Proyecto de Realidad Virtual: Realidad Aumentada en Herramientas de Robótica

Gonzalo Gabriel Fernández*
Facultad de Ingeniería, Universidad Nacional de Cuyo
Mendoza, Argentina
Email: *fernandez.gfg@gmail.com

Resumen

El presente documento detalla el proceso y producto final de la aplicación móvil desarrollada. Consiste en una aplicación educativa de realidad aumentada para dispositivos móviles con sistema operativo Android, que se comunica mediante protocolo TCP/IP con un programa desarrollado por el usuario en su computador con MATLAB, utilizando el toolbox *Robotics, Vision & Control* desarrollado por Peter Corke. El objetivo de la aplicación es la simulación, en un entorno de realidad aumentada, de los robots manipuladores serie estudiados en las cátedras vinculadas a la robótica.

El informe posee la siguiente estructura: Primero una descripción de los aspectos generales de la aplicación desarrollada y sus objetivos. Luego una descripción muy general de las escenas que la conforman, seguido de una sección con detalles más específicos de los componentes de cada una de esas escenas. Después, una descripción de todos los programas escritos a lo largo del proyecto en lenguaje C#, y una descripción de como se utilizaron las animaciones de Unity. Luego de todo esto una sección con los diagramas UML del proyecto, que se encuentran al final, con el objetivo de ayudar al lector con la comprensión de la dinámica entre los distintos componentes del proyecto. Luego, una explicación breve de como se trabajó con los modelos 3D seguido de la explicación del desarrollo en MATLAB. Por último, el modo en que se utiliza la aplicación, las conclusiones obtenidas, y el trabajo a futuro.

ÍNDICE

ÍNDICE DE FIGURAS

I. Introducción

La robótica es una de las ramas más importante de la carrera ingeniería en mecatrónica. Es un área de la industria muy interesante, con gran crecimiento y amplia salida laboral. En su estudio, se requiere el análisis de estructuras con cierto grado de complejidad en el espacio, y estas estructuras tridimensionales se someten a diversas transformaciones algebraicas que convierten el estudio en un desafío.

Con el fin de ayudar a esta comprensión de la robótica en espacio tridimensional, se propuso crear una herramienta de software que facilite la visualización de los manipuladores robóticos en el espacio y su movimiento. El medio seleccionado para llevar a cabo el desarrollo de esta consigna fue la realidad aumentada, por estar en auge, y además por dar una idea de su desempeño, por ejemplo, en el entorno real sobre el que el robot interactuaría en su posterior implementación física.

En la cátedra Robótica I y II, en la Facultad de Ingeniería de la Universidad Nacional de Cuyo, se utiliza como herramienta didáctica el toolbox *Robotics, Vision & Control* desarrollado por Peter Corke para MATLAB. Dicho toolbox posee distintas herramientas, pero en la cátedras mencionadas se realizan en profundidad análisis de la cinemática y dinámica de robots manipuladores serie. Este toolbox permite la simulación de los robots, tanto en una representación simple de la cadena cinemática con cilindros y prismas, como con los modelos 3d en formato STL del robot. Esta simulación es sobre un entorno virtual que es el visualizador 3D provisto por MATLAB, el cual consta simplemente del objeto en estudio con fondo blanco, no es el objetivo del software un entorno más complejo.

Considerando que en el ámbito académico en que se desarrolló el trabajo se utiliza este toolbox, se decidió tomar esta herramienta como punto de partida, y construir, con el motor de videojuegos multiplataforma Unity y el SDK (software development kit) Vuforia para realidad aumentada, una aplicación para teléfonos con sistema operativo Android que permita la visualización en realidad aumentada de los manipuladores que se simulan en MATLAB.

II. Descripción general de la aplicación

La aplicación, como ya se mencionó previamente, fue desarrollada para dispositivos móviles con sistema operativo Android. Fue desarrollada en el motor de videojuegos multiplataforma Unity, versión 2019.2.1f1, mediante su editor, y el IDE Visual Studio 2017 para la programación de scripts en lenguaje de programación C#. Para las herramientas de realidad aumentada se utilizó el SDK Vuforia 8.5.9, integrado en el editor de Unity.

La aplicación consta de tres escenarios distintos: Un menú principal, donde se encuentran toda la configuración de la comunicación y el robot a simular; una pantalla de carga, que se ejecuta en todos los cambios de escena y se expone una barra de carga para ver el estado en que se encuentra el levantamiento de recursos; y un último escenario en que se muestra la cámara y entra en acción Vuforia, mostrando en realidad aumentada el modelo 3D del robot a simular.

III. Descripción de escenas

El desarrollo en Unity consiste en la elaboración de diferentes escenas que se ejecutaran secuencialmente según cuáles sean las acciones del usuario. A continuación se expone cada una de las utilidas para la interfaz gráfica planteada.

III-A. Menú principal: MainMenu

En la figura 23 se expone la primer escena que requiere interacción con el usuario. Lo primero que se expone es el nombre de la aplicación, para la presentación se eligió el nombre *UnitLab: MATLAB + Unity*.

A la derecha de la escena se encuentra una imagen del robot a simular con el nombre del fabricante y nombre del modelo. Dado la limitante del tiempo en el trabajo, se decidió trabajar con un solo robot que es el IRB 910SC del fabricante ABB, y plantear la interfaz para a futuro implementar otros robots comerciales que también se utilizan en las cátedras de Robótica. También se planteó un botón de "Más información", donde en el futuro se implementaría una escena distinta donde se expongan otros detalles técnicos del robot que se está seleccionando.

A la izquierda se encuentra la configuración de la comunicación con MATLAB, donde se la habilita si se desea utilizar la aplicación con MATLAB, o se la deshabilita si se desea simular en un modo manual con *sliders*. También se encuentran dos casilleros editables por el usuario, donde se debe colocar la dirección IP del dispositivo que se está utilizando y el puerto por el que se establecerá la comunicación TCP/IP.

Por último se encuentran dos botones, a la izquierda el botón "Quit", para permitirle al usuario salir de la aplicación; y un botón "Start" para pasar a la escena de simulación contemplando la configuración planteada por el usuario.

III-B. Escena de carga: Loading

La escena de la figura 24, se utilizó como pantalla de transición entre una escena y otra, para que la aplicación no quede detenida mientras se cargan los recursos necesarios para mostrar en pantalla la escena siguiente. Se muestra en la parte superior el nombre de la aplicaicón, en la zona inferior de la pantalla los logos de las herramientas de software utilizadas (Unity, Vuforia y MATLAB), y en el centro una barra que muestra el estado en que se encuentra la carga.

III-C. Simulación manual: ManualScene

El usuario accede a la escena de la figura 25 cuando decide, en el menú principal, que ejecutará la simulación sin conexión con MATLAB. Por lo tanto, esta escena permite al usuario mover el robot en forma manual mediante los 4 *sliders* que se observan en la zona inferior de la pantalla. Además de ésto, arriba a la izquierda se encuentra nuevamente el nombre del fabricante y el modelo del robot que se está simulando; y arriba a la derecha un botón "Back" con el que el usuario puede volver al menú principal. Por supuesto, el componente principal de la escena se encuentra en el fondo, que es la cámara del dispositivo móvil ejecutada por Vuforia, donde se visualizará el robot.

III-D. Simulación con MATLAB: MatlabScene

La escena de la figura 26 es idéntica a la escena *ManualScene*, solo que esta vez el usuario accederá si en el menú principal habilitó la conexión con Matlab. Es por esto que en esta escena no se encuentran los *sliders* y hay mayor espacio para visualizar con la cámara del dispositivo y mediante Vuforia los movimientos ejecutados en MATLAB del robot seleccionado.

IV. Detalle de componentes de cada escena

Existen componenetes o *GameObject* que son comunes en todas las escenas de la aplicación. Los 3 principales son *MainCamera*, *Direction Light* y *Canvas*. Los primeros 2, por ser una aplicación en 2D sencilla no son de gran importancias, por lo que no se explican en este trabajo. En cambio, el *Canvas* es distinto en cada escena y posee los componentes más importantes de la aplicación.

IV-A. Componentes de escena Loading

Además de los comunes a todas las escenas mencionados previamente, un objeto *GameObject Background*, compuesto por un *Sprite Renderer* que posee la imagen de fondo de la escena (archivo *backgound.png*).

Luego, el *GameObject* principal que es el *Canvas* de la escena. Este objeto posee tres objetos imagen *MatlabLogo*, *UnityLogo* y *VuforiaLogo*, con los logos de las herramientas de software utilizadas (puede observarse en la figura 24). Dos objetos *Text*, con el título y "tagline" de la aplicación. Por último el objeto gráfico correspondiente a la barra de carga, de tipo *Slider*.

La escena posee el *GameObject SceneTransition*, con un *Canvas* que posee una imágen igual a la del fondo de la aplicación, un objeto *Animator* y un *Script* (detallado en la sección V-C) ambos con el mismo nombre. El funcionamiento de este objeto, vinculado con la animación de la escena, se expone detalladamente en la sección VI-B.

IV-B. Componentes de escena MainMenu

Exactamente igual que en la escena *Loading*, uno de los componentes de *MainMenu* es *Background* con la imagen de fondo de la aplicación.

Luego, el objeto principal *Canvas*, con los siguientes elementos: Un *EmptyObject* que agrupa dos objetos *Text* con el nombre de la aplicación y el "tagline", y también se encuentra el *EmptyObject MainMenu*, que contiene:

- Un objeto *Start* tipo *Button* del conjunto de objetos UI de Unity que lanza las escenas *ManualScene* o *MainScene*.
- Un objeto *Quit* también de tipo *Button* que permite al usuario salir de la aplicación.
- Un objeto *Text* con información.
- Un objeto *EnableConnection* de tipo *Toggle* que habilita o no el funcionamiento de la aplicación con MATLAB (habilitado la próxima escena es *MatlabScene*, inhabilitado será *ManualScene*).
- Un *EmptyObject* llamado *ConfigIP* con un objeto *Text* y otro *Input Field*, que básicamente permite al usuario introducir la IP de su dispositivo móvil.
- Un objeto *ConfigPort* idéntico a *ConfigIP* pero que permite al usuario introducir el puerto por el que se establecerá la comunicación.
- Un objeto *EmptyObject* llamado *RobotSelection*, que contiene un objeto *Text* y otro *Image* con el nombre y la imagen del robot a simular.
- Por último dos botones *NextRobot* y *PreviousRobot* que se dejan planteados para en el futuro incorporar más de un robot a la aplicación.

