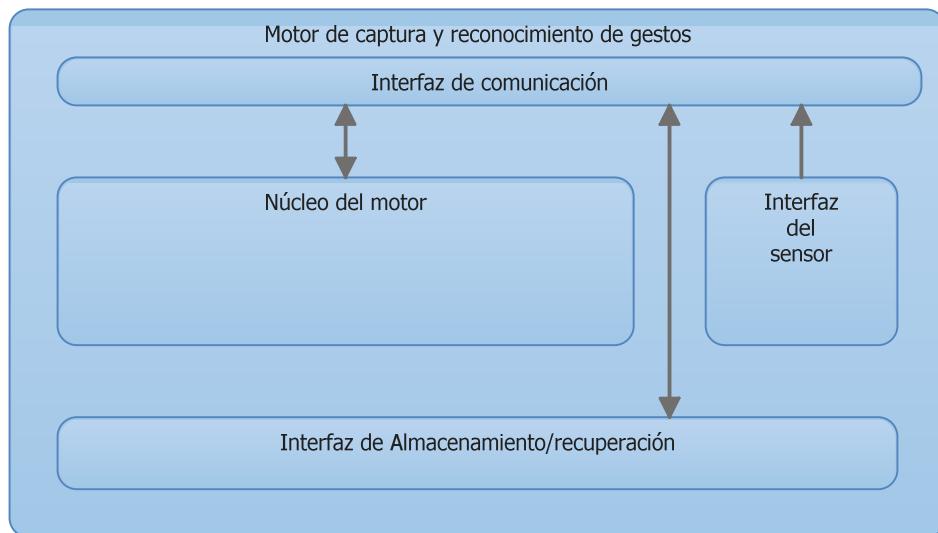


Diagrama 14: Arquitectura del sistema

Además en este diseño elegí separar las tareas de capturar y reconocer posturas en clases distintas, e incluso la forma en la que se reconocen de forma interna está separada en una clase que implementa la red de neuronas. Esto está pensado así para facilitar un posible cambio en el algoritmo de detección de posturas sin que este afecte al resto del sistema, bastaría con cambiar la clase que implementa la red por una que implemente por ejemplo los modelos ocultos de Markov.

b. Diseño de clases y responsabilidades

Tras identificar las principales clases y sus responsabilidades en el apartado del análisis, identifiqué algunas clases más complementarias y definí mas a fondo las responsabilidades de cada clase, dando lugar a la siguiente estructura:

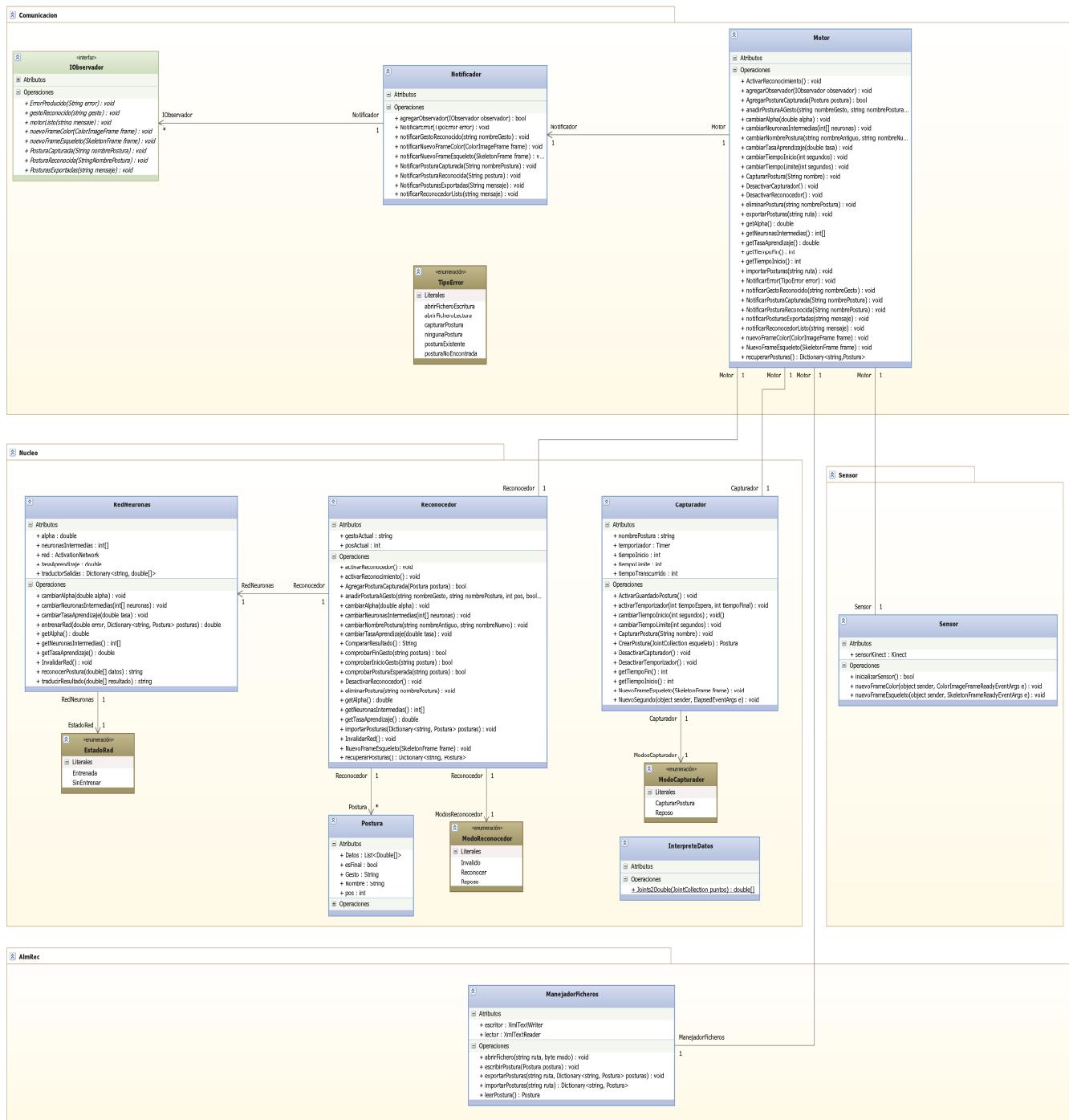


Diagrama 15: Clases del sistema

Como se puede observar, el diagrama anterior está organizado por subsistemas y muestra las relaciones entre las diferentes clases, así como los atributos y métodos que contienen. Los métodos se explicarán detalladamente en el punto siguiente del presente documento, mientras que las clases y sus atributos se explican a continuación, ordenadas por subsistema:

i. Interfaz de comunicación

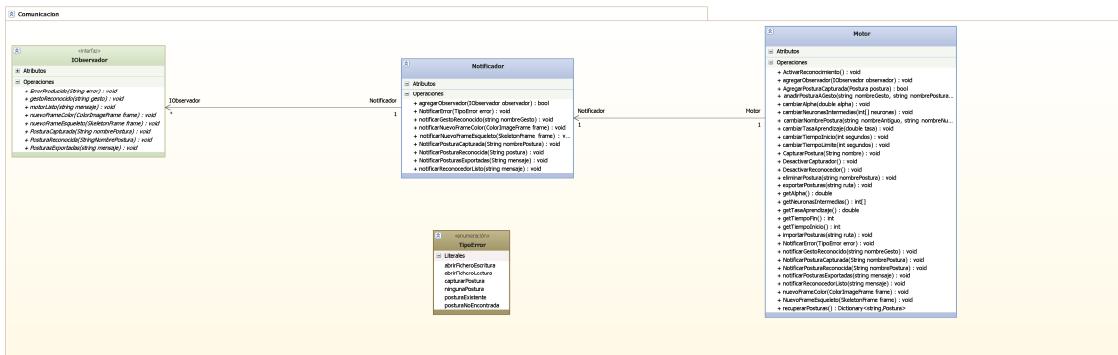


Diagrama 16: Clases de la interfaz de comunicación

| Nombre | IObservador |
|-------------|---|
| Tipo | Interfaz |
| Atributos | N/A |
| Descripción | Es la interfaz que deberán implementar las interfaces de usuario para recibir las notificaciones que lanzará el notificador cuando se reconozca una postura, un gesto o se produzca un error. Esta clase es parte del patrón de diseño “Observer” |

Tabla 41: Diseño de clase IObservador

| | |
|-------------|---|
| Nombre | Notificador |
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> • Observadores : List<IObservador> - Lista de observadores que recibirán las notificaciones producidas en esta clase. |
| Descripción | Los métodos que contiene la clase notificador son los necesarios para añadir nuevos observadores a la lista que recibirán las notificaciones y los métodos que generarán las notificaciones propiamente dichas, incluyendo los mensajes que acompañarán cada notificación. Es parte del patrón de diseño “Observer” |

Tabla 42: Diseño de clase Notificador

| Nombre | TipoError |
|-------------|--|
| Tipo | Enumeración |
| Atributos | <ul style="list-style-type: none"> • AbrirFicheroEscritura • AbrirFicheroLectura • capturarPostura • ningunaPostura • posturaExistente • posturaNoEncontrada |
| Descripción | Esta enumeración representa los diferentes tipos de error que se pueden notificar desde el motor. Si se produce algún error, la clase que lo detecte realizará una llamada a la clase notificador pasándole como parámetro un tipo de error de esta enumeración para que el notificador notifique un mensaje de error en consecuencia personalizado para cada error. |

Tabla 43: Diseño de clase TipoError

| Nombre | Motor |
|-------------|--|
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> Notificador : Notificador – Instancia de la clase notificador a la que redirigirá el motor cuando tenga que producirse una notificación. Capturador : Capturador - Instancia de la clase capturador que estará encargada de capturar las posturas del usuario. Reconocedor : Reconocedor – Instancia de la clase reconocedor que se encargará de decidir si el usuario está realizando una postura ó gesto conocido ó no. Sensor : Sensor – Instancia de la clase Sensor que enviará al motor los datos que recoja el sensor Kinect para que este lo distribuya al notificador, al capturador y al reconocedor. |
| Descripción | Esta clase sirve como punto de entrada para sistemas externos (como la interfaz de usuario) al motor. Además de manera interna sirve como controlador para redirigir las llamadas entre diferentes clases del sistema, de manera que todo el flujo de llamadas pase a través de esta clase, lo cual evita inconsistencias como instancias duplicadas de clases. |

Tabla 44: Diseño de clase Motor

ii. Interfaz del sensor

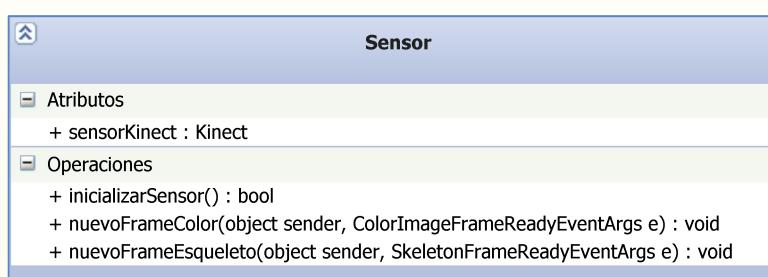


Diagrama 17: Clases de la interfaz del sensor

| Nombre | Sensor |
|-------------|---|
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> SensorKinect : Kinect – Representación del sensor Kinect que enviará los datos recogidos por el mismo al motor. Motor : Motor – Instancia de la clase motor a la que se enviarán los datos recogidos por el sensor Kinect. |
| Descripción | La principal función de esta clase es aislar el sensor Kinect del resto del sistema y redirigir los datos recabados por el mismo (los datos obtenidos por las cámaras de color y profundidad) al motor para que este lo distribuya a las clases que necesiten sus datos. |

Tabla 45:Diseño de la clase Sensor

iii. Núcleo del motor

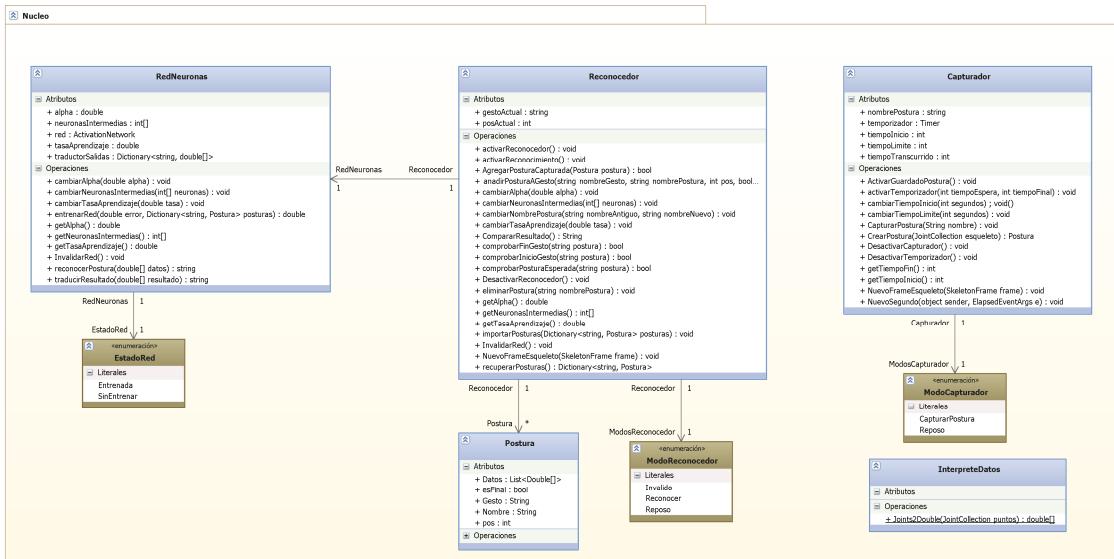


Diagrama 18: Clases del núcleo del motor

| Nombre | Postura |
|-------------|--|
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> Datos : List<Double> - Capturas de la posición de cada articulación reconocida por Kinect en un instante determinado de la postura. |
| | <ul style="list-style-type: none"> esFinal : bool – Valor que definirá si la postura es la última de un gesto o no. Gesto : String – Nombre del gesto al que pertenece la postura en caso de pertenecer a algún gesto. Pos : int – Posición de la postura dentro del gesto en caso de pertenecer a alguno. Nombre : String – Nombre de la postura. |
| Descripción | Esta clase representa una postura. Contiene todos los atributos que permiten definir la postura y el gesto al que pertenece. |

Tabla 46: Diseño de la clase postura

| | |
|-------------|--|
| Nombre | InterpreteDatos |
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> • N/A |
| Descripción | <p>Esta clase sirve para convertir los datos recogidos por el sensor Kinect (datos tipo JointCollection) en un array de double en el cual la posición de cada articulación está representada como la distancia en línea recta de cada una de sus coordenadas a las coordenadas de la articulación que Kinect llama “Spine central”</p> |

Tabla 47: Diseño de la clase InterpreteDatos

| Nombre | Capturador |
|-----------|--|
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> • nombrePostura : String – Nombre de la postura a capturar • temporizador : Timer – Temporizador que se activará al iniciar la captura de posturas. |

| | |
|-------------|--|
| | <ul style="list-style-type: none"> • tiempoInicio : int – Tiempo que se esperará en segundos antes de empezar a capturar una postura. • tiempoLimite : int – Tiempo que se esperará en segundos antes de dejar de intentar capturar una postura. • tiempoTranscurrido : int – Tiempo transcurrido en segundos desde que se activó el temporizador. • Motor : Motor – Instancia de la clase motor a la que se realizarán las llamadas para comunicar el capturador con el resto del sistema. • Modo : ModoCapturador – Modo en el que se encuentra el capturador en un determinado momento. |
| Descripción | <p>La clase capturador, como su nombre indica, tiene como objetivo el capturar los datos posicionales del cuerpo del usuario cuando este realiza una postura. Además contiene las funciones necesarias para modificar el tiempo que se esperará antes de capturar una postura y antes de dejar de hacerlo.</p> |

