Realidad virtual

Trabajo integrador:

Interfaz de personalización de brazos robóticos para simulación en Realidad Aumentada

Alumno: Olguin Nahuel

Legajo: 12297

Contenido

1. Introducción

Este proyecto busca aplicar algunos de los conceptos estudiados en la asignatura "Realidad Virtual". Unity es un motor de desarrollo en tiempo real que permite crear experiencias interactivas. En este caso se busca programar en Unity una interfaz que permita crear y modificar brazos robóticos para posteriormente ser proyectados en realidad aumentada, en la figura 1 se muestra como ejemplo un modelo de robot ABB en Unity.

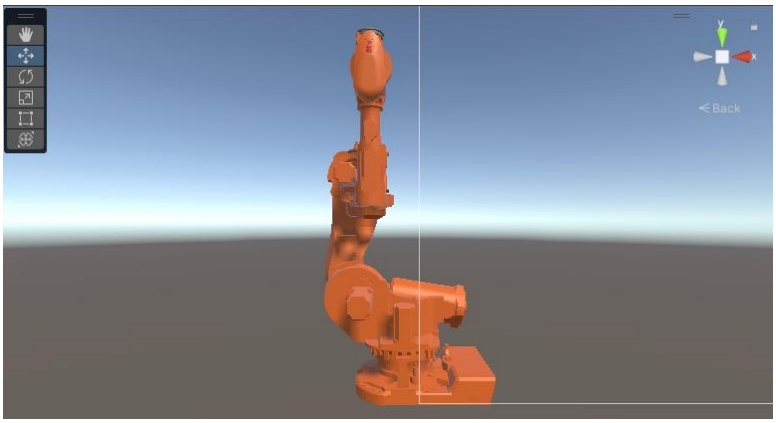
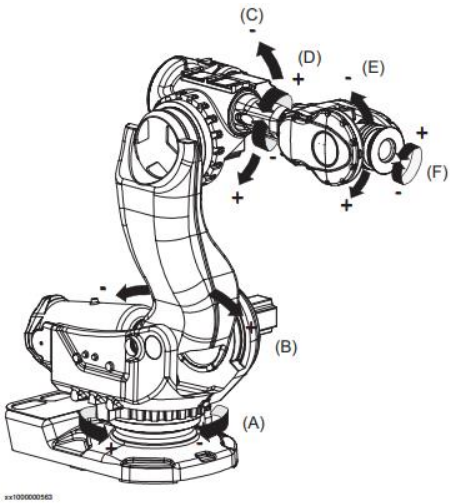


Figura 1: Modelo del robot en Unity

Haciendo uso de las herramientas de Unity el objetivo es controlar las articulaciones asociadas a cada eje del robot. En la figura 2 se muestran las articulaciones que deberían ser controladas por la aplicación para el caso del robot ABB utilizado como ejemplo.



Pos	Descripción	Pos	Descripción
A	Eje 1	D	Eje 4
B	Eje 2	E	Eje 5
C	Eje 3	F	Eje 6

Figura 2: Articulaciones controladas por el programa

Para desarrollar el código de control se hace uso del Entorno de Desarrollo Integrado (IDE) Visual Studio 2019, el cual es utilizado para desarrollar aplicaciones y juegos en diversos lenguajes de programación. Visual Studio es muy utilizado para escribir scripts de C# en Unity, ya que Visual Studio Tools ofrece un amplio conjunto de características que mejoran la escritura y depuración de scripts en C# para Unity facilitando el trabajo en proyectos.

2. Escenas

Resumen general

La aplicación cuenta con cinco escenas que se muestran en la figura 3, las cuales se describen brevemente a continuación:

1. Menú: Cuenta con botones que permiten acceder a las distintas escenas o salir de la aplicación.
2. Crear: Permite seleccionar que tipo de robot se llevara a simulación, siendo las opciones un robot personalizado o un robot prefabricado ABB.
3. Dimensionar: Permite modificar el tamaño de cada eslabón del robot.
4. Ensamblar: Permite ajustar la posición relativa de los eslabones, asi como establecer el eje de giro de las articulaciones correspondiente a cada eslabón.
5. Simular: Se accede a la cámara del dispositivo para captar la superficie sobre la cual se proyectara el robot en cuestión.

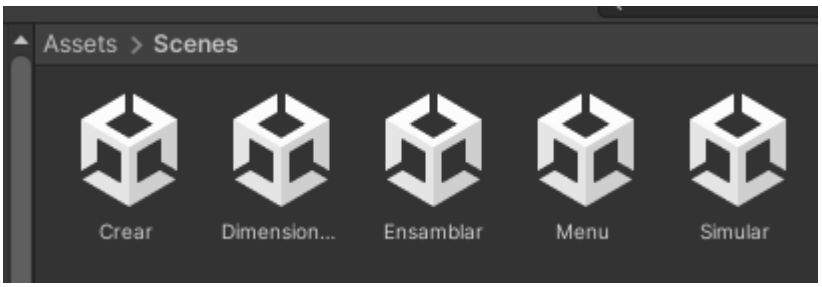


Figura 3: Lista de escenas que componen la aplicación

Menú

Primero comenzaremos explicando la interfaz que permite acceder a las distintas escenas que componen la aplicación. En la Figura 4, vemos que la interfaz cuenta con una imagen de fondo, un título y cinco botones.

Figura 4: Interfaz de menú principal

Todos los elementos que forman parte de esta interfaz están contenidos en un Canvas. El objeto Canvas de Unity es un Game Object que nos permite mostrar una capa de información entre la cámara y la escena de juego, de forma que dicha información siempre es visible en la escena, independientemente de la posición en la que nos encontremos. Todos los elementos UI (Interfaz de usuario), tales como imágenes de fondo, títulos, botones, selectores, deslizadores, etc; deben ser hijos de dicho Canvas como se observa en la figura 5.