IV-C. Componentes de escena ManualScene

ManualScene es la primer escena con los componentes de Vuforia. De hecho, el primer objeto en escena es *AR Camera*, lo provee Vuforia y es el componente básico para una aplicación de realidad aumentada ya que controla la cámara del dispositivo móvil, el audio del dispositivo (en esta aplicación no se utiliza), y contiene toda la información de la implementación de Vuforia que se está utilizando (para el desarrollo de la aplicación se utilizó *Vuforia Engine 8.5.9*).

El segundo componente, también vinculado a Vuforia, es *ImageTarget*. La aplicación desarrollada utiliza realidad aumentada basada en *targets*, esto quiere decir que las representación virtuales aparecerán en escena si Vuforia localiza una imagen en el entorno real, y las renderizará en relación a la posición y orientación de la imagen objetivo. La imagen utilizada se muestra en la figura 1, para poder utilizarla se preprocesa con el *Target Manager* en línea de Vuforia, que permite crear una base de datos con distintos objetivos y luego importar los *targets* como objetos procesados *ImageTarget* dentro del editor de Unity.

Figura 1. Imagen utilizada como *ImageTarget*, para levantar el modelo 3d del robot SCARA ABB 910SC.

Dentro del objeto *ImageTarget*, en forma de componente "hijo", se encuentran los *GameObject* asociados al modelo 3D del robot, su implementación se explica más detalladamente en la sección VIII.

La escena posee un objeto *Canvas*. Dentro de él, lo primero es un *EmptyObject* nombrado *Title* que contiene dos objetos *Text* con el nombre del modelo del robot que se esta simulando y el nombre del fabricante. Luego, un *EmptyObject* llamado *Sliders* que contiene tantos objetos tipo *Slider* como articulaciones activas posee el robot (en este caso 4 articulaciones), que controlaron según los valores que ingrese el usuario el

valor de cada variable articular del robot. Además, un objeto *BackButton* tipo *Button* del conjunto de objetos UI de Unity que lanza la escena *MainMenu*.

Por último, la escena posee un componente *SceneTransition* exactamente igual y con el mismo objetivo que el de la escena *MainMenu*, detallado en la sección IV-B.

IV-D. Componentes de escena MatlabScene

La escena *MatlabScene* tiene la mayoría de sus componentes exactamente igual que la escena *ManualScene*. Sus diferencias son la ausencia del componente *Sliders*, por lo que no existen los diferentes objetos UI *Slider*; y, lo que es más importante, el componete *MainCamera* agrega la clase *readSocket* del script (que se detalla en la sección V-G). Se elige *MainCamera* para agregar este componente vinculado a la comunicación con MATLAB simplemente para asegurar su existencia siempre que el usuario esté en la escena *MatlabScene*.

V. Descripción de programas escritos en C#

A continuación se detallan los *Scripts* desarrollados en C#, que luego son características de distintos *GameObject* de la aplicación y los dotan de funcionalidades específicas.

V-A. Serie de eventos en la interacción con el usuario: MainMenu

El *script* probablemente más importante de la aplicación es el de nombre *MainMenu.cs*. Aquí se encuentran todos los métodos relacionados con la interacción del usuario con componentes, por ejemplo, como botones, y los correspondientes eventos que se deben desencadenar.

En la figura 2 se puede observar el diagrama UML de la clase. Posee los atributos *GameObject connectionToggle*, *inputIP* e *inputPort* que se vinculan con los objetos en escena *EnableConnection*, *ConfigIP* y *ConfigPort* respectivamente. Además un objeto *GameObject sceneTransition* que agrega el componente *SceneTransition* en la escena del *script*.

Además posee los métodos:

- *Start*, que inicializa todos los atributos de la clase.
- Un método *EnableConnection*, que se ejecuta cuando el usuario habilite o deshabilite la conexión de la aplicación con MATLAB y escribe en *AppData* (sección V-B) la próxima escena a cargar en función de esto.
- Los métodos *EditedIP* y *EditedPort* que se ejecutan cuando el usuario modifica el casillero correspondiente y escriben en *AppData* la información actualizada.
- *PlayScene*, ejecutada cuando el usuario presione el botón *StartButton*, y carga la próxima escena correspondiente en base a la información en *AppData*.
- *BackScene*, método que se ejecuta cuando el usuario presione el botón *BackButton*, y carga la escena *MainMenu*.
- Por último, *QuitApp*, método que cierra la aplicación.

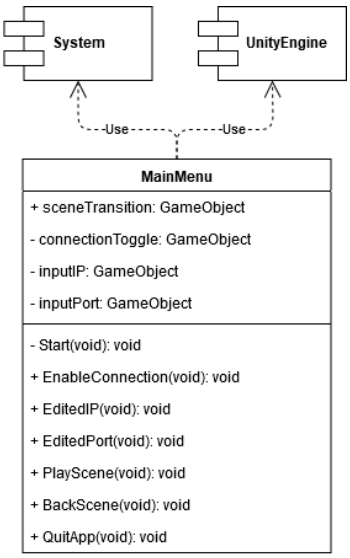

Figura 2. Diagrama UML del script con nombre *MainMenu.cs*.

V-B. Información estática de la aplicación: AppData

En el *script* de C# con nombre *AppData.cs* se encuentra toda la información estática de la aplicación. Por ejemplo, los parámetros de la comunicación TCP/IP como la dirección IP del dispositivo móvil y el puerto, la próxima escena a cargar, etc. Las clases estáticas en Unity permanecen inmutables entre escenas, por lo tanto es información que se encuentra disponible desde cualquier etapa de ejecución.

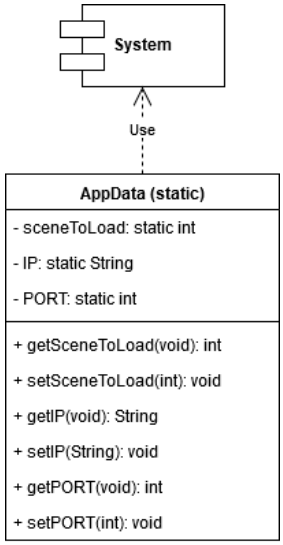

Figura 3. Diagrama UML de la clase estática AppData

En la figura 3 se puede observar el diagrama UML de ésta clase estática.

V-C. Transición entre escenas: SceneTransition

Clase con las rutinas necesarias a realizar para coordinar la carga de escenas y las diferentes animaciones que se ejecutan al pasar de una escena a otra.

Entre sus atributos se encuentra un objeto *Animator* de Unity, asociado a las animaciones a realizar; un objeto *AsyncOperation*, con información de la rutina asíncrona que carga la próxima escena; y un objeto *Slider*, que no es más que el objeto gráfico de la barra de carga. Sus métodos son *LoadScene*, con sobrecarga, si no recibe ningún parámetro lee la próxima escena a cargar desde *AppData*, ver sección V-B, y si recibe un entero carga la escena con el índice dado por tal número.

Además de esto, se encuentra el método *LoadAsyncronously*, que ejecuta la rutina asíncrona de carga de recursos para la próxima escena, se analiza el progreso de carga y se muestra con la barra de carga gráfica, y una vez finalizada la carga se setea un *Trigguer* que inicia la animación de transición. Por último, un método *OnFadeComplete* que se ejecuta al finalizar la animación y activa la siguiente escena.

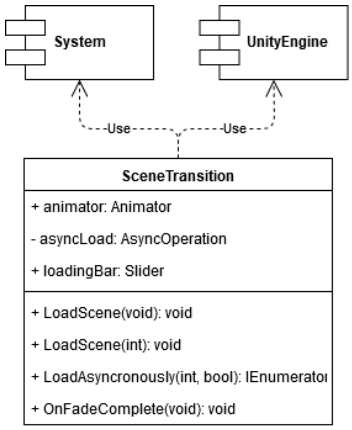

Figura 4. Diagrama UML de la clase SceneTransition

En la figura 4 se puede observar el diagrama UML de ésta clase.

V-D. Extensión de matrices de transformación: MatrixExtensions

En el archivo *MatrixExtensions.cs*, se encuentra una clase con tres métodos que extienden el uso de matrices de transformación homogénea.

Es una clase relacionada con los aspectos teóricos de robótica, dado que en las cátedras de Robótica se utiliza una descripción geométrica de las transformaciones realizadas a través de matrices de transformación homogénea 4x4, y Unity suele utilizar para la transformaciones otro método muy común que son los *Quaternions*, fue necesario la implementación de esta clase para trabajar con herramientas matemáticas conocidas.

En la figura 5 se puede observar el diagrama UML de ésta clase.

V-E. Información del manipulador: SerialLink

La clase *SerialLink*, en el archivo *SerialLink.cs*, es el nexo entre la aplicación de realidad aumentada y los componentes teóricos de robótica necesarios. Busca ser similar a la clase

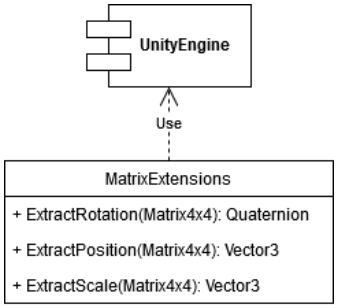

Figura 5. Diagrama UML de la clase MatrixExtensions

SerialLink del toolbox de Peter Corke para MATLAB. Basicamente contiene toda la información relacionada con el robot y los métodos necesarios para su manipulación.