Tabla 48: Diseño de la clase Capturador

| Nombre | ModoCapturador |
|-------------|---|
| Tipo | Enumeración |
| Atributos | <ul style="list-style-type: none"> • CapturarPostura • Reposo |
| Descripción | <p>Esta enumeración representa los distintos modos en los que puede encontrarse el capturador en un determinado momento. Se utiliza para saber si el capturador debe capturar una postura o no.</p> |

Tabla 49: Diseño de la clase ModoCapturador

| Nombre | Reconocedor |
|-------------|---|
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> • gestoActual : String – Nombre del gesto al que pertenece la última postura reconocida. Se utiliza para el reconocimiento de gestos. • posActual : int – Posición de la última postura reconocida dentro del gesto al que pertenece. Se utiliza para el reconocimiento de gestos. • Posturas : Dictionary<String, Postura> - Lista de posturas que es capaz de reconocer el sistema organizada en pares de nombre – postura. • Red : RedNeuronas – Instancia de la clase RedNeuronas a la que enviará los datos obtenidos por el sensor para saber si dichos datos corresponden a una postura en concreto o no. • Modo : ModoReconocedor – Modo en el que se encuentra el reconocedor en un instante determinado. |
| Descripción | <p>Esta clase se encarga de mantener una lista con las posturas reconocibles por el motor así como de enviar los datos obtenidos por el sensor Kinect a la red de neuronas. Además esta clase contiene las funciones necesarias para administrar las posturas y los gestos asociados a las mismas.</p> |

Tabla 50: Diseño de la clase Reconocedor

| Nombre | ModoReconocedor |
|-----------|---|
| Tipo | Enumeración |
| Atributos | <ul style="list-style-type: none"> • Inválido • Reconocer |

| | |
|-------------|--|
| | <ul style="list-style-type: none"> Reposo |
| Descripción | <p>Esta enumeración representa el modo en el que se encuentra el reconocedor en un instante determinado de tiempo, inválido si no ha entrenado la red de neuronas, reposo si está entrenada pero no está activado el reconocedor ó reconocer si el reconocedor está intentando reconocer posturas.</p> |

Tabla 51: Diseño de la clase ModoReconocedor

| Nombre | RedNeuronas |
|-------------|--|
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> Alpha : Double – Valor alpha de la red de neuronas. Se utiliza para el entrenamiento de la red. neuronasIntermedias : int[] – Número de neuronas ocultas que habrá en cada capa intermedia de la red de neuronas. Elegir un número adecuado de capas y de neuronas es vital para el correcto funcionamiento del sistema. Red : ActivationNetwork – Instancia de la red de neuronas. tasaAprendizaje : double – Valor de la tasa de aprendizaje para la red de neuronas. Se utiliza durante el entrenamiento de la red. traductorSalidas : Dictionary<string, double[]> - Diccionario que asocia arrays de double a nombres de posturas. Estado : EstadoRed – Estado de la red de neuronas en un determinado instante. |
| Descripción | <p>La clase RedNeuronas contiene los datos relativos a la red de neuronas así como métodos para crearla, entrenarla y realizar la traducción entre la salida de la misma y las posturas reconocibles. Esta clase es la que en base a los datos de entrada intentará clasificar los mismos en una de las posturas reconocibles.</p> |

Tabla 52: Diseño de la clase RedNeuronas

| Nombre | EstadoRed |
|-------------|--|
| Tipo | Enumeración |
| Atributos | <ul style="list-style-type: none"> Entrenada Sin entrenar |
| Descripción | <p>Esta enumeración representa el estado de la red de neuronas en un determinado instante. La red se puede encontrar entrenada o sin entrenar.</p> |

Tabla 53: Diseño de la clase EstadoRed

iv. Interfaz de almacenamiento/recuperación

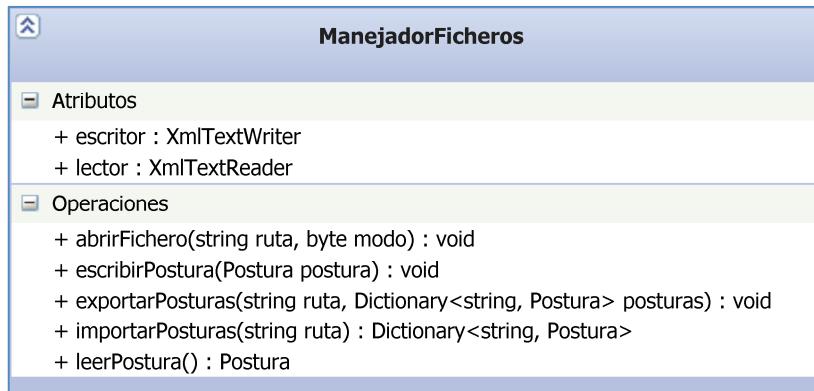


Diagrama 19: Clases de la Interfaz de almacenamiento/recuperación

| Nombre | ManejadorFicheros |
|-------------|--|
| Tipo | Clase |
| Atributos | <ul style="list-style-type: none"> Escriptor : XMLTextWriter – Clase de apoyo que escribe los datos en el fichero con formato XML. Lector : XMLTextReader – Clase de apoyo que lee los datos de un fichero con formato XML. Motor : Motor – Instancia de la clase motor a la que se llamará cuando el manejarFicheros deba comunicarse con otras partes del sistema. |
| Descripción | La función principal de esta clase es la escritura de las posturas a ficheros con formato XML y la posterior lectura de los mismos. |

Diagrama 20: Diseño de la clase ManejadorFicheros

c. Diseño de componentes del sistema

Tras haber realizado el diseño de las clases que componen cada subsistema y en base al mismo, muestro a continuación la descripción de cada componente del sistema, ordenado por subsistemas y dentro de los mismos por clase a la que pertenece el componente.

i. Interfaz de comunicación

1. Clase IObservador

| Nombre | ErrorProducido |
|----------------------------|--|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> Mensaje : String – El mensaje de error que se mostrará |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de proveer un mensaje de error que pueda ser mostrado al usuario. |

Tabla 54: IObservador.ErrorProducido

| Nombre | GestoReconocido |
|----------------------------|--|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> Gesto : String – El nombre del gesto que ha reconocido el motor |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de mostrar al usuario el nombre del gesto que ha reconocido el motor. |

Tabla 55: IObservador.GestoReconocido

| Nombre | MotorListo |
|----------------------------|---|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> Mensaje : String – El mensaje informativo que se hará llegar al usuario |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de mostrar al usuario un mensaje informando que el motor está listo para reconocer posturas. |

Tabla 56: IObservador.MotorListo

| Nombre | nuevoFrameColor |
|----------------------------|---|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> Frame : ColorImageFrame – El frame que ha recabado la cámara de color del sensor Kinect. |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de mandar el frame que ha conseguido el sensor Kinect a la interfaz de usuario, para poder mostrarlo por pantalla. |

Tabla 57: IObservador.nuevoFrameColor

| Nombre | nuevoFrameEsqueleto |
|----------------------------|---|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> Frame : SkeletonFrame – El frame que ha recabado la cámara de profundidad del sensor Kinect con los datos del esqueleto. |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de mandar el frame que ha conseguido el sensor Kinect a la interfaz de usuario para poder mostrarlo por pantalla |

Tabla 58: IObservador.nuevoFrameEsqueleto

| Nombre | PosturaCapturada |
|----------------------------|---|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombrePostura : String – El nombre de la postura que ha capturado el motor |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de mostrar al usuario un mensaje informando que el motor ha capturado correctamente una postura y el nombre de la misma. |

Tabla 59: IObservador.PosturaCapturada

| Nombre | PosturaReconocida |
|----------------------------|--|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombrePostura : String – El nombre de la postura que ha reconocido el motor |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de mostrar al usuario un mensaje informando que el motor ha reconocido correctamente una postura y el nombre de la misma. |

Tabla 60: IObservador.PosturaReconocida

| Nombre | PosturasExportadas |
|----------------------------|--|
| Clase | IObservador |
| Parámetros de entrada | <ul style="list-style-type: none"> • Mensaje : String – El mensaje informativo que se hará llegar al usuario |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente deberá ser implementado por las clases de la interfaz de usuario que implementen la interfaz IObservador. La función principal es la de mostrar al usuario un mensaje informando que se han exportado correctamente las posturas a un fichero. |

Tabla 61: IObservador.PosturasExportadas

2. Clase Notificador

| Nombre | AgregarObservador |
|----------------------------|---|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • observador : IObservador – La clase de interfaz de usuario a la que se deberán enviar las notificaciones |
| Parámetro de salida | Verdadero si se ha añadido el observador correctamente o falso en caso contrario. |
| Descripción del componente | Este componente simplemente agrega a la lista de |

| | |
|--|--|
| | observadores un nuevo observador, para que este pueda recibir las notificaciones que enviará el notificador. |
|--|--|

Tabla 62: Notificador.AgregarObservador

| Nombre | NotificarError |
|----------------------------|---|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • error : TipoError – El tipo de error que se ha producido. |
| Parámetro de salida | N/A |
| Descripción del componente | Elige un mensaje predeterminado según el tipo de error recibido y recorre toda la lista de observadores llamando a sus componentes ErrorProducido con el mensaje seleccionado. |

Tabla 63: Notificador.NotificarError

| Nombre | NotificarGestoReconocido |
|----------------------------|---|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombreGesto : String – El nombre del gesto que ha reconocido el motor. |
| Parámetro de salida | N/A |
| Descripción del componente | Recorre toda la lista de observadores llamando a sus componentes GestoReconocido con el nombre del gesto que ha reconocido el motor. |

Tabla 64: Notificador.NotificarGestoReconocido

| Nombre | NotificarNuevoFrameColor |
|----------------------------|--|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • frame : ColorImageFrame – El frame de la cámara de color que ha grabado el sensor Kinect. |
| Parámetro de salida | N/A |
| Descripción del componente | Recorre toda la lista de observadores llamando a sus componentes nuevoFrameColor con el frame de la imagen de color que ha recibido de la clase Sensor . |

Tabla 65: Notificador.NotificarNuevoFrameColor

| Nombre | NotificarNuevoFrameEsqueleto |
|----------------------------|--|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • frame : SkeletonFrame – El frame de la cámara de profundidad que ha grabado el sensor Kinect con los datos de posición del esqueleto. |
| Parámetro de salida | N/A |
| Descripción del componente | Recorre toda la lista de observadores llamando a sus componentes nuevoFrameEsqueleto con el frame de la imagen de profundida que ha recibido de la clase Sensor . |

Tabla 66: Notificador.NotificarNuevoFrameEsqueleto

| Nombre | NotificarPosturaCapturada |
|-----------------------|---|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombrePostura : String – El nombre de la |

| | |
|----------------------------|---|
| Parámetro de salida | postura que ha capturado el motor. |
| Descripción del componente | N/A |
| | Recorre toda la lista de observadores llamando a sus componentes posturaCapturada con el nombre de la postura que ha capturado el motor. |

Tabla 67: Notificador.NotificarPosturaCapturada

| Nombre | NotificarPosturaReconocida |
|----------------------------|---|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • postura : String – El nombre de la postura que ha reconocido el motor. |
| Parámetro de salida | N/A |
| Descripción del componente | Recorre toda la lista de observadores llamando a sus componentes posturaReconocida con el nombre de la postura que ha reconocido el motor. |

Tabla 68: Notificador.NotificarPosturaReconocida

| Nombre | NotificarPosturasExportadas |
|----------------------------|---|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • mensaje : String – El mensaje informativo que se enviará al usuario. |
| Parámetro de salida | N/A |
| Descripción del componente | Recorre toda la lista de observadores llamando a sus componentes posturasExportadas con el mensaje recibido por parámetro. |

Tabla 69: NotificarPosturasExportadas

| Nombre | NotificarReconocedorLista |
|----------------------------|---|
| Clase | Notificador |
| Parámetros de entrada | <ul style="list-style-type: none"> • mensaje : String – El mensaje informativo que se enviará al usuario. |
| Parámetro de salida | N/A |
| Descripción del componente | Recorre toda la lista de observadores llamando a sus componentes motorLista con el mensaje recibido por parámetro. |

Tabla 70: Notificador.NotificarReconocedorLista

3. Clase Motor

| Nombre | ActivarReconocimiento |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente ActivarReconocimiento de la clase Reconocedor . |

Tabla 71: Motor.ActivarReconocimiento

| Nombre | AgregarObservador |
|--------|-------------------|
| Clase | Motor |

| | |
|----------------------------|--|
| Parámetros de entrada | <ul style="list-style-type: none"> Observador : IObserver – La clase de interfaz de usuario a la que se deberán enviar las notificaciones. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente AgregarObservador de la clase Notificador . |

Tabla 72: Motor.AgregarObservador

| Nombre | AgregarPosturaCapturada |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> Postura : Postura – Una postura capturada por el capturador. |
| Parámetro de salida | Verdadero si se logra agregar la postura, falso en caso contrario. |
| Descripción del componente | Redirige la llamada al componente AgregarPosturaCapturada de la clase Reconocedor . |

Tabla 73: Motor.AgregarPosturaCapturada

| Nombre | AnadirPosturaAGesto |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> nombreGesto : String – El nombre del gesto al que se debe añadir la postura. nombrePostura : String – El nombre de la postura que se añadirá al gesto. Pos : Int – La posición de la postura dentro del gesto esFinal : bool – Si la postura es la última o no del gesto. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente anadirPosturaAGesto de la clase Reconocedor . |

Tabla 74: Motor.AnadirPosturaAGesto

| Nombre | cambiarAlpha |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> alpha : Double – El nuevo valor alpha que se quiere asignar para entrenar la red. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarAlpha de la clase Reconocedor . |

Tabla 75: Motor.CambiarAlpha

| Nombre | cambiarNeuronasIntermedias |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> neuronas : int[] – Las neuronas que habrá en cada capa oculta de la red de neuronas. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarNeuronasIntermedias de la clase Reconocedor . |

Tabla 76: Motor.CambiarNeuronasIntermedias

| Nombre | cambiarNombrePostura |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombreAntiguo : String – El nombre de la postura que se desea cambiar. • nombreNuevo : String – El nombre que se quiere asignar a la postura. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarNombrePostura de la clase Reconocedor . |

Tabla 77: Motor.CambiarNombrePostura

| Nombre | cambiarTasaAprendizaje |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • tasa : Double – El nuevo valor para la tasa de aprendizaje que se quiere asignar a la red de neuronas durante su entrenamiento. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarTasaAprendizaje de la clase Reconocedor . |

Tabla 78: Motor.CambiarTasaAprendizaje

| Nombre | cambiarTiempolInicio |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • segundos : int – El nuevo valor en segundos que se quiere asignar al tiempo de espera antes de iniciar la captura de posturas. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarTiempolInicio de la clase Capturador . |

Tabla 79: Motor.CambiarTiempolInicio

| Nombre | cambiarTiempoLimite |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • segundos : int – El nuevo valor en segundos que se quiere asignar al tiempo de espera antes de dejar de capturar posturas. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarTiempoLimite de la clase Capturador . |

Tabla 80: Motor.CambiarTiempoLimite

| Nombre | capturarPostura |
|-----------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombre : String – El nombre de la postura que se va a capturar. |
| Parámetro de salida | N/A |

| | |
|----------------------------|--|
| Descripción del componente | Redirige la llamada al componente capturarPostura de la clase Capturador . |
|----------------------------|--|

Tabla 81: Motor.CapturarPostura

| Nombre | DesactivarCapturador |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente desactivarCapturador de la clase Capturador . |

Tabla 82: Motor.DesactivarCapturador

| Nombre | DesactivarReconocedor |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente desactivarReconocedor de la clase Reconocedor . |

Tabla 83: Motor.DesactivarReconocedor

| Nombre | EliminarPostura |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombrePostura : String – El nombre de la postura que se desea eliminar |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente eliminarPostura de la clase Reconocedor . |

