TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TUXTLA GUTIÉRREZ
DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

INGENIERÍA ELECTRÓNICA

Informe Técnico de Residencia Profesional

NOMBRE DEL PROYECTO

SISTEMA DE CAPACITACIÓN DE UN BRAZO ROBÓTICO VIRTUAL DE 2 GRADOS DE LIBERTAD BAJO REALIDAD AUMENTADA.

PRESENTA

Dimas de la Peña López.

N° de Control

14270565

Empresa: Centro de Investigaciones en Óptica, A.C.

Asesor Interno: M. en C. Osvaldo Brindis Velázquez

Asesor Externo: Dr. Carlos Alberto Paredes Orta

Resumen

Los procesos de capacitación y enseñanza para el manejo de dispositivos en el área de robótica avanzan sin pausa, y en áreas donde la eficiencia y la rapidez, así como la claridad de los conceptos presentados a los alumnos es de vital importancia es fundamental contar con las herramientas que le permitan al estudiante comprender el funcionamiento de éstos dispositivos. Una de las principales tendencias tecnológicas durante los últimos años es la creación de gemelos digitales. A grandes rasgos, la apuesta por esta tecnología no es más que la generación de una réplica virtual de un producto que simula el comportamiento de su homólogo físico, con el objetivo de monitorearlo y analizar su funcionamiento.

El objetivo de este proyecto es la elaboración de una aplicación de realidad aumentada controlada desde un dispositivo Android que permita visualizar el gemelo digital de un brazo robótico de 2 grados de libertad. Esta aplicación nos da el poder para controlar el movimiento del gemelo digital, la programación de rutinas y muestra los posibles ángulos de los brazos, así como la posición del efector final, calculados a partir de sus fórmulas de cinemática inversa y directa respectivamente. Además se busca que ésta misma aplicación pueda conectarse al robot real y permita un control simultaneo del robot y de su gemelo digital.

Esta aplicación se muestra como una herramienta práctica que le permite a los estudiantes comprender el funcionamiento y entender los cálculos de cinemática del robot, así como aprender a programar rutinas, para después ser llevadas a la práctica en el robot real usando la misma aplicación. Los resultados muestran un aprendizaje favorable por parte de los estudiantes y se presenta como una herramienta realmente útil para estudiar el funcionamiento y teoría detrás de los brazos robóticos. Se podrían realizar investigaciones adicionales para adaptar la aplicación a robots que cuenten con mayores grados de libertad, o robots que cuenten con otros tipos de articulaciones (lineales, cilíndricas, esféricas).

Índice

Índice de figuras

Justificación

El presente proyecto se enfocará al desarrollo de una aplicación de realidad aumentada que nos permita visualizar el gemelo digital de un brazo robótico de 2 grados de libertad, así como el control del gemelo digital y del robot a través de ésta.

El desarrollo de dicho proyecto nos proporcionará una nueva tecnología educativa que brindará a estudiantes en el CIO una nueva herramienta para mejorar su rendimiento estudiantil. Poder visualizar en un robot virtual conceptos de róbotica le permitirá al estudiante un mayor aprendizaje y compresión de éstos ya que éstos son más tangibles con un ejemplo visual.

Además el proyecto nos brindará conocimiento en distintas áreas de la ingeniería como electrónica, programación e industrial. Algunos de los beneficios que éste proyecto nos ofrece son conocimiento en el desarrollo de aplicaciones de realidad aumentada, creación de gemelos digitales para control y monitoreo remoto de robots, desarrollo de interfaces en Android para enseñanza y capacitación, entre muchas otras.

El proyecto tiene como base investigaciones anteriores relacionadas al desarrollo de aplicaciones de realidad aumentada aplicadas a la industria y la robótica, así que expande el conocimiento previo, pero también abre muchas puertas a futuras investigaciones que podrían adaptar la aplicación a nuevos modelos de robot que cuentan con mayor número de grados de libertad o con diferente tipo de articulaciones, la aplicación se puede adaptar a plataformas diferentes a android, el control remoto se puede realizar a través de diferentes metodos de comunicación además de wifi y también se pueden monitorear nuevos parámetros, así que el proyecto invita a una continua investigación para seguir expandiendo el conocimiento y a la creación de nueva y mejor tecnología.

Objetivos

OBJETIVO GENERAL

Realizar un sistema de realidad aumentada demostrativo para capacitación de la manipulación de un brazo robótico virtual de 2 grados de libertad bajo la técnica de realidad aumentada.

OBJETIVOS ESPECÍFICOS

- Construir un robot de 2 grados de libertad (brazo mecánico, actuadores y controlador.)
- Modelar el robot haciendo uso de sofware CAD (Diseño Asistido por Computadora) y hacer los cálculos de la cinemática directa e inversa para el modelado.
- Seleccionar marcadores y realizar configuración en Unity para objetos en realidad aumentada.
- Desarrollar interfaz en Unity para control de movimiento de modelo 3D del robot y mostrar los cálculos de cinemática directa e inversa.
- Realizar la comunicación vía wifi entre la aplicación y el robot para el control del mismo.
- Desarrollar una base de datos para almacenar rutinas del robot.
- Exportar aplicación a Android, realizar pruebas y ajustes para encontrar posibles errores e integrar a la plataforma de capacitación.

Problemas a resolver

- La materia de mecatrónica en el CIO estudia temas de robótica entre los cuales se encuentran temas tales como la cinemática del robot, y a pesar de que existen programas para simulación que nos entregan datos como matlab, muchas veces la ausencia de una representación visual de esta información evita que el estudiante pueda internalizar y comprender plenamente éstos conceptos.

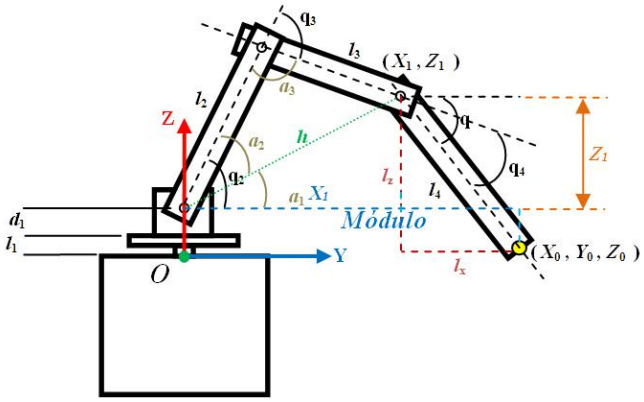


Imagen 1. Cinemática de un robot de 3 grados de libertad (Fuente: ResearchGate.Net)

- Las aplicaciones de realidad aumentada y el uso de gemelos digitales son nuevos campos dentro de la rama de la tecnología que aun son muy poco explorados y que cuentan con mucho potencial. Debido a esto en la actualidad aun no existe ninguna aplicación en la plataforma Android enfocada a robótica y que nos permita el control simultaneo de un robot así como de su gemelo digital.

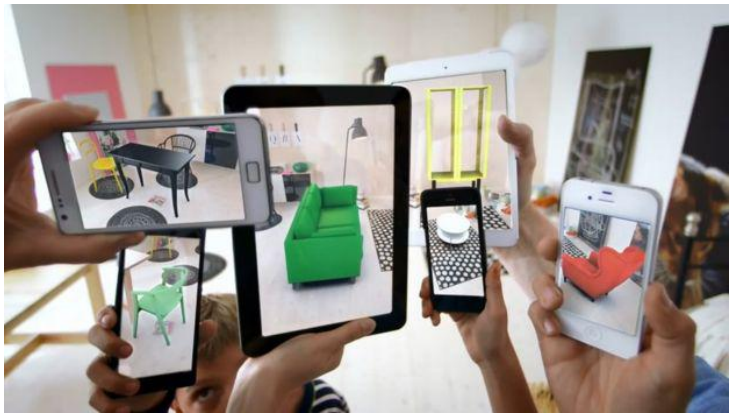


Imagen 2. Aplicación de realidad aumentada (Fuente: BBC News)

Marco Teórico

Robots

Si queremos modelar el gemelo digital de un brazo robótico, lo primero que tenemos que entender es el concepto de robot. Una definición oficial robot proviene del Robot Institute of America (RIA): *"un robot es un manipulador multifuncional reprogramable diseñado para mover material, piezas, herramientas o dispositivos especializados a través de movimientos programados variables para la realización de una variedad de tareas."*

El control y simulación de robots requiere del desarrollo de modelos matemáticos (geométricos, cinemáticos, etc) dependiendo de los objetivos, la importancia de la tarea a realizar y el funcionamiento esperado. Equipados con estos modelos matemáticos, podremos desarrollar métodos para planificar y controlar los movimientos del robot para realizar tareas específicas. Los manipuladores robóticos están compuestos de enlaces conectados por uniones para formar una cadena cinemática. Las articulaciones son típicamente rotativas (revolutivas) o lineales (prismáticas). Una junta giratoria es como una bisagra y permite la rotación relativa entre dos enlaces. Una articulación prismática permite un movimiento relativo lineal entre dos enlaces. (Spong, 2005)

Denotamos las juntas revolutivas por R y las prismáticas por P, y las dibujamos como se muestra en la siguiente figura.

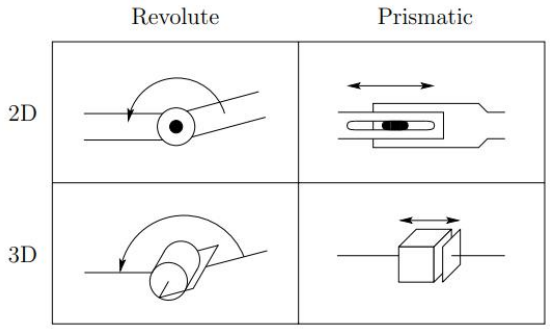


Imagen 3. Representación simbólica de articulaciones (Fuente: Robot Modeling and Control (Spong, 2005))

Las variables de la articulación son denotadas por θ para una articulación revolutiva y d para la articulación prismática.

Se dice que un objeto tiene n grados de libertad (DOF) si su configuración puede especificarse mínimamente mediante n parámetros. Por lo tanto, el número de DOF es igual a la dimensión del espacio de configuración. Para un robot manipulador, el número de articulaciones determina el número DOF. Un manipulador que tiene más de seis enlaces se denomina manipulador cinemáticamente redundante. La dificultad de controlar un manipulador aumenta rápidamente con el número de enlaces. (Aguilar, 2011)

El robot con el que vamos a trabajar cuenta con dos articulaciones revolutivas y es denominado un robot de 2 grados de libertad.

Los manipuladores de robots pueden clasificarse según varios criterios, como su fuente de energía o la forma en que se actúan las juntas, su geometría o estructura cinemática, su área de aplicación prevista o su método de control. Dicha clasificación es útil principalmente para determinar qué robot es el adecuado para una tarea determinada.

Fuente de alimentación. Por lo general, los robots tienen alimentación eléctrica, hidráulica o neumática. Los actuadores hidráulicos no tienen rival en cuanto a su velocidad de respuesta y capacidad de producción de torque. Por lo tanto, los robots hidráulicos se utilizan principalmente para levantar cargas pesadas. Los inconvenientes de los robots hidráulicos son que tienden a derramar fluido hidráulico, requieren mucho más equipo periférico (como bombas, que requieren más mantenimiento) y son ruidosos. Los servomotores DC o AC son cada vez más populares, ya que son más baratos, limpios y silenciosos. Los robots neumáticos son económicos y simples, pero no se pueden controlar con precisión. Como resultado, los robots neumáticos están limitados en su rango de aplicaciones y popularidad. **Nosotros vamos a trabajar con servomotores DC debido a las ventajas mencionadas.**

Método de control. Los robots se clasifican por método de control en robots servo y no servo. Los primeros robots fueron no servo robots. Estos robots son esencialmente dispositivos de circuito abierto cuyo movimiento se limita a paradas mecánicas predeterminadas, y son útiles principalmente para la transferencia de materiales. De hecho, de acuerdo con la definición dada anteriormente, los robots de parada fija apenas califican como robots. Los servo robots usan el control de computadora de circuito cerrado para determinar su movimiento y, por lo tanto, son capaces de ser dispositivos verdaderamente multifuncionales y reprogramables. **Por su método de control, nuestro robot es servocontrolado.**

El tipo más simple de robot en esta clase es el robot punto a punto. A un robot punto a punto se le puede enseñar un conjunto discreto de puntos, pero no hay control sobre la ruta del efector final entre los puntos enseñados. A estos robots generalmente se les enseña una serie de puntos, los puntos se almacenan y reproducen. En los robots de trayectoria continua, por otro lado, se puede controlar toda la trayectoria del efector final. Estos son los robots más avanzados y requieren los controladores de computadora más sofisticados y el desarrollo de software. **Debido a la aplicación que le vamos a dar, nosotros vamos a diseñar un robot punto a punto.**

Geometría. **El robot con el que vamos a trabajar es considerado un robot en serie.** Existe una clase distinta de manipuladores consiste en el llamado robot paralelo. En un manipulador paralelo, los enlaces están dispuestos en una cadena cinemática cerrada en lugar de abierta. Sus cinemáticas y dinámicas son más difíciles de obtener que las de los robots de enlace en serie y, por lo tanto, generalmente se tratan solo en aplicaciones más avanzadas.

Modelo cinemático de un Robot Planar de 2 grados de libertad

El problema de la cinemática es describir el movimiento del manipulador sin tener en cuenta las fuerzas y los pares que causan el movimiento. La descripción cinemática es, por lo tanto, geométrica. Primero consideramos el problema de la cinemática directa, que consiste en determinar la posición y orientación del efector final dados los valores para las variables conjuntas del robot. El problema de la cinemática inversa es determinar los valores de las variables conjuntas dada la posición y orientación del efector final. (Callejas, 2006)

Para nuestro proyecto consideraremos un tipo idealizado de robot incrustado en el plano cartesiano estándar, llamado robot plano, el cual contará con dos grados de libertad y cuyas articulaciones serán revolutivas. La representación gráfica de nuestro robot se puede ver a continuación.

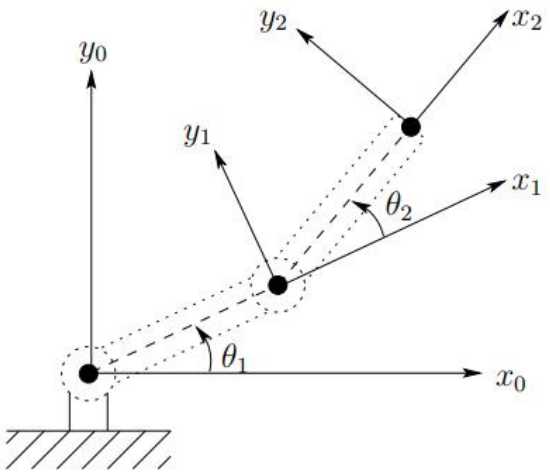


Imagen 4. Robot Planar de 2 grados de libertad (Fuente: Robot Modeling and Control (Spong, 2005))

Primero abordaremos el problema de la cinemática directa, y luego el de la cinemática inversa de nuestro robot.

CINEMÁTICA DIRECTA

El problema de la cinemática directa se refiere a la relación entre las articulaciones individuales del robot manipulador y la posición y orientación de la herramienta o efector final. (Montiel, 2019)

“Las variables de unión son los ángulos entre los enlaces en el caso de juntas giratorias, y la extensión del enlace en el caso de juntas prismáticas o deslizantes.” (Rodríguez; Mckinley; Ramírez, 2017)

Usualmente se suele utilizar el método de representación sistemátíca de Denavit-Hartenberg, el cual hace uso de matrices de transformación homogenea. Este método ofrece la ventaja de conocer tanto la posición final de manipulador como la posición de cada una de sus articulaciones. (Ramírez; Rubiano, 2012)

El análisis cinemático de un manipulador de n enlaces puede ser extremadamente complejo y las convenciones que se usan para el análisis también lo son. Sin embargo es posible llevar a cabo un análisis cinemático avanzado incluso sin respetar estas convenciones, debido a que estamos trabajando con un manipulador plano de dos enlaces y por lo tanto se puede realizar un análisis geométrico.

Es habitual establecer un sistema de coordenadas fijo, llamado marco base al que se hace referencia a todos los objetos, incluido el efector final. En este caso, establecemos el marco de coordenadas base $o_0x_0y_0$ en la base del robot.

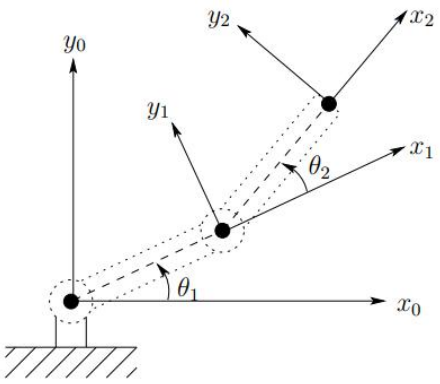


Imagen 5. Marcos coordinados para robot plano de dos enlaces. (Fuente: Robot Modeling and Control (Spong, 2005))

Las coordenadas (x, y) de la herramienta se expresan en este marco de coordenadas como

$$x = x_2 = \alpha_1 \cos \theta_1 + \alpha_2 \cos (\theta_1 + \theta_2) \quad (1.1)$$

$$y = y_2 = \alpha_1 \sin \theta_1 + \alpha_2 \sin (\theta_1 + \theta_2) \quad (1.2)$$

en donde α_1 y α_2 son las longitudes de los dos enlaces, respectivamente. Los valores de $\mathbf{x_2}$ y $\mathbf{y_2}$ son los valores que buscamos con la cinemática directa, ya que nos indican la posición de nuestro efector final. Las ecuaciones (1.1) y (1.2) se denominan **ecuaciones cinemáticas directas para este brazo**.