Los atributos de la clase son los siguientes:

- Un arreglo de objetos clase *Link*, asociado a cada articulación del robot como se explica en la sección V-F.
- Un arreglo de *floats* con el valor numérico de las variables articulares.
- Un arreglo donde se vinculan los modelos 3d del robot en escena.
- Valores numéricos *a1*, *a2* y *alpha4*, específicos de la descripción DH del robot que se decidió simular. En el futuro se generalizarán estos parámetros.

Los métodos de la clase son los siguientes:

- Los métodos *Start* y *Update*, característicos de Unity, el primero se ejecuta una vez en la instanciación de la clase y el segundo una vez por cuadro en ejecución.
- Método *updateConfig*, que actualiza los objetos *link* en base a la posición, escala y orientación del robot simulado. Se utilizan los métodos implementados en la clase *MatrixExtension*.
- El método *updateDH*, que se encarga de la actualización de los parámetros variables en la descripción DH, en los objetos link en base a la configuración articular actual.
- Método *setSliderQ*, específico del modo manual de la aplicación, asociado a un determinado *Slider*, y se actualiza el *q* correspondiente en base al valor ingresado por el usuario.

En la figura 6 se puede observar el diagrama UML de ésta clase.

V-F. Descripción Denavit-Hartenberg: Link

La clase *Link* busca ser similar a la clase con mismo nombre del toolbox para MATLAB de Peter Corke, y poseer la descripción de Denavit Hartenberg de el eslabón correspondiente.

Los parámetros de la clase son los correspondientes a la descripción de Denavit-Hartenberg de cada eslabón: *theta*, *d*, *a*, *alpha* y *sigma*. También posee un último atributo que asocia la instancia de la clase con el *GameObject* correspondiente en escena.

La clase posee un constructor, que se utiliza en la instancia del *SerialLink* del robot a simular, donde se inicializan todos los atributos antes mencionados.

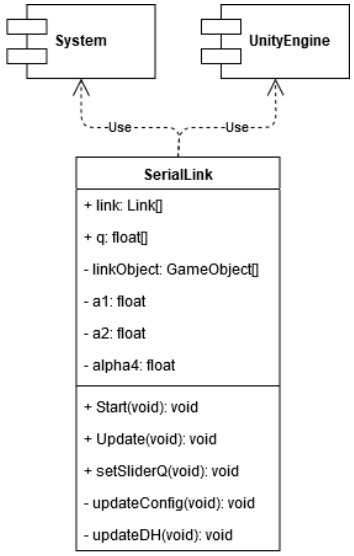

Figura 6. Diagrama UML de la clase SerialLink

Por último, la clase posee un método *getT* que devuelve la matriz de transformación homogénea del eslabón en su posición y orientación actual, dada la descripción DH correspondiente.

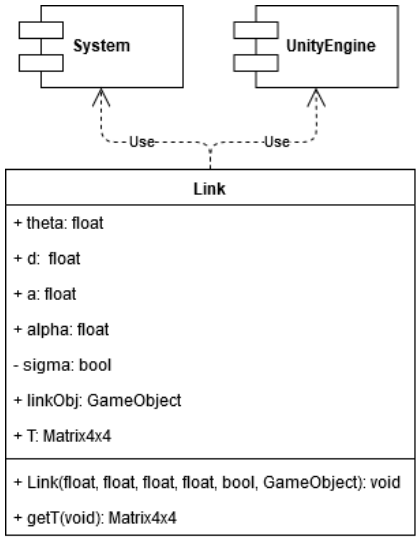

Figura 7. Diagrama UML de la clase Link

En la figura 7 se puede observar el diagrama UML de ésta clase.

V-G. Comunicación TCP/IP, servidor en Unity: readSocket

La clase *readSocket* es la encargada de la comunicación entre la aplicación y MATLAB. Abre el socket como servidor TCP/IP, y espera a recibir mensages de la aplicación del usuario en MATLAB.

Los atributos de esta clase son:

- *listener*, de tipo *TcpListener* de la librería *System.NET* de C#. Es el objeto que permanentemente escucha la recepción de mensajes en el *socket*.
- Un *String* llamado *msg*, donde se guardará el mensaje recibido por el socket para su procesamiento.
- Un *String* con nombre *IP*, donde se guarda la dirección IP del dispositivo móvil donde se encuentra la aplicación.
- Un entero denominado *PORT*, donde se guarda el número de puerto por el que se establecerá la comunicación.
- Por último un *GameObject* llamado *RobotArm*, que apunta al objeto en escena donde se encuentra toda la información relacionada con el robot.

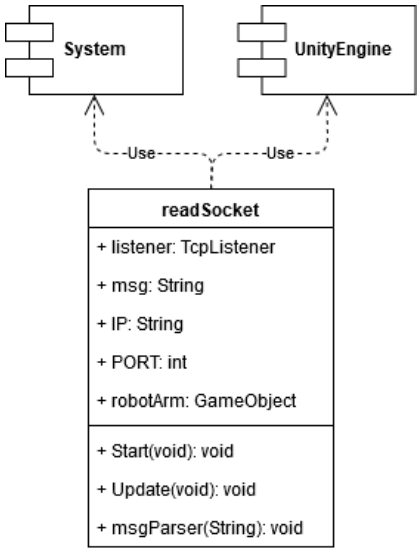

Figura 8. Diagrama UML de la clase *readSocket*.

En la sección X se explica más detalladamente la implementación de la comunicación entre MATLAB y la aplicación móvil desarrollada en Unity.

En la figura 8 se puede observar el diagrama UML de ésta clase.

VI. Descripción de animaciones

VI-A. Animaciones

Se implementaron 4 animaciones que en escencia tienen el objetivo de hacer que las transiciones entre escenas sean más prolijas y agradables visualmente: Dos de *Fade In* donde la escena comienza con una imagen igual al *background* limpia superpuesta sobre el contenido de la escena y que cambia su *alpha* hasta hacerse completamente transparente; y dos de *Fade Out*, que basicamente es la animación opuesta a la de *Fade In*. Se implementaron dos tipos: las animaciones *FadeInLoading* y *FadeOutLoading* para la entrada y salida de la escena *Loading* específicamente; y las animaciones *Fade In* y *Fade Out* para la entrada y salida del resto de las escenas.

VI-B. Animator y controladores

Para los objetos *Animator* y sus respectivos controladores con el mismo nombre, se aplica la misma diferencia que para las animaciones. Esto es importante porque los eventos ejecutados en la escena *Loading* son distintos al resto de las escenas.

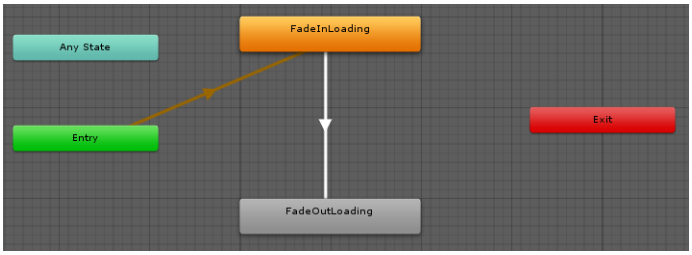

Figura 9. Máquina de estados de *Animator SceneTransition*, con *Controller* del mismo nombre

En la figura 9, se observa la máquina de estados utilizada en la escena *Loading*. Aquí puede observarse que luego de la entrada a la escena se ejecuta la animación *FadeInLoading*, y luego una transición a la animación *FadeOutLoading* cuando se ejecute un *trigger FadeOut*.

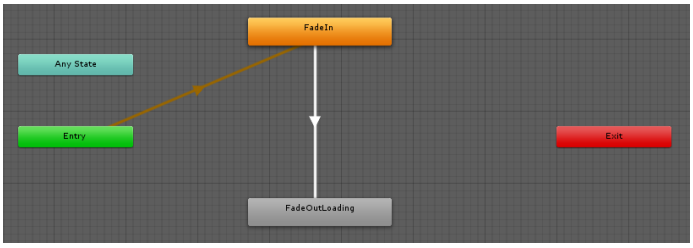

Figura 10. Máquina de estados de *Animator SceneTransition*, con *Controller SceneLoading*

En la figura 10, se observa la máquina de estados utilizada en el resto de las escenas para la transición. Puede observarse que es prácticamente igual a la de la figura 9, la única salvedad es que en lugar de la animación *FadeInLoading* se encuentra la animación *FadeIn*. La diferencia entre estas dos animaciones es que la primera, al finalizar, llama un evento que comienza la carga de recursos de la próxima escena (funcionamiento de la escena *Loading*). En el resto de las escenas esto no es necesario, por lo tanto la animación *FadeIn* no tiene este evento al finalizar.

La segunda diferencia es el controlador, el controlador de la máquina de la figura 9 es *SceneTransition*, mientras que el controlador de la máquina de la figura 10 es *SceneLoading*.

VII. Diagramas UML

Para exponer de una forma práctica y clara la forma en que se comunican los distintos elementos de la aplicación descriptos a lo largo del informa, se realizaron los siguientes diagramas UML de las escenas.

En la figura 14, se muestra el diagrama UML de la escena *Loading*.

Puede que la escena de mayor complejidad sea la de *Loading*, por la rutina asíncrona de carga de recursos. Por lo tanto, para más claridad, en la figura 15 se expone el diagrama de secuencia de esta escena.

En la figura 16, se muestra el diagrama UML de la escena *MainMenu*.

En la figura 11, se expone el diagrama de secuencia que sucede en la entrada de la escena *MainMenu*. Esta secuencia es exactamente igual en todos los inicio de escenas que no sean la escena *Loading*.

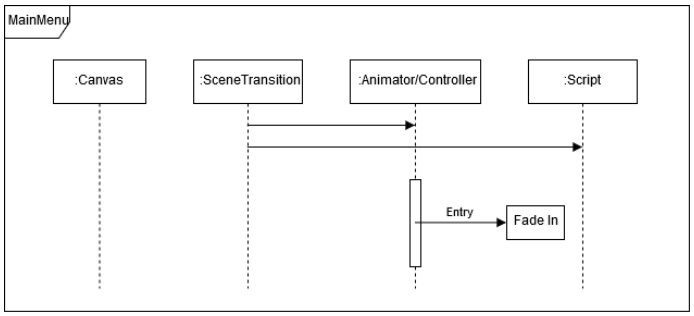

Figura 11. Diagrama de secuencia de la escena *MainMenu*, en la entrada a la escena.