Tabla 84: Motor.EliminarPostura

| Nombre | ExportarPosturas |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • ruta : String – La ruta (con el nombre incluido) del fichero en el que se quieren guardar las posturas. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente exportarPosturas de la clase ManejadorFicheros . |

Tabla 85: Motor.ExportarPosturas

| Nombre | GetAlpha |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | El valor alpha de la red de neuronas : Double |
| Descripción del componente | Redirige la llamada al componente getAlpha de la clase Reconocedor . |

Tabla 86: Motor.GetAlpha

| Nombre | GetNeuronasIntermedias |
|--------|------------------------|
| | |

| Clase | Motor |
|----------------------------|--|
| Parámetros de entrada | N/A |
| Parámetro de salida | Las neuronas que hay en cada capa oculta de la red de neuronas : Int[] |
| Descripción del componente | Redirige la llamada al componente getNeuronasIntermedias de la clase Reconocedor . |

Tabla 87: Motor.GetNeuronasIntermedias

| Nombre | GetTasaAprendizaje |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | La tasa de aprendizaje de la red de neuronas : Double |
| Descripción del componente | Redirige la llamada al componente getTasaAprendizaje de la clase Reconocedor . |

Tabla 88: Motor.GetTasaAprendizaje

| Nombre | GetTiempoFin |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | El tiempo en segundos antes de dejar de intentar capturar una postura: Int |
| Descripción del componente | Redirige la llamada al componente getTiempoFin de la clase Capturador . |

Tabla 89: Motor.GetTiempoFin

| Nombre | GetTiempoinicio |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | El tiempo en segundos antes de empezar a capturar una postura: Int |
| Descripción del componente | Redirige la llamada al componente getTiempoinicio de la clase Capturador . |

Tabla 90: Motor.GetTiempoinicio

| Nombre | ImportarPosturas |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • ruta : String – Ruta (nombre del fichero incluido) del fichero desde el que se desean cargar las posturas. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente importarPosturas de la clase ManejadorFicheros . |

Tabla 91: Motor.ImportarPosturas

| Nombre | NotificarError |
|-----------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • error : TipoError – El tipo de error que se ha producido en el motor |

| | |
|----------------------------|--|
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente notificarError de la clase Notificador . |

Tabla 92: Motor.NotificarError

| Nombre | NotificarGestoReconocido |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombreGesto : String – El nombre del gesto que se ha reconocido. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente notificarGestoReconocido de la clase Notificador . |

Tabla 93: Motor.NotificarGestoReconocido

| Nombre | NotificarPosturaCapturada |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombrePostura : String – El nombre de la postura que se ha capturado. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente notificarPosturaCapturada de la clase Notificador . |

Tabla 94: Motor.NotificarPosturaCapturada

| Nombre | NotificarPosturaReconocida |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • nombrePostura : String – El nombre de la postura que se ha reconocido. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente notificarPosturaReconocida de la clase Notificador . |

Tabla 95: Motor.NotificarPosturaReconocida

| Nombre | NotificarPosturasExportadas |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • mensaje : String – El mensaje informativo que se mostrará al usuario. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente notificarPosturasExportadas de la clase Notificador . |

Tabla 96: Motor.NotificarPosturasExportadas

| Nombre | NotificarReconocedorListo |
|----------------------------|--|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> • mensaje : String – El mensaje informativo que se mostrará al usuario. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente notificarReconocedorListo de la clase Notificador . |

Tabla 97: Motor.NotificarReconocedorListo

| Nombre | NuevoFrameColor |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> frame : ColorImageFrame – El frame de la cámara de color del sensor Kinect. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente notificarNuevoFrameColor de la clase Notificador . |

Tabla 98: Motor.NuevoFrameColor

| Nombre | NuevoFrameEsqueleto |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | <ul style="list-style-type: none"> frame : SkeletonFrame – El frame de la cámara de profundidad del sensor Kinect con los datos e posición del esqueleto. |
| Parámetro de salida | N/A |
| Descripción del componente | Comprueba si el reconocedor y/o el capturador están activos y en caso de estarlo llama a sus componentes NuevoFrameEsqueleto . Después llama a dicho componente de la clase notificador . |

Tabla 99: Motor.NuevoFrameEsqueleto

| Nombre | RecuperarPosturas |
|----------------------------|---|
| Clase | Motor |
| Parámetros de entrada | N/A |
| Parámetro de salida | Una estructura con todas las posturas reconocibles por el sistema : Dictionary<String, Postura> |
| Descripción del componente | Redirige la llamada al componente RecuperarPosturas de la clase Reconocedor y devuelve el resultado de dicha llamada. |

Tabla 100: Motor.RecuperarPosturas

ii. Interfaz del sensor

1. Clase Sensor

| Nombre | InicializarSensor |
|----------------------------|---|
| Clase | Sensor |
| Parámetros de entrada | N/A |
| Parámetro de salida | Verdadero si se ha podido inicializar el sensor o falso si se ha producido algún error : bool |
| Descripción del componente | Inicializa los flujos de video de la cámara de color del sensor Kinect y de los datos del esqueleto del mismo. Después asocia al evento AllFramesReady del sensor Kinect el componente NuevosFrames de la clase Sensor |

Tabla 101: Sensor.InicializarSensor

| Nombre | NuevosFrames |
|--------|--------------|
| Clase | Sensor |

| | |
|----------------------------|---|
| Parámetros de entrada | <ul style="list-style-type: none"> Sender : object – Instancia que ha disparado el evento. E : AllFramesReadyEventArgs – Parámetros asociados al evento disparado |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente se dispara cada vez que las cámaras del sensor Kinect tienen un nuevo frame listo. Extrae el frame del esqueleto y de la cámara de color y se los pasa a los componentes NuevoFrameEsqueleto y NuevoFrameColor respectivamente. Ambos componentes pertenecen a la clase Sensor |

Tabla 102: Sensor.NuevosFrames

| Nombre | NuevoFrameEsqueleto |
|----------------------------|---|
| Clase | Sensor |
| Parámetros de entrada | <ul style="list-style-type: none"> E : SkeletonFrame – Frame con los datos de posición del esqueleto que ha detectado el sensor Kinect. |
| Parámetro de salida | N/A |
| Descripción del componente | Reenvia el frame al componente NuevoFrameEsqueleto de la clase Motor para que esta lo distribuya a las clases que lo necesiten. |

Tabla 103: Sensor.NuevoFrameEsqueleto

| Nombre | NuevoFrameColor |
|----------------------------|--|
| Clase | Sensor |
| Parámetros de entrada | <ul style="list-style-type: none"> E : ColorImageFrame – Frame con la imagen capturada por la cámara de color de Kinect. |
| Parámetro de salida | N/A |
| Descripción del componente | Reenvia el frame al componente NuevoFrameColor de la clase Motor para que esta lo distribuya a las clases que lo necesiten. |

Tabla 104: Sensor.NuevoFrameColor

iii. Núcleo del motor

1. Clase InterpreteDatos

| Nombre | JointCollection |
|----------------------------|---|
| Clase | InterpreteDatos |
| Parámetros de entrada | <ul style="list-style-type: none"> Puntos : JointCollection – Colección de datos posicionales del usuario que ha recabado el sensor Kinect. |
| Parámetro de salida | Un array de tipo double con las coordenadas de cada articulación recabada por el sensor expresadas en función de su distancia a la articulación “Spine” : double[] |
| Descripción del componente | Calcula la distancia de cada coordenada de cada articulación a la misma coordenada de la articulación “spine” para que todas las coordenadas sean sobre el |

el mismo centro, y después las almacena en un array de double, que es el tipo de dato admitido por la red de neuronas.

Tabla 105: InterpreteDatos.Joints2Double

2. Clase Capturador

| Nombre | ActivarGuardadoPostura |
|----------------------------|--|
| Clase | Capturador |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Cambia el modo del capturador a CapturarPostura |

Tabla 106: Capturador.ActivarGuardadoPostura

| Nombre | ActivarTemporizador |
|----------------------------|--|
| Clase | Capturador |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Reinicia el contador de segundos que han transcurrido desde que se activara el temporizador de la clase Capturador y luego lo inicia. |

Tabla 107: Capturador.ActivarTemporizador

| Nombre | CambiarTiempoinicio |
|----------------------------|--|
| Clase | Capturador |
| Parámetros de entrada | <ul style="list-style-type: none"> Segundos : int – Tiempo en segundos que deberá esperar el capturador desde que se pulsa el botón de iniciar captura hasta que pasa a modo capturarPostura |
| Parámetro de salida | N/A |
| Descripción del componente | Cambia el tiempo que esperará el capturador antes de pasar a modo capturarPostura |

Tabla 108: Capturador.CambiarTiempoinicio

| Nombre | CambiarTiempolimite |
|----------------------------|---|
| Clase | Capturador |
| Parámetros de entrada | <ul style="list-style-type: none"> Segundos : int – Tiempo en segundos que deberá esperar el capturador desde que se pulsa el botón de iniciar captura hasta que vuelve a modo reposo |
| Parámetro de salida | N/A |
| Descripción del componente | Cambia el tiempo que esperará el capturador antes de volver a modo reposo |

Tabla 109: Capturador.CambiarTiempolimite

| Nombre | CapturarPostura |
|-----------------------|--|
| Clase | Capturador |
| Parámetros de entrada | <ul style="list-style-type: none"> Nombre : String – Nombre que recibirá la postura que se va a capturar |

| Parámetro de salida | N/A |
|----------------------------|---|
| Descripción del componente | Este componente activa el temporizador para dar tiempo al usuario a colocarse en la postura deseada, después desactiva el reconocedor para no interferir y finalmente asigna el nombre de la postura al atributo del capturador para tenerlo listo una vez se capturen los datos posicionales. |

Tabla 110: Capturador.CapturarPostura

| Nombre | DesactivarCapturador |
|----------------------------|--|
| Clase | Capturador |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Cambia el modo del capturador a receso |

Tabla 111: Capturador.DesactivarCapturador

| Nombre | DesactivarTemporizador |
|----------------------------|---|
| Clase | Capturador |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Apaga el temporizador de la clase capturador para que deje de contar segundos. |

Tabla 112: Capturador.DesactivarTemporizador

| Nombre | getTiempolInicio |
|----------------------------|--|
| Clase | Capturador |
| Parámetros de entrada | N/A |
| Parámetro de salida | El tiempo que esperará el capturador antes de pasar a modo capturarPostura tras pulsar el botón de capturar posturas : int |
| Descripción del componente | Devuelve el valor del atributo TiempolInicio |

Tabla 113: Capturador.GetTiempolInicio

| Nombre | getTiempoFin |
|----------------------------|---|
| Clase | Capturador |
| Parámetros de entrada | N/A |
| Parámetro de salida | El tiempo que esperará el capturador antes de pasar a modo receso tras pulsar el botón de capturar posturas : int |
| Descripción del componente | Devuelve el valor del atributo TiempoLimite |

Tabla 114: Capturador.GetTiempoFin

| Nombre | nuevoFrameEsqueleto |
|----------------------------|---|
| Clase | Capturador |
| Parámetros de entrada | <ul style="list-style-type: none"> • Frame : SkeletonFrame – El frame con los datos del esqueleto recabados por Kinect. |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente abre los datos del frame, comprueba si |

se estaba siguiendo algún esqueleto, en caso de ser así se extraen los datos de dicho esqueleto y se crea una nueva **postura**, transformando previamente los datos mediante el componente **Joints2Doble** de la clase **InterpreteDatos**. Finalmente se almacena la nueva postura en la lista de posturas del **reconocedor**, se desactivan el **capturador** (mediante el componente **DesactivarCapturador**) y el temporizador (componente **DesactivarTemporizador**) y se notifica que se ha capturado una postura mediante el componente **NotificarPosturaCapturada** del motor

Tabla 115: Capturador.NuevoFrameEsqueleto

| Nombre | NuevoSegundo |
|----------------------------|--|
| Clase | Capturador |
| Parámetros de entrada | <ul style="list-style-type: none"> Sender : Object – La instancia que ha provocado el evento E : ElapsedEventArgs – Los parámetros del evento. |
| Parámetro de salida | N/A |
| Descripción del componente | Comprueba si el tiempo que ha pasado es igual a Tiempoinicio ó a Tiempolímite, ambos atributos de la clase capturador. En caso de coincidir con Tiempoinicio llama al componente ActivarCapturador , si coincide con Tiempolímite llama al componente DesactivarCapturador . En cualquier caso, finalmente añade uno al tiempo transcurrido. |

Tabla 116: Capturador.NuevoSegundo

3. Clase Reconocedor

| Nombre | ActivarReconocedor |
|----------------------------|---|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Pone el reconocedor en modo reconocer |

Tabla 117: Reconocedor.ActivarReconocedor

| Nombre | ActivarReconocimiento |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Este componente comprueba en qué modo se encuentra el reconocedor, si está en modo inválido comprueba que haya posturas en la lista de posturas. De ser así se entrena el motor mediante el componente EntrenarRed de la clase RedNeuronas y después notifica que se ha entrenado correctamente la red mediante el componente NotificarReconocedorListo de la clase Motor . En caso de que no hubiera posturas se notifica el error mediante el componente NotificarError de la clase |

Motor. Finalmente, e independientemente de las condiciones anteriores se desactiva el capturador mediante el componente **DesactivarCapturador** de la clase **Motor** y se llama al componente **ActivarReconocedor**.

Tabla 118: Reconocedor.ActivarReconocimiento

| Nombre | AgregarPosturaCapturada |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> Postura : Postura – La postura que se desea agregar a la lista de posturas del reconocedor |
| Parámetro de salida | N/A |
| Descripción del componente | Comprueba si existe ya una postura con ese nombre en el reconocedor. De ser así agrega los datos de la postura a los que ya tenía de la misma, en caso contrario añade la postura completa al diccionario de posturas. |

Tabla 119: Reconocedor.AgregarPosturaCapturada

| Nombre | AnadirPosturaAGesto |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> nombreGesto : String – El nombre del gesto al que se añadirá la postura nombrePostura : String – El nombre de la postura a añadir. Pos : Int – La posición que ocupará la postura dentro del gesto. esFinal : bool – Si la postura es la última del gesto o no. |
| Parámetro de salida | N/A |
| Descripción del componente | Comprueba si ya existe una postura con ese nombre en el diccionario de posturas del reconocedor. En caso de existir se cambian los valores nombreGesto, pos y esfinal de dicha postura por los recibidos por parámetro. En caso de no existir la postura se llama al componente NotificarError de la clase Motor y se notifica que no existe ninguna postura con ese nombre. |

Tabla 120: Reconocedor.AnadirPosturaAGesto

| Nombre | cambiarAlpha |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> alpha : Double – El nuevo valor alpha que se quiere asignar para entrenar la red. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarAlpha de la clase RedNeuronas . |

Tabla 121: Reconocedor.CambiarAlpha

| Nombre | cambiarNeuronasIntermedias |
|--------|----------------------------|
| Clase | Reconocedor |

| | |
|----------------------------|---|
| Parámetros de entrada | <ul style="list-style-type: none"> neuronas : int[] – Las neuronas que habrá en cada capa oculta de la red de neuronas. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarNeuronasIntermedias de la clase RedNeuronas . |

Tabla 122: Reconocedor.CambiarNeuronasIntermedias

| Nombre | cambiarNombrePostura |
|----------------------------|---|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> nombreAntiguo : String – El nombre de la postura que se desea cambiar. nombreNuevo : String – El nombre que se quiere asignar a la postura. |
| Parámetro de salida | N/A |
| Descripción del componente | Comprueba si existe en la lista de posturas una postura cuyo nombre coincide con nombreNuevo. De ser así se llama al componente NotificarError de la clase Motor para informar que no se puede asignar ese nombre a la postura. En caso de no existir se busca la postura cuyo nombre coincide con nombreAntiguo y se cambia por nombreNuevo. |