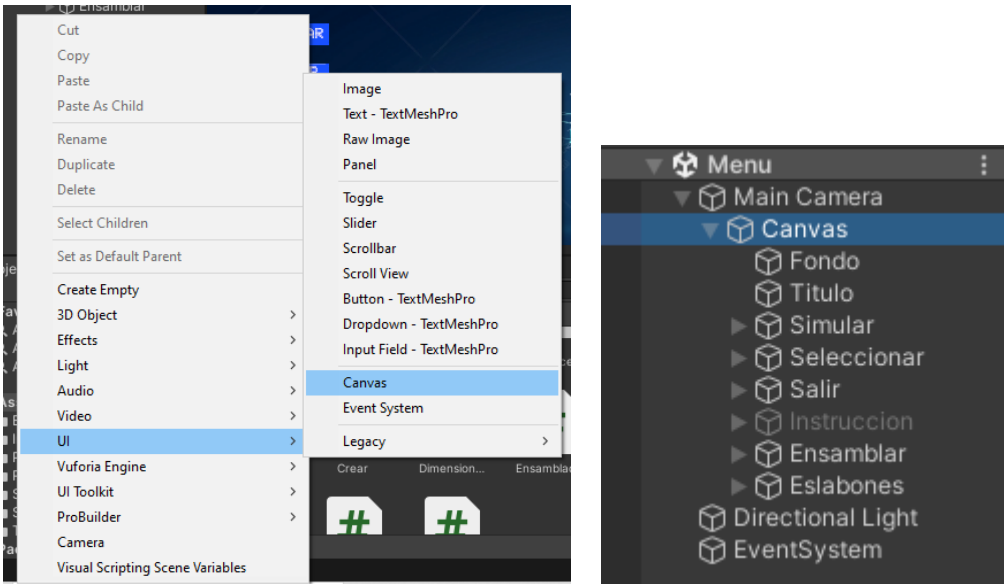


Figura 5: Implementación de Canvas y elementos de UI en Unity

En Unity, el objeto “Button” se utiliza para crear botones interactivos en la interfaz de usuario. Los botones se utilizan para realizar acciones cuando el usuario hace clic en ellos, en este caso permiten avanzar a otras escenas o salir de la aplicación. En la figura 6 se muestra la ubicación de los botones en la lista de objetos UI.

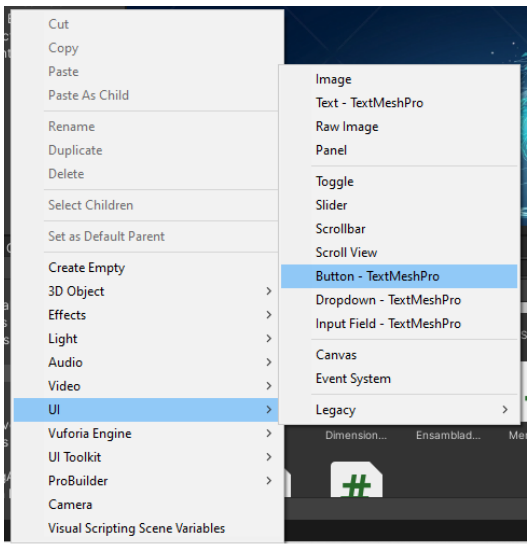


Figura 6: Implementación de Botones en Unity

En la figura 7 se muestra como ejemplo el botón “Ensamblar”, el cual accede al Script “Menú inicial” y la funcion “Go_Ensamblar”. Esta funcion hace que al presionar dicho botón se cargue la escena cuyo nombre es “Ensamblar”, siempre y cuando en la escena actual exista el objeto correspondiente al robot sobre el cual se está trabajando, el cual se identifica con el nombre “Robot”.

El resto de botones funcionan de forma análoga al botón “Ensamblar” utilizado como ejemplo, cada uno con su respectiva funcion.

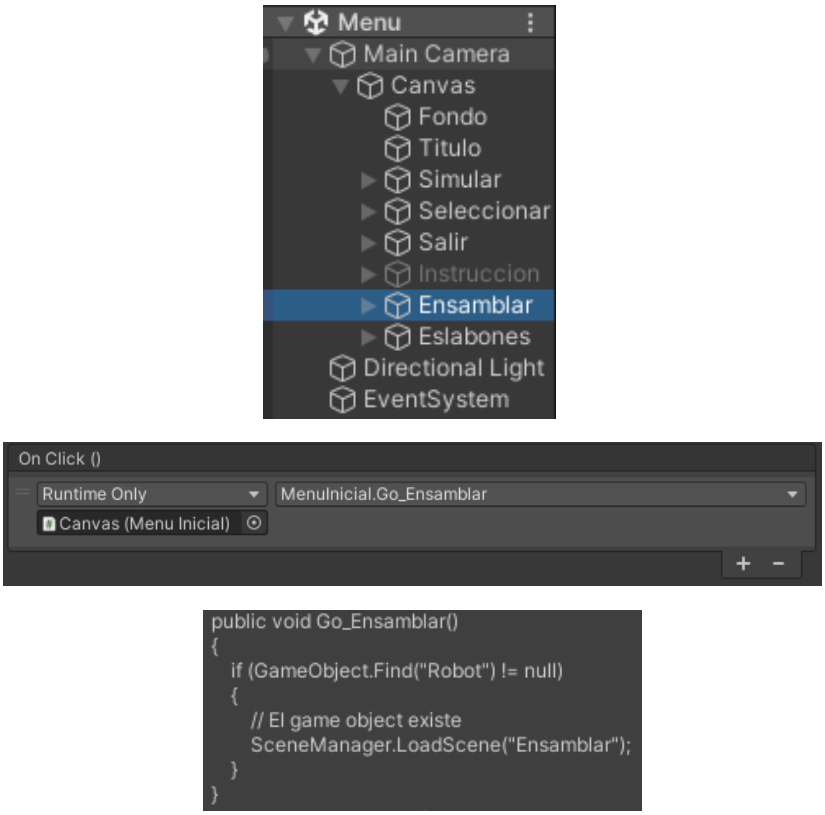


Figura 7: Configuración del botón "Ensamblar"