En la figura 17, el diagrama de secuencia de la escena *MainMenu*, para un caso de uso del usuario, donde se configura primero la dirección IP del dispositivo móvil del usuario, luego el puerto en el que se establecerá la comunicación, y por último la habilitación o no de la conexión con MATLAB.

En la figura 12, se muestra el diagrama de secuencia de la escena *MainMenu*, para el caso en que el usuario presiona el botón *QuitButton* para salir de la aplicación.

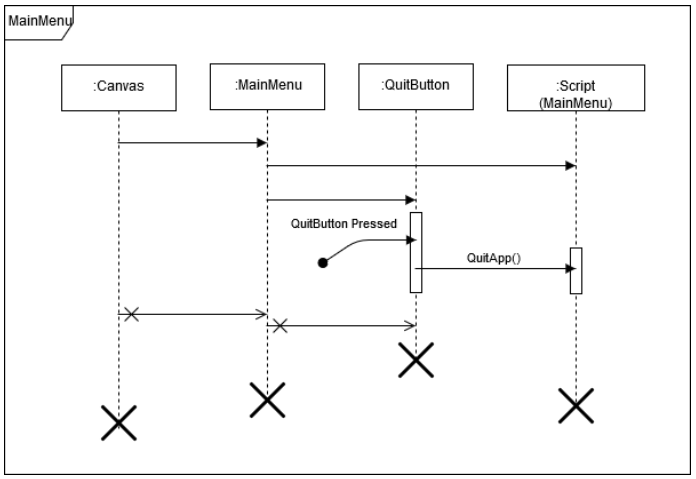

Figura 12. Diagrama de secuencia de la escena *MainMenu*, usuario presiona *QuitButton*.

En la figura 18, el diagrama de secuencia de la escena *MainMenu* para el caso en el que el usuario presiona el botón *StartButton* lanzando la escena *Loading* que cargará la escena *ManualScene* o *MatlabScene* según cuál haya sido la configuración que introduzco el usuario previamente.

En la figura 20 el diagrama UML de la escena *ManualScene*.

La escena *ManualScene* tiene entrada con animación exactamente igual que la escena *MainMenu*, por lo tanto no se consideró necesario exponer su diagrama de secuencia. Lo importante de esta escena es cómo es la secuencia de eventos cuando el usuario interactúa con alguno de los *Sliders* que modifican la posición y orientación de los eslabones. Por lo tanto, en la figura 19 se encuentra dicho diagrama de secuencia para una articulación genérica.

Ante la presión del botón *BackButton* de la escena *ManualScene*, la aplicación responde exactamente igual a la salida de la escena *MainMenu*, por lo tanto el diagrama de secuencia es similar al de la figura 18, y no se considera necesario repetirlo.

En la figura 21 el diagrama UML de la escena *MatlabScene*, donde se puede apreciar mejor la diferencias básicas entre esta escena y la escena *ManualScene*, cuyo diagrama UML se encuentra en la figura 20.

Por último, se expone en la figura 22 el diagrama de secuencia de la escena *MatlabScene* ante la recepción de un mensaje desde la aplicación en MATLAB del usuario.

El resto de los diagramas de secuencia, que serían la entrada y salida de la escena mediante la interacción del usuario con el botón *BackButton*, no se muestran por ser exactamente igual a los de la escena *ManualScene*.

Es importante destacar que los diagramas UML se encuentran en una versión simplificada o reducida. Muchos objetos a los que se le asigna directamente el tipo, por ejemplo *Text*, en realidad en su implementación Unity crea un objeto *GameObject* que agrega un *script Text*. Esto se expone simplificado porque si el árbol del diagrama se expande en este grado de especificidad se torna complejo y de gran tamaño, perdiendo el objetivo que es el entendimiento del proyecto de una forma práctica.

VIII. Implementación de modelos 3D del robot en Unity

Como se menciona en la sección IV-C, en las escenas *ManualScene* y *MatlabScene* se encuentran objetos *ImageTarget* dentro de los cuales se encuentran los componentes asociados a los modelos 3D del robot.

Lo primero, es un *EmptyObject* denominado *RobotArm*, dentro del cual existen tantos *EmptyObject* como eslabones tiene el manipulador que se está simulando (en el caso de lo implementado 4), con nombre *Link* seguido del número de índice empezando por 0. Dentro de cada uno de estos objetos *LinkX* se encuentra el modelo 3D del eslabón en forma de *Prefab* (se obtiene simplemente arrastrando el modelo STL del eslabón a la escena).

Esta estructura de objetos se eligió así para poder lidiar con el movimiento relativo de los eslabones y sus sistemas de referencia. Cada uno de los archivos STL tiene sistema de referencia iguales al del eslabón base, y es necesario que tengan el sistema de referencia asignado en la descripción de Denavit-Hartenberg del robot. Por lo tanto, para poder correr los sistemas de referencia, se modificó en concordancia con la descripción el componente *Transform* de cada *Prefab* de los distintos eslabones. De esta forma, luego se pueden ejecutar transformaciones homogéneas a los objetos *LinkX* y los modelos 3D se moverán respecto del sistema de referencia correcto.

IX. Desarrollo en MATLAB: Extensión del toolbox *Robotics, Vision & Control*

El otro componente fundamental del trabajo es la implementación en MATLAB del software necesario para que la aplicación del usuario pueda comunicarse con la aplicación en su dispositivo móvil y realizar la simulación en realidad aumentada. Para esto, es importante destacar que si se observa el código fuente del toolbox *Robotics, Vision & Control* de Peter Corke, la herramienta fue programada con paradigma orientado a objetos. Esto significa que se puede explotar la gran ventaja de este paradigma de desarrollo, que es la modularidad.

Dicho esto, se procedió implementando una clase *UnityLink* que extiende o hereda los componentes de la clase principal del toolbox *SerialLink*, como se observa en la figura 13.

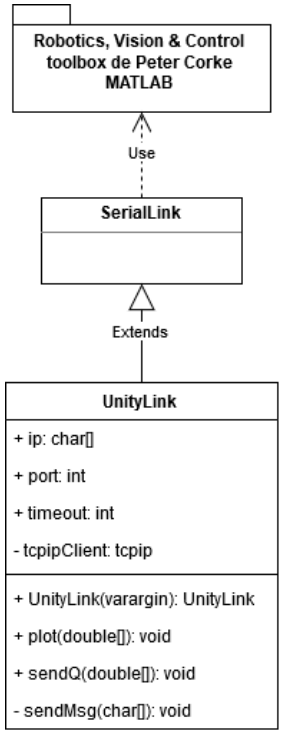

Figura 13. Diagrama UML de la extensión del toolbox *Robotics, Vision & Control* implementada

Esta clase agrega los atributos específicos de la comunicación TCP/IP: *ip*, dirección IP del dispositivo móvil; *port*, puerto elegido para la comunicación; y *timeout*, tiempo de espera antes de la desconexión. Además, se implementaron los métodos: *sendMsg(msg)*, método privado, también relacionado con la comunicación y basicamente envía la cadena de caracteres *msg*; y los métodos públicos al usuario *sendQ(q)* donde el usuario envía la configuración articular q a la aplicación móvil; y una sobrecarga del método *plot* del toolbox, que además de hacer la representación original en MATLAB utiliza el método anterior para hacer la representación equivalente en la aplicación móvil.

X. Sobre la comunicación entre MATLAB y la aplicación móvil: Protocolo TCP/IP

Como ya se mencionó previamente, el protocolo de comunicación seleccionado para la comunicación entre MATLAB y la aplicación móvil desarrollada con Unity es el protocolo TCP/IP.

El modelo TCP/IP es usado para comunicaciones en redes y, como todo protocolo, describe un conjunto de guías generales

de operación para permitir que un equipo pueda comunicarse en una red. TCP/IP provee conectividad de extremo a extremo especificando cómo los datos deberían ser formateados, direccionados, transmitidos, enrutados y recibidos por el destinatario. TCP/IP es un conjunto de protocolos que permiten la comunicación entre los ordenadores pertenecientes a una red. La sigla TCP/IP significa Protocolo de control de transmisión/Protocolo de Internet.

X-A. Justificación de la elección de TCP/IP sobre otros protocolos

El motivo de la elección sobre la utilización de, por ejemplo, Bluetooth, es la posibilidad de no limitar la aplicación a un dispositivo móvil que se encuentre en el mismo espacio físico que se encuentra el computador corriendo MATLAB, sino permitirle al usuario su uso a través de internet por ejemplo, o rápidamente efectuar la comunicación entre distintos dispositivos móviles con la aplicación o distintos computadores con MATLAB. Otra posible elección hubiera sido el protocolo UDP, sin embargo este protocolo tendría la ventaja únicamente de ser más rápido, y al hacer pruebas con TCP/IP quedó en evidencia que la velocidad de comunicación era suficiente, y se tiene la ventaja de que TCP/IP da confirmación de la recepción de todos los paquetes enviados. Además, no es de menor importancia que TCP/IP esté implementado en las librerías tanto de MATLAB como en el *core* del lenguaje C#.

X-B. Aspectos generales de la implementación

El protocolo TCP/IP es un tipo de comunicación servidor-cliente. Para implementarlo se decidió que el dispositivo móvil corriendo la aplicación desarrollada sea el servidor, y la aplicación en MATLAB tome el rol de cliente. La justificación de esta elección fue el modo en el que se suele trabajar en MATLAB con el toolbox. Después de analizarlo, se decidió que en lugar de que la aplicación móvil este solicitando a intervalos periódicos el estado articular de la simulación para efectuar cambios (lo que sucedería si la elección fuera MATLAB como servidor y dispositivo móvil como cliente), MATLAB envíe mensajes únicamente cuando en la simulación se efectuó un cambio. También para que MATLAB tome el rol de servidor, se requeriría que el usuario tenga en MATLAB un proceso corriendo permanentemente con lo que es en Unity el *TcpListener*, y en MATLAB se suele trabajar en forma secuencial, sin paralelización de procesos. Sin embargo, la decisión tomada tiene la desventaja que no se puede plantear una situación donde MATLAB envía mensajes a múltiples dispositivos corriendo la aplicación en la misma red, algo que hubiera sido interesante y se deja planteado como trabajo a futuro.