CINEMÁTICA INVERSA

Ahora, dados los ángulos de unión θ1, θ2 podemos determinar las coordenadas efector final x e y. Para ordenarle al robot que se mueva a la ubicación A necesitamos lo inverso; es decir, necesitamos las variables conjuntas θ1, θ2 en términos de las coordenadas x y y. Este es el problema de la cinemática inversa. En otras palabras, dados x y y, deseamos resolver los ángulos de las articulaciones. Como las ecuaciones cinemáticas directas no son lineales, una solución puede no ser fácil de encontrar, ni existe una solución única en general. (Cox; Little; O'Shea, 2008)

Podemos ver en el caso de un mecanismo plano de dos enlaces que puede no haber solución, por ejemplo, si las coordenadas (x, y) dadas están fuera del alcance del manipulador. Si las coordenadas dadas (x, y) están dentro del alcance del manipulador, puede haber dos soluciones como se muestra en la figura, las denominadas configuraciones de *codo hacia arriba* y *codo hacia abajo*, o puede haber exactamente una solución si el manipulador debe estar completamente extendido para llegar al punto.

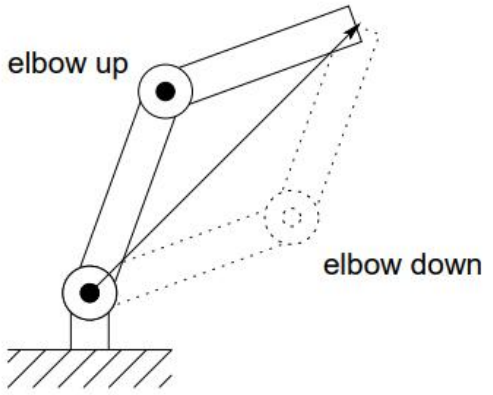


Imagen 6. Múltiples soluciones de cinemática inversa (Fuente: Robot Modeling and Control (Spong, 2005))

Considerando el siguiente diagrama:

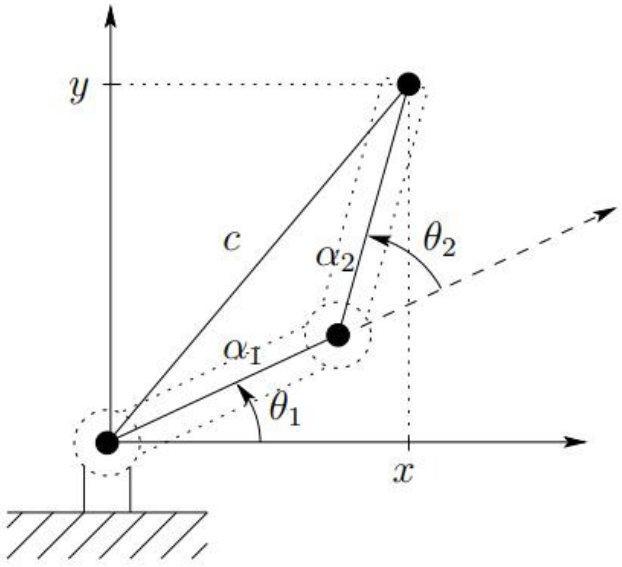


Imagen 7. Resolviendo los ángulos articulares de un brazo planar de dos enlaces (Fuente: Robot Modeling and Control (Spong, 2005))

Usando la Ley de cosenos vemos que el ángulo θ_2 viene dado por:

$$\cos\theta_2 = \frac{x^2 + y^2 - \alpha_1^2 - \alpha_2^2}{2\alpha_1\alpha_2} := D \tag{1.4}$$

Ahora podríamos determinar θ_2 como:

$$\theta_2 = \cos^{-1}(D) \quad (1.5)$$

Sin embargo, una mejor manera de encontrar θ_2 es notar que si cos (θ_2) viene dado por la ecuación (1.4) entonces sin (θ_2) se da como:

$$\sin(\theta_2) = \pm\sqrt{1 - D^2} \quad (1.6)$$

y, por lo tanto, θ_2 se puede encontrar por

$$\theta_2 = \tan^{-1}\frac{\pm\sqrt{1 - D^2}}{D} \quad (1.7)$$

La ventaja de este último enfoque es que las soluciones codo arriba y codo abajo se recuperan eligiendo los signos positivos y negativos en la ecuación (1.7), respectivamente. Finalmente, conociendo θ_2 y aplicando expresiones trigonométricas y aplicando analogamente el mismo análisis podemos llegar a θ_1, el cual ahora se da como:

$$\theta_1 = \tan^{-1}(y/x) - \tan^{-1}\left(\frac{\alpha_2 \sin\theta_2}{\alpha_1 + \alpha_2 \cos\theta_2}\right) \quad (1.8)$$

Observe que el ángulo θ1 depende de θ2. Esto tiene sentido físicamente ya que esperaríamos requerir un valor diferente para θ1, dependiendo de la solución elegida para θ2 (codo arriba o codo abajo).

Las ecuaciones (1.7) y (1.8) se denominan **ecuaciones cinemáticas inversas para este brazo**.

Software CAD (Solidworks)

Para poder crear un modelo 3D del robot, se hace uso de Software CAD (Diseño Asistido por Computadora). Los programas CAD son una herramienta indispensable para la concepción de proyectos. Los software de diseño asistido por ordenador van muchos más allá que los programas de dibujo, además de presentarnos los planos de los proyectos, nos ayudan a hacer todos los cálculos previos a la creación o construcción de un objeto. Los programas CAD facilitan la concepción 2D y el diseño 3D de objetos, así como la modelización 3D.

Los software CAD han reemplazado los programas para dibujo técnico en el sector de la industria. Los programas CAD 3D son sumamente útiles para empresas que operan en sectores variados, también lo son para estudiantes y hasta para particulares que desean ocuparse de sus propios diseños.

Si se habla de diseño asistido por computadora desde una perspectiva de diseño mecánico, esto permite la generación de piezas y productos partiendo muchas de las veces mediante bocetos los cuales se pueden extruir para la generación de su volumen y por medio de la misma interfaz del paquetería de CAD definir su geometría mediante diversas operaciones de vaciado, recortes, chaflanes, barrenados, etcétera. La mayoría de la paquetería de diseño mecánico cuenta con herramientas de cálculo rápido de las configuraciones dibujadas para la obtención de centros de gravedad, momentos de inercia, masa y propiedades físicas muy útiles en los procesos de cálculo y diseño ingenieril. (Solorzano, 2012)

Uno de los mejores y más usados softwares para diseño mecánico es **Solidworks.** SolidWorks es un programa de diseño mecánico en 3D con el que puedes crear geometría 3D usando sólidos paramétricos, la aplicación esta enfocada a diseño de producto, diseño mecánico, ensambles, y dibujos para taller. SolidWorks diseña de forma que va dejando un historial de operaciones para que puedas hacer referencia a ellas en cualquier momento. Como herramienta de diseño 3D es fácil

de usar, acompaña al ingeniero mecánico y el diseñador industrial en su desempeño diario.

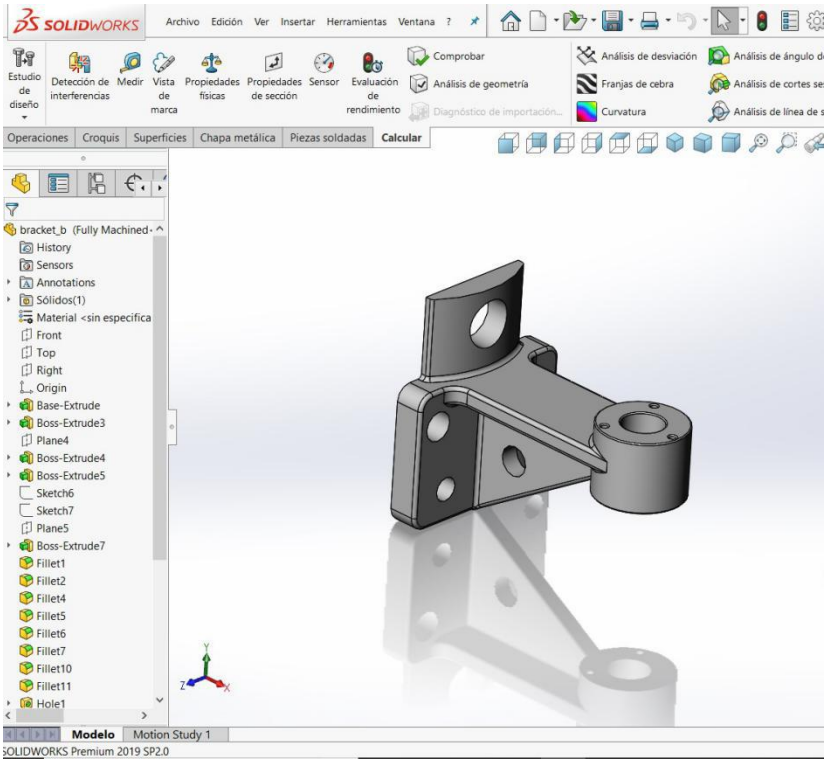


Imagen 8. Interfaz de Solidworks (Fuente: solidworks.com)

Con Solidworks puedes diseñar piezas mecánicas en 3D, evaluar ensambles de varias piezas y producir dibujos de fabricación para el taller, ademas puedes manejar los datos de diseño en su sistema de administración PDM y llevar un control de las versiones de dibujos.

El paquete de simulación de SolidWorks garantiza la calidad y el funcionamiento de sus diseños antes de comprometerse a fabricarlos. Las exhaustivas herramientas de análisis le permiten probar digitalmente los modelos para obtener una percepción técnica de gran valor al comienzo del proceso de diseño. (Toapanta, 2015)

Realidad Aumentada

Para poder entender el funcionamiento del proyecto, hay que tener claro que es la realidad aumentada.

La Realidad Aumentada es una tecnología que complementa la percepción e interacción con el mundo real y permite al usuario estar en un entorno real aumentado con información adicional generada por el ordenador. Esta tecnología está introduciéndose en nuevas áreas de aplicación como son entre otras la reconstrucción del patrimonio histórico, el entrenamiento de operarios de procesos industriales, marketing, el mundo del diseño interiorista y guías de museos. (Guaitara, 2014)

El término de realidad aumentada sería acuñado por Tom Caudell a principios de los 90s y en 1997, Ronald Azuma, el denominado padre de la realidad aumentada, codificó las características de la misma. A partir de esa fecha las experiencias han tenido un crecimiento exponencial, en los que han sucedido los avances y desarrollos que han hecho madurar ésta tecnología. (Torres, 2013)

Imagen 9. Ejemplo de aplicación usando realidad aumentada (Fuente: pokemongo.com)

Bajo el parámetro de realidad aumentada se agrupan así aquellas tecnologías que permiten la superposición, en tiempo real, de imágenes, marcadores o información

generados virtualmente, sobre imágenes del mundo real. De esta manera se crea un entorno en el que la información y los objetos virtuales se fusionan con los objetos reales ofreciendo una experiencia tal para el usuario que puede llegar a pensar que forma parte de su realidad habitual.

Sin embargo, la realidad aumentada no se ha quedado limitada a su conceptualización, ya se han desarrollado muchas aplicaciones tecnológicas que la incluyen y además que demuestran su potencial para actividades humanas. (Vargas, 2016)

Ahora la pregunta principal reside en *¿Como funciona la realidad aumentada?*

Cuando se habla de un sistema de RA, se establece la conjunción de una serie de elementos, tanto de hardware y de software, que se relacionan entre si, permitiendo la creación, visualización y consulta de datos digitales en este contexto de funcionamiento.

De esta manera los componentes básicos del sistema son

Hardware:

o Un ordenador, el cual puede ser un PC o un dispositivo móvil (tableta,

teléfono inteligente o gafas).

o Un monitor o dispositivo de visualización de los datos.

o Una cámara para la captura de los datos del entorno y que actúa como

rastreador.

Software:

o Una aplicación o programa que se ejecute desde el dispositivo a utilizar.

o Servicios web o un servidor de contenidos de RA.

Existen diversas taxonomías de RA, que clasifican este tipo de tecnología de acuerdo a las diversas formas de implementaciones que se presentan. Una clasificación inicial la cual está definida por las características del hardware a utilizar, divide a los sistemas de RA en dos tipos: fija y móvil. Mientras que un sistema móvil aporta al usuario la posibilidad de efectuar desplazamientos sobre un medio determinado, los sistemas fijos son inflexibles en este aspecto. ()

En nuestro caso haremos uso de la RA móvil, debido a que la aplicación está diseñada para dispositivos Android.

Otra clasificación de RA está relacionada con el tipo de reconocimiento o lectura que realiza el sistema. Esta nueva clasificación tiene en cuenta los niveles o grados de complejidad del objeto marcador o del reconocimiento. Se pueden emplear diferentes tipos de objetos reconocibles (trackables):

- **Marcadores (markers)**: se caracterizan por la simplicidad gráfica de su presentación, que es una derivación de matrices de pixeles con profundidades de color de 1 bit, de tal modo que pueden ser reconocidas fácilmente empleando cámaras digitales.

Imagen 10. Marcador QR (Fuente: MODELO DE REALIDAD AUMENTADA Y NAVEGACIÓN PEATONAL (Joo, 2011))

- **Imágenes:** para este caso, el objeto reconocible que da lugar a la activación de la RA corresponde a imágenes que actúan como marcadores con una mayor elaboración en su diseño gráfico. Este tipo de RA se denomina “sin marcadores”

(markerless), ya que finalmente no se emplea un código, sino que más bien el identificador es una imagen con diferentes dimensiones, texturas y colores.

Imagen 11. Markerless presente en una caja de medicamentos (Fuente: MODELO DE REALIDAD AUMENTADA Y NAVEGACIÓN PEATONAL (Joo, 2011))

- Objetos: En la actualidad, existen aplicaciones que son capaces de reconocer objetos completos, tridimensionales y con texturas complejas como son piezas mecánicas, maquetas, caras o incluso edificaciones. Esto ha significado la evolución de los rastreadores, permitiendo la posibilidad de prescindir de una gráfica impresa o intrusiva.

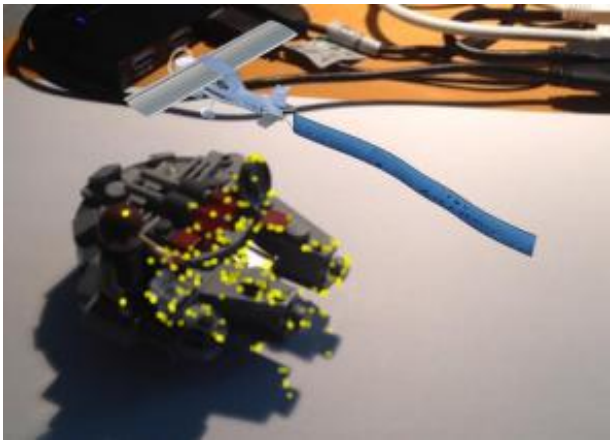


Imagen 12. Objeto como marcador para RA (Fuente: MODELO DE REALIDAD AUMENTADA Y NAVEGACIÓN PEATONAL (Joo, 2011))

El proyecto hace uso de la detección de imágenes como marcadores que generan la realidad aumentada.

Los niveles de complejidad en la detección de los objetos reconocidos –o su inexistencia–, están directamente relacionados con las altas prestaciones de hardware existentes, en donde los niveles de procesamiento, la posibilidad de capturar imágenes en alta definición y la utilización de sensores complementarios, permiten experiencias menos complejas en su implementación y con un mayor grado de presencialidad en el visualización de la información digital.

Una de las variantes más atractivas de la RA es su posibilidad de portabilidad, mediante tabletas y teléfonos inteligentes, los cuales ya cuentan con una potencia similar a la que poseen los ordenadores portátiles o de escritorio. La visualización de contenidos en RA tiene así como elemento añadido un contexto de movilidad, que permite la creación de actividades ubicuas, sin las limitaciones espaciales de los lugares fijos, ni la necesidad de entornos cuidadosamente acondicionados para su correcto funcionamiento.

En la actualidad, tabletas y teléfonos inteligentes cuentan con herramientas que permiten una adecuada implementación de contenidos de RA y que definen este nuevo contexto “móvil”.

- Pantallas portátiles de alta resolución las cuales permiten la visualización y la implementación de la información conjunta de datos digitales y la realidad.

- Cámaras que detectan la información presentada en la realidad.

En la actualidad, estas características están presentes en tabletas y teléfonos inteligentes, que, aunque no son dispositivos vestibles, mantienen todas las características esenciales para la generación de un ambiente de RAM programado.

Realidad Aumentada como herramienta de aprendizaje

Dentro de la incorporación de nuevas herramientas en los ambientes educativos, la RA es una de las más destacadas e importantes tecnologías desarrolladas para este fin (Adams, 2005; L. Johnson, Smith, Willis, Levine, & Haywood, 2011; Martínez Landa, 2015). Esto se encuentra relacionado con las características que tiene la RA cómo tecnología, particularmente en lo concerniente a la posibilidad de generar un entorno inmersivo e interactivo.

Imagen 13. Realidad aumentada como herramienta de aprendizaje (Fuente: omniumgames.com)

La sensación de inmersión que tienen los usuarios tiene importantes consecuencias en contextos educativos digitales (Dede, 2009):

- La posibilidad de generar múltiples perspectivas de visualización en las que el usuario puede cambiar el contexto de referencia, y que pueden pasar de ser exocéntricas (la visualización desde el exterior del objeto) a egocéntricas (una visualización desde el interior del objeto).

- El contexto de un aprendizaje situado en donde, en un mismo nivel, el contexto y el conocimiento temático son aplicados para mejorar su comprensión y proceso de aprendizaje. Generar entornos virtuales de aprendizaje completos es un proceso complejo, sin embargo estos modelamientos de un mundo virtual o aumentado, permitirían a los estudiantes comprender los contenidos en un contexto real.