3. Escena “Crear”

Funcionamiento

La interfaz de la escena “Crear” se muestra en la figura 8, donde se observan los siguientes elementos:

1. Botón “Menú” que permite regresar a la escena del menú principal.
2. Botón “ABB” que genera un robot prefabricado “ABB IRB 7600” para ser utilizado como ejemplo.
3. Elemento llamado “Dropdown”, el cual es un menú desplegable que permite a los usuarios elegir un valor de una lista de opciones. En este caso las opciones son el número de eslabones que conforman nuestro robot personalizado (En la figura 8 se observa que esta seleccionada la opción “1”).
4. Botón “Crear” que genera una primera versión del robot personalizable basándose en la cantidad de eslabones que fueron seleccionados anteriormente.
5. Botones “–Zoom/+Zoom” que permiten alejar o acercar la cámara al robot creado.
6. Botón “Siguiente” que permite avanzar a la siguiente escena de personalización del robot.

Figura 8: Interfaz de la escena “Crear”

Programación

En la figura 9 se observa la funcion que se ejecuta al presionar el botón “ABB”. Lo primero que se verifica es si ya existe un robot en la escena actualmente, en caso de que si exista se elimina dicho robot. Posteriormente se procede a generar el nuevo robot cargando un modelo prefabricado con el nombre “Robot ABB”, se le asigna el nombre “Robot” y se le añade como componente un script con el nombre “No_Destruir”, el cual permite que el robot no se destruya al cargar una nueva escena.

```
public void Boton_ABB()
{
   //Eliminamos el robot prefabricado
   if (GameObject.Find("Robot") != null)
   {
      Robot = GameObject.Find("Robot");
      Destroy(Robot.GetComponent<No_Destruir>());
      Destroy(Robot);
   }

   // Instanciamos el nuevo robot
   GameObject myPrefab = Resources.Load<GameObject>("Robot ABB");
   Robot = Instantiate(myPrefab);
   Robot.name = "Robot";
   // Adds a component named "NombreDelScript" to the GameObject named "Robot"
   Robot.AddComponent<No_Destruir>();
}
```


```
public class No_Destruir : MonoBehaviour
{
   // Start is called before the first frame update
   void Start()
   {
      DontDestroyOnLoad(gameObject);
   }
```

Figura 9: Funcion del botón "ABB" y script "No_Destruir"

Al presionar el botón "ABB" se presentara en la escena un robot ABB IRB 7600 como el observado en la figura 10.

Figura 10: Efecto al presionar el botón "ABB"

Ahora se procede a explicar la funcion perteneciente al botón “Crear”. El procedimiento al comienzo es idéntico al botón “ABB”, con la diferencia de que no se carga el modelo de robot ABB, sino que se genera un Game Object vacío con el nombre “Robot”, como se observa en la figura 11.

```
public void Boton_Crear()
{
   //Eliminamos el robot prefabricado
   if (GameObject.Find("Robot") != null)
   {
      Robot = GameObject.Find("Robot");
      Destroy(Robot.GetComponent<No_Destruir>());
      Destroy(Robot);
   }

   // Instanciamos el nuevo robot
   Robot = new GameObject
   {
      name = "Robot"
   };
```

Figura 11: Función que crea el objeto del nuevo robot personalizable

Posteriormente, se observa en la figura 12, que se carga la primera pieza del robot personalizado, la cual es un modelo prefabricado llamado “Base”. A esta pieza se le asigna el nombre “Base” y un Tag o etiqueta también con el nombre “Base”. Posteriormente se hace uso del componente “Transform” del objeto para posicionarlo, rotarlo y escalarlo con respecto al sistema de referencia global. Finalmente la base es asignada como objeto hijo del Game Object “Robot” creado anteriormente.

```
//Generamos la base
GameObject myPrefab = Resources.Load<GameObject>("Base");
eslabon = Instantiate(myPrefab);
eslabon.name = ("Base");
eslabon.tag = "Base";
eslabon.transform.position = new Vector3(0, 0, 0);
eslabon.transform.localScale = new Vector3(1, 0.2f, 1);
eslabon.transform.SetParent(Robot.transform);
```

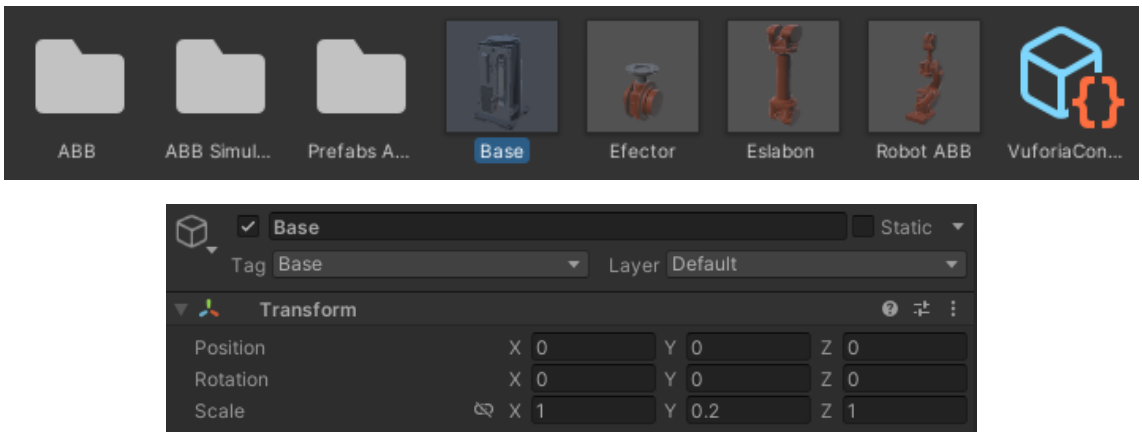
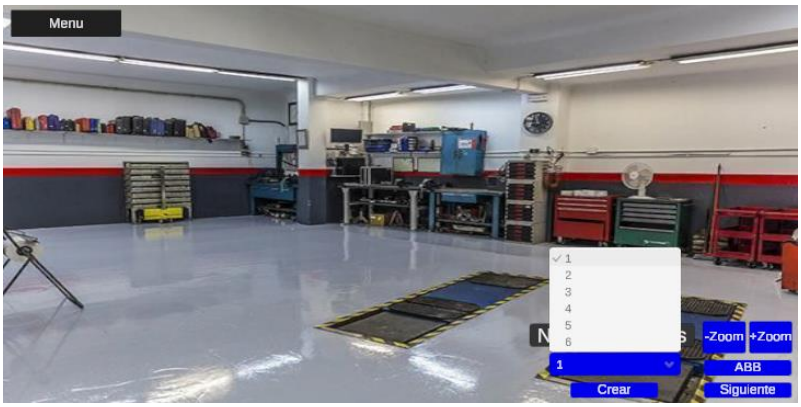