XI. Modo de uso de la aplicación móvil para establecer la comunicación con MATLAB

El modo en que el usuario debe usar la aplicación móvil para efectuar exitosamente la simulación con datos de su programa en MATLAB es el siguiente:

1. Crear el programa en MATLAB, basado en un manipulador tipo *UnityLink*, con por ejemplo un bucle que calcula las diferentes configuraciones articulares deseadas y luego las envía mediante el método *UnityLink.SendQ*.
2. Abrir la aplicación móvil desarrollada en Unity.
3. En la pantalla principal de la aplicación introducir la dirección IP del dispositivo móvil, y el puerto en el que se desea establecer la comunicación (se propone puerto 8888).
4. Al objeto *UnityLink* creado en el programa en MATLAB, cambiarle sus atributos IP y PORT por los mismos introducidos en la aplicación (Es importante que la IP sea de la misma red que la computadora).
5. Habilitar la conexión con MATLAB en la aplicación móvil con la opción *EnableConnection*.
6. Proceder a la escena *MatlabScene* presionando en la aplicación el botón *Start*.
7. Apuntar al target impreso, donde aparecerá el modelo 3D del robot en realidad aumentada.
8. Ejecutar el programa en MATLAB para que envíe la secuencia de configuraciones articulares. El robot en la aplicación móvil debería moverse si la conexión fue exitosa.

Para comprobar que la comunicación entre la aplicación desarrollada en Unity y MATLAB efectivamente funciona, se implementó un programa sencillo donde se crea un objeto *UnityLink* con la matriz de Denavit-Hartenberg correspondiente al robot manipulador SCARA IRB 910SC. Luego, se realiza un bucle donde se le envían una serie de consignas articulares consecutivas con el método *sendQ* para que la simulación anime al robot en un movimiento suave.

XII. Conclusiones

El producto final obtenido, es una aplicación para dispositivos móviles Android que funciona fluido, tiene una interfaz gráfica muy sencilla que da al usuario las opciones necesarias y suficientes para tener la simulación funcionando rápidamente. El marcador es una imagen del robot que se está simulando, en esta caso el robot del fabricante ABB modelo *IRB 910SC*. La herramientas utilizadas fueron las adecuadas, tanto Unity como Vuforia son intuitivos y aunque no se poseía nada de experiencia en su uso y los tiempos eran limitados, se llego a desarrollar una aplicación que cumple las metas propuestas.

A pesar de existir muchos detalles a mejorar, la comunicación TCP/IP entre la computadora corriendo MATLAB y el dispositivo móvil con la aplicación funciona cumpliendo las expectativas, no provocó inconvenientes ni retrasos y la simulación lograda para movimientos de prolongada duración del robot son suaves y sin saltos. Es necesario destacar que las pruebas se realizaron en redes locales donde la cantidad de dispositivos conectados era mínima.

El toolbox *"Robotics, Vision & Control"* para MATLAB está desarrollado, valga la redundancia, en lenguaje de programación MATLAB, pero con un paradigma orientado a objetos, por la tanto los conocimientos brindados por la carrera fueron útiles a la hora de plantear una extensión del toolbox que se comunique con la aplicación.

XIII. Trabajo a futuro

Dadas las limitaciones del tiempo del trabajo quedan aspectos que se desearía continuar desarrollando. Algunos de estos aspectos son:

- Incorporación de otros modelos de robot comerciales de distintos fabricantes que se utilicen en el programa de Robótica I y Robótica II, para que el estudiante pueda apreciar sus dimensiones reales, y su desempeño en un espacio de trabajo similar al real.
- Mejorar la interfaz para los comandos manuales al robot desde la aplicación.
- Mejorar el aspecto visual de la aplicación, incorporar la posibilidad de utilizar el dispositivo móvil en posición vertical.
- Limpiar el código y crear *prefabs* de Unity que permitan que el proyecto sea más escalable.
- Ampliar la comunicación con MATLAB a otras herramientas de robótica distintas, por ejmplo, el uso de ROS (*Robot Operating System*) con el software de simulación Gazebo. Esto se vio imposibilitado dado la poca experiencia con la herramienta.
- Trabajar sobre el protocolo de comunicación, sobre el protocolo TCP/IP utilizado. Analizar y estudiar con mayor profundidad los diferentes aspectos de la comunicación implementada. Corregir errores, y plantear la posibilidad de hacer *broadcast* para la comunicación con múltiples dispositivos móviles o la posibilidad de una interfaz de comunicación distinta.
- Trabajar con distintos marcadores para los aspectos de realidad aumentada.
- Plantear la posibilidad de simular trabajos cooperativos entre múltiples robot en un entorno de realidad aumentada.
- Continuar con el desarrollo de la clase *UnityLink* para facilitarle al usuario el desarrollo de su aplicación en MATLAB.

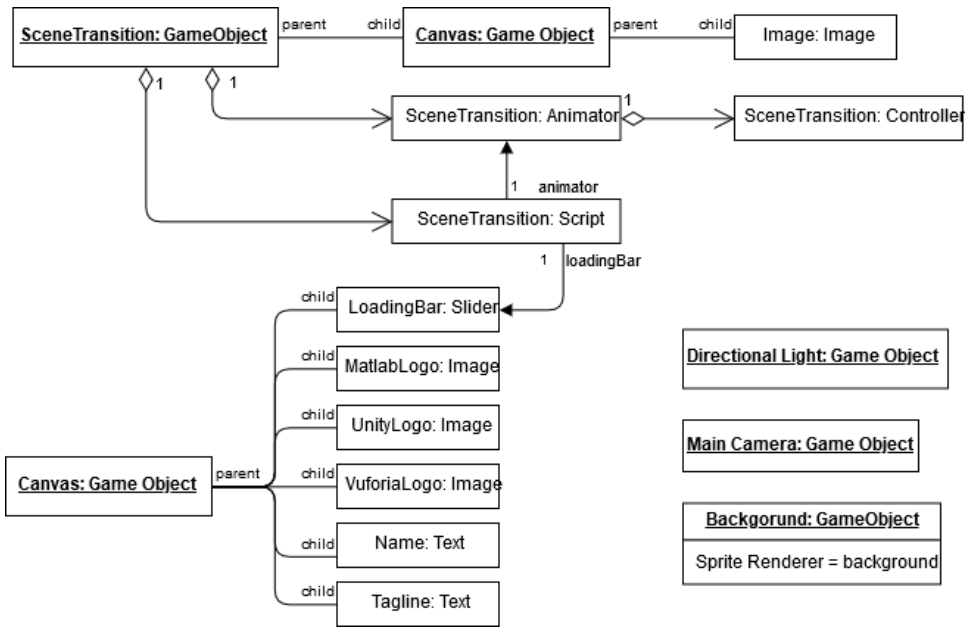

Figura 14. Diagrama UML de escena *Loading*.