- Los estudiantes podrían aplicar su conocimiento adquirido en nuevos contextos o situaciones, generándose un fenómeno de transferencia de contenidos o de aprendizaje.

De la misma manera, González (2013) presenta razones complementarias para la utilización de la RA en educación: permite la utilización de contenidos didácticos que no son posibles de otra manera y aporta elementos como interactividad, elementos lúdicos, experimentación y trabajo colaborativo.

Adicionalmente, una serie de estudios han probado los efectos de la RA en diferentes dimensiones y temáticas educativas, como son procesos y cambio conceptual (Shelton & Stevens, 2004); emulación en trabajos de laboratorio (Andujar, Mejías, & Marquez, 2011); espacialidad y territorialización (Huang, Schmidt, & Gartner, 2012); aprendizaje basado en consultas (Squire & Klopfer, 2007), entre otros. Los resultados de estos estudios demuestran las actitudes positivas hacia esta herramienta, particularmente en lo concerniente a la satisfacción en el uso y la percepción de utilidad.

Unity para desarrollo de aplicaciones de realidad aumentada

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X, Linux. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataformas entre las que se encuentran: WebGL, Windows, OS, OS X, GNU/Linux, iOS, **Android (que es la plataforma en la que buscamos desarrollar el proyecto),** Windows Phone, PlayStation, Xbox, Nintendo, etc.

Unity puede usarse junto con Blender, 3ds Max, Maya, y otros software de modelado 3D. Los cambios realizados a los objetos creados con estos productos se actualizan automáticamente en todas las instancias de ese objeto durante todo el proyecto sin necesidad de volver a importar manualmente.

Los programadores pueden utilizar UnityScript (un lenguaje personalizado inspirado en la sintaxis ECMAScript), **C# (el cual es usado para el desarrollo de éste proyecto)** o Boo (que tiene una sintaxis inspirada en Python).

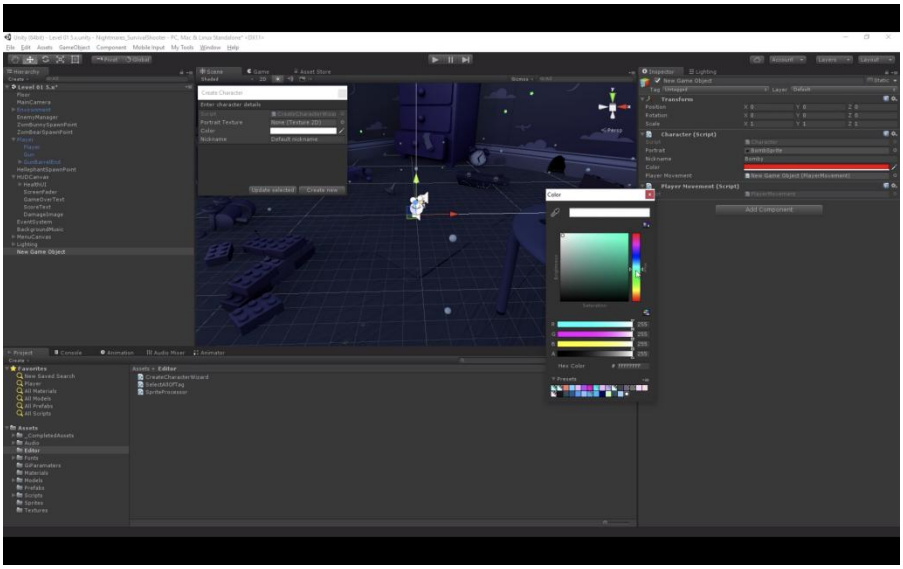


Imagen 14. Interfaz de Unity (Fuente: Propia)

ASSETS EN UNITY

Unity también incluye Unity Asset Server - una solución de control de versiones para todos los assets de juego y scripts. Un asset es una representación de cualquier item que puede ser utilizado en su juego o proyecto. Un asset podría venir de un archivo creado afuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos que Unity soporta.

Dentro de estos assets, se encuentran los kits de desarrollo de software (SDK), los cuales son generalmente un conjunto de herramientas de desarrollo de software que permite a un desarrollador de software crear una aplicación informática para un sistema concreto. Estos SDK nos permiten desarrollar aplicaciones en Unity que nos permitan, por ejemplo:

- **DESARROLLAR Y EXPORTAR APLICACIONES A LA PLATAFORMA ANDROID (ANDROID SDK)**
- **CREAR APLICACIONES QUE CUENTEN CON REALIDAD AUMENTADA (VUFORIA)**
- **CONEXIÓN VÍA WIFI ENTRE UNITY Y DISPOSITIVOS A TRAVÉS DE LENGUAJE ARDUINO (UDUINO)**
- **CREACIÓN DE BASES DE DATOS (SQLITE)**

Como se puede observar las herramientas que nos proporcionan los Assets de Unity, así como la potencia y versatilidad del software nos permitirán desarrollar nuestra aplicación contando con todas las funciones solicitadas.

Procedimiento y descripción de las actividades realizadas

Construir un robot de 2 grados de libertad (Brazo mecánico, actuadores y controlador)

BRAZO MECÁNICO

La primera parte en el desarrollo del proyecto fue la construcción del brazo robótico. Primero empezamos con la construcción del brazo mecánico. El software que se decidió usar para el diseño de cada una de la piezas del robot fue Solidworks. Una vez diseñadas las piezas y tomando en cuenta las medidas de éste diseño, posteriormente se cortaron y ensamblaron las piezas El material que se escogió usar para el brazo fue acero. Se eligió este material debido a que es muy resistente y con un buen diseño nos permite un robot robusto y durable.

El corte de laminas de acero se realizó con mucha facilidad debido a que el centro de investigaciones cuenta con un taller de diseño mecánico que nos permitió la realización de éstas piezas.

Además se cuenta con una base de madera que sujeta nuestro brazo mecánico. Esta base de madera cuenta con una pequeña plataforma cuya unica función es elevar un poco el brazo mecánico.

El brazo mecánico principal está formado por tres piezas principales. La primera sujeta nuestro brazo mecánico a una base de madera y sostiene uno de los servomotores. A continucación podemos ver el diseño en Solidworks de la base de madera unida a la primera pieza de nuestro brazo mecánico y posteriormente como se ve una vez realizadas ambas piezas.

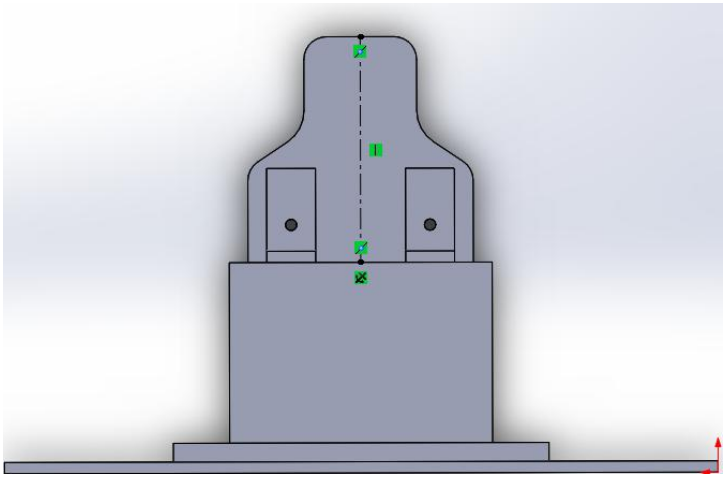


Imagen 15. Vista frontal de la pieza unida a la base (Fuente: Propia)

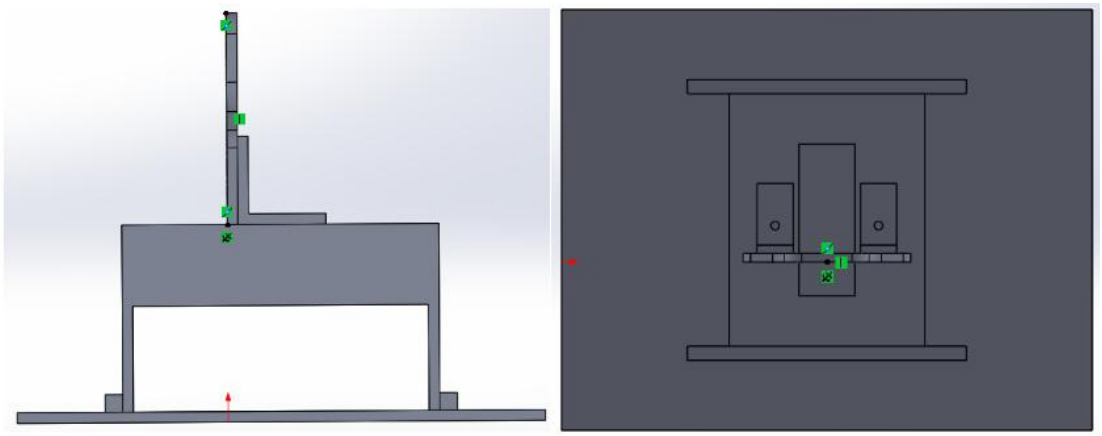


Imagen 16. Vistas lateral y superior de la pieza unida a la base (Fuente: Propia)

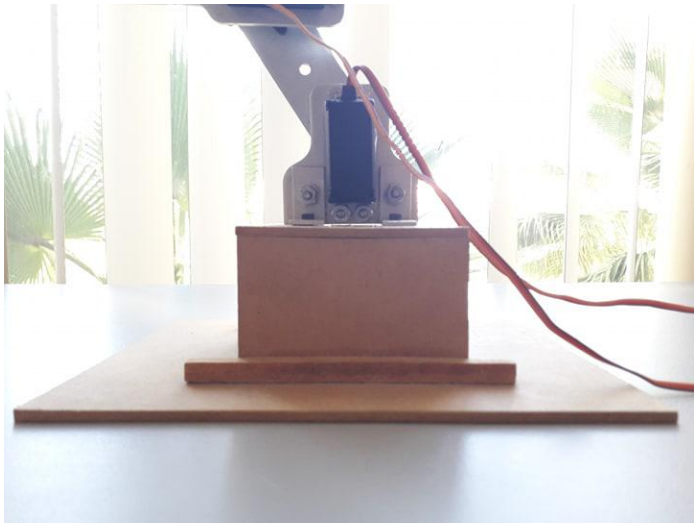


Imagen 17. Fotografía de la pieza unida a la base (Fuente: Propia)

La segunda pieza es el primer brazo de nuestro robot. Esta pieza conecta las otras dos piezas que cuentan con servomotores. La longitud que existe entre los centros de rotación de los servomotores es importante, ya que es la que determina la longitud del primer brazo, esta longitud es la que se toma en cuenta para los cálculos posteriores de cinemática de nuestro robot. Como se puede observar en la imagen, **la longitud de nuestro primer brazo es de 7.5 cm**.

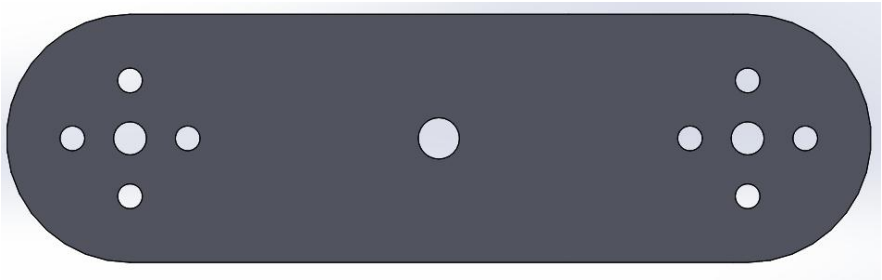


Imagen 18. Vista frontal del primer brazo (Fuente: Propia)

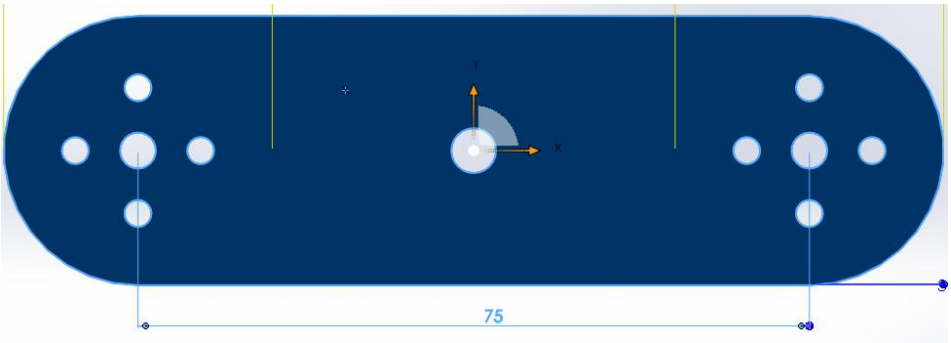


Imagen 19. Longitud del primer brazo (Fuente: Propia)

Imagen 20. Fotografía del primer brazo (Fuente: Propia)

Finalmente contamos con la tercer pieza, que es el segundo brazo de nuestro robot. Esta pieza sujeta el segundo servomotor y se encuentra conectada al primer brazo. También es importante encontrar la distancia entre el centro de rotación de nuestro servo y la punta del brazo, se conoce como la longitud del segundo brazo. De acuerdo a nuestro ensamble final, **esta longitud es de 6 cm**.

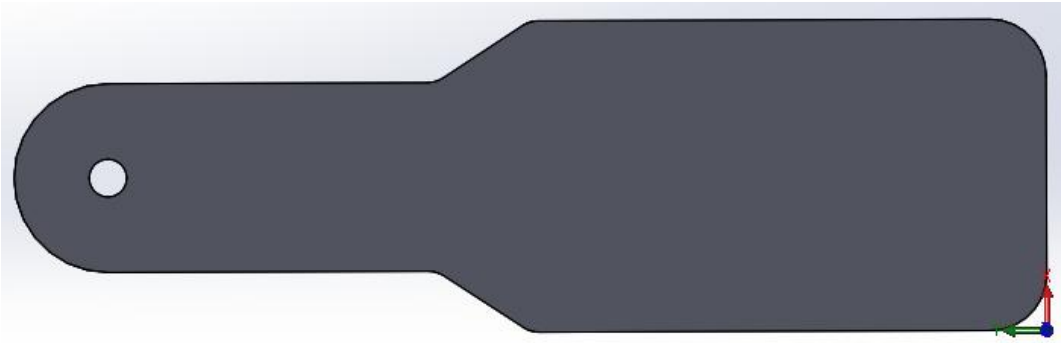


Imagen 21. Vista frontal del segundo brazo (Fuente: Propia)

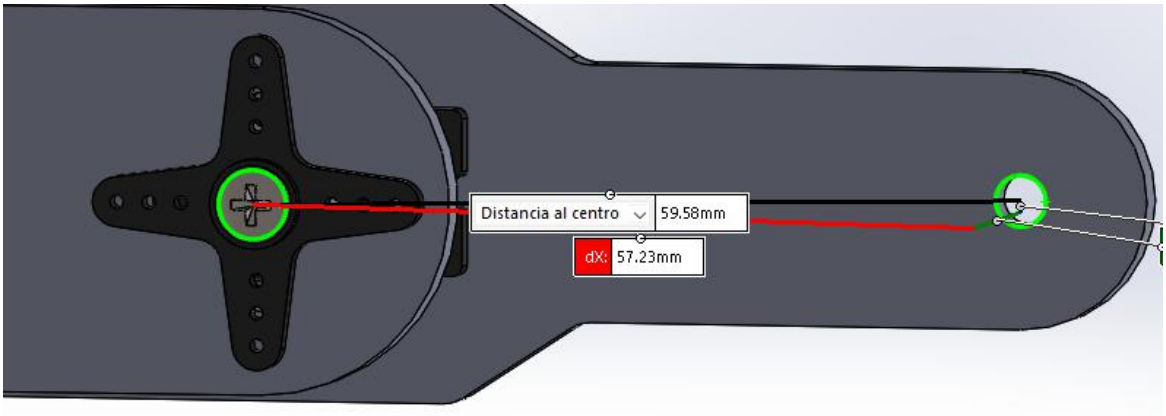


Imagen 22. Longitud del segundo brazo (Fuente: Propia)

Imagen 23. Fotografía del segundo brazo (Fuente: Propia)

ACTUADORES

Los actuadores que utilizamos para el movimento de nuestros brazos y los que determinan los ángulos de éstos fueron servos analógicos de motor DC. Los que decidimos usar fueron los **Servo giro limitado HD-3001HB**. Estos servos son unos de los servos estándar más populares y cuentan con las características necesarias para la realización de nuestro proyecto. Dos rodamientos de bolas ayudan a reducir la fricción y mejorar el rendimiento. El cable se termina con un conector tipo "JR" estándar. A continuación se muestran las características del servo:

- Torque(4.8V): 3.5 kg-cm (48.6 oz/in)

- Velocidad: 0.14 sec (4.8V) │ 0.12 sec (6.0V)

- Rango de operación: 4.8 ~ 6.0 DC Volts

- Peso : 43.0 g (1.52 oz)

- Tipo de Motor ：Motor DC

- Tipo de engranaje : Plastico

- Temperatura de operación: -20℃~60℃

- Frecuencia de trabajo: 1520μs / 50hz

Imagen 24. Servo giro limitado HD-3001HB (Fuente: pololu.com)

Para la alimentación de nuestro circuito de control y para alimentar a nuestros servos, contamos con una simple fuente de voltaje. Esta cuenta con una terminal que nos proporciona con 5V y con suficiente corriente para alimentar facilmente nuestro robot.