Tabla 123: Reconocedor.CambiarNombrePostura

| Nombre | cambiarTasaAprendizaje |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> tasa : Double – El nuevo valor para la tasa de aprendizaje que se quiere asignar a la red de neuronas durante su entrenamiento. |
| Parámetro de salida | N/A |
| Descripción del componente | Redirige la llamada al componente cambiarTasaAprendizaje de la clase RedNeuronas . |

Tabla 124: Reconocedor.CambiarTasaAprendizaje

| Nombre | comprobarFinGesto |
|----------------------------|---|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> postura : String – El nombre de la postura que se quiere comprobar si es la última del gesto actual. |
| Parámetro de salida | Verdadero si la postura es la última del gesto actual, falso en caso contrario. |
| Descripción del componente | Comprueba si la postura es la última del gesto que se está reconociendo en un determinado instante. |

Tabla 125: Reconocedor.ComprobarFinGesto

| Nombre | comprobarInicioGesto |
|-----------------------|---|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> postura : String – El nombre de la postura que se quiere comprobar si es la primera de algún gesto |
| Parámetro de salida | Verdadero si la postura es la primera de algún gesto, falso en caso contrario. |

| | |
|----------------------------|--|
| Descripción del componente | Comprueba si la postura tiene asignado un gesto y además si es la primera del mismo. |
|----------------------------|--|

Tabla 126: Reconocedor.ComprobarInicioGesto

| Nombre | comprobarPosturaEsperada |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> postura : String – El nombre de la postura que se quiere comprobar si es la siguiente del gesto actual. |
| Parámetro de salida | Verdadero si la postura es la siguiente del gesto actual, falso en caso contrario. |
| Descripción del componente | Comprueba si la postura es la siguiente del gesto que se está reconociendo en un determinado instante. |

Tabla 127: Reconocedor.ComprobarPosturaEsperada

| Nombre | desactivarReconocedor |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Pone el reconocedor en modo Reposo |

Tabla 128: Reconocedor.DesactivarReconocedor

| Nombre | eliminarPostura |
|----------------------------|---|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> postura : String – El nombre de la postura que se desea eliminar del reconocedor. |
| Parámetro de salida | N/A |
| Descripción del componente | Comprueba si la postura a eliminar existe en el diccionario de posturas del reconocedor , si existe, se elimina del diccionario y se llama al componente InvalidarRed de la clase Reconocedor . En caso de no existir se notifica el error llamando al componente NotificarError de la clase Motor . |

Tabla 129: Reconocedor.EliminarPostura

| Nombre | GetAlpha |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | El valor alpha de la red de neuronas : Double |
| Descripción del componente | Redirige la llamada al componente getAlpha de la clase RedNeuronas . |

Tabla 130: Reconocedor.GetAlpha

| Nombre | GetNeuronasIntermedias |
|-----------------------|---|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | Las neuronas que hay en cada capa oculta de la red de neuronas : Int[] |

| | |
|----------------------------|--|
| Descripción del componente | Redirige la llamada al componente getNeuronasIntermedias de la clase RedNeuronas . |
|----------------------------|--|

Tabla 131: Reconocedor.GetNeuronasIntermedias

| Nombre | GetTasaAprendizaje |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | La tasa de aprendizaje de la red de neuronas : Double |
| Descripción del componente | Redirige la llamada al componente getTasaAprendizaje de la clase RedNeuronas . |

Tabla 132: Reconocedor.GetTasaAprendizaje

| Nombre | ImportarPosturas |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> posturas : Dictionary<String, Postura> - El diccionario con las posturas que se desea importar. |
| Parámetro de salida | N/A |
| Descripción del componente | Cambia el diccionario actual de posturas de la clase reconocedor por el recibido por parámetro y llama al componente InvalidarRed . |

Tabla 133: Reconocedor.ImportarPosturas

| Nombre | InvalidarRed |
|----------------------------|---|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Pone el reconocedor en modo Invalido y llama al componente InvalidarRed de la clase RedNeuronas . |

Tabla 134: Reconocedor.InvalidarRed

| Nombre | NuevoFrameEsqueleto |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | <ul style="list-style-type: none"> frame : SkeletonFrame – El frame con los datos de posición del esqueleto recabados por el sensor Kinect. |
| Parámetro de salida | N/A |
| Descripción del componente | <p>Este componente extrae los datos del frame y comprueba si el sensor Kinect estaba siguiendo a algún usuario. En caso de ser así se llama al componente Joints2Double de la clase Interprete datos y el resultado de dicha llamada se pasa al componente ReconocerPostura de la clase RedNeuronas. Si esta última llamada devuelve algún resultado distinto a un string vacío se ha reconocido una postura por lo que se notifica usando el componente notificarPosturaReconocida de la clase Motor y se pasa a intentar reconocer un gesto. Para ello se utilizan los componentes comprobarInicioGesto,</p> |

comprobarFinGesto y **comprobarPosturaEsperada**. El reconocimiento se realiza siguiendo la máquina de estados explicada en el **Anexo II: Máquina de estados para el reconocimiento de gestos**.

Tabla 135: Reconocedor.NuevoFrameEsqueleto

| Nombre | RecuperarPosturas |
|----------------------------|--|
| Clase | Reconocedor |
| Parámetros de entrada | N/A |
| Parámetro de salida | El diccionario de posturas reconocibles : Dictionary<String, Postura> |
| Descripción del componente | Devuelve el diccionario de posturas reconocibles, es decir, el atributo posturas de la clase Reconocedor . |

Tabla 136: Reconocedor.RecuperarPosturas

4. *Clase RedNeuronas*

| Nombre | cambiarAlpha |
|----------------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | <ul style="list-style-type: none"> alpha : Double – El nuevo valor alpha que se quiere asignar para entrenar la red. |
| Parámetro de salida | N/A |
| Descripción del componente | Cambiar el atributo alpha de la clase RedNeuronas por el recibido por parámetro. |

Tabla 137: RedNeuronas.CambiarAlpha

| Nombre | cambiarNeuronasIntermedias |
|----------------------------|---|
| Clase | RedNeuronas |
| Parámetros de entrada | <ul style="list-style-type: none"> neuronas : int[] – Las neuronas que habrá en cada capa oculta de la red de neuronas. |
| Parámetro de salida | N/A |
| Descripción del componente | Cambiar el atributo neuronasIntermedias de la clase RedNeuronas por el recibido por parámetro. |

Tabla 138: RedNeuronas.CambiarNeuronasIntermedias

| Nombre | cambiarTasaAprendizaje |
|----------------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | <ul style="list-style-type: none"> tasa : Double – El nuevo valor para la tasa de aprendizaje que se quiere asignar a la red de neuronas durante su entrenamiento. |
| Parámetro de salida | N/A |
| Descripción del componente | Cambiar el atributo tasaAprendizaje de la clase RedNeuronas por el recibido por parámetro. |

Tabla 139: RedNeuronas.CambiarTasaAprendizaje

| Nombre | entrenarRed |
|-----------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | <ul style="list-style-type: none"> error : Double – El porcentaje máximo de fallo objetivo al entrenar la red. Posturas : Dictionary<string, Postura> - El |

| | |
|----------------------------|---|
| | diccionario de posturas que se utilizará para el entrenamiento. |
| Parámetro de salida | El porcentaje de error alcanzado tras el entrenamiento : Double |
| Descripción del componente | Este componente recorre el diccionario de posturas que recibe como entrada y va rellenando un nuevo diccionario que traducirá las salidas de la red con el nombre de la postura que tengan asociada. Después crea unas listas de valores de entrada y salida en base a los datos de las posturas que recibe como parámetro de entrada pero aleatorizando el orden que ocupará cada dato dentro de dichas listas. Finalmente se crea la red de neuronas con los parámetros de red almacenados en los atributos de la clase RedNeuronas y se realiza el entrenamiento con las listas de orden aleatorio. |

Tabla 140: RedNeuronas.EntrenarRed

| Nombre | GetAlpha |
|----------------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | N/A |
| Parámetro de salida | El valor alpha de la red de neuronas : Double |
| Descripción del componente | Devuelve el valor del atributo alpha de la clase RedNeuronas |

Tabla 141: RedNeuronas.GetAlpha

| Nombre | GetNeuronasIntermedias |
|----------------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | N/A |
| Parámetro de salida | Las neuronas que hay en cada capa oculta de la red de neuronas : Int[] |
| Descripción del componente | Devuelve el valor del atributo neuronasIntermedias de la clase RedNeuronas |

Tabla 142: RedNeuronas.GetNeuronasIntermedias

| Nombre | GetTasaAprendizaje |
|----------------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | N/A |
| Parámetro de salida | La tasa de aprendizaje de la red de neuronas : Double |
| Descripción del componente | Devuelve el valor del atributo tasaAprendizaje de la clase RedNeuronas |

Tabla 143: RedNeuronas.GetTasaAprendizaje

| Nombre | InvalidarRed |
|----------------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | N/A |
| Parámetro de salida | N/A |
| Descripción del componente | Cambia el estado de la red a Sin entrenar |

Tabla 144: RedNeuronas.InvalidarRed

| Nombre | ReconocerPostura |
|----------------------------|---|
| Clase | RedNeuronas |
| Parámetros de entrada | <ul style="list-style-type: none"> datos : double[] – Los datos posicionales que ha recabado el sensor Kinect tras ser transformados por el componente Joints2Double |
| Parámetro de salida | El nombre de la postura que se ha reconocido o un string vacío en caso de no haberse reconocido ninguna postura: String |
| Descripción del componente | Este componente pasa los datos que recibe como parámetro a la red de neuronas y calcula la salida. Después utiliza dicha salida al llamar al componente TraducirResultado y devuelve el resultado de dicho componente. |

Tabla 145: RedNeuronas.ReconocerPostura

| Nombre | ReconocerPostura |
|----------------------------|--|
| Clase | RedNeuronas |
| Parámetros de entrada | <ul style="list-style-type: none"> resultado : double[] – La salida de la red de neuronas. |
| Parámetro de salida | El nombre de la postura que se ha reconocido o un string vacío en caso de no haberse reconocido ninguna postura : String |
| Descripción del componente | Este componente busca en el diccionario de traducción de salidas a que postura le corresponde la salida que ha dado la red de neuronas y devuelve el nombre de la postura a la que corresponde o un string vacío en caso de no corresponder a ninguna postura. |

Tabla 146: RedNeuronas.ReconocerPostura

iv. Interfaz de Almacenamiento/Recuperación

1. Clase ManejadorFicheros

| Nombre | abrirFichero |
|----------------------------|---|
| Clase | ManejadorFicheros |
| Parámetros de entrada | <ul style="list-style-type: none"> Ruta : String – La ruta del fichero que se desea abrir. Modo : Byte – El modo en que se desea abrir el fichero, 0 para escritura ó 1 para lectura. |
| Parámetro de salida | N/A |
| Descripción del componente | Inicializa los atributos lector (si el modo es 1) ó escritor (si el modo es 0) para la ruta especificada en la entrada. |

Tabla 147: ManejadorFicheros.AbrirFichero

| Nombre | escribirPostura |
|-----------------------|---|
| Clase | ManejadorFicheros |
| Parámetros de entrada | <ul style="list-style-type: none"> Postura : Postura – La postura que se desea escribir en el fichero |

| | |
|-----------------------------------|--|
| Parámetro de salida | N/A |
| Descripción del componente | Escribe una postura en el fichero XML abierto actualmente según el esquema definido en el apartado E: Diseño de los ficheros de datos, del presente documento. |

Tabla 148: ManejadorFicheros.escribirPostura

| | |
|-----------------------------------|--|
| Nombre | exportarPosturas |
| Clase | ManejadorFicheros |
| Parámetros de entrada | <ul style="list-style-type: none"> Ruta : String – La ruta del fichero en el que se quieren guardar las posturas. Posturas : Dictionary<String, Postura> - El diccionario con las posturas que se desean exportar. |
| Parámetro de salida | N/A |
| Descripción del componente | Escribe el inicio de un documento XML en la ruta especificada, después recorre todo el diccionario de posturas llamando al componente escribirPostura por cada postura del diccionario. Finalmente se escribe el cierre del documento. |

Tabla 149: ManejadorFicheros.exportarPosturas

| | |
|-----------------------------------|--|
| Nombre | importarPosturas |
| Clase | ManejadorFicheros |
| Parámetros de entrada | <ul style="list-style-type: none"> Ruta : String – La ruta del fichero desde el que se quieren cargar las posturas. |
| Parámetro de salida | El diccionario con las posturas que se han leído desde el fichero. Dictionary<string, Postura> |
| Descripción del componente | Lee el inicio de un fichero XML en la ruta especificada y después llama al componente leerPostura hasta que alcanza el final del fichero. |

Tabla 150: ManejadorFicheros.importarPosturas

| | |
|-----------------------------------|---|
| Nombre | leerPostura |
| Clase | ManejadorFicheros |
| Parámetros de entrada | N/A |
| Parámetro de salida | La postura que se ha leído desde el fichero |
| Descripción del componente | Lee una postura desde el fichero XML abierto actualmente según el esquema definido en el apartado E: Diseño de los ficheros de datos, del presente documento. |

Tabla 151: ManejadorFicheros.leerPostura

d. Diagramas de secuencia de los principales casos de uso

i. UC-01: Capturar una postura

Nota: Dado que este caso de uso es muy extenso y se basa principalmente en el lanzamiento de eventos de forma externa al usuario he decidido dividirlo en tres partes, la primera la desencadena el usuario y pone en marcha el caso de uso, la siguiente la desencadena el temporizador, y la última la desencadena el sensor, al tener listo un nuevo frame del esqueleto.