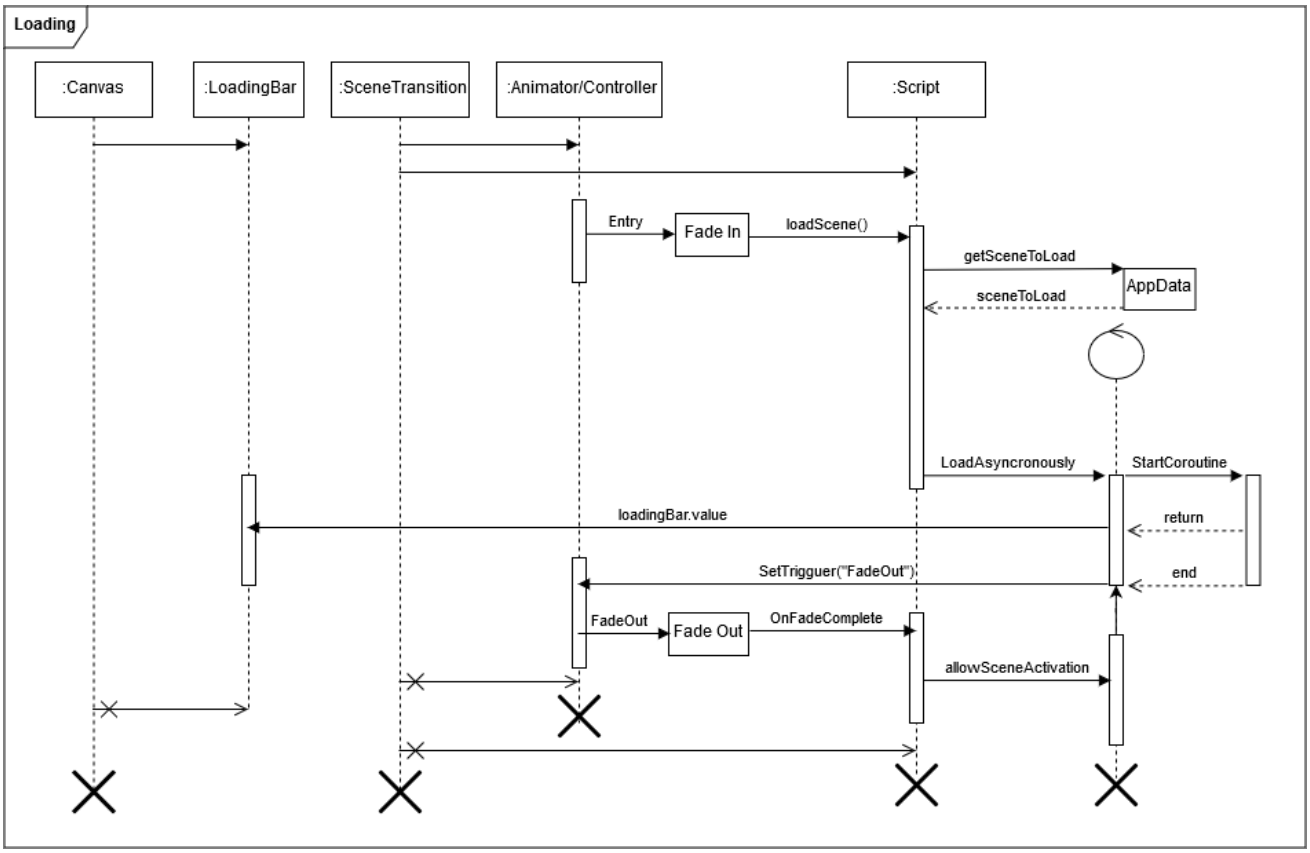

Figura 15. Diagrama de secuencia de escena *Loading*

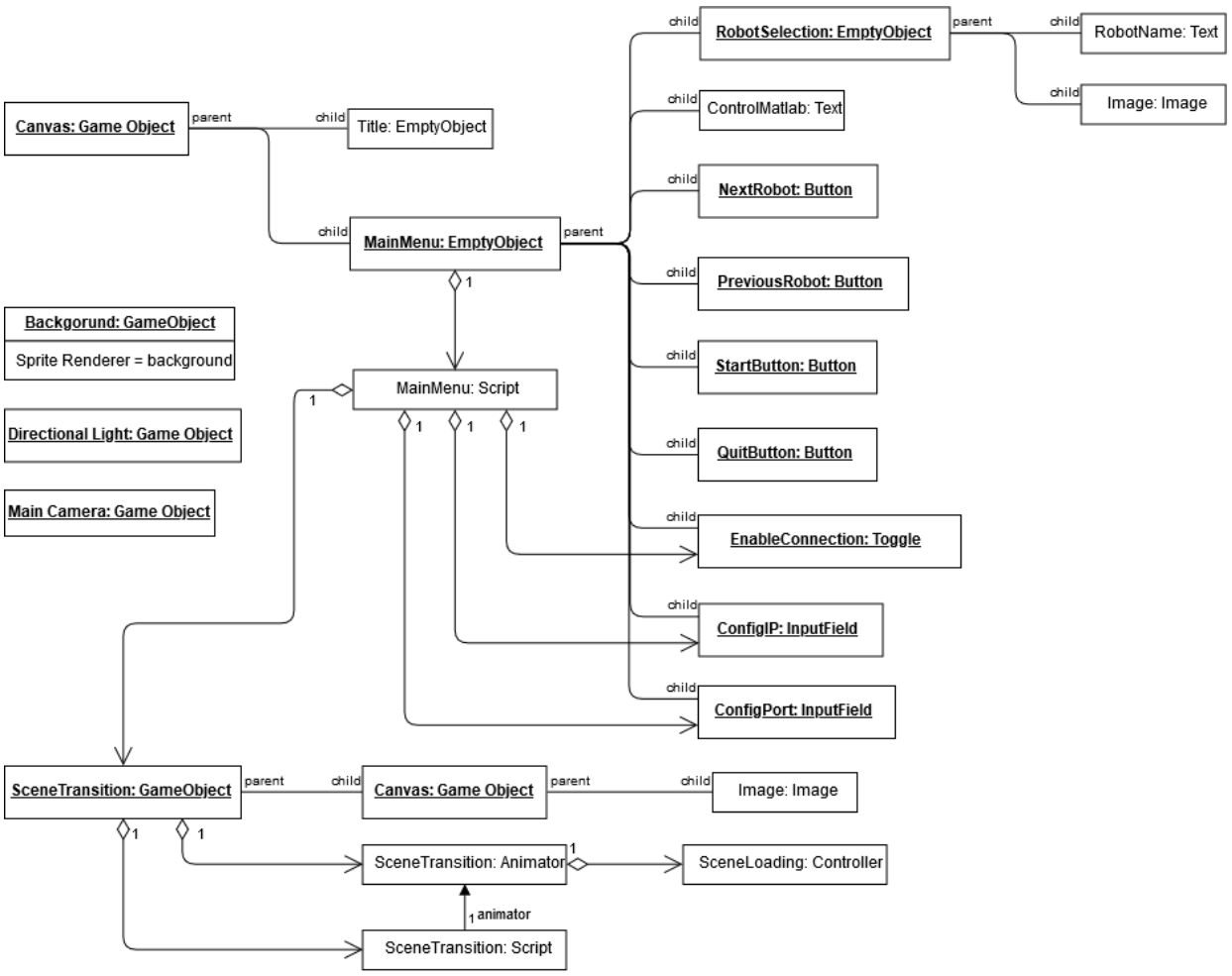

Figura 16. Diagrama UML de escena *MainMenu*.

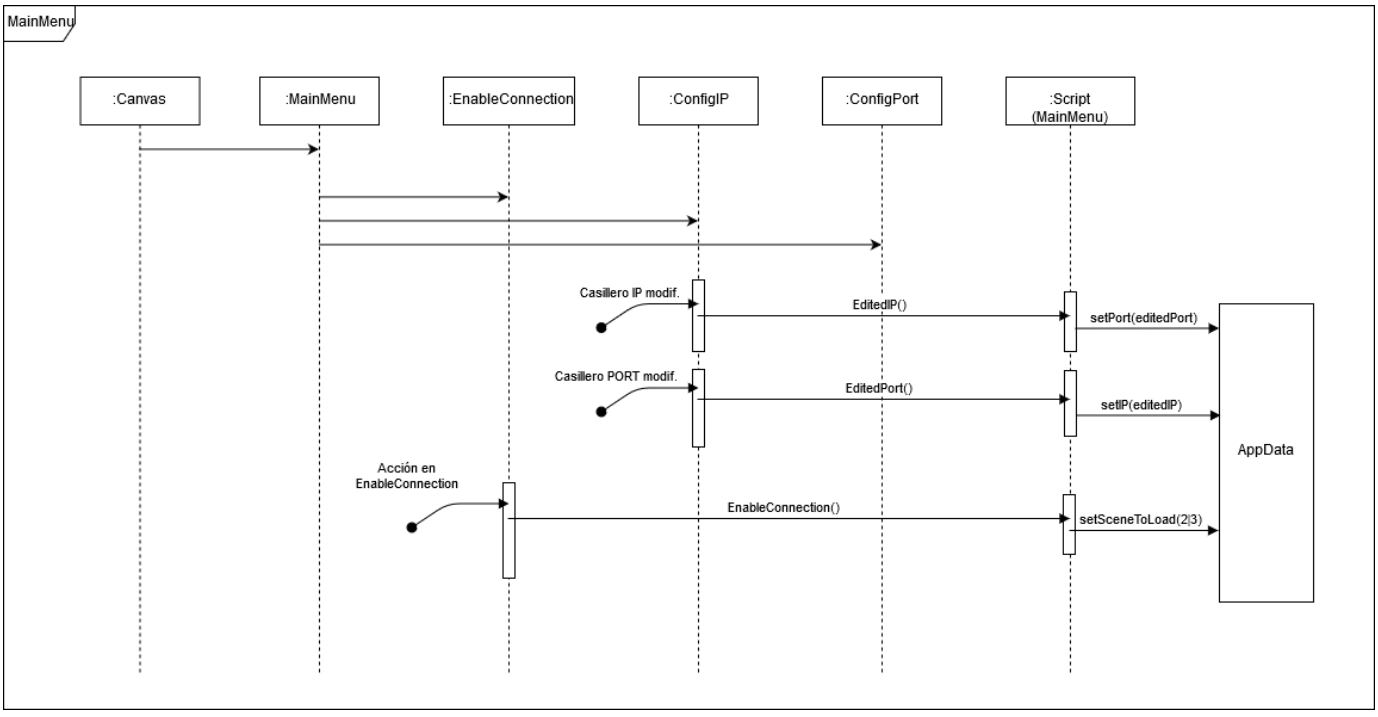

Figura 17. Diagrama de secuencia de la escena *MainMenu*, con la configuración la aplicación

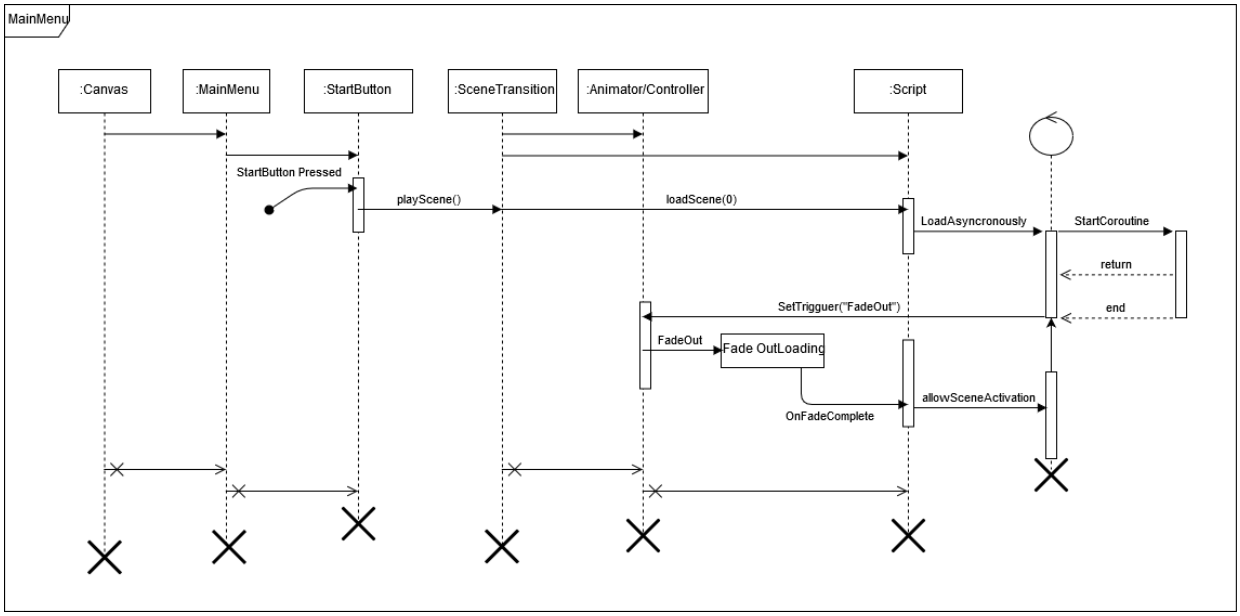

Figura 18. Diagrama de secuencia de la escena *MainMenu*, usuario presiona *StartButton*.