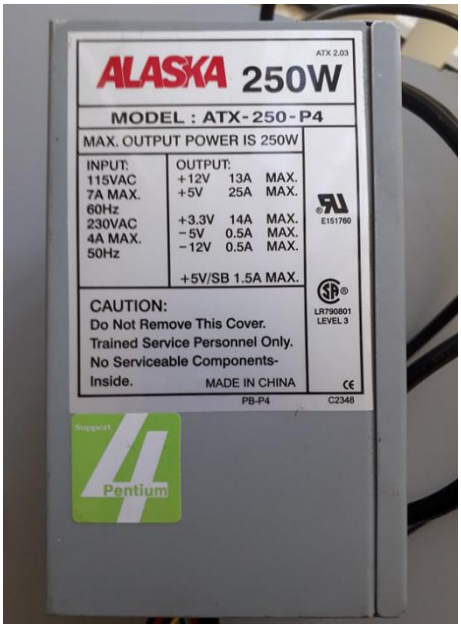


Imagen 25. Fuente usada para alimentar el robot (Fuente: Propia)

A continuación podemos ver el ensamblaje en Solidworks en el que se puede ver el robot, así como las fotografías del robot.

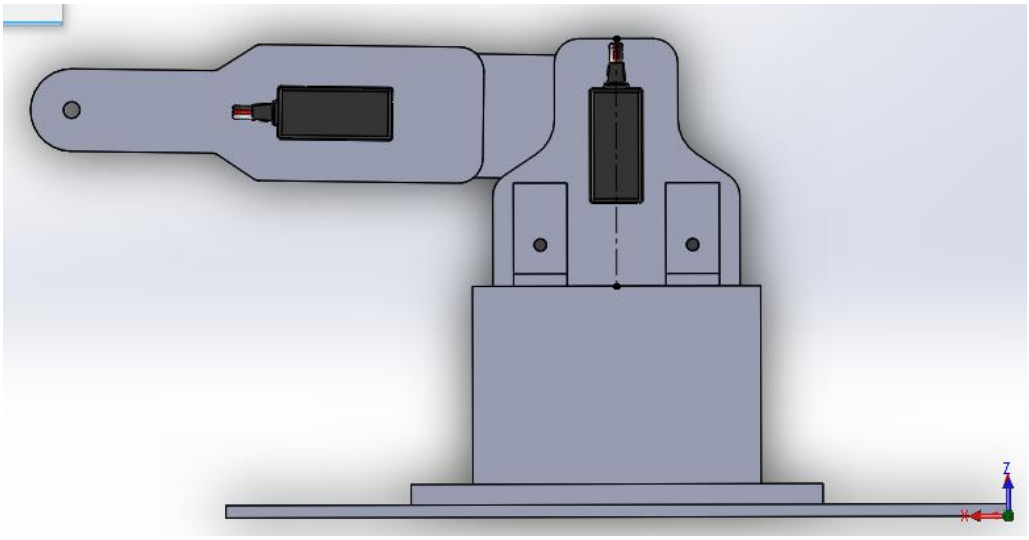


Imagen 26. Vista lateral 1 del robot (Fuente: Propia)

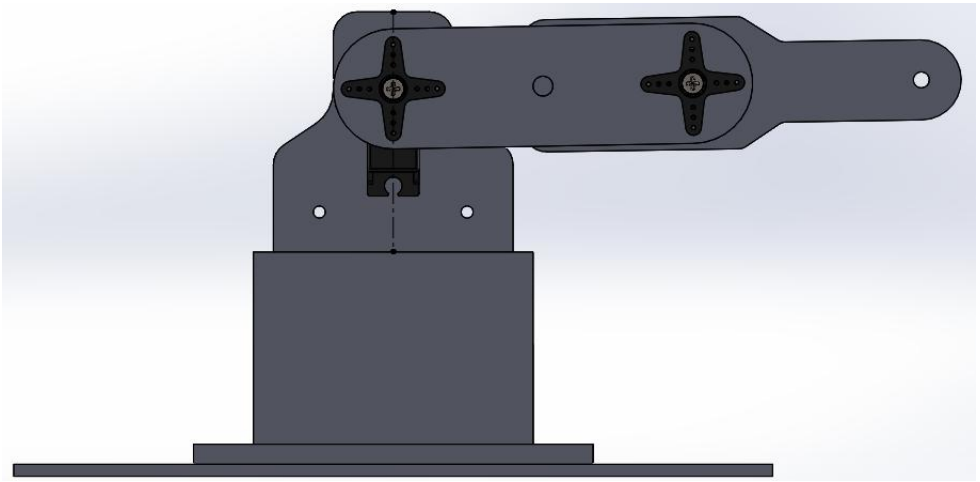


Imagen 27. Vista lateral 2 del robot (Fuente: Propia)

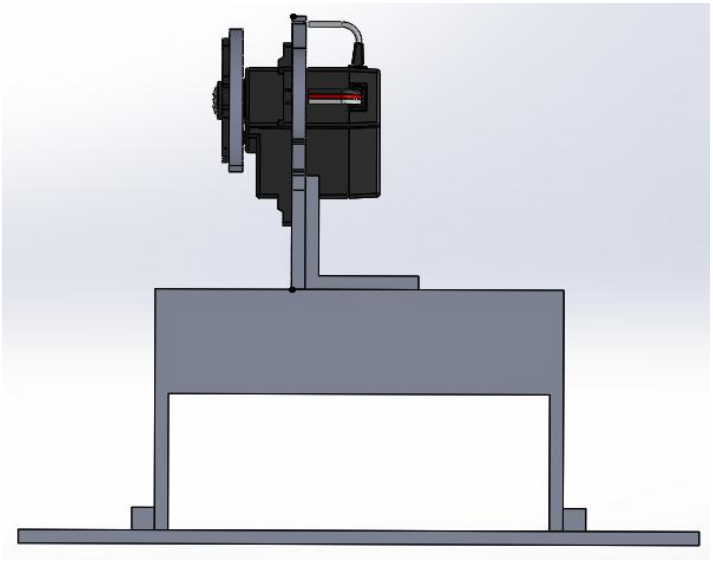


Imagen 28. Visa frontal del robot (Fuente: Propia)

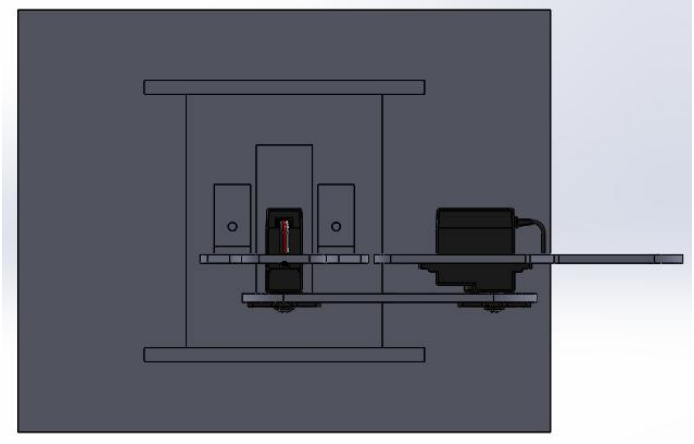


Imagen 29. Vista superior del robot (Fuente: Propia)

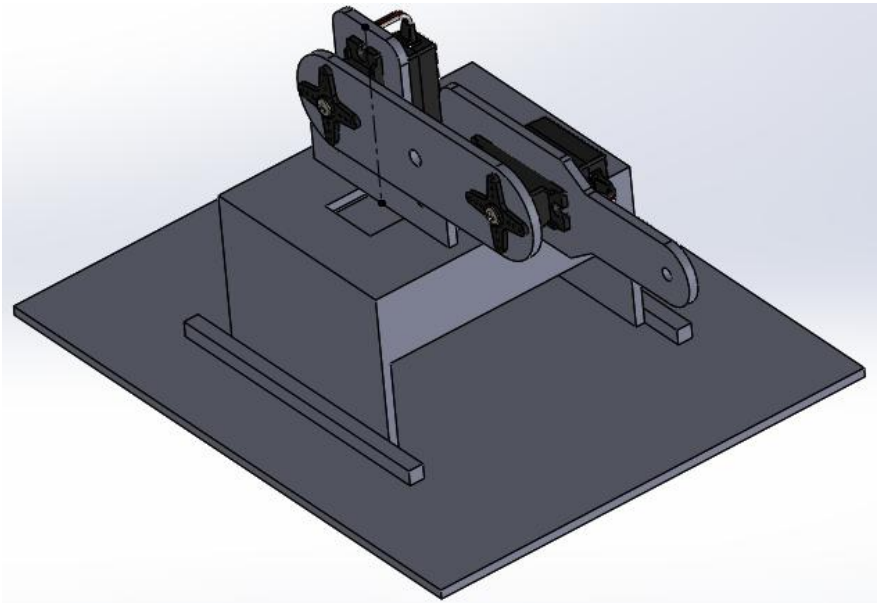


Imagen 30. Vista isométrica del robot (Fuente: Propia)

Imagen 31. Vista frontal del robot (Fuente: Propia)

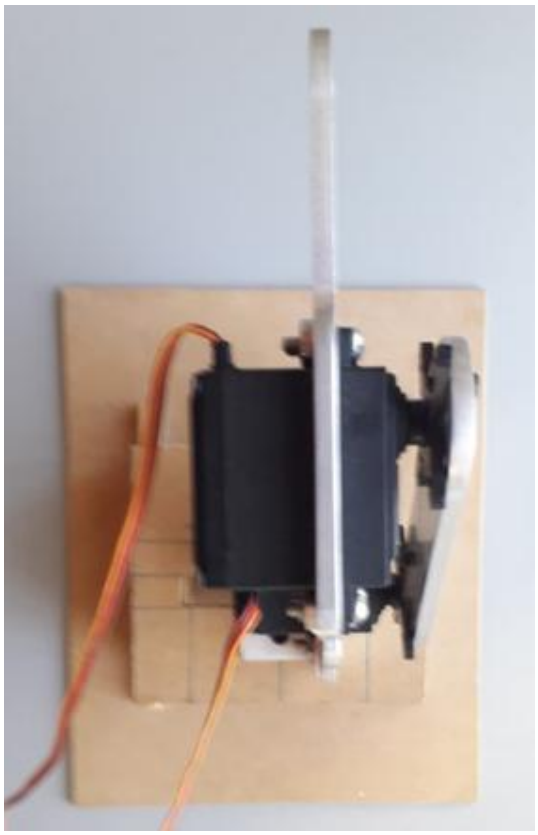


Imagen 32. Vista superior del robot (Fuente: Propia)

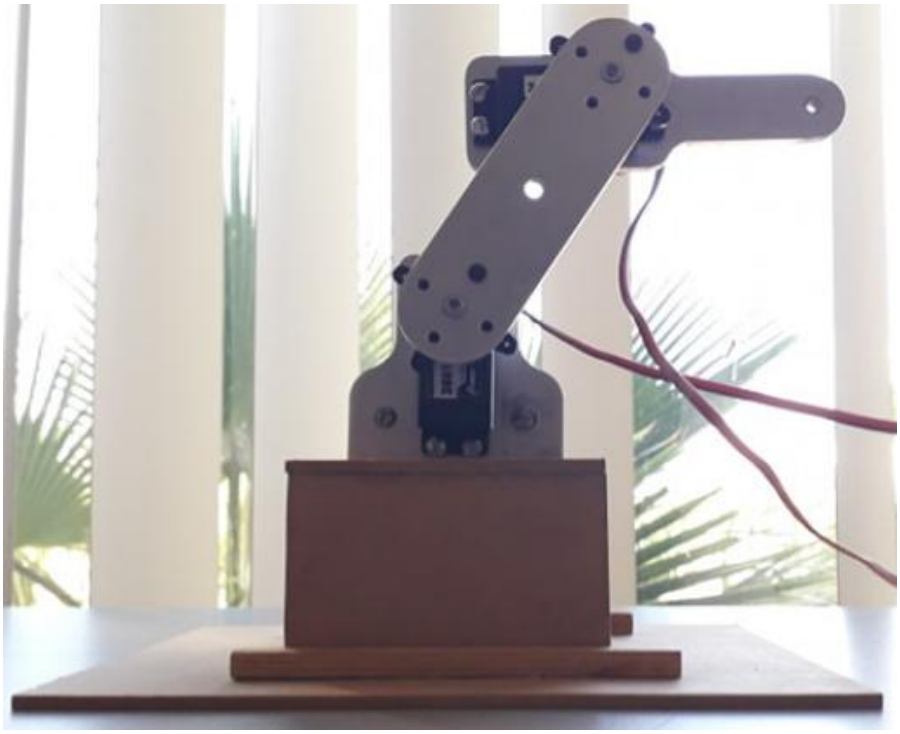


Imagen 33. Vista lateral del robot (Fuente: Propia)

CONTROLADOR

Para controlar el robot se necesitó el uso de un microcontrolador que contara con comunicación wifi para poder recibir los comandos a distancia desde nuestro dispositivo Android, y que además contara con pines PWM, ya que éstos pines son los que nos permiten controlar los ángulos a los que se encuentran los brazos del robot.

Aunque existían diversas opciones, como por ejemplo Arduinos con Shield de Wifi, al final decidí hacer uso de el **ESP8266**. La decisión se tomo debido a que es un chip pequeño que cuenta con todas las funcionalidades que se buscan, y a diferencia de otras opciones, el ESP8266 no necesita de un microcontrolador extra, debido a que contiene un micro integrado, por lo que reduce mucho los costos y el espacio que ocupa.

El ESP8266 es un chip de bajo costo Wi-Fi con una pila TCP/IP completa y un microcontrolador. Éste chip cuenta con las siguientes características:

- CPU RISC de 32-bit: Tensilica Xtensa LX106 a un reloj de 80 MHza

- RAM de instrucción de 64 KB, RAM de datos de 96 KB

- Capacidad de memoria externa flash QSPI - 512 KB a 4 MB* (puede soportar hasta 16 MB)

- IEEE 802.11 b/g/n Wi-Fi

- Tiene integrados: TR switch, balun, LNA, amplificador de potencia de RF y una red de adaptación de impedancias

- Soporte de autenticación WEP y WPA/WPA2

- 16 pines GPIO (Entradas/Salidas de propósito general)

- SPI, I²C,

- Interfaz I²S con DMA (comparte pines con GPIO)

- Pines dedicados a UART, más una UART únicamente para transmisión que puede habilitarse a través del pin GPIO2

- 1 conversor ADC de 10-bit

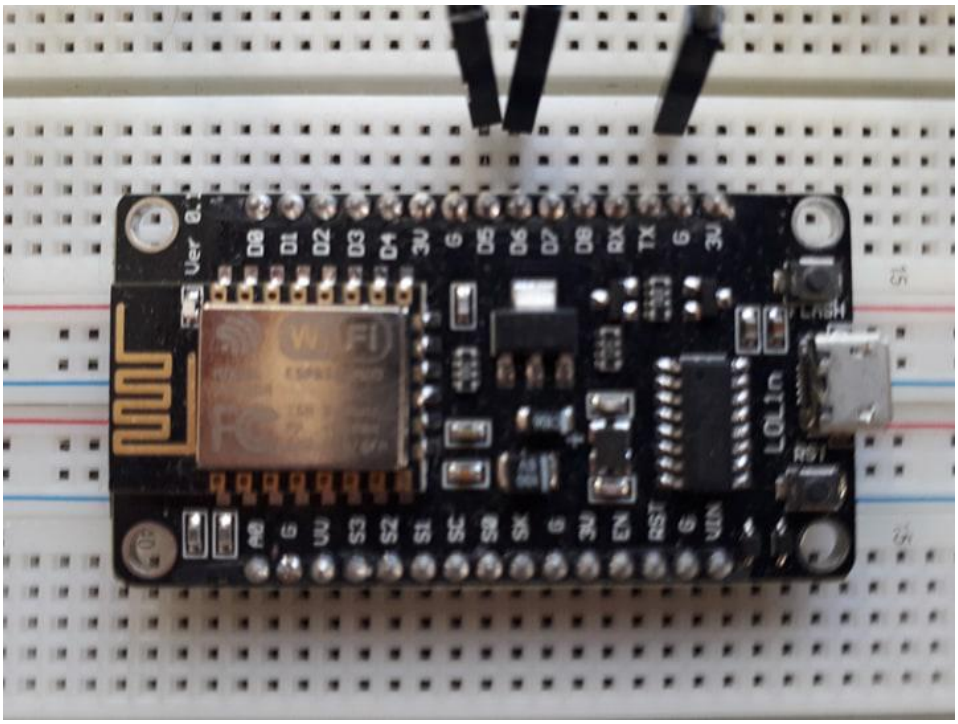


Imagen 34. ESP8266 usando en nuestro proyecto (Fuente: Propia)

A continuación podemos ver la distribución de pines en el ESP8266:

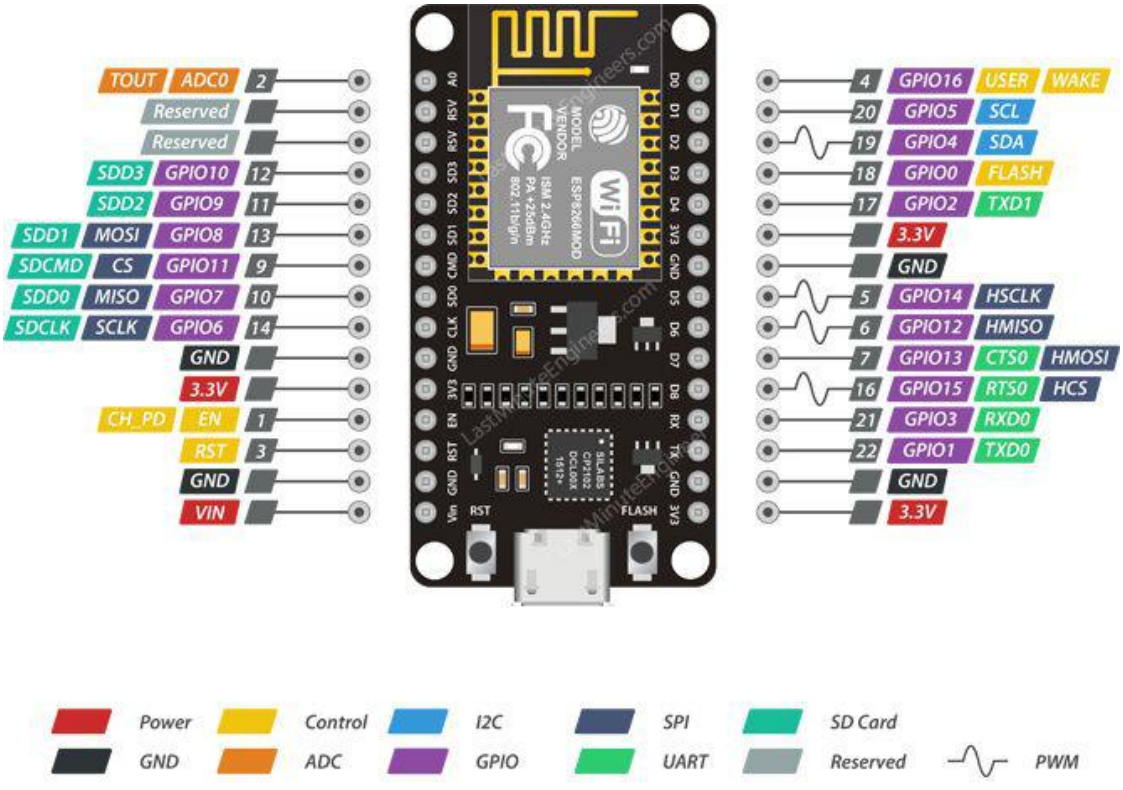


Imagen 35. Distribución de pines en el ESP8266 (Fuente: lastminuteengineers.com)

Como podemos observar, nuestro microcontrolador, cuenta con multiples puertos para PWM, los puertos que nosotros vamos a usar para nuestro proyecto son el **GPIO14** y el **GPIO12**.