Figura 12: Instanciación y posicionamiento de la base del robot

El mismo procedimiento se repite para el resto de eslabones. En la figura 13 se observa que la variable “N_eslabones” almacena el valor seleccionado del menú desplegable que indica la cantidad de eslabones que forman al robot. Se inicia un bucle for que avanza desde el valor “i=1” hasta el valor “N_eslabones+1”. El valor “i=0” está asignado a la base, el valor “i=N_eslabones+1” está asignado al efector final y los valores intermedios de “i” están asignados al resto de eslabones.




```
//Generamos los eslabones
N_eslabones = int.Parse(Num_elegido.text);
for (int i = 1; i < N_eslabones + 2; i++)
{
   if (i == N_eslabones+1)
   {
      myPrefab = Resources.Load<GameObject>("Efector");
      eslabon = Instantiate(myPrefab);
      eslabon.name = ("Eslabon " + i);
      eslabon.tag = "Eslabon";
      eslabon.transform.position = new Vector3(0, 0.8f + 1.2f * (i - 1)-0.4f, 0);
      eslabon.transform.localScale = new Vector3(0.25f, 0.25f, 0.25f);
      eslabon.transform.SetParent(Robot.transform);
   }
   else
   {
      myPrefab = Resources.Load<GameObject>("Eslabon");
      eslabon = Instantiate(myPrefab);
      eslabon.name = ("Eslabon " + i);
      eslabon.tag = "Eslabon";
      eslabon.transform.position = new Vector3(0, 0.8f + 1.2f * (i - 1), 0);
      eslabon.transform.localScale = new Vector3(0.25f, 0.6f, 0.25f);
      eslabon.transform.SetParent(Robot.transform);
   }
}
```

Figura 13: Instanciación y posicionamiento de los eslabones del robot

Al seleccionar la cantidad de eslabones y presionar el botón crear aparecerá un robot como el observado en la figura 14.

Figura 14: Efecto al presionar el botón "Crear"

Posteriormente se configuran las articulaciones que permiten conectar a las distintas partes del robot. Inicialmente comenzamos accediendo al componente “Rigidbody” de la pieza “Base”, como se observa en la figura 15, para anular los efectos de la gravedad y congelar su posición de acuerdo a los valores de su correspondiente componente “Transform” (Este congelamiento solo se desactiva cuando comience la simulación en realidad aumentada).

```
//Configuramos las hingejoints y los rigidbodys
//Tomamos el gameobject de la base
Base = GameObject.FindGameObjectWithTag("Base");
Base.AddComponent<Rigidbody>();
Rigidbody rb_anterior = Base.GetComponent<Rigidbody>();
rb_anterior.useGravity = false;
rb_anterior.constraints = RigidbodyConstraints.FreezePosition | RigidbodyConstraints.FreezeRotation;
```

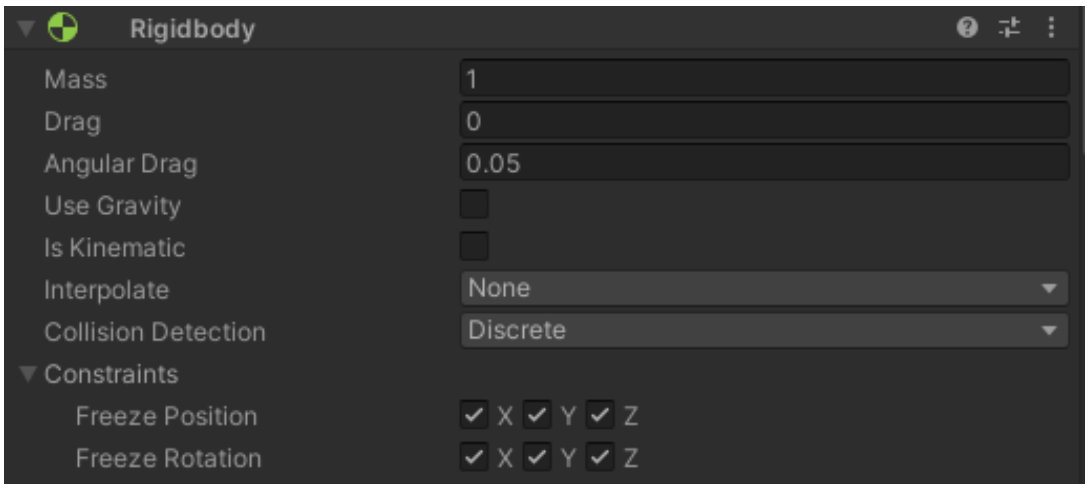


Figura 15: Fijamiento de la posición de la base del robot

Se repite el mismo procedimiento para el resto de eslabones como se observa en la figura 16, generando una lista llamada “Eslabones” que almacena el resto de los eslabones y la configuración de todos ellos se realiza a través de un ciclo “for”.

```
//Armamos una lista con los eslabones (No olvidar poner el tag al eslabon)
Eslabones = GameObject.FindGameObjectsWithTag("Eslabon");
foreach (GameObject eslabon in Eslabones)
{
    //Rigidbody
    eslabon.AddComponent<HingeJoint>();
    Rigidbody rb = eslabon.GetComponent<Rigidbody>();
    rb.useGravity = false;
    rb.constraints = RigidbodyConstraints.FreezePosition | RigidbodyConstraints.FreezeRotation;
```