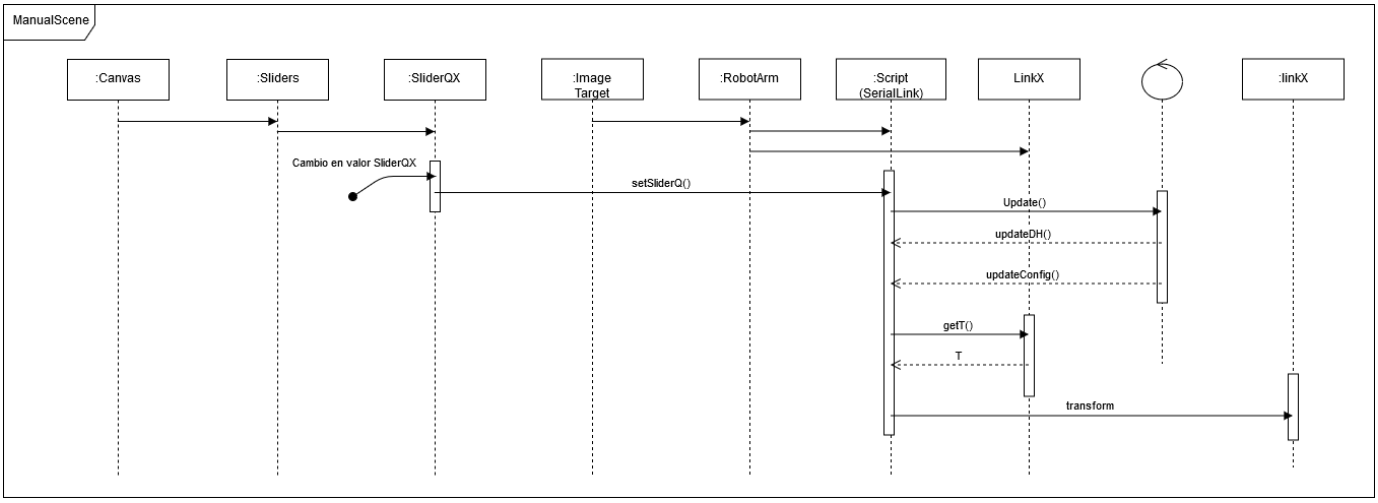

Figura 19. Diagrama de secuencia de la escena *ManualScene*, con usuario interactuando con uno de los *Sliders*.

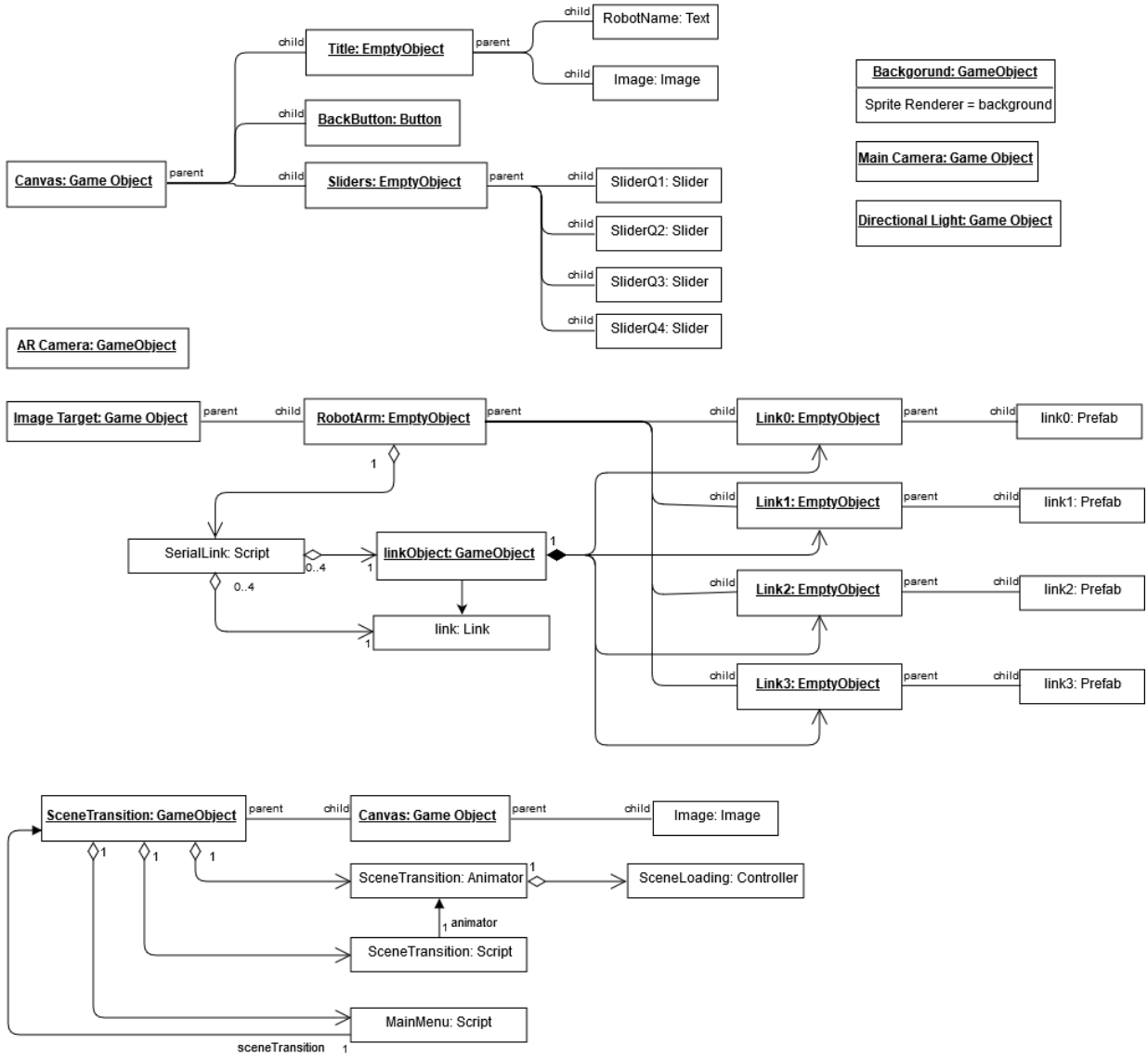

Figura 20. Diagrama UML de escena *ManualScene*.

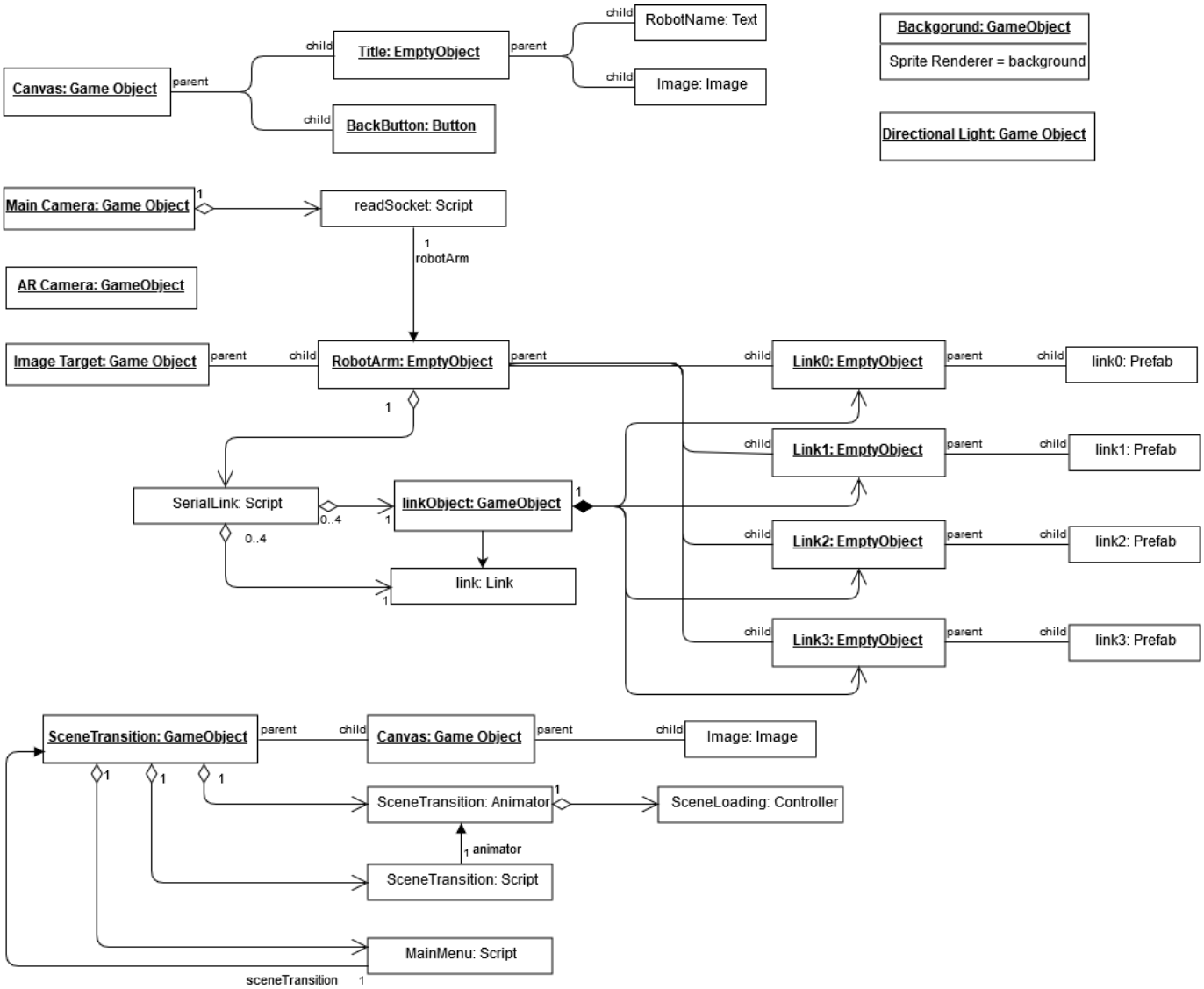

Figura 21. Diagrama UML de la escena *MatlabScene*.