1. UC-01: Parte 1

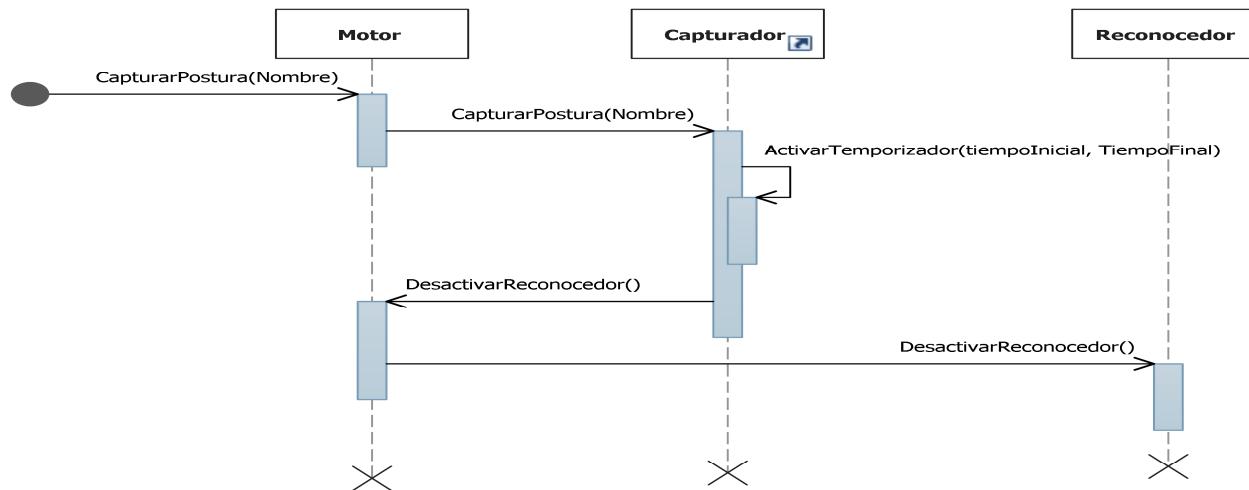


Diagrama 21: Diagrama de secuencia de UC-01, Parte 1

2. UC-01: Parte 2

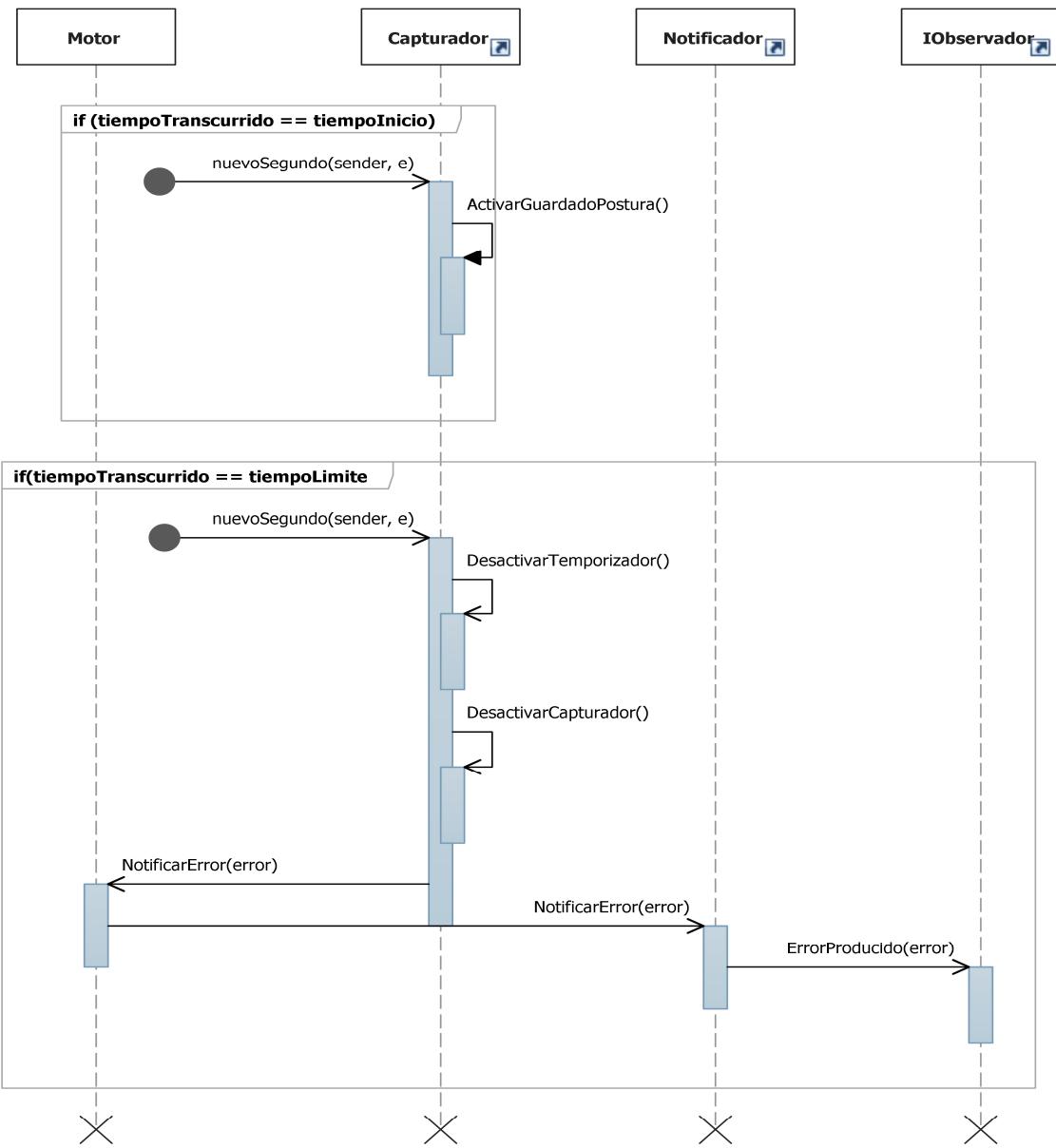


Diagrama 22: Diagrama de secuencia de UC-01, Parte 2

3. UC-01: Parte 3

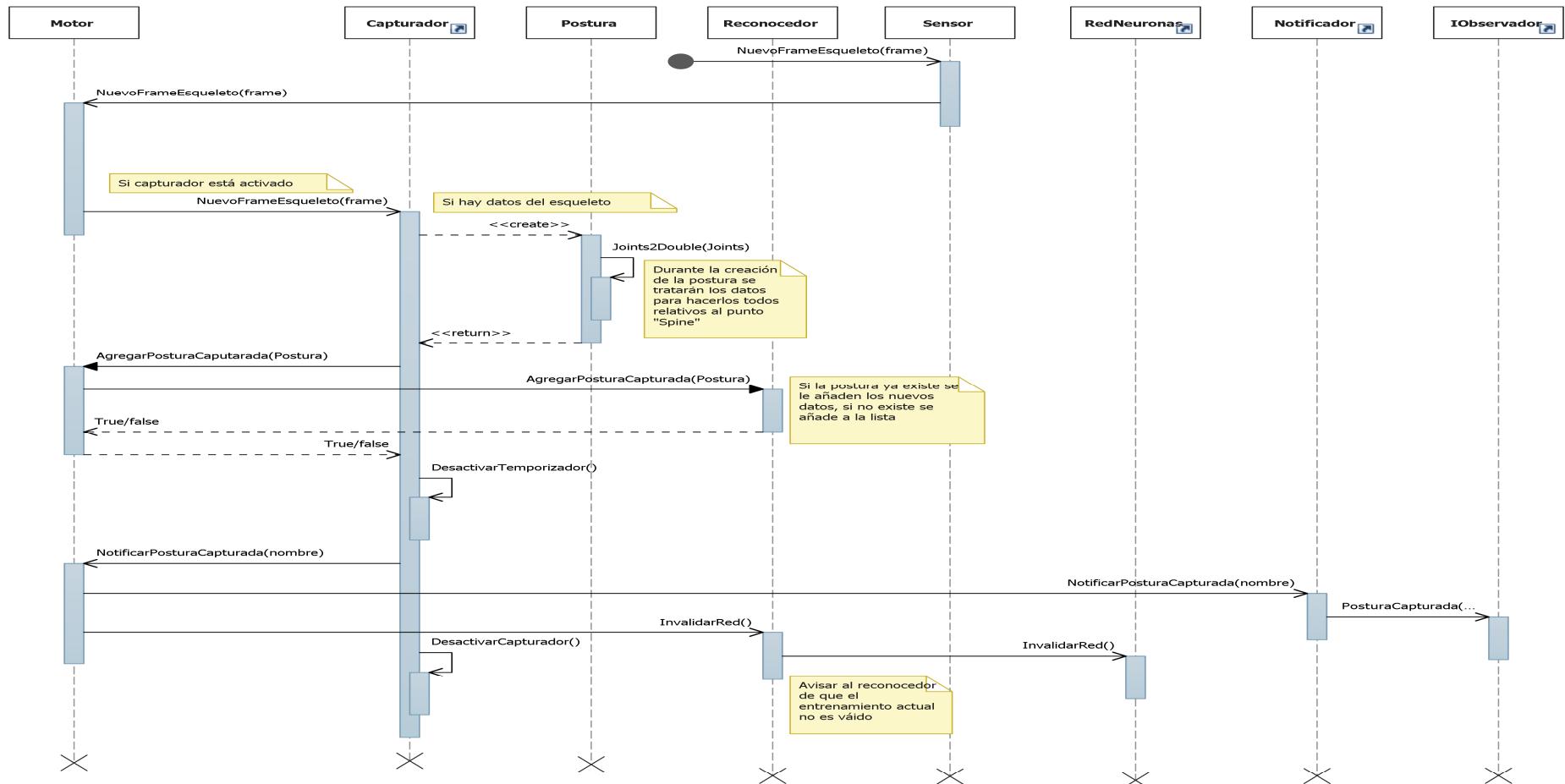


Diagrama 23: Diagrama de secuencia de UC-01, Parte 3

ii. UC-02: Ver posturas guardadas

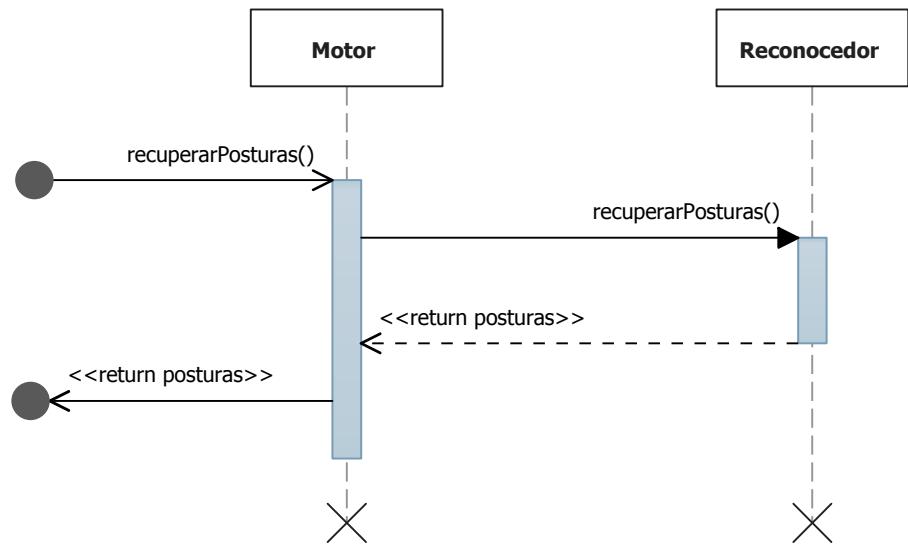


Diagrama 24: Diagrama de secuencia de UC-02

iii. UC-03: Cambiar el nombre de una postura

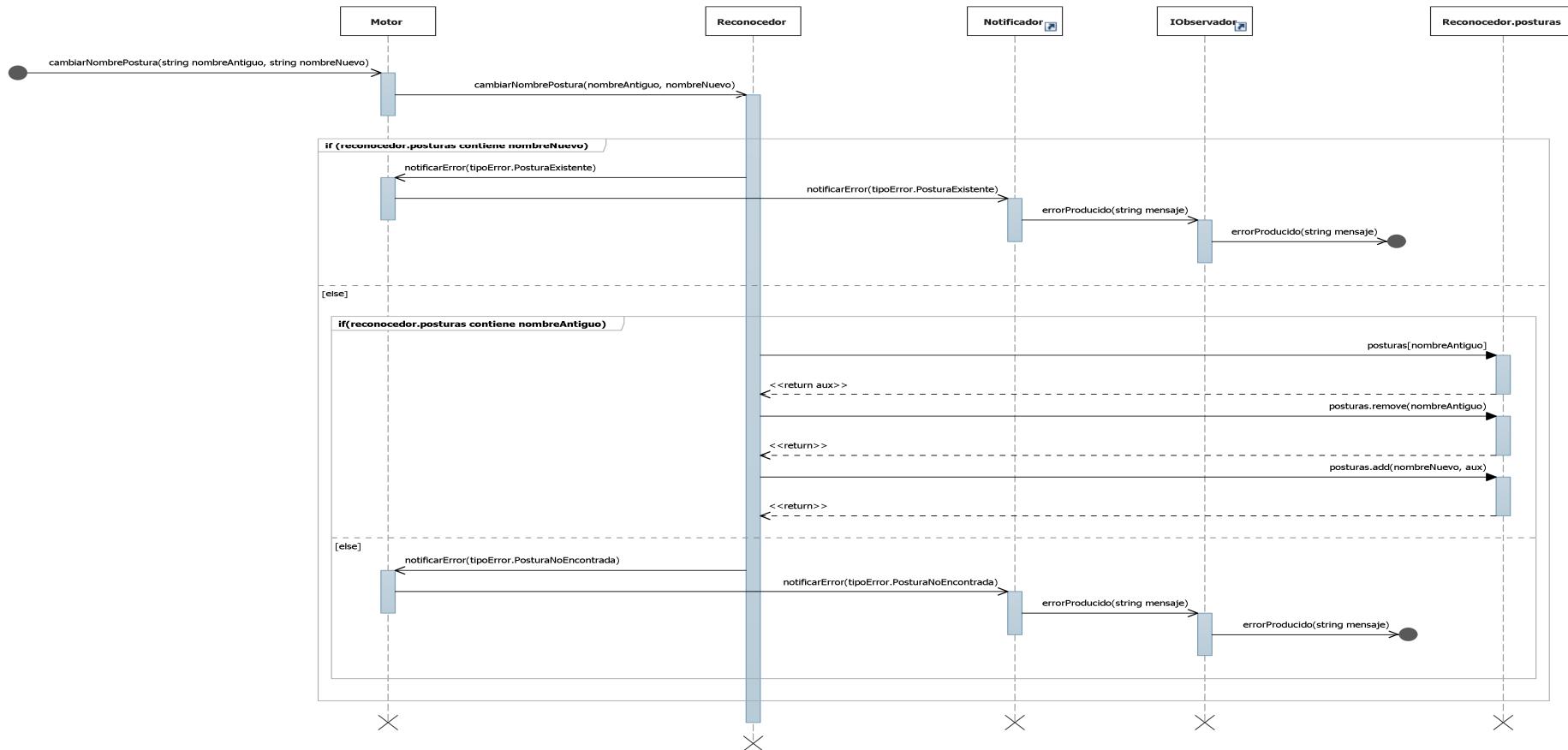


Diagrama 25: Diagrama de secuencia de UC-03

iv. UC-04: Eliminar una postura

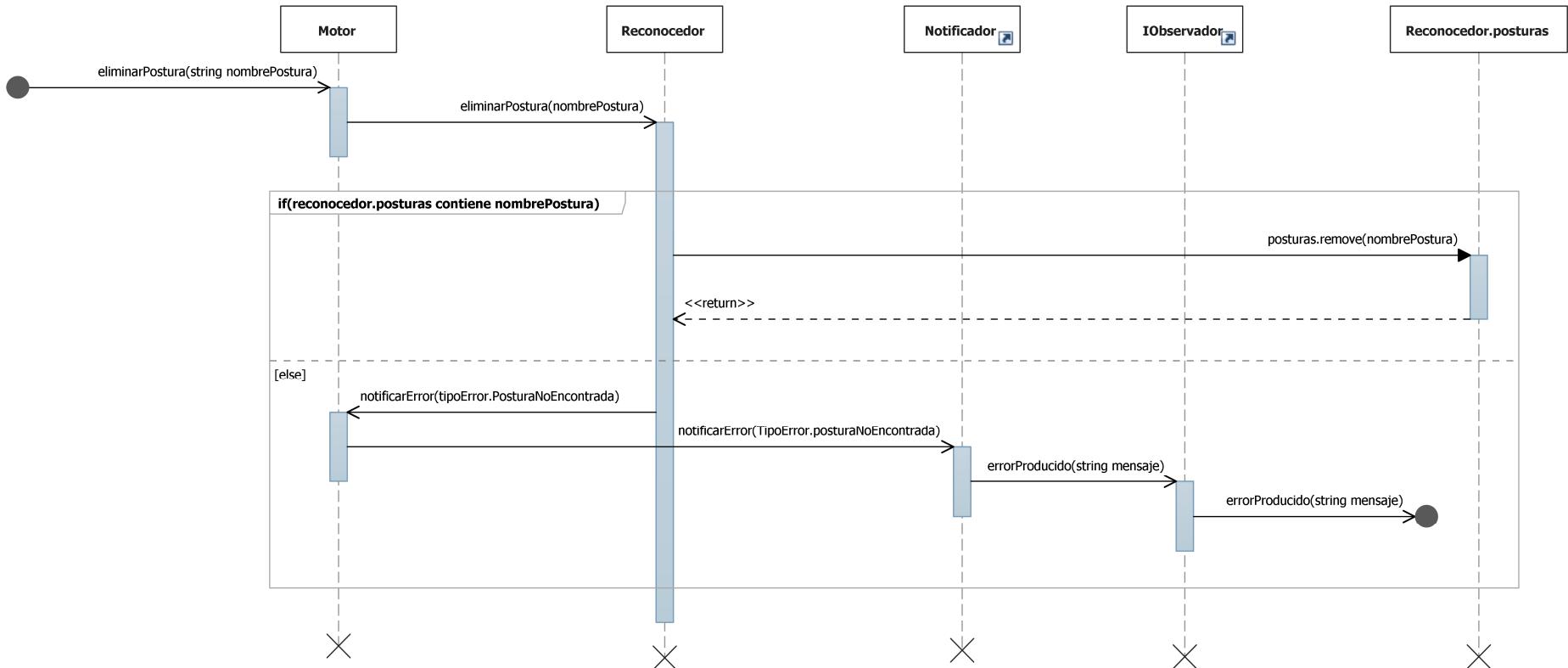


Diagrama 26: Diagrama de secuencia de UC-04

v. UC-05: Asignar un gesto a una postura

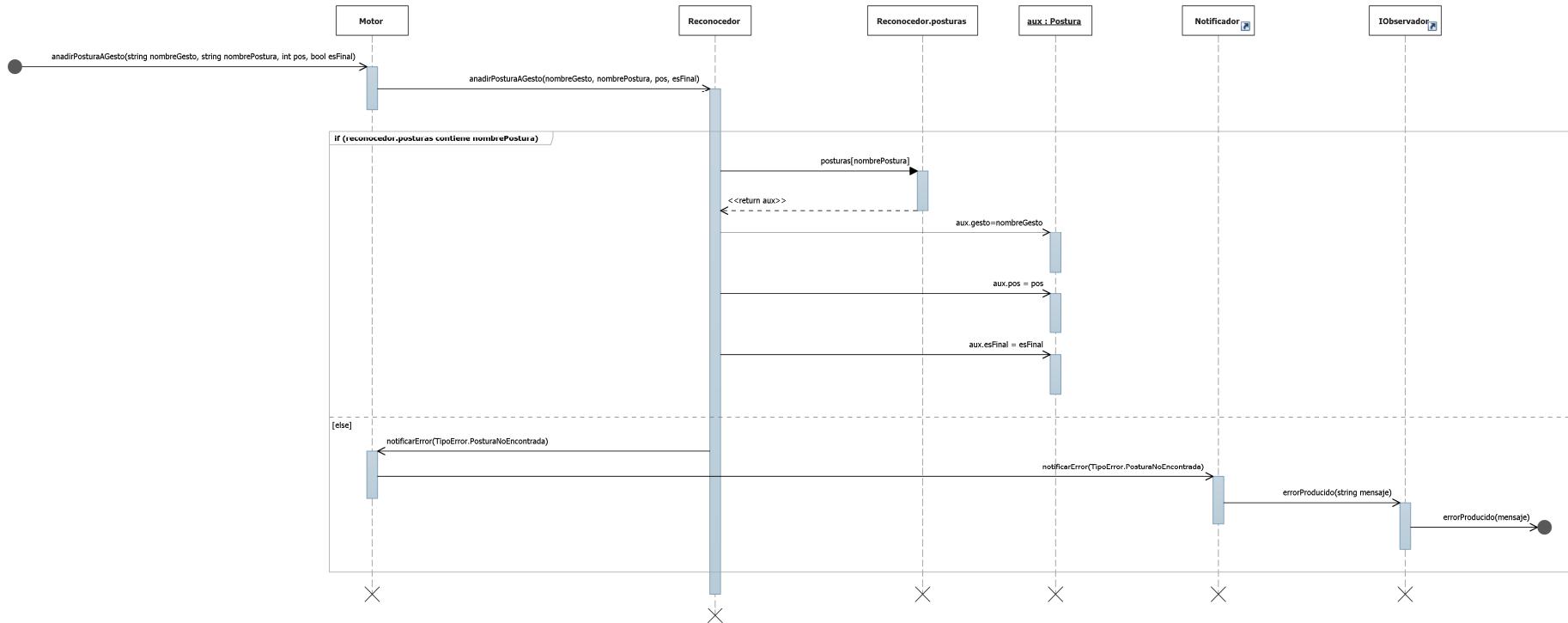


Diagrama 27: Diagrama de secuencia de UC-05

vi. UC-06: Eliminar el gesto de una postura

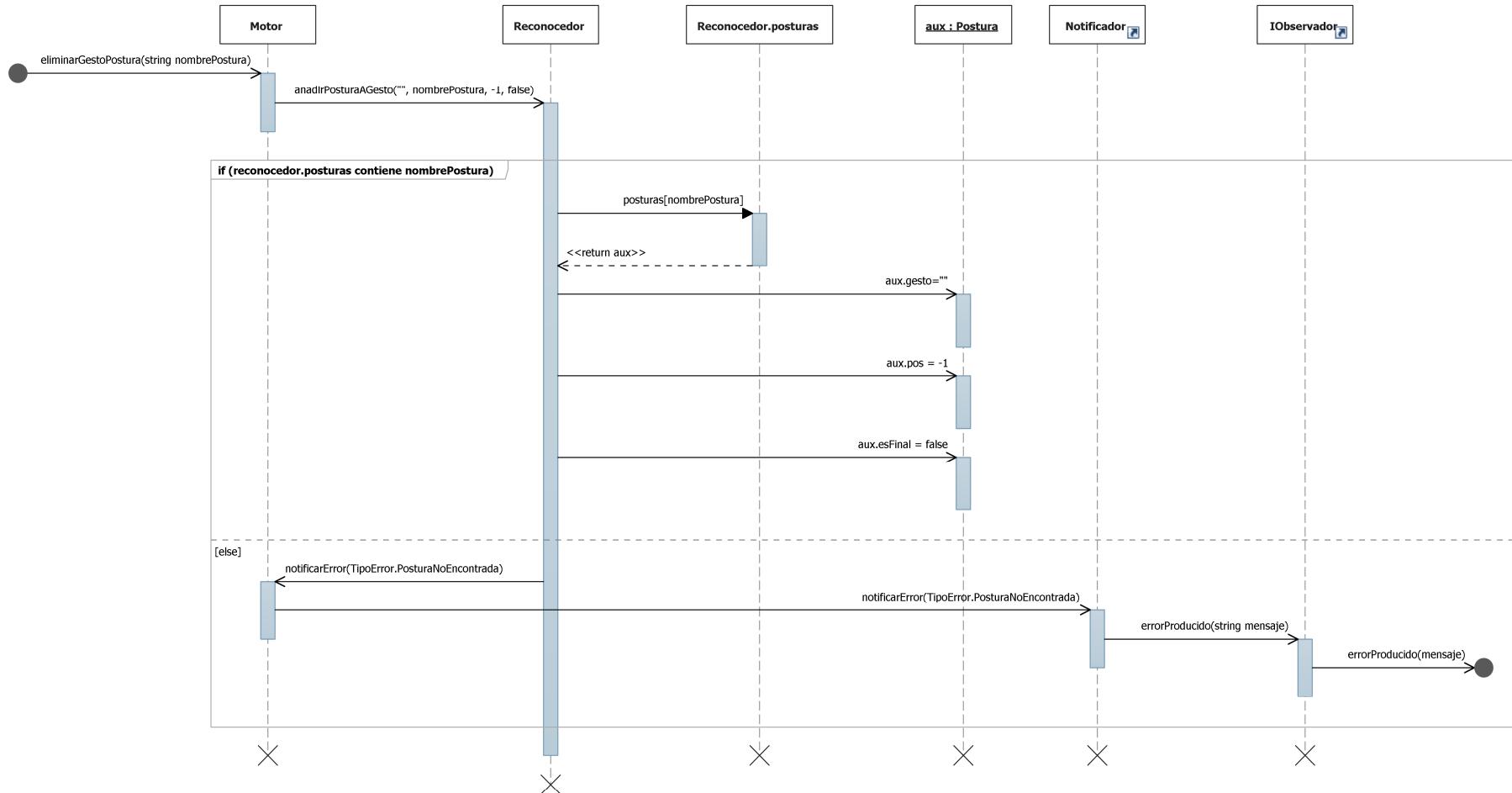


Diagrama 28: Diagrama de secuencia de UC-06