Figura 16: Fijamiento de la posición de todos los eslabones del robot

Además de configurar el componente “Rigidbody” de los elementos, también se configura el componente “hingejoint” como se observa en la figura 17, el cual corresponde a las articulaciones del sistema. En donde “Connected Body” establece cuales son los dos cuerpos conectados, “anchor” indica la posición en la que se encuentra la articulación a lo largo de la pieza, donde (0, -1, 0) corresponde al extremo inferior de la pieza; y “axis” indica el eje de giro, siendo (0, 0, 1) el giro alrededor del eje z.

```
//Hingejoint
//Cuerpo conectado
HingeJoint hinge = eslabon.GetComponent<HingeJoint>();
hinge.connectedBody = rb_anterior;
//Anchor
hinge.anchor = new Vector3(0, -1, 0);
//Eje de giro
hinge.axis = new Vector3(0, 0, 1);
```

Figura 17: Configuración de las articulaciones

Para controlar la posición de las articulaciones se hace uso del componente “hingeSpring” el cual establece un sistema masa-resorte-amortiguador en cada articulación. Donde la variable “spring” almacena el valor del coeficiente de elasticidad y la variable “damper” almacena el coeficiente de amortiguamiento. Además, también se pueden establecer los límites de rotación que presenta la articulación a través del componente “hingelimits”.

```
//Activamos spring
hinge.useSpring = true;
JointSpring hingeSpring = hinge.spring;
hingeSpring.spring = 1000000; // Reemplaza 100f con el valor deseado
hingeSpring.damper = 1000000; // Reemplaza 0f con el valor deseado
hinge.spring = hingeSpring;
//Activamos limites
hinge.useLimits = true;
JointLimits hingeLimits = hinge.limits;
hingeLimits.min = -90;
hingeLimits.max = 90;
hinge.limits = hingeLimits;
```

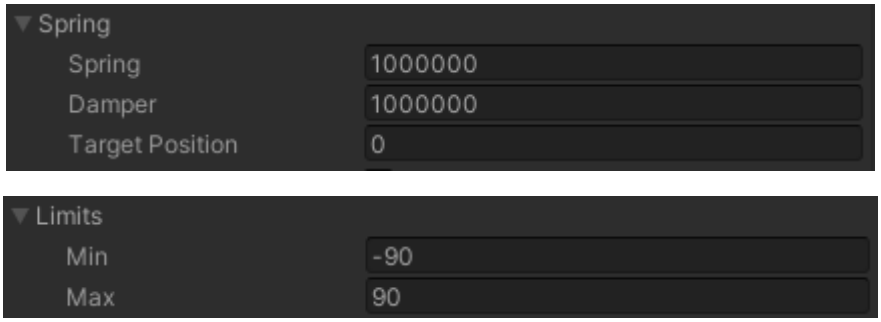


Figura 18: Configuración de los controladores y límites de las articulaciones

4. Escena “Dimensionar”

Funcionamiento

La interfaz de la escena “Dimensionar” se muestra en la figura 19, donde se observan los siguientes elementos:

1. Botón Menú que permite regresar al menú principal.
2. Menú desplegable que permite seleccionar que pieza del robot se está dimensionando.
3. Casillas para ingresar valores numéricos que serán asignados como el radio y altura del eslabón en cuestión.
4. Un botón “Aplicar” que aplica el dimensionamiento a la pieza de acuerdo a los valores que fueron introducidos en las casillas de radio y altura.
5. Un botón siguiente que permite avanzar a la siguiente escena.

Figura 19: Interfaz de la escena "Dimensionar"

Programación

Lo primero que se debe realizar, como se observa en la figura 20, es la actualización del menú desplegable de acuerdo al robot que tenemos actualmente; por ejemplo si tenemos un robot con tres eslabones, el menú desplegable debe ofrecer esas tres opciones. El Game Object correspondiente al menú desplegable tiene el nombre “Selector_eslabones”, se eliminan las opciones anteriores con la funcion “ClearOptions”, luego se almacenan los nombres de los eslabones en una lista llamada “Options” y finalmente se añade esa lista al menú desplegable a través de la funcion “AddOptions”.

```
//Actualizamos la lista de seleccion de eslabones
Selector_eslabones.ClearOptions();
//Create la lista con las nuevas opciones
List<string> Options = new();
foreach (GameObject eslabon in Eslabones)
{
    Options.Add(eslabon.name);
}
//Aderimos las nuevas opciones a la lista
Selector_eslabones.AddOptions(Options);
```

Figura 20: Actualización del menú desplegable

Posteriormente se busca visualizar únicamente la pieza seleccionada en el menú desplegable, mientras que las demás permanecen desactivadas. Para ello se inicia de forma cíclica un bucle "for" que recorre todos los eslabones del robot, como se observa en la figura 21, si el nombre del eslabón del ciclo actual "eslabon.name" coincide con el nombre de la opción seleccionada en el menú desplegable "Eslabon_elegido.text" entonces se activa esa pieza y el Game Object correspondiente a la cámara de la escena se posiciona sobre la misma para que su visualización sea centrada. El resto de eslabones se desactivan consiguiendo que sean imperceptibles para la cámara.

```
void Update()
{
    if (saliendo == false)
    {
        foreach (GameObject eslabon in Eslabones)
        {
            if (Eslabon_elegido.text == eslabon.name)
            {
                eslabon_actual = eslabon;
                eslabon.SetActive(true);
                Camara.transform.position = new Vector3(Camara.transform.position.x, eslabon.transform.position.y, Camara.transform.position.z);
            }
            else
            {
                eslabon.SetActive(false);
            }
        }
    }
}
```