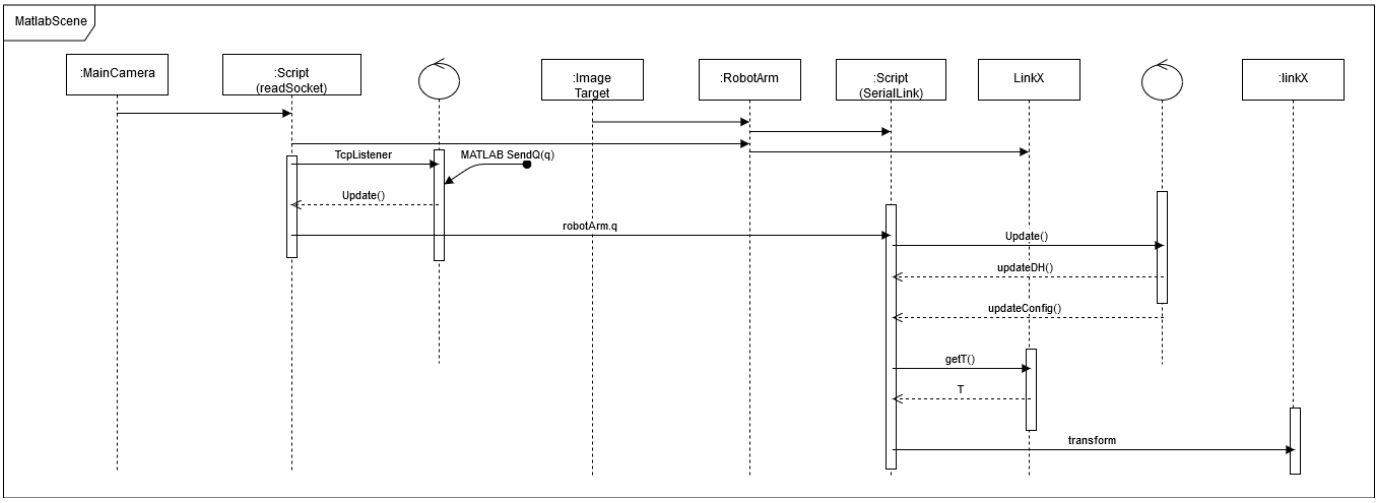

Figura 22. Diagrama de secuencia de la escena *MatlabScene*, ante un mensaje de MATLAB.

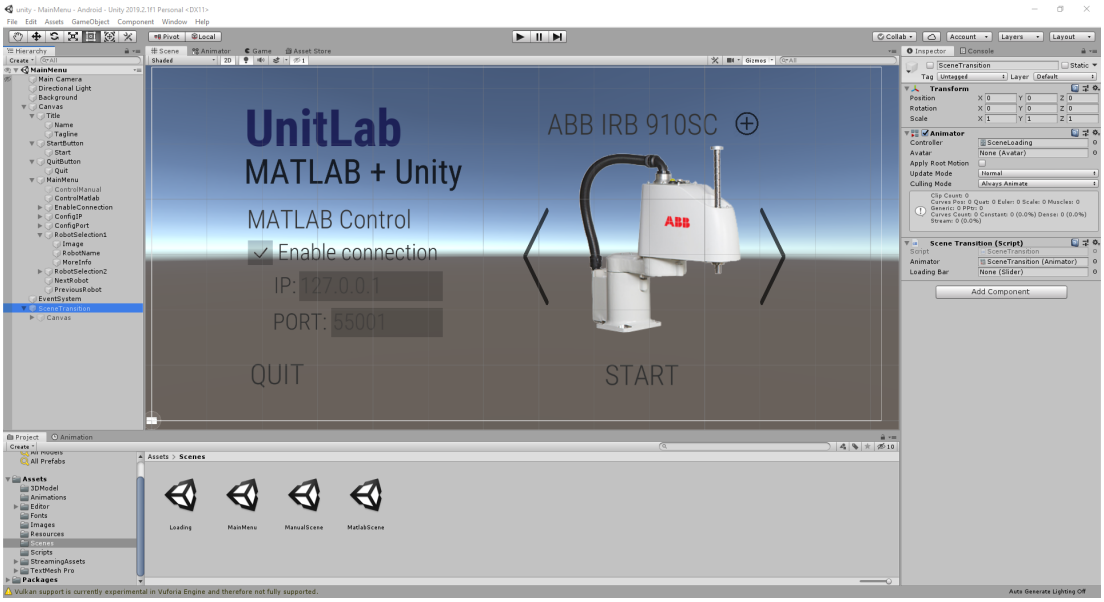

Figura 23. Captura de pantalla del editor de Unity, con la escena *MainMenu* abierta.

Figura 24. Captura de pantalla del editor de Unity, con la escena *Loading* abierta.

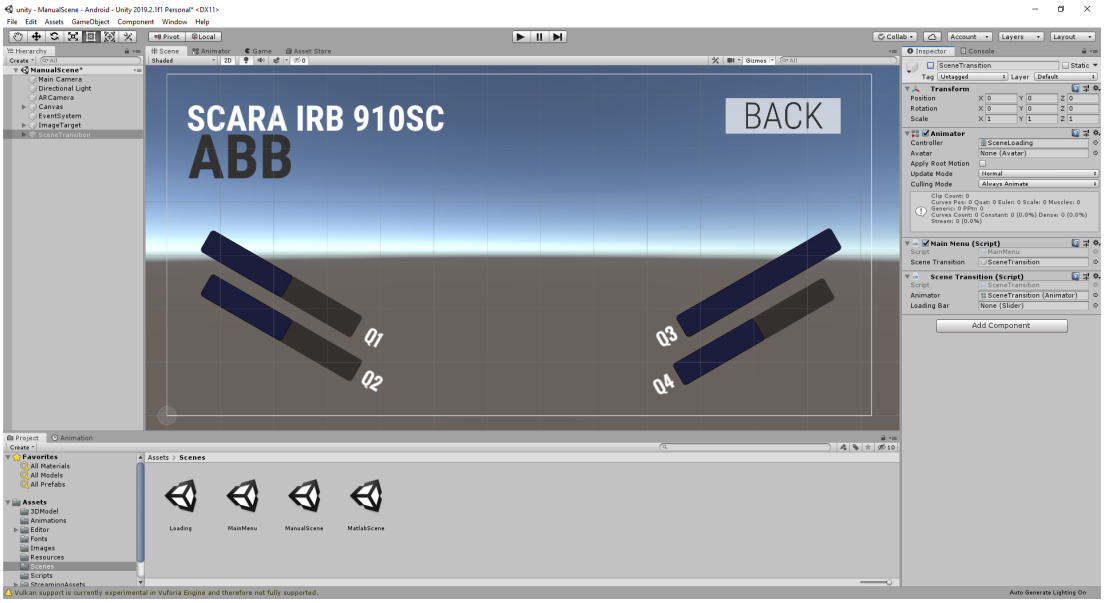

Figura 25. Captura de pantalla del editor de Unity, con la escena *ManualScene* abierta.

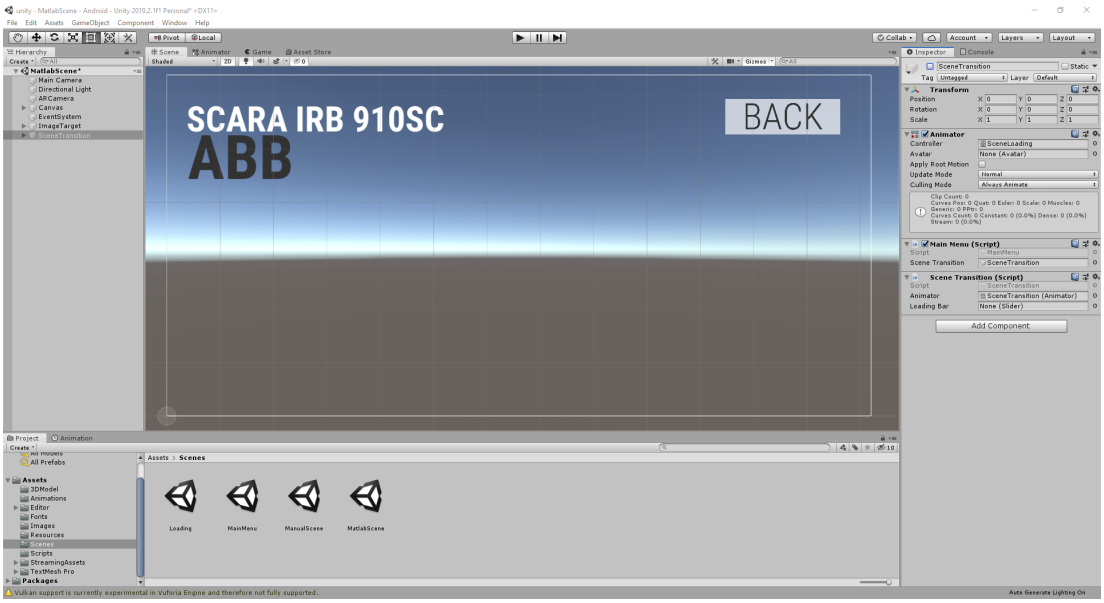

Figura 26. Captura de pantalla del editor de Unity, con la escena *MatlabScene* abierta.