Ahora el siguiente problema al que nos enfrentamos es el de como programar nuestro ESP8266. Existe un software especial que la empresa nos proporciona para controlar el ESP8266. Sin embargo, existe una alternativa que es mucho más fácil y con la que todos estamos familiarizados, **Arduino.**

Si bien el ESP8266 no es un dispositivo propio de Arduino, existen librerías que nos permiten controlarlo como si fuera uno. Para poder hacer ésto seguimos los siguientes pasos.

COMO CONTROLAR EL ESP8266 DESDE ARDUINO

Debemos tener ya instalado nuestro Arduino IDE con versión 1.6.4 o superior. Seguidamente vamos a archivo>Preferencias y en la casilla "Gestor de URLs Adicionales de Tarjetas" agregamos:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

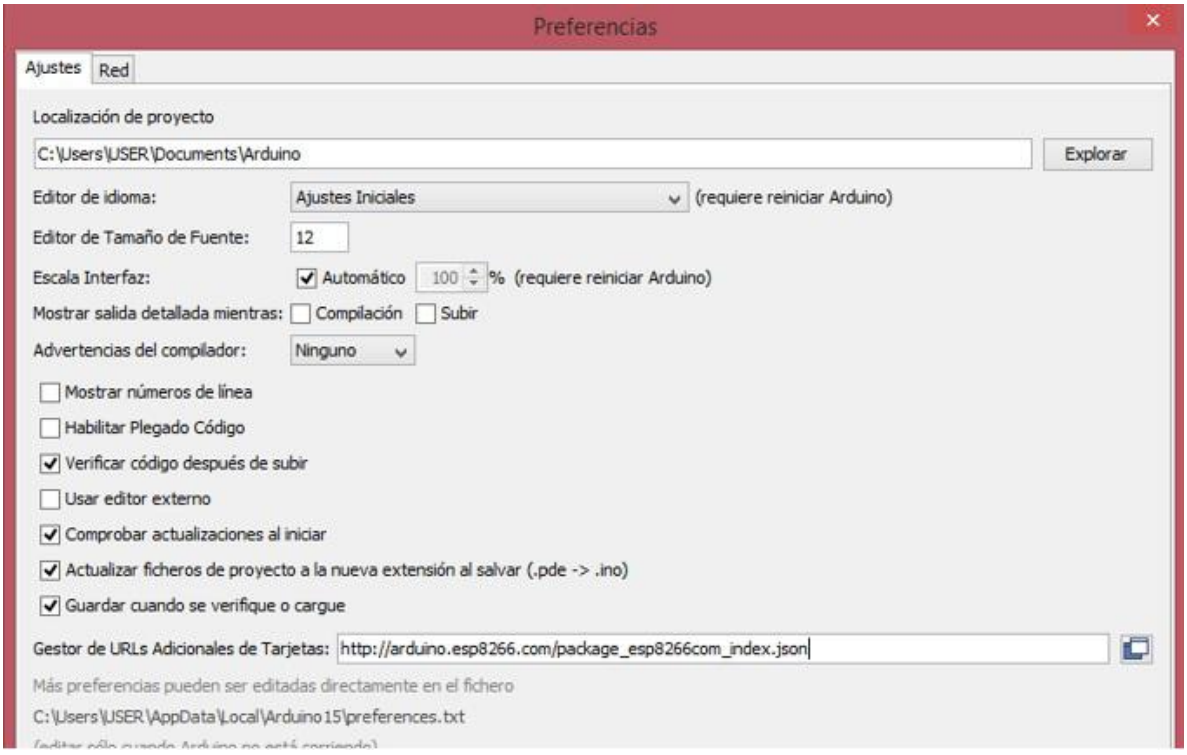


Imagen 36. ESP8266 en Arduino paso 1 (Fuente: Propia)

Seguidamente vamos a Herramientas>placa: … >Gestor de Tarjetas

Y buscamos en la lista “esp8266 by ESP8266 Community“, lo seleccionamos e instalamos.

Imagen 37. ESP8266 en Arduino paso 2 (Fuente: Propia)

La instalación va a demorar un poco, al finalizar, el ítem del ESP8266 les debe marcar como instalado. Ahora en herramientas>placas, deben de estar las nuevas placas instaladas.

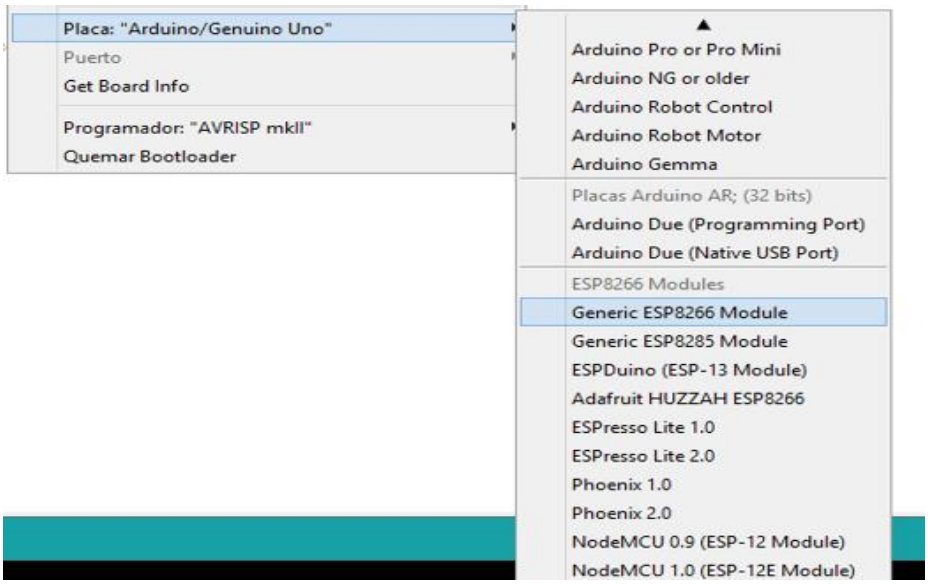


Imagen 38. ESP8266 en Arduino paso 3 (Fuente: Propia)

La placa que vamos a usar y que hay que seleccionar en éste menú es la que dice **NodeMCU 1.0 (ESP-12E Module).**

En el ESP8266 la resolución del PWM es de 10 bits a diferencia de un arduino que es de 8bits. Otra diferencia positiva para el ESP8266 es que podemos modificar la frecuencia del PWM, siendo por defecto de 1KHZ.

Posteriormente, cuando se llegue a la parte de la conexión WiFi con nuestra aplicación, se explicará el código usado para el control de los servomotores y la creación del servidor a través del cual nuestra aplicación se podrá conectar vía WiFi a nuestro robot.

Finalmente solo queda explicar como queda la conexión final de nuestro robot con la placa ESP8266, esta conexión se muestra a continuación.

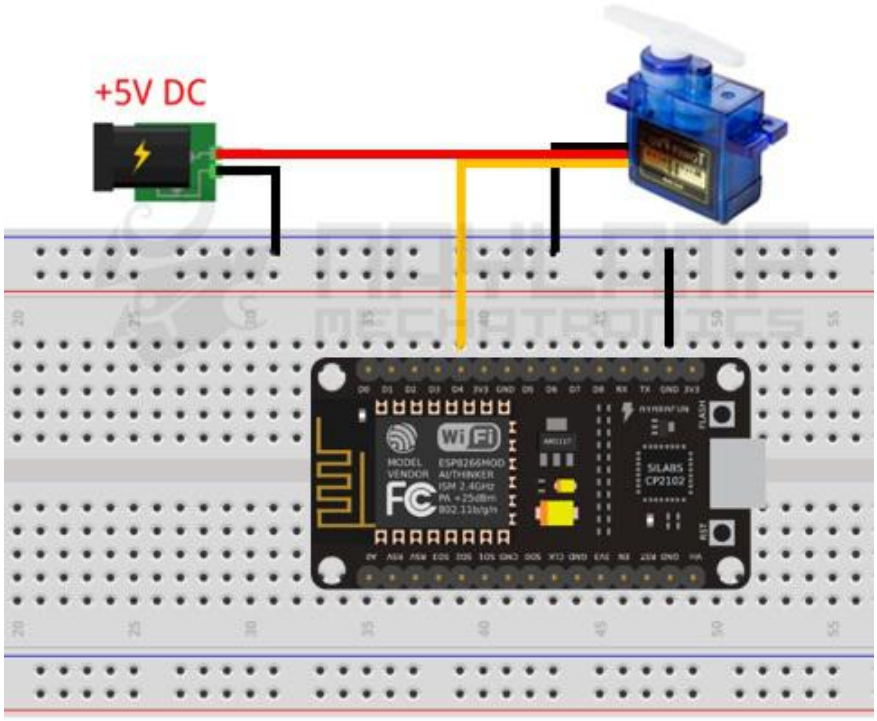


Imagen 39. Conexión del ESP8266 con el servomotor (Fuente: Propia)

Como se puede observar, la manera de conectar los servos a las salidas es sumamente sencilla, se realiza de la misma manera para ambos servomotores. Se alimentan los servomotores con la fuente, se realiza una tierra común entre la fuente, los servos y el micro y finalmente se conectan las lineas de control de los servos a las salidas PWM (**GPIO14** y el **GPIO12**) de nuestro ESP8266. Una vez que contamos con nuestro robot completo y su parte de control, es momento de enfocarnos en el modelo 3D que será usado en Unity para el gemelo digital en realidad aumentada.

Modelar el robot haciendo uso de software CAD (Diseño Asistido por Computadora) y hacer los cálculos de la cinemática directa e inversa para el modelado

Como ya fue mencionado anteriormente, cada una de la piezas del robot 3D fueron diseñadas y luego ensambladas en el software Solidworks. El formato con el que las piezas se guardan en solidworks es .SLDPRT y el ensamblaje final es guardado en formato .SLDASM.

El software con el que se va a desarrollar nuestra aplicación es Unity. Para poder visualizar el gemelo digital en realidad aumentada necesitamos un modelo en 3D para importarlo en nuestra aplicación y trabajar con el. Fue acá en donde me tope con uno de los primeros problemas.

Unity puede leer modelos 3D en formato .fbx, .dae, .blend, .3ds, .dxf, y .obj, sin embargo nuestras piezas están en .SLDPRT y .SLDASM. Para poder importar el modelo 3D del robot a Unity primero tenía que convertir las piezas a un formato que Unity pudiera leer. Para poder hacer ésto decidí usar el programa llamado **Blender**.

Blender es un programa informático multi plataforma, dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales. La ventaja que tiene Blender es la capacidad de exportar los modelos 3D a formato **.blend**, los cuales se pueden importar sin ningún problema a Unity.

Lo primero que tenemos que hacer es exportar nuestras piezas de Solidworks a formato **.wrl**. Este formato se puede a importar a Blender para poder realizar la conversión. Para realizar la conversión de nuestro robot a éste formato basta con ir a Archivo > Guardar como y seleccionar VRML en la lista que nos aparece, y finalmente seleccionamos VRML97 (VRML 2.0) en las opciones que nos presenta, porque ésta es la versión que se puede importar a Blender.

Una vez que tenemos nuestro robot en formato .wrl, lo importamos desde Blender. A continuación podemos ver como la pieza se importó exitósamente a nuestro programa.

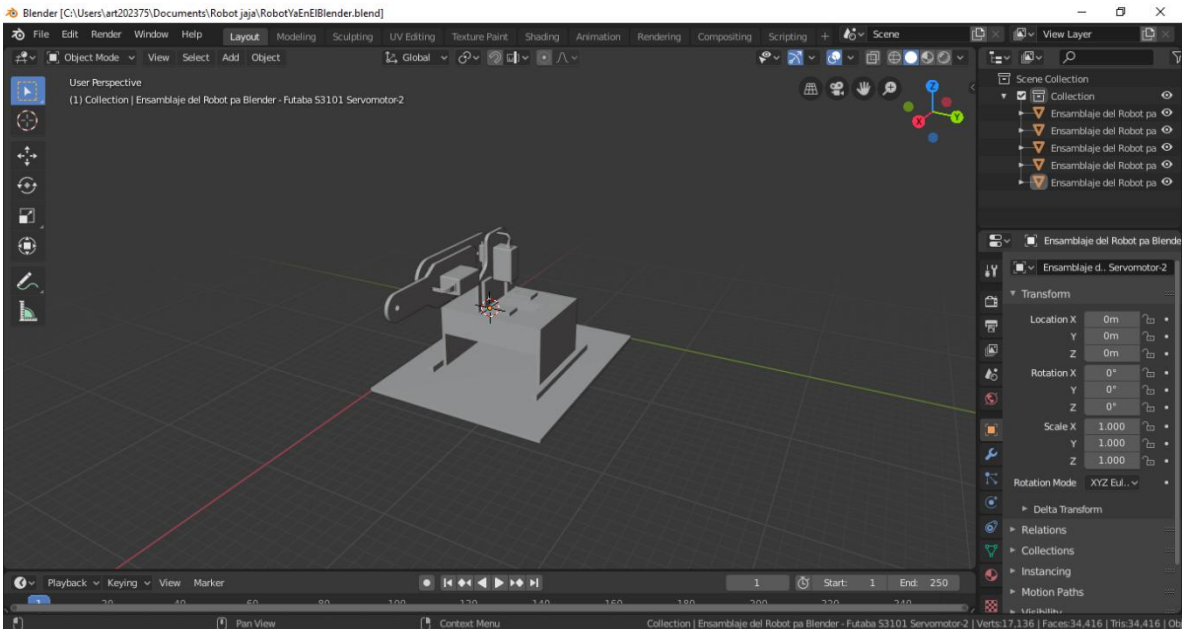


Imagen 40. Robot en formato .wrl importado a Blender (Fuente: Propia)

La conversión a formato **.blend** se realiza fácilmente, lo único que tenemos que hacer es ir a Archivo > Guardar Como y finalmente lo guardamos como archivo .blend.

Una vez que tenemos nuestro modelo en 3D listo, lo podemos importar sin ningún problema a Unity.

Abrimos Unity y creamos un nuevo proyecto. Este proyecto crea una carpeta en la cual se almacenan todos los recursos del programa. En la carpeta de nuestro proyecto automáticamente se crea una carpeta llamada **Assets,** lo único que tenemos que hacer para cargar el robot a Unity es copiar el archivo que creamos en Blender a esta carpeta.

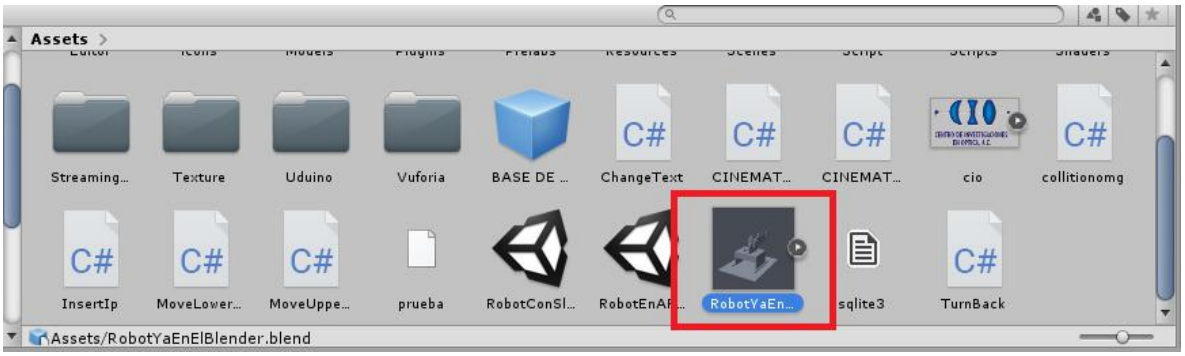


Imagen 41. Modelo 3D de blender en la carpeta de Assets del proyecto (Fuente: Propia)

Para cargar el robot en nuestro proyecto de Unity basta con arrastrarlo a nuestra escena en Unity.