Figura 21: Enfoque en la pieza seleccionada

Una vez seleccionado el eslabón a dimensionar y asignados los valores de radio y altura en sus respectivas casillas, solo queda aplicar los cambios a la pieza en cuestión. Lo primero a realizar es extraer de las casillas de texto los caracteres numéricos de radio y altura para almacenarlos en variables string llamadas "soloNumeros_Radio" y "soloNumeros_Altura", como se observa en la figura 22. Finalmente las variables string se castean como variables tipo float para introducirlas en la parte de "Scale" del componente "Transform" de la pieza.

```
public void Aplicar_cambios()
{
    //Tomamos solo los numeros del inputfield
    string soloNumeros_Radio = new((Radio.text).Where(char.IsDigit).ToArray());
    string soloNumeros_Altura = new((Altura.text).Where(char.IsDigit).ToArray());
    //Modificamos el tamaño del eslabon
    eslabon_actual.transform.localScale = new Vector3(float.Parse(soloNumeros_Radio)/100, float.Parse(soloNumeros_Altura)/100/2, float.Parse(soloNumeros_Radio)/100);
}
```

Figura 22: Aplicar cambios de dimensionamiento a la pieza

5. Escena “Ensamblar”

Funcionamiento

La interfaz de la escena “Ensamblar” se muestra en la figura 23, donde se observan los siguientes elementos:

1. Selector de pieza que especifica cual es la pieza sobre la cual se van a realizar las distintas operaciones de traslación o rotación (en la figura 23 esta seleccionada la pieza “Base”).
2. Selector de operación que permite especificar si se quieren realizar traslaciones o rotaciones sobre la pieza seleccionada (en la figura 23 esta seleccionada la operación traslación).
3. Botones con flechas en los tres ejes “xyz” que aplican las operaciones de traslación o rotación en el sentido indicado por las flechas.
4. Botones “xyz” correspondientes al eje de articulación, que establece la dirección del eje de giro correspondiente a la pieza en cuestión.
5. Botones “-Zoom/+Zoom” permiten acercar o alejar la cámara de la escena con respecto a la pieza seleccionada.
6. Botón “Menú” permite volver al menú principal.
7. Botón “Siguiente” permite avanzar a la siguiente escena.

Figura 23: Interfaz de escena ensamblar

Programación

Lo primero a realizar, como se observa en la figura 24, es generar un Game Object con forma de eje cilíndrico que permite visualizar de forma más clara el eje de giro de la articulación de la pieza en cuestión. Para ello se instancia un objeto con forma de cilindro, al cual se le asigna una etiqueta con nombre “Eje”.

```
//Instanciamos un cilindro q representa el eje de rotacion
cylinder = GameObject.CreatePrimitive(PrimitiveType.Cylinder);
cylinder.tag = "Eje";
cylinder.transform.SetParent(eslabon_actual.transform);
cylinder.GetComponent<Renderer>().material.color = Color.red;
```

Figura 24: Instanciación del objeto cilíndrico que simboliza el eje de rotación

Para posicionar el cilindro en la posición del eje de giro de la articulación se le asigna como padre la pieza sobre la cual se está trabajando (la cual está indicada el menú desplegable), como se observa en la figura 25. Finalmente se ajusta su componente “Transform” para posicionarlo adecuadamente. Notar que la posición de este cilindro esta igualada a la del eje de giro de la articulación en cuestión la cual es “articulación.anchor”.

```
//Ajustamos el eje al eslabon correspondiente
cylinder.transform.SetParent(eslabon_actual.transform);
if (eslabon_actual.name != cylinder.transform.parent.name)
{
    cylinder.transform.SetParent(eslabon_actual.transform);
}
articulacion = eslabon_actual.GetComponent<HingeJoint>();
cylinder.transform.localPosition = articulacion.anchor;
cylinder.transform.localScale = new Vector3(0.2f, 5f, 0.2f);
```

Figura 25: posicionamiento del objeto "Eje"

Ahora, como se ve en la figura 26, para igualar la orientación del cilindro con la orientacion del eje accedemos al componente “axis” de la articualciones y de acuerdo a la misma se rota el cilindro de una forma especifica para cada caso.

```
//Ajustamos el cilindro de acuerdo al eje de giro
if (articulacion.axis == new Vector3(1, 0, 0))
{
    cylinder.transform.localRotation = Quaternion.Euler(0, 0, 90);
}
else if (articulacion.axis == new Vector3(0, 1, 0))
{
    cylinder.transform.localRotation = Quaternion.Euler(0, 0, 0);
}
else if (articulacion.axis == new Vector3(0, 0, 1))
{
    cylinder.transform.localRotation = Quaternion.Euler(90, 0, 0);
}
```

Figura 26: orientación del objeto "Eje"

Ahora para aplicar los movimientos sobre la pieza al apretar los botones, primero se verifica que tipo de desplazamiento esta seleccionado en el menu desplegable cuyo nombre es “Desplazamiento _elegido.text”. Si esta seleccionada la opcion “Trasladar” se aplica una operación de translacion de “0.1 unidades” en el sentido indicado por el boton (en este caso en el eje “X” positivo); por otro lado, si esta seleccionada la opcion “Rotar” se aplica una operación de rotacion donde se gira la pieza en “2°” en el sentido especificado por el boton.

```
public void Mover_Xpos()
{
    if (Desplazamiento_elegido.text == "Trasladar")
    {
        eslabon_actual.transform.Translate(new Vector3(0.1f, 0, 0));
    }

    if (Desplazamiento_elegido.text == "Rotar")
    {
        eslabon_actual.transform.Rotate(new Vector3(2, 0, 0));
    }
}
```