vii. UC-07: Cambiar tiempos de espera

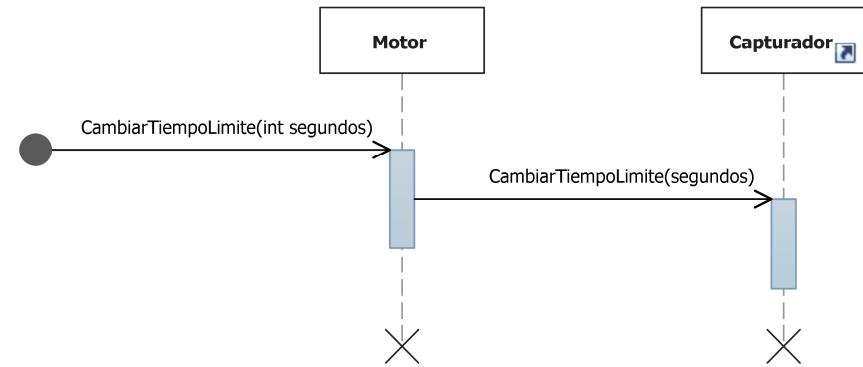
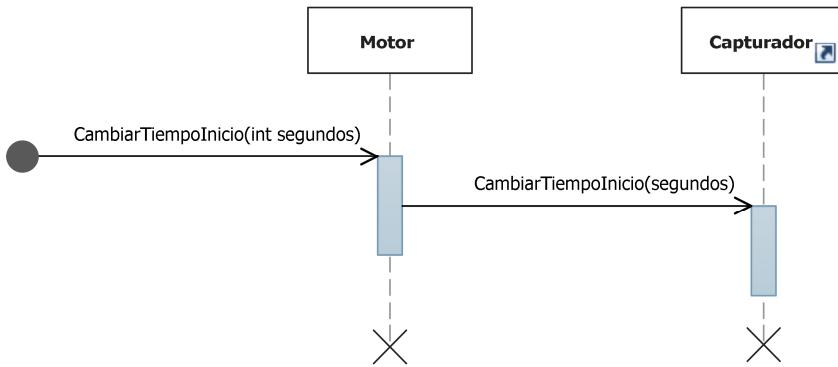


Diagrama 29: Diagramas de secuencia de UC-07

Nota: El diagrama de la izquierda representa el cambio del tiempo de espera antes de poner el capturador en modo capturar postura y el de la derecha el tiempo de espera antes de poner el capturador en modo reposo.

viii. UC-08: Activar reconocimiento de posturas y gestos

Nota: Al igual que pasaba con el caso de uso UC-01, este caso es demasiado extenso y también se basa en eventos por lo cual se separará de nuevo, esta vez en dos partes, la primera activada por el usuario y la segunda activada por el sensor, al tener un nuevo frame del esqueleto listo. Además en la segunda parte del diagrama he decidido omitir la secuencia para el reconocimiento de gestos ya que esta se puede ver de una forma mucho mas clara en el Anexo II: Diagrama de estados para el reconocimiento de gestos.