El siguiente paso es ponerle texturas a nuestro robot. La razón para ponerle texturas a las partes del robot no es otra más que estética y de claridad visual. El hecho de tener texturas hace que el gemelo digital sea más parecido al robot real.

Las únicas tres texturas que necesitamos crear para nuestro robot son la **madera** de la base, el **acero** de los brazos y el **plástico negro** del que están hechos los servomotores.

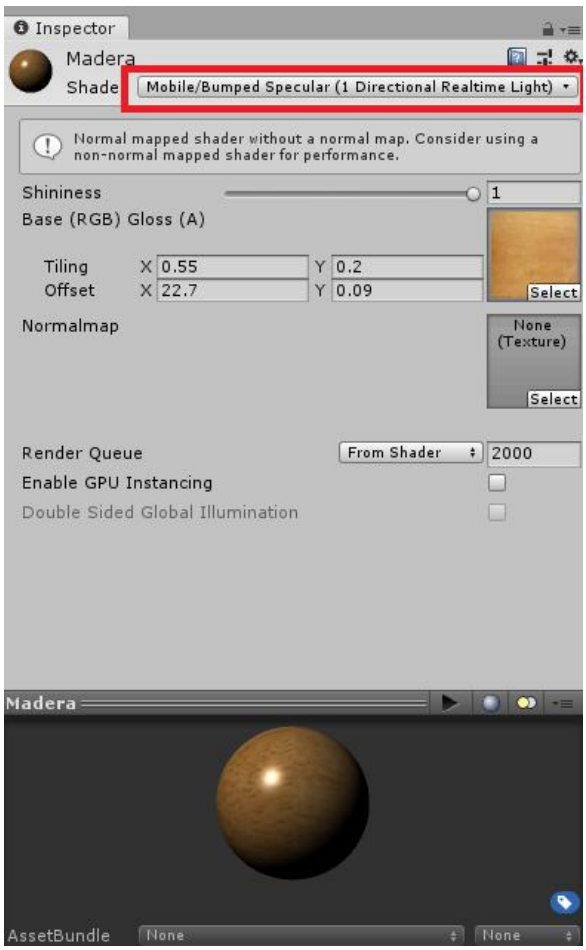


Imagen 42. Creación de textura del robot (Fuente: Propia)

Para crear las texturas solo necesitamos descargar una imagen que muestre una textura de madera para poder cargarla desde Unity. Lo único que debemos cuidar al momento de crear nuestras texturas es que el tipo de textura sea apta para dispositivos móviles, ya que la aplicación está destinada a Android.

Una vez creadas las tres texturas que necesitamos, arrastramos las texturas desde nuestra carpeta de Assets directamente a las piezas del robot a las que deseamos aplicar las texturas. En la siguiente imagen se pueden ver las tres texturas creadas y como se ven una vez aplicadas al modelo del robot.

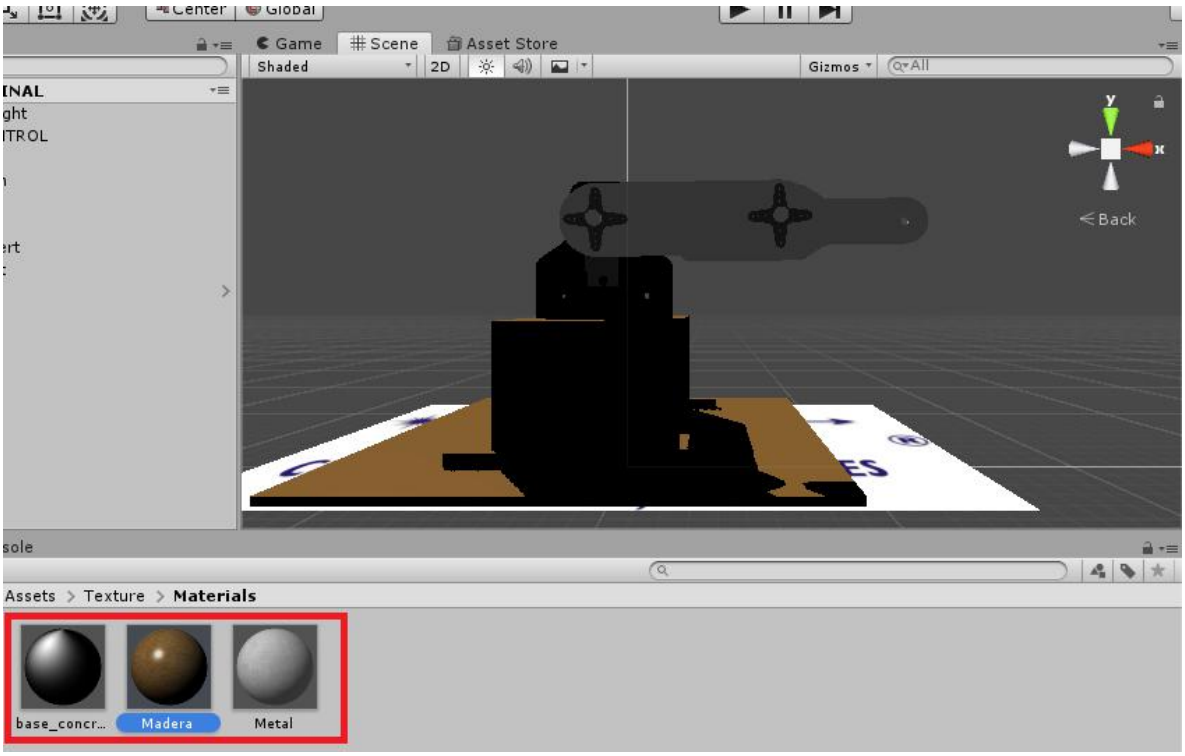


Imagen 43. Tres texturas creadas y aplicadas al modelo 3D (Fuente: Propia)

CÁLCULOS DE CINEMÁTICA DIRECTA E INVERSA DEL ROBOT

Dentro de los objetivos del proyecto se encuentra que nuestra aplicación realice los cálculos de cinemática directa e inversa, para ésto se busca mostrar en la interfaz la posición del **efector final**, así como los dos posibles ángulos que pueden tener las articulaciones (**codo arriba** y **codo abajo**) a partir de la posición del efector final.

Para poder hacer esto, primero debemos de determinar las ecuaciones de cinemática directa y de cinemática inversa de nuestro robot, para posteriormente programar ésta característica en nuestra aplicación. Recurrimos a las fórmulas que obtuvimos en el capítulo **Modelo cinemático de un Robot Planar de 2 grados de libertad** de nuestro Marco Teórico.

Las fórmulas de cinemática directa de nuestro robot nos indican que la posición del efector final está dada por:

$$x = \alpha_1 \cos \theta_1 + \alpha_2 \cos (\theta_1 + \theta_2)$$

$$y = \alpha_1 \sin \theta_1 + \alpha_2 \sin (\theta_1 + \theta_2)$$

α_1 es la longitud del primer brazo de nuestro robot y α_2 es la longitud del segundo brazo. De acuerdo a nuestro diseño, las longitudes de los brazos son:

$$\alpha_1 = 7.5 \text{ cm}$$

$$\alpha_2 = 6 \text{ cm}$$

Si sustituimos estos valores en nuestras fórmulas encontramos las ecuaciones de cinemática directa que definen a nuestro robot.

$$\mathbf{x = 7.5 \cos \theta_1 + 6 \cos (\theta_1 + \theta_2)}$$

$$\mathbf{y = 7.5 \sin \theta_1 + 6 \sin (\theta_1 + \theta_2)}$$

Esta fórmula nos permitirá determinar la posición del efector final en función de los ángulos de los brazos de nuestro robot.

Para encontrar las ecuaciones de cinemática inversa recurrimos a las fórmulas que encontramos. El ángulo dos viene dado por:

$$\theta_2 \quad = \quad \tan^{-1} \frac{\pm\sqrt{1 - D^2}}{D}$$

Donde D está definido como:

$$\cos\theta_2 \quad = \quad \frac{x^2 + y^2 - \alpha_1^2 - \alpha_2^2}{2\alpha_1\alpha_2} := D$$

x = coordenada del eje x del efector final

y = coordenada del eje y del efector final

α_1 = 7.5 cm

α_2 = 6 cm

De ésta manera podemos determinar el $\boldsymbol{\theta_2}$ basándonos en la posición que en su momento tenga el efector final. Finalmente calculamos el $\boldsymbol{\theta_1}$ usando la fórmula:

$$\theta_1 \quad = \quad \tan^{-1}(y/x) - \tan^{-1}\left(\frac{\alpha_2 \sin\theta_2}{\alpha_1 + \alpha_2 \cos\theta_2}\right)$$

Haciendo uso de ambas ecuaciones encontraremos los posibles ángulos para llegar a la posición del efector final y habremos resuelto el problema de la cinemática inversa.

El enfoque ahora será mostrar que el modelo 3D se puede controlar desde nuestro programa y posteriormente se integran las ecuaciones de cinemática directa e inversa que acabamos de discutir.

Seleccionar marcadores y realizar configuración en Unity para objetos en realidad aumentada

Teniendo nuestro modelo 3D en Unity es momento de empezar a trabajar la parte de realidad aumentada. Lo primero que necesitamos para empezar a desarrollar nuestro gemelo digital en AR es **Vuforia.** Vuforia es una plataforma de desarrollo de aplicaciones de Realidad Aumentada (AR) y Realidad Mixta (MR), con seguimiento y rendimiento robustos en una variedad de hardware (incluidos dispositivos móviles).

Como vimos previamente, nosotros vamos a trabajar con marcadores tipo **imagen**. Vuforia nos permite subir marcadores a su sitio web para posteriormente usarlos para generar el modelo 3D en AR a partir de ellos.

El instalador de Unity trae por defecto el SDK de Vuforia por lo que único que necesitamos hacer es registrar una **llave** en su página web (lo cuál es gratuito) para poder registrar el producto. Una vez que hemos registrado el producto, lo primero que se debe hacer es activar Vuforia en nuestro proyecto de Unity.

Para activar Vuforia, se accede a la configuración del Reproductor desde Edición> Configuración del proyecto, luego se selecciona la categoría del Reproductor y selecciona la pestaña del dispositivo móvil (Android en nuestro caso). En el panel de configuración de XR, habilita la propiedad Vuforia Augmented Reality Support.

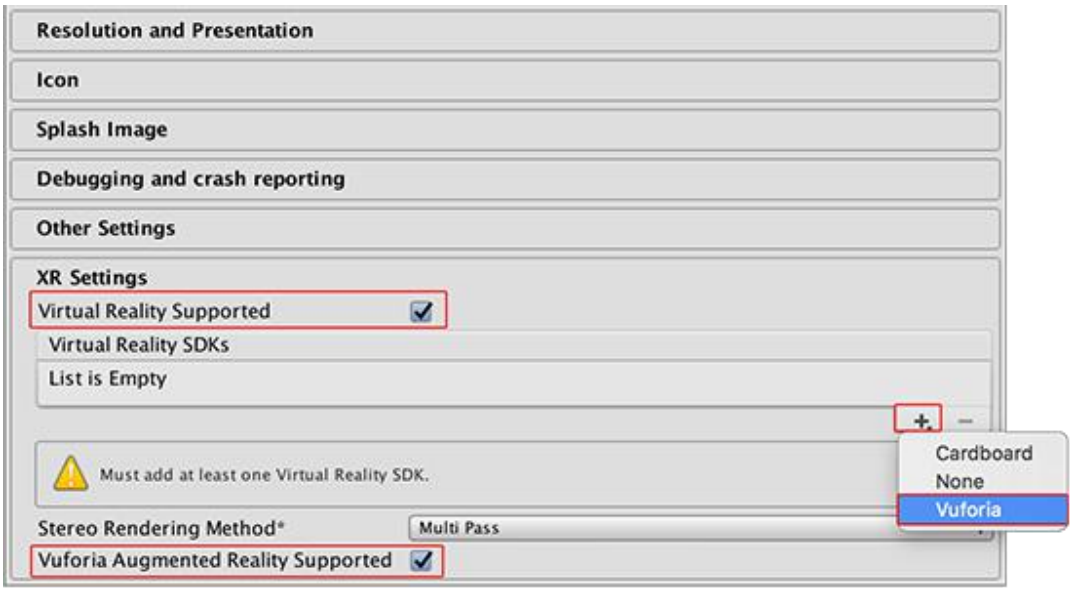


Imagen 44. Activando Vuforia en Unity (Fuente: Propia)

Al momento de activar Vuforia tu escena ahora contiene dos **GameObjects**: una cámara principal y una luz direccional. Se debe agregar una nueva cámara AR a la escena para habilitar la funcionalidad AR, así que eliminamos el objeto de cámara principal actual de la escena. Para agregar una cámara AR a la escena, vamos a GameObject> Vuforia> AR Camera. Unity también nos pedirá que importe sus **Assets** de Vuforia. Seleccionamos Importar, y Unity importa todos los archivos Vuforia necesarios en el proyecto.

La ventana Proyecto muestra 4 carpetas nuevas, una de las cuales se llama Vuforia. El código y los activos en esta carpeta proporcionan la funcionalidad principal de **AR**. Las otras carpetas proporcionan escenas de muestra, recursos, herramientas y complementos.

Imagen 45. Carpetas en Assets que pertenecen a Vuforia (Fuente: Propia)

La **camara AR** es la que le permite a nuestra aplicación usar la cámara de nuestro celular para buscar el marcador. Cuando la cámara detecta el marcador, coloca encima de éste el gemelo digital. Ahora es momento de la selección y creación de marcadores.

SELECCIÓN Y CREACIÓN DE MARCADOR

Volvemos a la web de Vuforia, que es en la cual registramos nuestro producto y en este caso en lugar de a la licencia accedemos a Target Manager. Pulsamos sobre Add DataBase donde le pondremos un nombre y en nuestro caso le pusimos de nombre “RobotAR”.

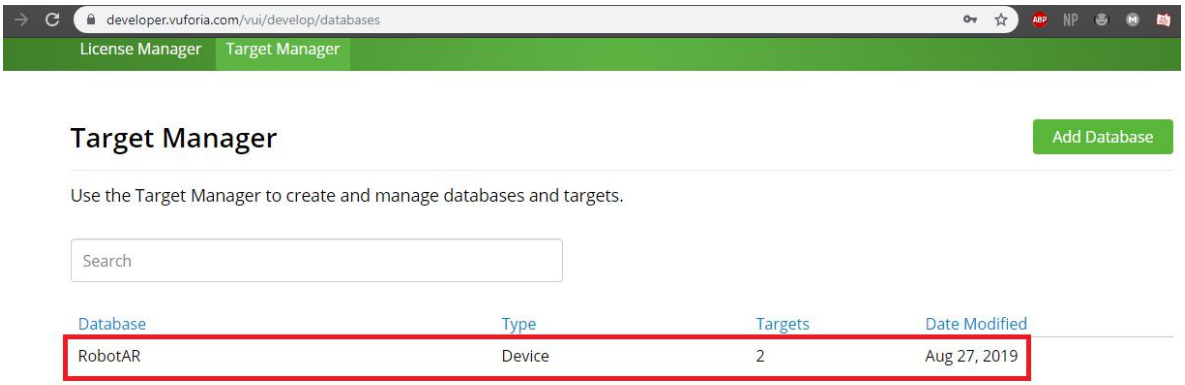


Imagen 46. Creación de base de datos de marcadores (Fuente: Propia)

Es en esta base de datos es que agregaremos la imagen que nos servirá de marcador para nuestra aplicación. Picándole a la base de datos añadiremos nuestro marcador. Al pulsar Add target, nos saldrán varios tipos, usaremos single image, pero se pueden usar objetos 3D reales. La imagen usada tiene que tener un máximo de 2mb y ser jpg o png. Además debemos saber el ancho de la misma ya que habrá que ponerlo en la caja donde pone Width y darle un nombre. Yo he usado la siguiente imagen:

Imagen 47. Marcador usado en el proyecto (Fuente: Propia)

Como podemos ver, la página nos indica un score con estrellas, el hecho de que tenga **5 estrellas** nos dice que la cámara detectará con mayor facilidad los patrones de lineas de nuestro marcador, lo que hará que nuestra aplicación genere más fácilmente el gemelo digital. Una vez subida, pulsamos sobre Download Database (All) y marcamos Unity Editor. Esto generará un unitypackage que importaremos haciendo doble clic.

Imagen 48. Generando el unitypackage para cargar nuestro marcador en Unity (Fuente: Propia)

Esto hará que automáticamente Unity cargue la imagen que subimos a la base de datos y la importe en nuestro proyecto, lo que sigue es indicar que objetos queremos que se muestren en la cámara al detectar el marcador.

CONFIGURANDO EL ROBOT PARA QUE SEA DETECTADO CON EL MARCADOR

Ahora añadiremos nuestra imagen como marcador a la escena, pulsando sobre Hierarchy botón derecho del ratón, Vuforia e Imagen. Esto nos crea un ImageTarget que al seleccionarlo nos muestra en el Inspector sus opciones. En Database al pinchar se nos muestra como opción la nuestra que seleccionamos. Y si no lo hace de forma automática, seleccionamos nosotros la nuestra.