Figura 27: Operación de traslación o rotación de la pieza

Por ultimo al apretar los botones “xyz” correspondiente al sector de eje de giro en la interfaz de la figura 23, se debe modificar el vector “axis” del componente “hingejoint” de la pieza en cuestion. Para ello simplemente se accede a dicho vector a traves del comando “articulacion.axis” y se le asignan los valores de acuerdo al boton apretado.

```
public void Eje_x()
{
    articulacion.axis = new Vector3(1, 0, 0);
}
```

Figura 28: Ajuste del eje de giro en la articulación

6. Escena “Simular”

Funcionamiento

La interfaz de la escena “Simular” se muestra en la figura 29, donde se observan los siguientes elementos:

1. Botón de menú que permite volver al menú inicial.
2. Menú desplegable que permite seleccionar el eslabón a controlar.
3. Un Slider o deslizador que permite establecer la posición objetivo de la articulación correspondiente al eslabón seleccionado.

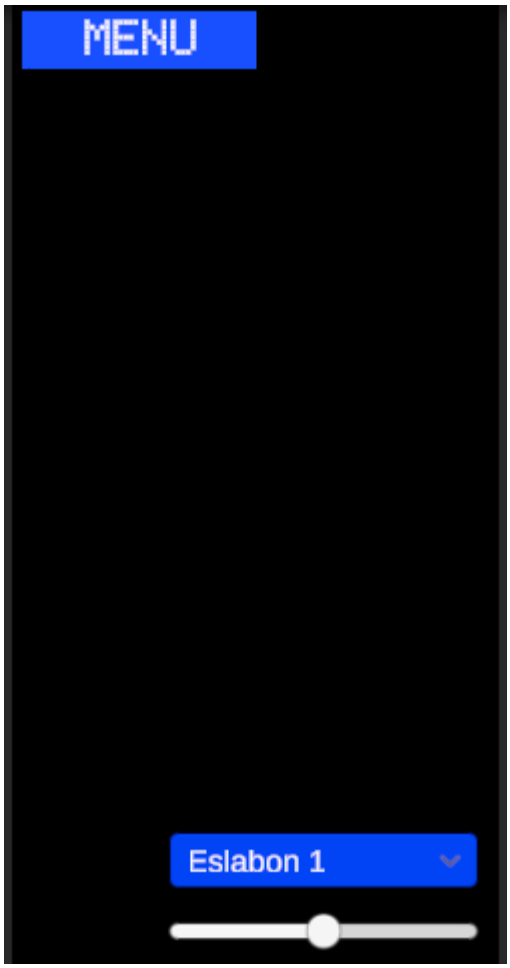


Figura 29: Interfaz de la escena "Simular"

Para aplicar realidad aumentada en el brazo robótico a simular se hace uso de Vuforia, el cual es un kit de desarrollo de software (SDK) para dispositivos móviles que permite la creación de aplicaciones de realidad aumentada (AR).

Primero se introduce, como se observa en la figura 30, un Game Object “VRCamera” perteneciente al kit de Vuforia Engine, la cual es una cámara de realidad aumentada que permite renderizar objetos virtuales en el mundo real.

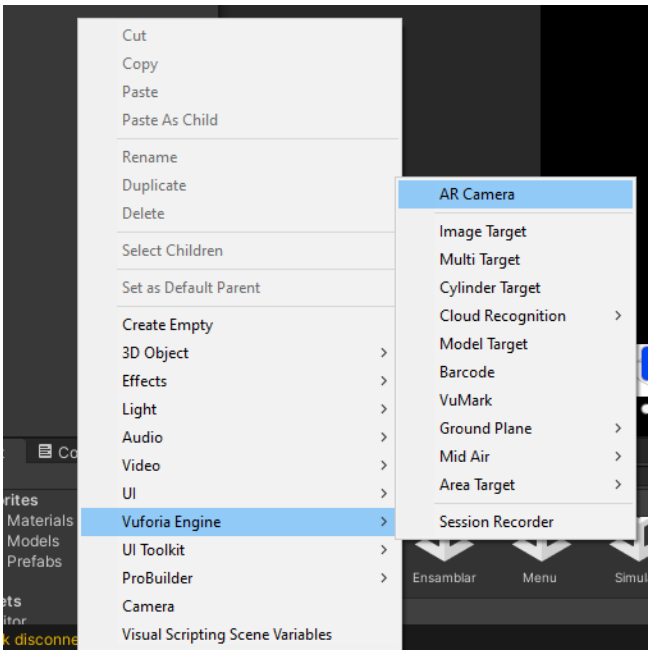


Figura 30: Implementación del objeto "VRCamera"

Luego se agrega otro Game Object del kit de Vuforia el cual es "Image Target" (figura 31) al cual se le carga una imagen de referencia que Vuforia Engine puede detectar y rastrear para proyectar el robot a simular. El motor detecta y rastrea la imagen comparando las características naturales extraídas de la imagen de la cámara con la base de datos de recursos conocida. En este caso, la imagen a detectar es la de una estrella.