1. UC-08: Parte 1

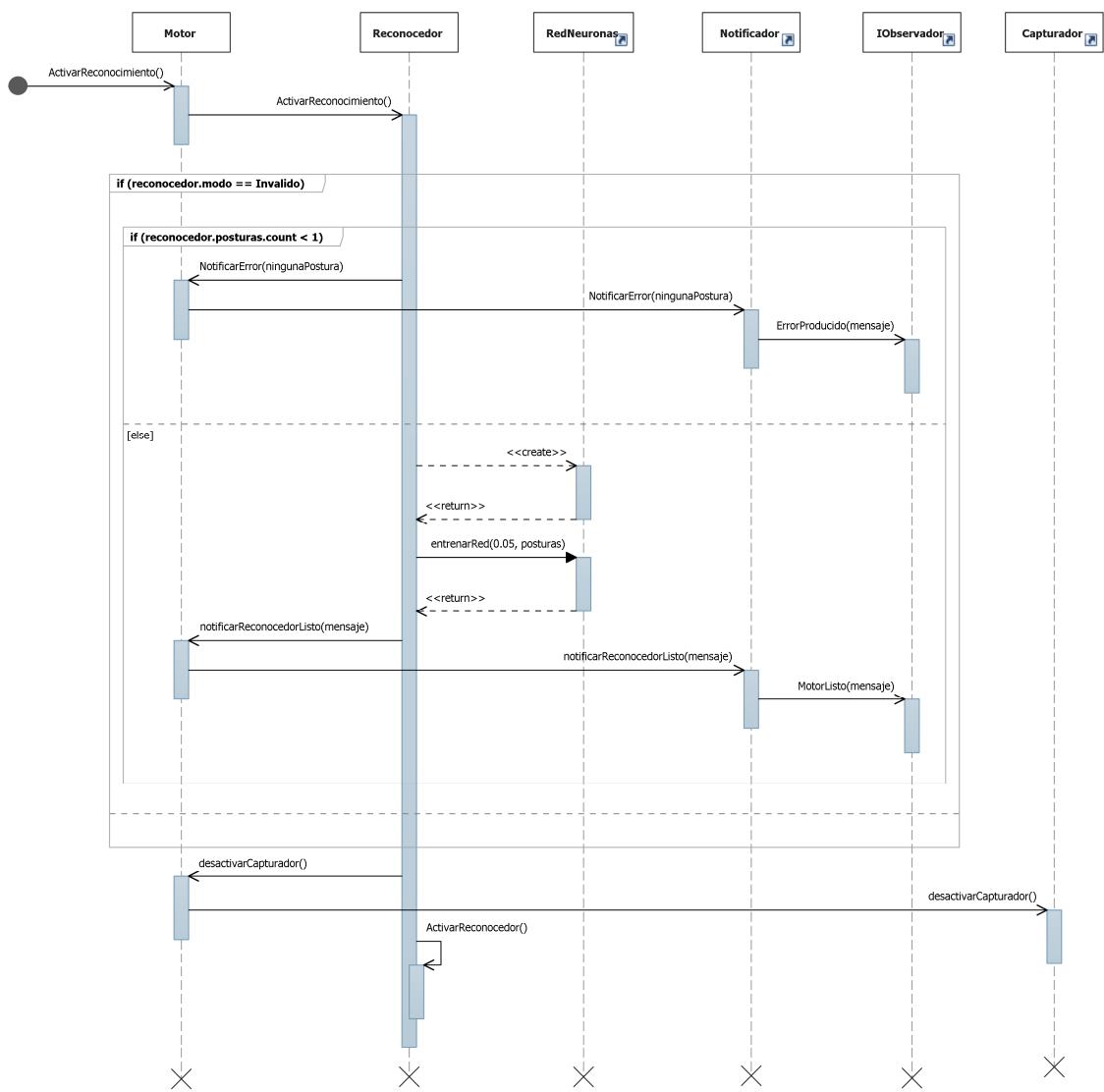


Diagrama 30: Diagrama de secuencia de UC-08, Parte 1

2. UC-08: Parte 2

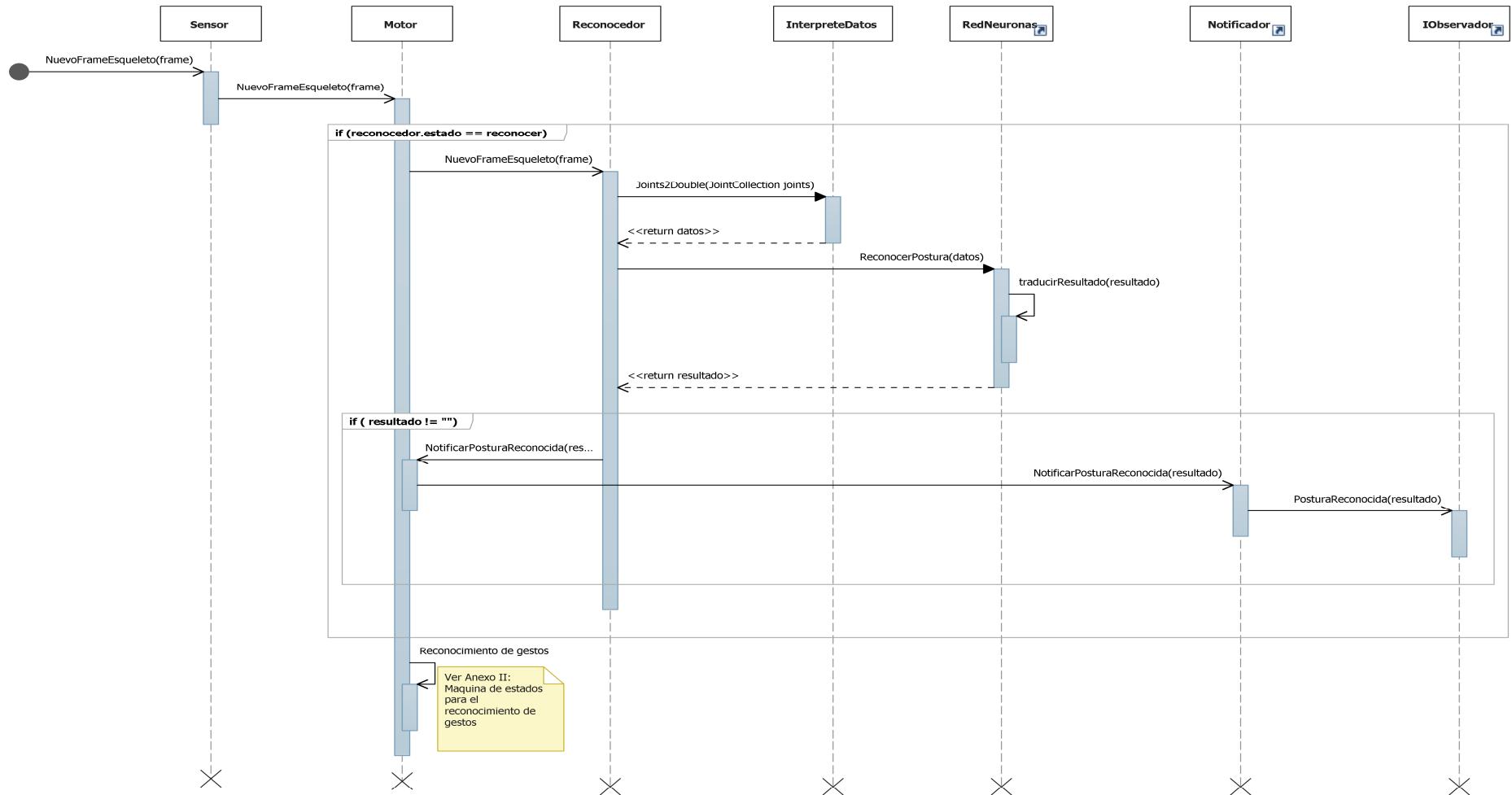


Diagrama 31: Diagrama de secuencia de UC-08, Parte 2

ix. UC-09: Exportar posturas

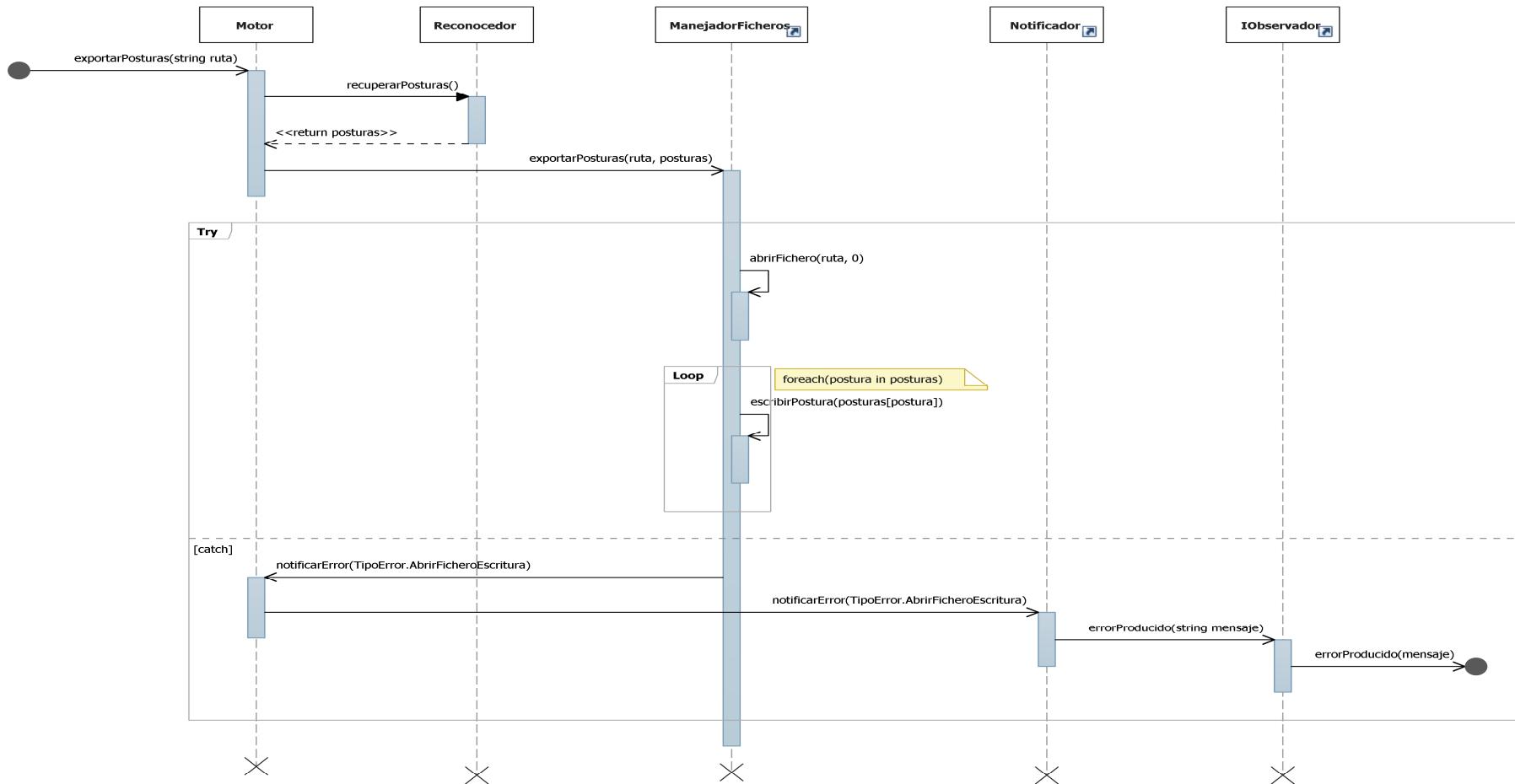


Diagrama 32: Diagrama de secuencia de UC-09

x. UC-10: Importar posturas

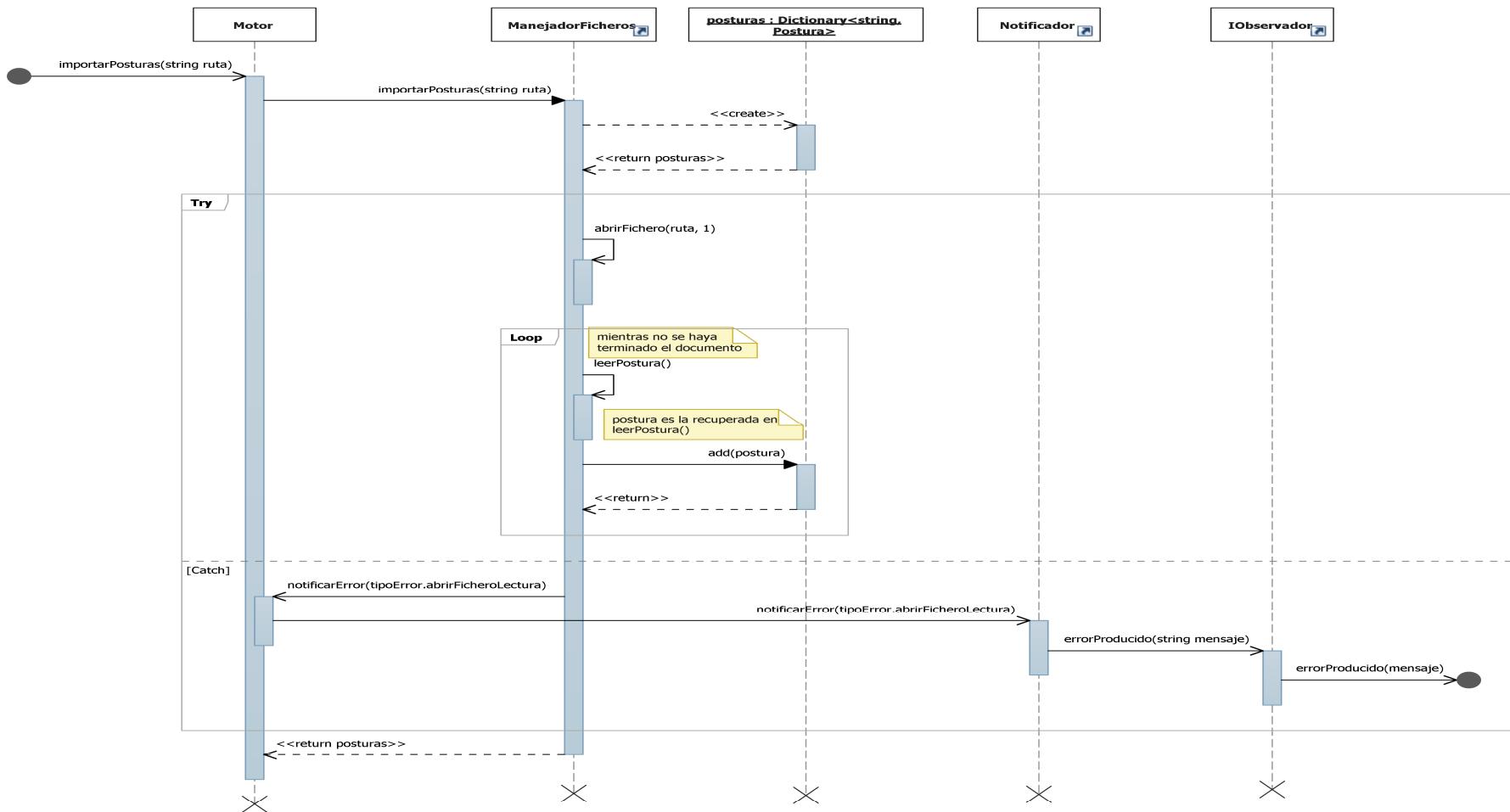


Diagrama 33: Diagrama de secuencia de UC-10

e. Diseño de los ficheros de datos

Para guardar los datos de las posturas dudé entre el formato XML y el formato binario que ofrece .NET. El motivo por el que finalmente escogí usar un formato XML es que este formato es fácil de leer desde un editor de texto normal y de modificar, lo cual implica que se pueden añadir posturas de varios ficheros de una manera sencilla con un simple editor de texto, mientras que con el fichero binario esa opción quedaría automáticamente descartada. Además en el framework .NET 4.0 existen multitud de herramientas que facilitan la lectura y escritura de archivos de formato XML, lo cual implica que no es necesario crear un parseador complicado.

En cuanto al formato interno del mismo fichero XML decidí crear el siguiente esquema, que recoge las características necesarias para el funcionamiento del motor de cada postura.

```
<Posturas>
    <Postura>
        <Nombre>Postura 1</Nombre>
        <Gesto>Gesto 1</Gesto>
        <Pos>3</Pos>
        <esFinal>false</esFinal>
        <Dato>0.02,0.1,-0.3...0.5</Dato>
        <Dato>0.025,0.1,-0.27...0.54</Dato>
    </Postura>
    ...
    <Postura>
    ...
</Postura>
</Posturas>
```

Como se puede ver, en dicho esquema por cada postura se inserta en el XML un “tag” “postura” con el nombre de la misma, el gesto al que pertenece si lo hay, la posición que ocupa en el mismo, si es final o no y una serie repetitiva de datos, que son las coordenadas del cuerpo respecto al centro de la columna de cada captura que se haya realizado de esa postura.

f. Mockups de interfaz de usuario

En cuanto a la interfaz de usuario dudé entre dejarla sin implementar, para que cada desarrollador tuviera la suya propia, implementar una sencilla de línea de comandos ó implementar una interfaz gráfica.

Finalmente opté por implementar la interfaz gráfica de manera que el sistema sirviera por si solo para algunas de las funciones que estaba contemplado (como por ejemplo el entrenamiento de deportistas, se puede realizar directamente utilizando la biblioteca con la interfaz de usuario por defecto).

Lo que buscaba en la interfaz es que fuera fácil gestionar las opciones del sistema y las posturas, y que además el usuario pudiera verse reflejado en la pantalla para estar seguro de que estaba realizando bien las posturas a la hora de capturarlas.

En base a esas especificaciones pensé en los siguientes mockups, los cuales finalmente fueron implementados casi en su totalidad.

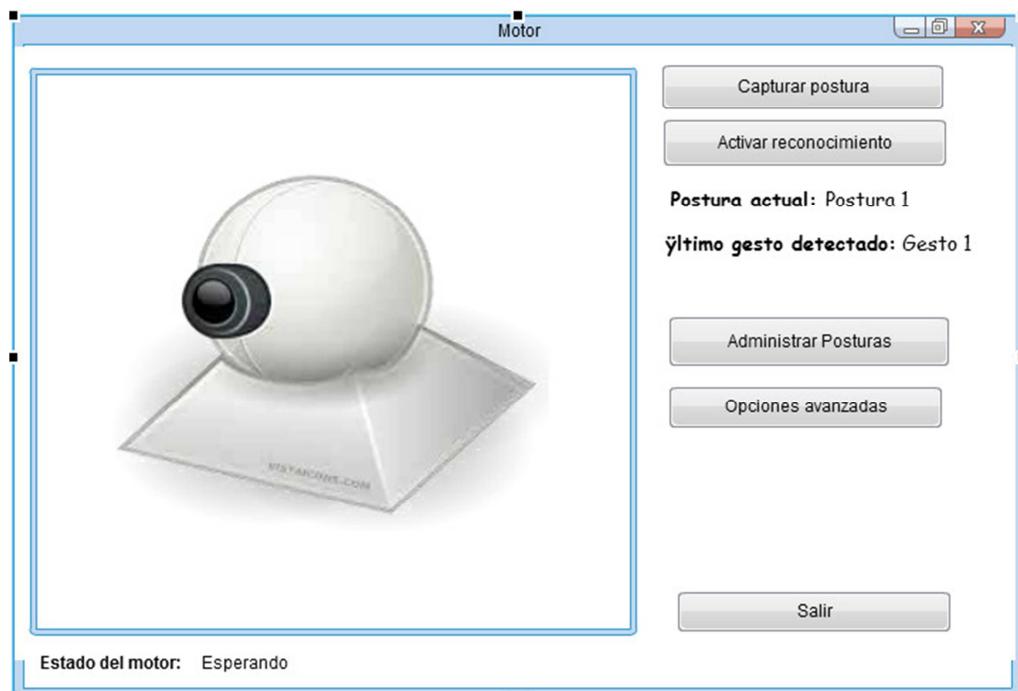
i. Pantalla principal

En la pantalla principal opté por añadir la imagen de color obtenida por el sensor Kinect, para que el usuario se viera realizando la postura. Esto es útil a la hora de capturarla para estar seguro de que la realiza correctamente.

Además añadí los botones a las funciones mas importantes del programa, la captura de posturas y la activación del modo reconocimiento. Debajo de dichos botones está un texto informativo que anuncia que postura o gesto se ha reconocido por última vez, para poder realizar pruebas o usar la interfaz ya para reconocer gestos.

Bajo dicho texto se encuentran los botones para entrar a los submenús de opciones avanzadas y para administrar las posturas.