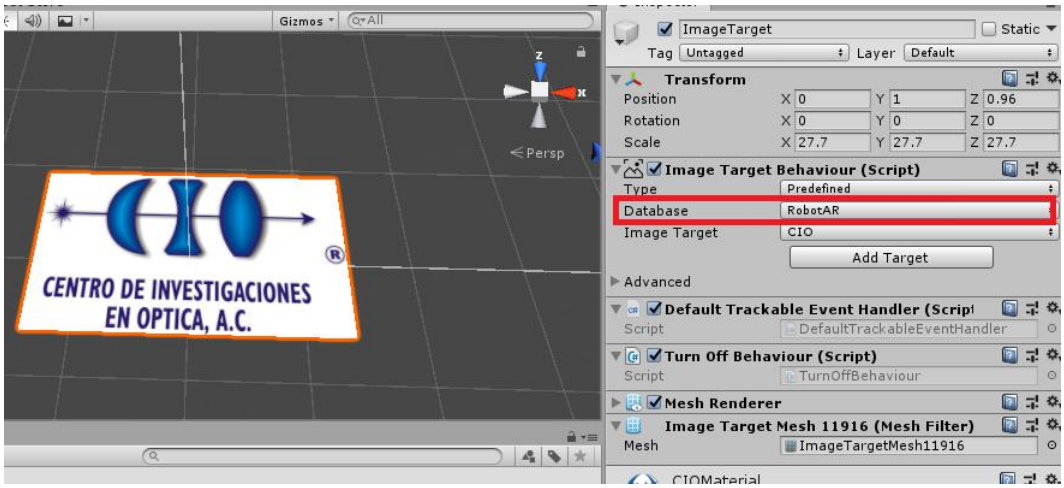


Imagen 49. Seleccionando nuestra imagen como marcador en Unity (Fuente: Propia)

Ahora tomamos el modelo 3D de nuestro robot. Lo escalamos y colocamos donde queremos que aparezca cuando nuestra cámara detecte nuestro marcador. Es muy importante señalar que el objeto que queremos que se muestre sobre el marcador debe estar **dentro del marcador (ImageTarget) en la jerarquía de nuestro proyecto.**

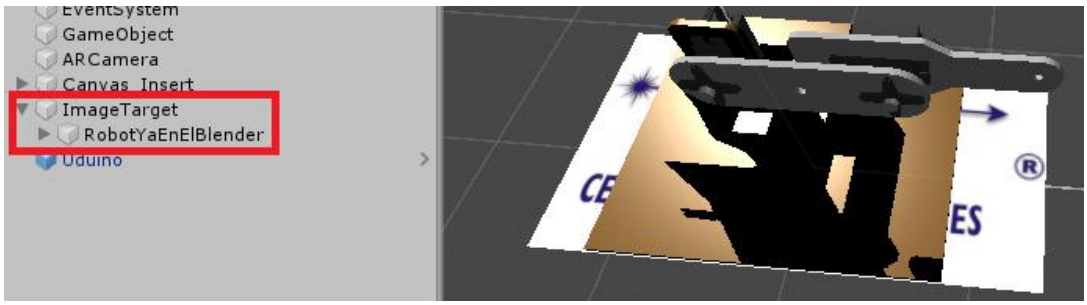


Imagen 50. El robot se encuentra sobre el marcador y además dentro de ImageTarget en la jerarquía (Fuente: Propia)

Finalmente usamos la cámara de nuestra computadora para comprobar si Vuforia está detectando el marcador y generando nuestro gemelo digital.

Imagen 51. Gemelo digital (Fuente: Propia)

Posteriormente se calculó la relación que había entre el tamaño del marcador y el tamaño del robot y se imprimió un marcador que nos produjera un gemelo digital de un tamaño exacto al del robot. De esta manera si colocamos el robot a lado de su gemelo digital ambos son exactamente iguales.

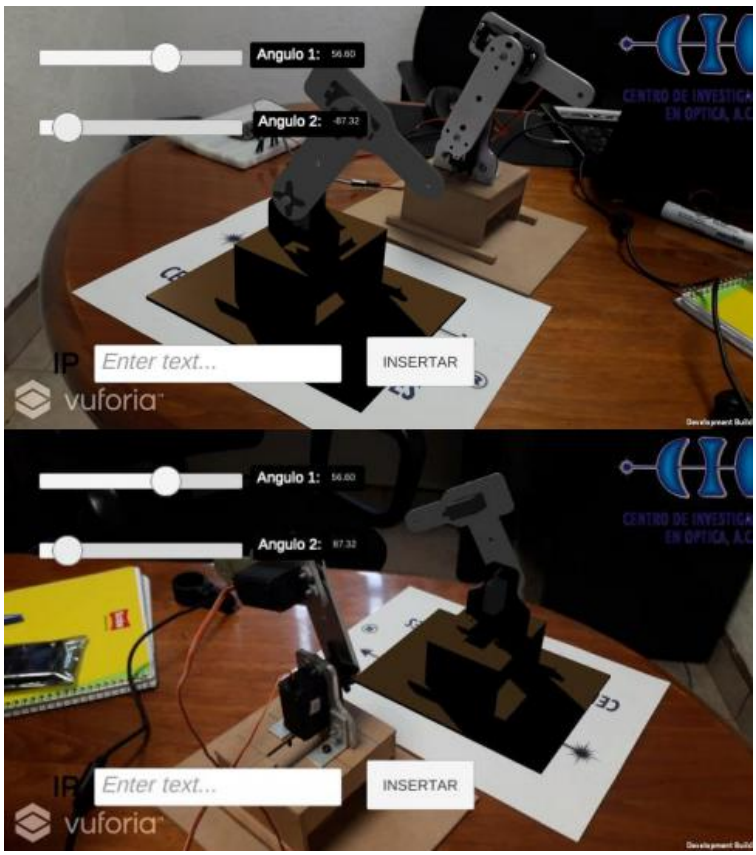


Imagen 52. Gemelo digital colocado a lado del robot real (Fuente: Propia)

Desarrollar interfaz en Unity para control de movimiento de modelo 3D del robot y mostrar los cálculos de cinemática directa e inversa

Una vez teniendo el gemelo digital del robot en Unity funcionando con la cámara AR es momento de desarrollar una interfaz que nos permita controlar el valor de los ángulos de los brazos. La parte de la interfaz en la que se encuentran todos los botones, indicadores y controles se conoce como **GUI** la cual es la abreviación de *Graphical User Interface* lo cual se traduce al español como **Interfaz Gráfica de Usuario**, los elementos del GUI se encuentran distribuido en lo que en Unity se conoce como un **Canvas**. Cada Canvas es una pantalla que cuenta con distintos elementos de monitoreo o de control de nuestra aplicación. Lo que se coloca en el Canvas es lo que nosotros veremos en la pantalla de nuestro dispositivo Android.

Para poder controlar nuestro robot creamos nuestro Canvas, en este se colocan dos controles deslizantes a los que llamaremos **Sliders**. Moviendo estos Sliders es que seleccionaremos el valor de los ángulos de nuestros brazos. Además de los Sliders vamos a colocar dos objeto tipo **Text** que no son mas que cajas de texto que no permitirán visualizar los ángulos en nuestra pantalla.

Imagen 53. Sliders que nos permitirán controlar el movimiento del robot (Fuente: Propia)

Una vez que ya tenemos los elementos en nuestra interfaz que nos permiten mover el robot es momento de entender un poco el código que se usó para controlar los brazos.

Lo primero que hicimos fue crear un objeto vacío en nuestra jerarquía, dentro de este objeto almacenaremos todos los códigos que nos permitirán controla el movimiento del robot. El código está programado en lenguaje **C#** y además se hace uso de la aplicación **Visual Studio** para escribir este código. A continuación se mostrará el funcionamiento de estos programas.

Para poder controlar la rotación de los brazos lo que hacemos primero es crear los puntos de articulación en Unity, estos puntos de articulación a los que nombré **joint_0** y **joint_1** son dos pequeños cilindros que cree en Unity y que coloqué en los puntos de articulación de mi robot. La función de joint_0 y joint_1 es hacer **que las piezas de mi robot giren en torno a éstas**.

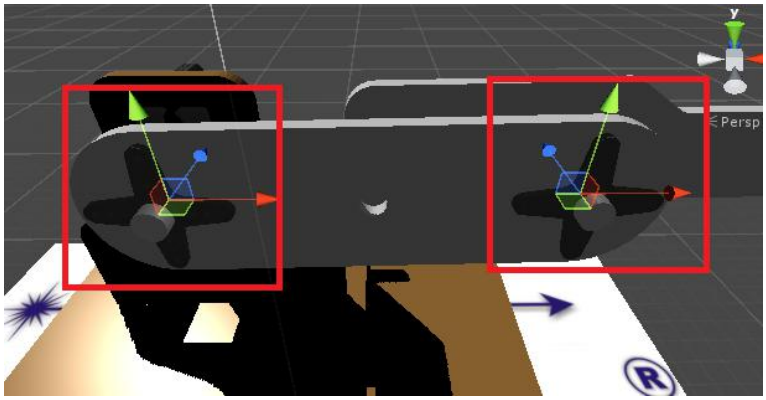


Imagen 54. joint_0 y joint_1 (Fuente: Propia)

Otra cosa que es muy importante destacar es que **tanto la pieza que vas a mover como la articulación (joint) deben de estar en el mismo nivel en la jerarquía.** También es importante mencionar que **joint_1 debe de estar dentro de el brazo completo en la jerarquía** esto se debe a que cuando la primera articulación se mueve, se mueve todo el brazo y por lo tanto la segunda articulación también cambia de posición.

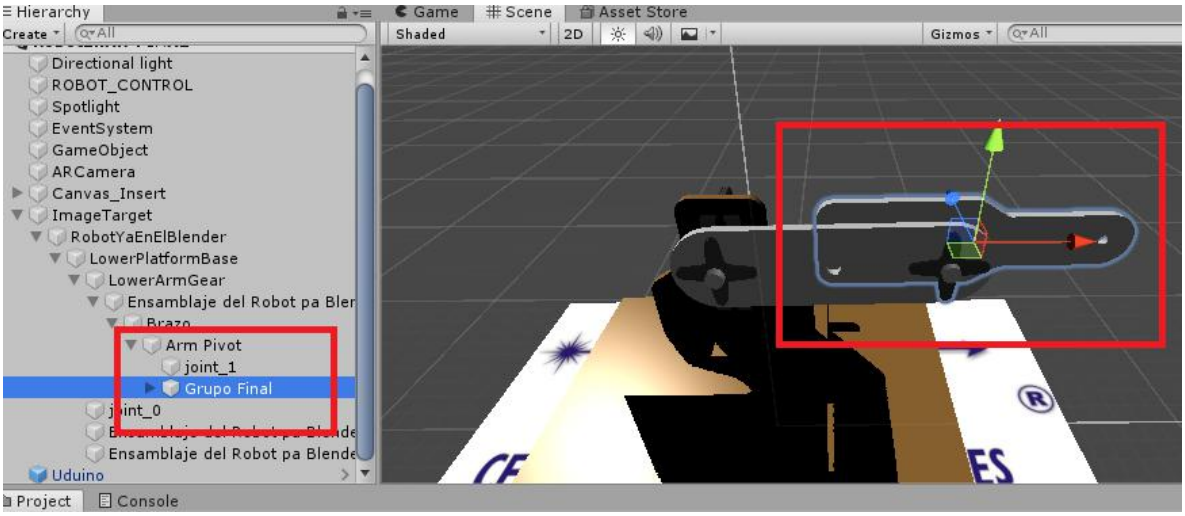


Imagen 55. Importancia de la jerarquía para el movimiento del robot (Fuente: Propia)

Una vez que creamos, colocamos en posición y ordenamos en la jerarquía las articulaciones vamos a entender que es lo que realmente hace el programa para mover al robot.

Es importante destacar que a lo largo de éste reporte técnico se explorará únicamente el **núcleo del programa**, es decir no se explicará cada una de las lineas del código, ya que son programas bastante extensos en los que explicar cada declaración de variables, asignación de valores o uso de rutinas sería un mero desperdicio de tiempo, pero si cabe remarcar que se entrará en detalle con algunas partes del código que puedan resultar de interés.

Tenemos dos códigos principales para controlar el movimiento de nuestro robot. Uno para controlar el movimiento del brazo inferior y uno para el movimiento del brazo superior.

Lo primero que hace el código para calcular el el ángulo del brazo inferior es crear dos vectores, uno de ellos es la **base de nuestro robot**. Tomamos este vector porque es completamente horizontal al suelo, esto nos permite que cuando el primer brazo está completamente horizontal, el ángulo sea de 0° y cuando el brazo se mueva hacia arriba o hacia abajo éste ángulo aumente. El segundo vector que tomamos **el vector que se forma desde la primera articulación hasta la segunda**. A continuación calculamos el valor del ángulo que se forma entre éstos

dos vectores, éste valor es el ángulo del primer brazo. Pero también debemos de tomar en cuenta la dirección en la que se mueve este brazo. Si el brazo se mueve hacia arriba de 0° el ángulo tiene signo **positivo** y si el brazo se mueve hacia abajo de 0° el ángulo tiene signo **negativo**. Así que agregamos al código condicionantes **if - else** que nos permitan determinar el signo de nuestro ángulo. Ya que nuestro ángulo cuenta con signo, lo podemos guardar en una variable. Ésta variable nos va a servir para mandarla a la caja de texto de **Ángulo 1** lo cual nos permitirá mostrar en la interfaz el valor del ángulo.

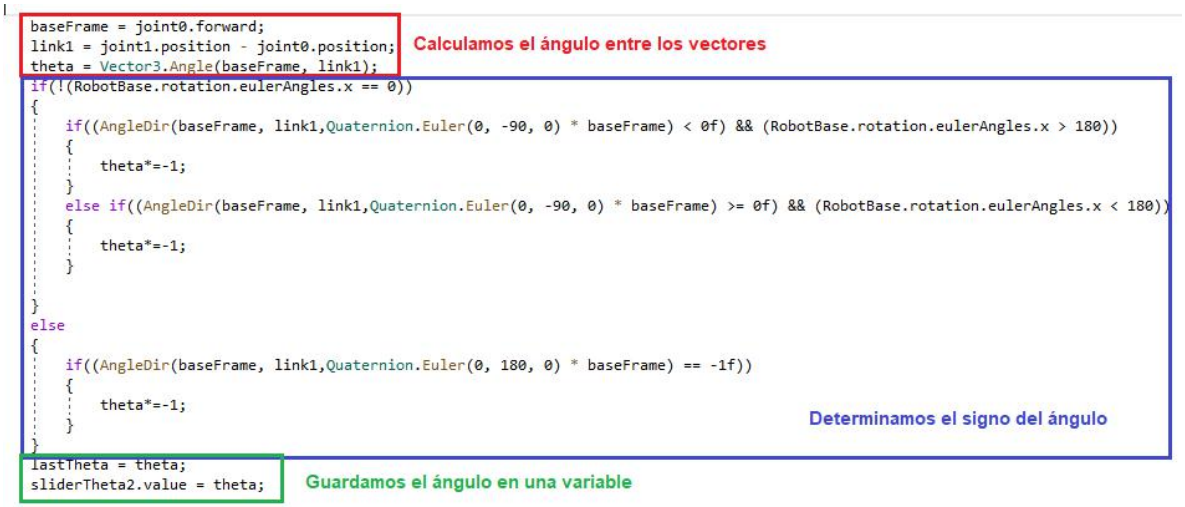


Imagen 56. Programa para calcular el ángulo del brazo inferior (Fuente: Propia)

Una vez que ya contamos con el código que nos permite **calcular éste ángulo**, es momento de ver el código que nos permite el **movimiento del brazo**.

Usamos un condicionante **if**, y le decimos a nuestro programa que si el Slider **se mueve** tome el valor del slider y lo guarde en una variable, de ahi manda esa variable a una pequeña rutina que lo que hace es **rotar el brazo inferior a la posición que indica el Slider** y esta rotación la hace respecto a **joint_0**. El programa se encuentra dentro de un **void Update ()**, lo que ésto significa es que el cálculo de los valores de los ángulos y la detección de cambios en el slider se realizan cada **frame**. Cada segundo en Unity ocurren 60 **frames,** lo que quiere decir que el programa se ejecuta continuamente.

Ahora es momento de pasar al movimiento del segundo brazo, el brazo superior. El procedimiento es muy parecido al del brazo inferior, con una simple diferencia, ahora los vectores que usamos para calcular el ángulo son diferentes. En éste caso el vector de referencia horizontal es el vector que se forma de **joint_0** a **joint_1.** Para el segundo vector creamos un punto al que le pondremos **joint_2**, el cual colocaremos en la punta del brazo superior, así el segundo vector sería el que se forma de **joint_1** a **joint_2**. Cuando los vectores son paralelos el ángulo de la segunda articulación es 0°. El procedimiento para calcular el signo es el mismo y de igual manera se guarda el valor para mostrarlo en la caja de texto **Ángulo 2,** lo mismo sucede con la parte del código para el movimiento del brazo superior, la diferencia es que ahora el brazo superior rota respecto a **joint_1**.

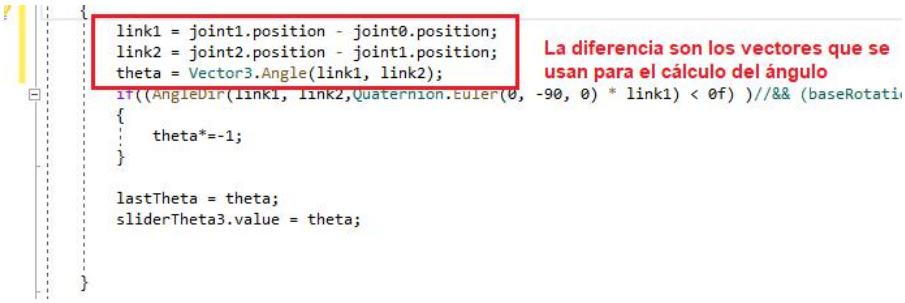


Imagen 57. Programa para calcular el ángulo del brazo superior (Fuente: Propia)

Finalmente lo que tenemos que hacer para que el programa realice los movimientos es indicarle cada uno de los elementos mencionados en el código, arrastrándolos desde nuestra jerarquía hasta el inspector de nuestro código.