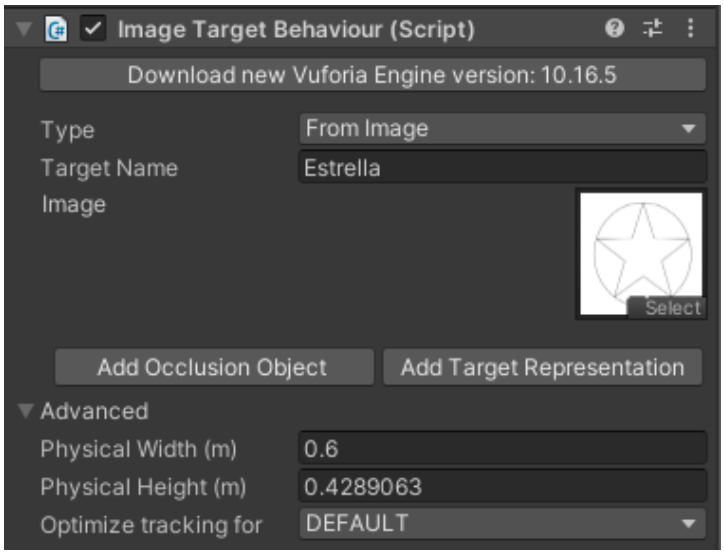


Figura 31: Implementación del objeto Image Target

Programación

Lo primero a realizar es una copia del robot fabricado que será utilizada por la simulación, sin olvidar desactivar al robot original para no ver dos robots en una misma escena. Como se observa en la figura 32, se debe realizar una copia para asignarle como padre al Game Object "Image Target" debido a que al original ya se encuentra dentro del grupo que no pueden destruirse al cambiar de escena "DontDestroyOnLoad".

```
//Hacemos una copia del robot y desactivamos el robot original
Robot_prefab = GameObject.Find("Robot");
Robot_prefab.SetActive(false);

//Instanciamos una copia del robot que sirve para simular
Robot = Instantiate(Robot_prefab);
Destroy(Robot.GetComponent<No_Destruir>());
Robot.transform.SetParent(BaseAR.transform);
Robot.SetActive(true);
```

Figura 32: generación de copia del robot para simular

Otra cosa que se hace previa a la simulación es crear un vector que almacena la posición de cada eslabón dadas por el "Slider", con el objetivo de no perder la posición del eslabón actual en el deslizador al cambiar de eslabón en el menú desplegable, como se observa en la figura 33. El deslizador toma valores entre "0" y "1", inicialmente consideramos la posición intermedia la cual es "0.5".

```
//Armamos lista q almacena los valores de posicion de cada eslabon
posiciones_deslizador = new float[Robot.transform.childCount];
for (int i = 1; i < posiciones_deslizador.Length; i++)
{
    posiciones_deslizador[i] = (float)0.5;
}
```

Figura 33: Almacenamiento de las posiciones de los eslabones

Para establecer la posición objetivo que será comunicada al controlador, se debe tomar el valor en el deslizador de la interfaz y realizar un cálculo, utilizando este último y los límites de la articulación en cuestión (figura 34).

$$posicion\_obj = (limite\_sup - limite\_inf).valor\_deslizador - limite\_inf$$

```
//Almacenamos la posicion del deslizador para la articulacion en cuestion
posiciones_deslizador[N_eslabon_actual] = Deslizador.value;

//Calculamos la posicion objetivo y almacenamos esta posicion al deslizador correspondiente
posicion = (eslabon.GetComponent<HingeJoint>().limits.max - eslabon.GetComponent<HingeJoint>().limits.min) * Deslizador.value + eslabon.GetComponent<HingeJoint>().limits.min;
```

Figura 34: Calculo de la posición objetivo

Finalmente la posición objetivo calculada debe introducirse en la casilla "Target position" del componente "hingejoint" de la articulación en cuestión (figura 35).

```
//Posicion objetivo aplicada al eslabon
HingeJoint hinge = eslabon.GetComponent<HingeJoint>();
JointSpring hingeSpring = hinge.spring;
hingeSpring.targetPosition = posicion;
hinge.spring = hingeSpring;
```

Figura 35: Posición objetivo comunicada al controlador

7. Conclusiones

El proyecto de creación de una aplicación que permite crear un robot para posteriormente ser proyectado en realidad aumentada, como se observa en la figura 36, ha sido un éxito. La aplicación ha sido desarrollada utilizando tecnologías en realidad aumentada, lo que permite a los usuarios crear robots realistas y detallados. La aplicación también logra permitir a los usuarios interactuar con sus robots en tiempo real, lo que hace que la experiencia sea muy divertida e inmersiva.

Figura 36: Simulación ejemplo como resultado del proyecto

La aplicación ha sido probada por un grupo de usuarios y el resultado ha sido muy positivo. Los usuarios han valorado la facilidad de uso de la aplicación, la calidad de los modelos de robots y la posibilidad de interactuar con los robots en tiempo real.

El proyecto ha ampliado mis conocimientos en el campo de la realidad aumentada, asi como en el uso del motor de desarrollo de videojuegos “Unity”. La aplicación puede ser utilizada para una gran variedad de propósitos, incluyendo la educación y el entretenimiento.

A continuación se presentan algunas de las posibles mejoras para la aplicación:

- Aumentar la cantidad de modelos prefabricados que pueden ser utilizados como ejemplos, además del robot ABB IRB 7600 con el que se cuenta actualmente.
- Aumentar la cantidad de modelos que pueden utilizarse para cada una de las partes del robot personalizado, es decir poder seleccionar entre un amplio abanico de modelos tanto para la base, eslabones y efector final.
- Ampliar la personalización de la partes, como por ejemplo cambios de colores, solo cambiar tamaño de ciertas partes, etc.
- Incluir la posibilidad de modificar otras propiedades de las articulaciones, como por ejemplo: límites, velocidades máximas, aceleraciones máximas, etc.
- Incluir efectos dinámicos en la simulación, asi como la implementación de motores en las articulaciones contemplando efectos de inercia y fuerzas externas a las que se someta el robot.

8. Bibliografía

- Página web oficial de Unity: https://unity.com/es
- Unity Learn: https://learn.unity.com/
- Unity Asset Store: https://assetstore.unity.com/
- Unity Forums: https://forum.unity.com/
- Unity Docs: https://docs.unity.com/
- Página web oficial Vuforia: https://developer.vuforia.com/