Finalmente está el botón de salir que sale de la aplicación.



ii. Menú de administrar posturas

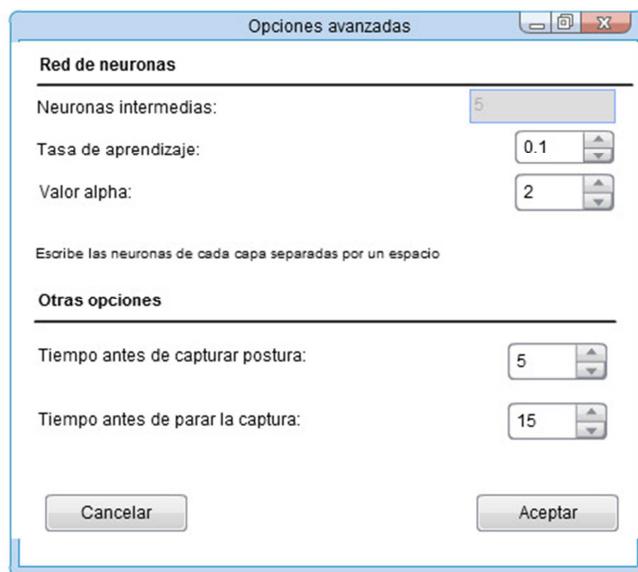
En esta pantalla se encuentran las diferentes opciones que permiten gestionar las posturas y los gestos asociados a las mismas. A la izquierda hay botones que permiten importar o exportar las posturas y una lista con todas las posturas que actualmente están en memoria principal. Cuando se selecciona alguna postura bajo su nombre aparecen el gesto asociado a la misma, la posición dentro del gesto y si es final o no.

En el lateral derecho, están los botones que dependen de la postura seleccionada. Dichos botones permiten al usuario cambiar el nombre de la postura, eliminarla o asignarla/desasignarla un gesto.



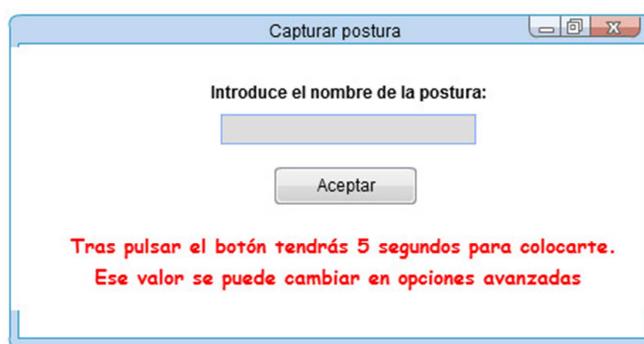
iii. Menú de opciones avanzadas

Como finalmente implementé una red de neuronas, y la eficacia de esta depende en gran medida de los parámetros de la propia red, así como del conjunto de entrada, en esta pantalla se permiten modificar dichos parámetros, así como el tiempo que espera el motor antes de capturar una postura y el que espera antes de dejar de intentar capturar.



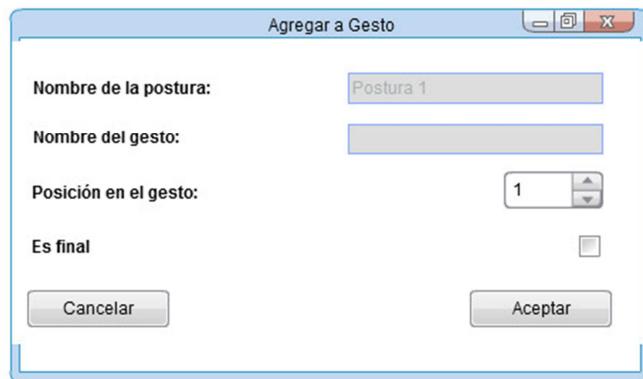
iv. Capturar una postura

Al pulsar el botón de capturar una postura se debe introducir el nombre de la postura a capturar, por lo que en la interfaz se debe mostrar una ventana que permita al usuario introducir dicho nombre.



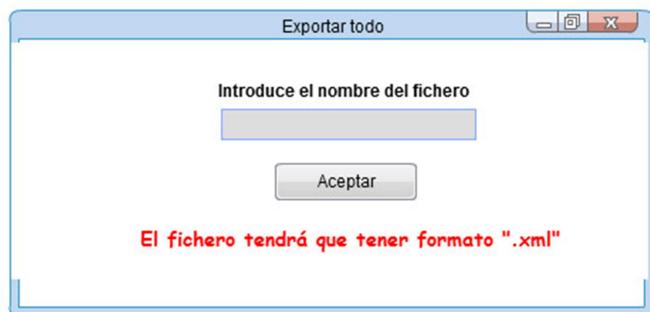
v. Agregar gesto

En la pantalla de administrar posturas se permite agregar una postura a un gesto, y para esto se necesitan los datos del gesto. Por ello elegí que se mostrara una nueva ventana que contuviera los campos necesarios para llenar con el gesto.



vi. Importar ó exportar posturas

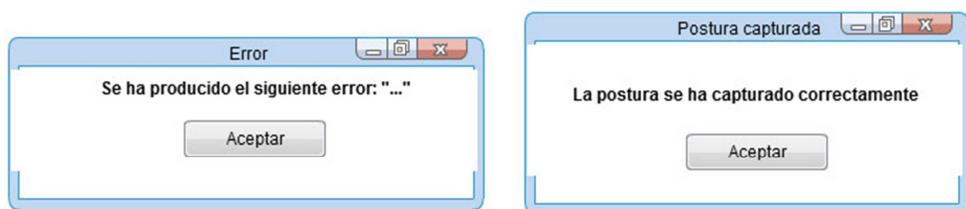
Dado que en ambos casos solo se necesita saber la ruta del fichero desde el que se va a importar o al que se va a exportar, ambas funcionalidades comparten la misma ventana, la cual contiene el campo en el que se introducirá la ruta.



vii. Mensajes de error/información

Por último pensé que el motor tendría que notificar algunas cosas al usuario, con lo que sería necesario un último tipo de ventana que contuviera los mensajes de que se ha producido un error o mensajes de información exitosa, como por ejemplo que se ha logrado capturar una postura.

Ambos tipos de mensaje comparten el siguiente formato de ventana:



6. Resultado de las pruebas de aceptación

La siguiente tabla resume el resultado de la realización de las pruebas definidas en el apartado 3.F del presente documento.

| ID | Nombre | Objetivo | Resultados |
|-------|---------------------------------------|---|-----------------------------|
| GP-01 | Grabar una postura | Se probará que el sistema es capaz de almacenar correctamente en memoria principal los datos sobre la posición de las diferentes partes del cuerpo en un instante determinado | Todas las pruebas superadas |
| GP-02 | Gestionar gestos asociados a posturas | Se probará que se pueden asignar y eliminar correctamente gestos a una postura. | Todas las pruebas superadas |
| GP-03 | Reconocer una postura | Se probará que el sistema es capaz de reconocer una postura que ha sido capturada previamente por el sistema | Todas las pruebas superadas |
| GP-04 | Reconocer un gesto | Se probará que el sistema es capaz de reconocer un gesto previamente definido por el usuario | Todas las pruebas superadas |
| GP-05 | Gestionar posturas | Se probará que el sistema puede cambiar el nombre de posturas o eliminarlas | Todas las pruebas superadas |
| GP-06 | Exportar posturas | Se probará el guardado de la información de las posturas a un fichero | Todas las pruebas superadas |
| GP-07 | Importar posturas | Se probará el guardado y la recuperación del estado del motor. | Todas las pruebas superadas |
| GP-08 | Opciones | Se probará que el motor puede guardar y cargar automáticamente las posturas capturadas y que se puede modificar el tiempo de espera para la captura | Todas las pruebas superadas |

Tabla 152: Resumen del resultado de las pruebas

7. Conclusiones

a. Conclusiones de los resultados

El presente Trabajo de Fin de Grado (TFG) consiste en el prototipado de una aplicación que utilice el sensor Kinect de Microsoft. Como aplicación escogí un motor de captura y reconocimiento de gestos y posturas puesto que dicho motor permitiría a los usuarios definir las posturas y gestos en tiempo de ejecución, teniendo por tanto la aplicación muchas y diversas utilidades (Facilitar el desarrollo de aplicaciones mayores, fisioterapia, entrenamiento de deportistas...).

Este trabajo me ha servido para aplicar la mayoría de conocimientos que he adquirido a lo largo de toda la carrera. He podido poner en práctica como sería el ciclo de vida casi al

completo de todo un sistema desde que se plantea el problema hasta que el sistema estaría listo para el paso a producción.

Además, el TFG involucraba el sensor Kinect, que es una tecnología con la que no estaba familiarizado lo cual me ha obligado a estudiar su funcionamiento y a mejorar mis habilidades de programación.

También me he tenido que familiarizar con los lenguajes en los que está desarrollado el proyecto (C# y XAML para la interfaz), ya que tan solo tenía conocimientos básicos de uno de ellos de experiencias anteriores en prácticas que había realizado a lo largo de la carrera.

Por último, para la realización del reconocimiento he tenido que valorar entre diferentes algoritmos que podrían servir y conocer sus ventajas e inconvenientes, lo cual me ha aportado una visión mas amplia del problema y me ha ayudado a decantarme por las redes de neuronas (en gran medida porque ya estaba familiarizado con su algoritmo y sabía que dicho sistema podría dar buenos resultados en este tipo de sistema). Cabe destacar, que también he tenido que familiarizarme con la biblioteca de clases de código abierto de Aforge.Net, que es un conjunto de clases que implementan algoritmos de inteligencia artificial. Esto fue necesario porque no tenía recursos (tiempo principalmente) para el desarrollo completo de un perceptrón multicapa.

En cuanto a los objetivos del TFG creo que he cumplido todos los objetivos:

-El sistema permitirá al usuario definir sus propios gestos y posturas en tiempo de ejecución.

-El sistema reconocerá las posturas y gestos que el usuario haya definido.

-El sistema contará con mecanismos que permitan personalizar la interfaz de usuario así como incluir el sistema completo como un componente en aplicaciones mayores.

Como se puede comprobar utilizando la aplicación, los dos primeros objetivos se cumplen (el segundo en un alto porcentaje de las ocasiones, ya que todos los sistemas de reconocimiento de gestos tienen un pequeño porcentaje de fallo). En cuanto al último objetivo, para poder cumplirlo implementé el patrón de diseño “observer” y asigné las responsabilidades de actuar como controlador y entrada del sistema a la clase Motor, lo cual permite que el sistema sea integrado como un subsistema en otros sistemas mayores.

b. Líneas futuras del sistema

Aunque he cumplido los objetivos del TFG, ha habido varias funcionalidades que por falta de tiempo no he podido implementar pero me hubiera gustado. En una nueva versión de este prototipo sería útil añadir las siguientes funcionalidades ó cambios:

- **Paralelizar el procesamiento de la imagen con el reconocimiento de las posturas.** Esto sería útil puesto que en los ordenadores actuales la tendencia va claramente hacia los procesadores de varios nucleos y el sistema actualmente solo aprovecha uno de ellos. Estas dos tareas son independientes entre si, así que se podrían paralelizar asignando un procesador a cada tarea.

- **Automatizar la extracción de posturas intermedias que definen un gesto.** Actualmente el usuario tiene que definir todas las posturas que componen un gesto, desde la primera hasta la última. Sería útil diseñar un algoritmo que dadas la posición inicial y final del gesto, fuera capaz de extraer varias posturas intermedias (con un número de ellas definido por el usuario) al realizar el usuario el gesto, de modo que el usuario no tuviera que involucrarse en dicho proceso de una forma tan activa.
- **Permitir que una postura pertenezca a varios gestos.** Actualmente cada postura solo puede tener un gesto asociado, sería útil que una postura pudiera pertenecer a varios gestos, ya que hay muchos gestos que comparten posturas intermedias.
- **Aprovechar el reconocimiento de voz del sensor Kinect.** Aunque fue una de las primeras ideas que tuve pronto la descarté para centrarme en los componentes básicos, sería muy interesante añadir soporte para algunos comandos de voz que puedan sustituir al temporizador y a algunos botones de la interfaz gráfica, así el usuario en vez de pulsar el botón de capturar postura y correr a colocarse en una zona adecuada, solo tendría que decir en voz alta y clara una instrucción cuando ya esté colocado en la postura deseada.

8. Anexo I: Posturas comunes

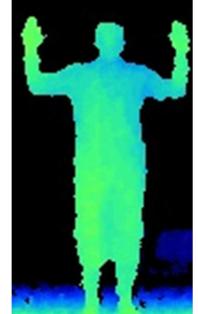
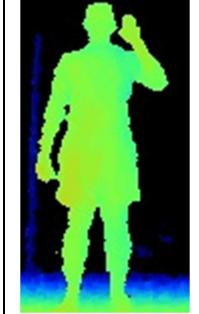
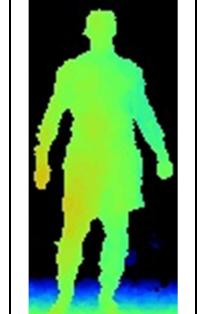
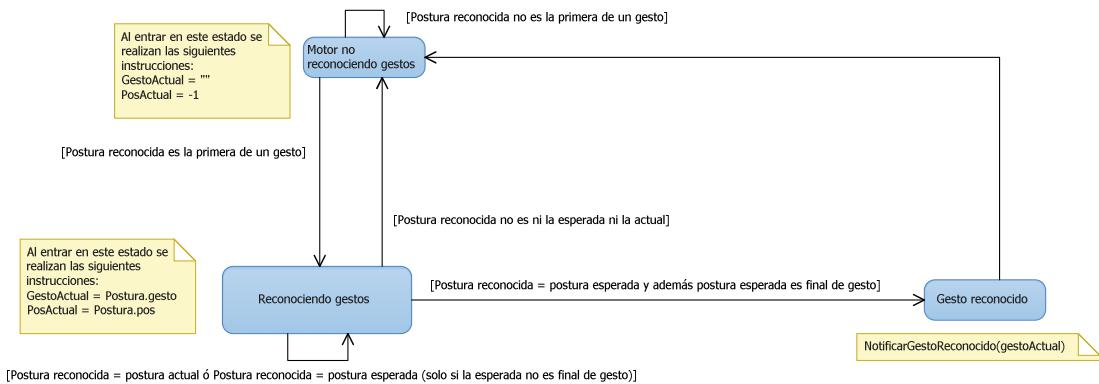
| | | | | |
|---|---|---|--|---|
|  |  |  |  |  |
| Brazos extendidos | Brazos levantados | Brazo derecho pegado al cuerpo | Brazo derecho separado del cuerpo | De pie normal |

Tabla 153: Posturas comunes

9. Anexo II: Maquina de estados del reconocimiento de gestos



10. Anexo III: Manual de usuario

El manual de usuario se encuentra en el fichero “Manual de usuario.docx” que se encuentra en la carpeta “Documentación” del proyecto.

11. Anexo III: Bibliografía

- [1] Microsoft, Kinect For Windows SDK. <http://www.microsoft.com/en-us/kinectforwindows/>
- [2] Codeplex, Kinect SDK Dynamic Time Warping (DTW) Gesture recognition. <http://kinectdtw.codeplex.com/>
- [3] Microsoft, Kinect For Windows Programming guide. <http://msdn.microsoft.com/en-us/library/hh855348.aspx>
- [4] Aforge.net, Biblioteca de clases Aforge.Net <http://aforgenet.com/>
- [5] Daniel Ramage, Hidden Markov Models Fundamentals. <http://cs229.stanford.edu/section/cs229-hmm.pdf>
- [6] Jose Mª Valls, Perceptrón multicapa. <http://eva.evannai.inf.uc3m.es/et/docencia/rn-inf/documentacion/Tema3-MLP.pdf>
- [7] Pavel Senin, Dynamic Time Warping Algorithm Review. <http://csl.ics.hawaii.edu/techreports/08-04/08-04.pdf>