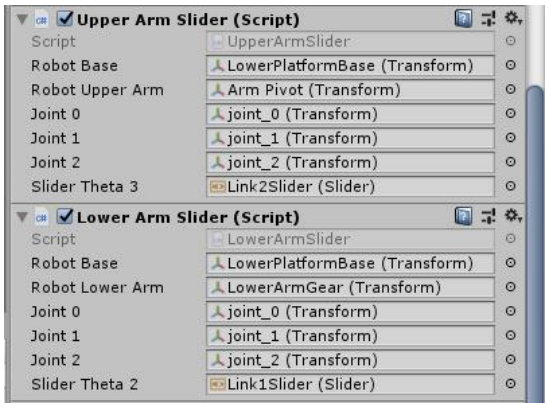


Imagen 58. Inspector de los códigos para mover el robot (Fuente: Propia)

Siempre que creamos un código en Unity se debe indicar al programa a que objetos nos referimos, esto se hace arrastrando los objetos hacia el inspector para que el programa pueda realizar su función. Así que éste paso solo se mencionará en ésta ocasión porque es el primer código que es mencionado en el reporte.

Una vez que tenemos el código que mueve y muestra los ángulos, podemos hacer la parte que hace los cálculos de cinemática directa e inversa de nuestro robot. Usaremos la fórmulas calculadas anteriormente para calcular la posición del efector final en el caso de la **cinemática directa** y para calcular los dos posibles pares de ángulos para llegar a la posición del efector final en el caso de la **cinemática inversa**.

Lo primero que hacemos es crear dos cajas de texto que son las que van a mostrar los valores que calculemos, así que la creamos en la parte inferior de nuestra interfaz.

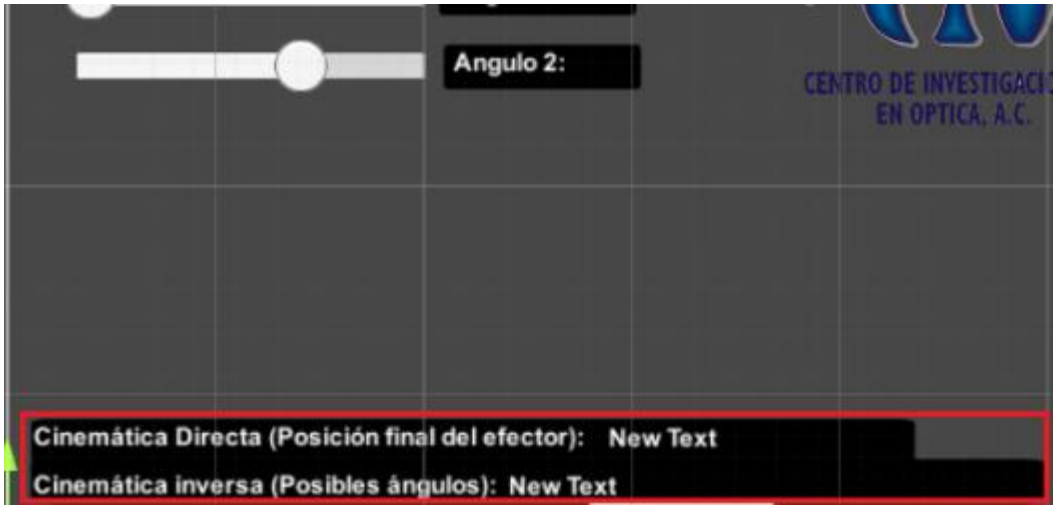


Imagen 59. Cajas de texto para mostrar los cálculos de cinemática (Fuente: Propia)

El código es bastante sencillo, lo que hacemos es tomar los valores de los ángulos que se obtiene de los Sliders y de ahí se procede a realizar los cálculos. Unity cuenta con una librería de matemáticas con funciones trigonométricas, así que éstos cálculos son fáciles de realizar, lo único que hay que tener en cuenta es que hay que transformar los valores de los ángulos de grados a radianes, porque las funciones trigonométricas en Unity trabajan con radianes.

```
public void ChangeOfValueArm1()
{
    a1 = (float)((slider1.value) *3.1416)/180;
}

1 referencia
public void ChangeOfValueArm2()
{
    a2 = (float)((slider2.value) * 3.1416) / 180;
}

0 referencias
public void ShowCinDerValues()
{

}

0 referencias
void Update()
{
    x = (7.5 * (Mathf.Cos(a1))) + (6 * (Mathf.Cos(a1 + a2)));
    y = (7.5 * (Mathf.Sin(a1))) + (6 * (Mathf.Sin(a1 + a2)));
    string sliderMessage = "x: " + x.ToString("F2") + " cm     y: " + y.ToString("F2") + " cm";
    cindirtext.text = sliderMessage;
}
```

Convertimos los ángulos a radianes

Se realizan los cálculos de cinemática directa y se envían a la caja de texto

Imagen 60. Cálculo de la cinemática directa (Fuente: Propia)

Se realizan ambos cálculos cada uno en su propio script con las fórmulas establecidas anteriormente y finalmente se envían estos valores a la caja de texto.

A continuación podemos ver el movimiento de los brazos y los ángulos de las articulaciones en la parte superior, así como los resultados de los cálculos de cinemática en la parte inferior.

Imagen 61. Movimiento de los brazos y cálculos de cinemática (Fuente: Propia)

Realizar la comunicación vía wifi entre la aplicación y el robot para el control del mismo

Para realizar la conexión entre nuestra aplicación y la tarjeta ESP8266 se hace uso de Arduino para la creación de un servidor que se aloja en una dirección IP establecida y a la cual se accede a través de un puerto definido en la configuración. Una vez que Arduino ha creado este servidor, podemos acceder desde nuestra aplicación en Unity haciendo uso de las herramientas que nos proporciona el SDK de Uduino.

Primero se explicará el código de Arduino que le permite al ESP8266 crear un servidor y controlar los servomotores y posteriormente se explicará como se envían comandos desde Unity para el control de los servomotores.

CÓDIGO EN ARDUINO

El código de Arduino es bastante sencillo, usa dos librerías principalmente, **Servo.h** y **Uduino_WiFi.h**, siendo ésta última la que permite una fácil conexión con Unity.

Lo que hace nuestro programa es crear un objeto tipo **uduino** y luego creamos todos los comandos que nos permiten realizar diferentes acciones desde Unity, como mandar valores digitales y análogos a los diferentes pines de la targeta, leer valores digitales y analógicos o inicializar y desconectar los servos entre otros. Estos comandos cuentan cada uno con su propia rutina que se ejecuta cuando la aplicación manda a llamar alguno de los comandos. Estos comandos se leen y pueden visualizar a través del puerto serial.

```
#include<Uduino_WiFi.h>
Uduino_WiFi uduino("uduinoBoard"); // Declare and name your object

// Servo
#include <Servo.h>
#define MAXSERVOS 8

void setup()
{
  Serial.begin(9600);
#if defined (__AVR_ATmega32U4__) // Leonardo
  while (!Serial) {}
#elif defined(__PIC32MX__)
  delay(1000);
#endif

  uduino.connectWifi("ZTE BLADE L7A", "08a5469b4dc8");

  uduino.addCommand("s", SetMode);
  uduino.addCommand("d", WritePinDigital);
  uduino.addCommand("a", WritePinAnalog);
  uduino.addCommand("rd", ReadDigitalPin);
  uduino.addCommand("r", ReadAnalogPin);
  uduino.addCommand("br", BundleReadPin);
  uduino.addCommand("b", ReadBundle);

  uduino.addInitFunction(InitializeServos);
  uduino.addDisconnectedFunction(DisconnectAllServos);
}
```

Se incluyen las librerías y se define el objeto tipo uduino

Comandos que nos permiten controlar la placa desde Unity

Imagen 62. Definición de objeto tipo uduino y creación de comandos (Fuente: Propia)

La parte del código que controla los servomotores es bastante sencilla. Cuenta con funciones como inicializar servos, preparar los servos, desconectar uno o todos los servos, actualizar el valor PWM y obtener el índice del PIN conectado.

```
Servo servos[MAXSERVOS];
int servoPinMap[MAXSERVOS];

void InitializeServos() {
  for (int i = 0; i < MAXSERVOS - 1; i++ ) {
    servoPinMap[i] = -1;
    servos[i].detach();
  }
}

void SetupServo(int pin) {
  if (ServoConnectedPin(pin))
    return;

  int nextIndex = GetAvailableIndexByPin(-1);
  if (nextIndex == -1)
    nextIndex = 0;
  servos[nextIndex].attach(pin);
  servoPinMap[nextIndex] = pin;
}


void DisconnectServo(int pin) {
  servos[GetAvailableIndexByPin(pin)].detach();
  servoPinMap[GetAvailableIndexByPin(pin)] = 0;
}
```

```
bool ServoConnectedPin(int pin) {
  if (GetAvailableIndexByPin(pin) == -1) return false;
  else return true;
}

int GetAvailableIndexByPin(int pin) {
  for (int i = 0; i < MAXSERVOS - 1; i++ ) {
    if (servoPinMap[i] == pin) {
      return i;
    } else if (pin == -1 && servoPinMap[i] < 0) {
      return i; // return the first available index
    }
  }
  return -1;
}

void UpdateServo(int pin, int targetValue) {
  int index = GetAvailableIndexByPin(pin);
  servos[index].write(targetValue);
  delay(10);
}

void DisconnectAllServos() {
  for (int i = 0; i < MAXSERVOS; i++) {
    servos[i].detach();
    digitalWrite(servoPinMap[i], LOW);
    servoPinMap[i] = -1;
  }
}
```

Imagen 63. Código para el control de los servos (Fuente: Propia)

Finalmente tenemos la parte que nos permite crear el servidor indicándole la red a la que nos vamos a conectar y la contraseña de esta. A continuación al correr el programa, nuestra tarjeta ESP8266 nos indica la dirección IP en la que se alojará a la cual debemos acceder desde nuestra aplicación así como el puerto al que hay que conectarse.

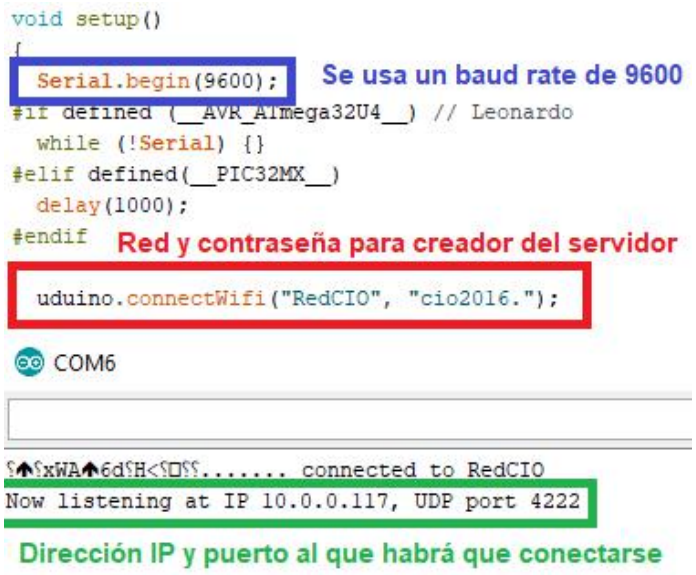


Imagen 64. Configuración de conexión de la tarjeta (Fuente: Propia)

CÓDIGO EN UNITY

Una vez que ya tenemos establecido éste servidor en nuestra tarjeta ESP8266, es momento de hablar de la configuración y el código que nos permite controlar el robot desde nuestra aplicación.

Para el control se usa el SDK **Uduino**. Uduino es un complemento de Unity que simplifica la comunicación entre Arduino y Unity. Tiene todas las características de mapeo que la mayoría de los desarrolladores necesitan para sus instalaciones interactivas. Funciona de manera eficiente en todos los principales sistemas operativos de escritorio, y presenta ejemplos y utiliza una API simple y legible.

Primero se importan los Assets de Uduino en nuestro proyecto de la misma manera que se ha hecho con los SDK anteriores y se inserta un objeto tipo Uduino

en nuestro proyecto, éste objeto es el que nos permite ver la configuración de la conexión que se realiza con Arduino via WiFi.

Haciendo click sobre éste objeto podemos ver y cambiar la configuración para poder controlar nuestro robot.

El tipo de conexión que usaremos es WiFi, así que la seleccionamos, colocamos un Baud Rate de 9600 que coincida con el que se indicó en Arduino para que puedan comunicarse. Como el puerto se mantiene estático cada vez que se conecta la placa se ingresa el valor, el cual es **42222**. El valor de la dirección IP se deja en blanco debido a que se ingresará desde nuestra aplicación.

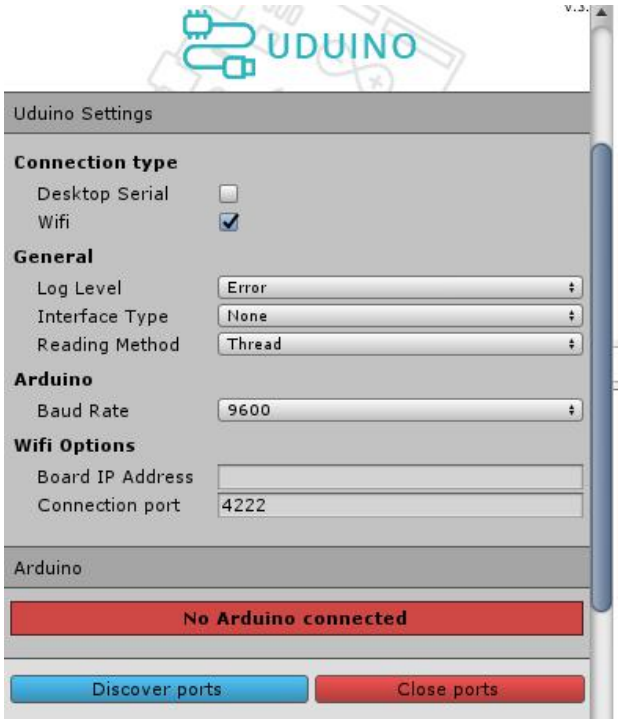


Imagen 65. Configuración de Uduino. (Fuente: Propia)

Una vez que tenemos la configuración básica, nos pasamos a la pestaña de configuración avanzada, en donde realizaremos algunos cambios. La mayor parte de la configuración que viene por defecto es la indicada, pero necesitamos indicarle que estamos trabajando con un **NodeMCU** en el tipo de tarjeta. Además que se desactiva la casilla de **Discover on Play** ya que no queremos que busque placas cuando empieza a correr la aplicación, si no cuando nosotros le indiquemos.

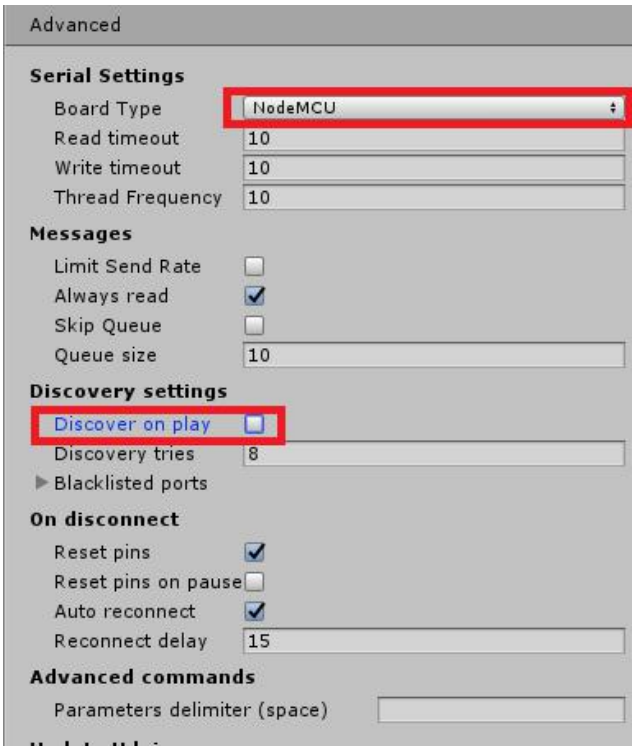


Imagen 66. Configuración avanzada de Uduino (Fuente: Propia)

Una vez que tenemos la configuración establecida es hora de crear los controles que nos faltan y de programar la conexión por medio de la IP, así como el control de los servomotores.

Los únicos dos controles que tenemos que agregar a nuestra aplicación son los que nos permitirán conectarnos a la placa haciendo uso de la dirección IP. Así que agregamos una caja de texto y un botón que nos permitirán acceder al servidor.

Imagen 67. Caja de texto y botón para ingresar dirección IP (Fuente: Propia)

Ya que contamos con todos los elementos en la interfaz gráfica que nos permiten la conexión y el control del robot procedemos a explicar el código. Los scripts que se usan son el que nos permite conectarnos a la dirección IP de la tarjeta y el que una vez conectados nos permite mover los brazos.

En el primero lo que hacemos es primero descubrir el objeto tipo Uduino en nuestra escena, luego solicitamos el valor que se ingresa a la caja de texto y lo guardamos en una variable para después establecer la dirección IP en nuestro objeto tipo Uduino y finalmente usamos el comando de Uduino para descubrir puertos. Todo éste código lo adjuntamos a nuestro botón y lo activamos cuando se hace un click sobre éste.

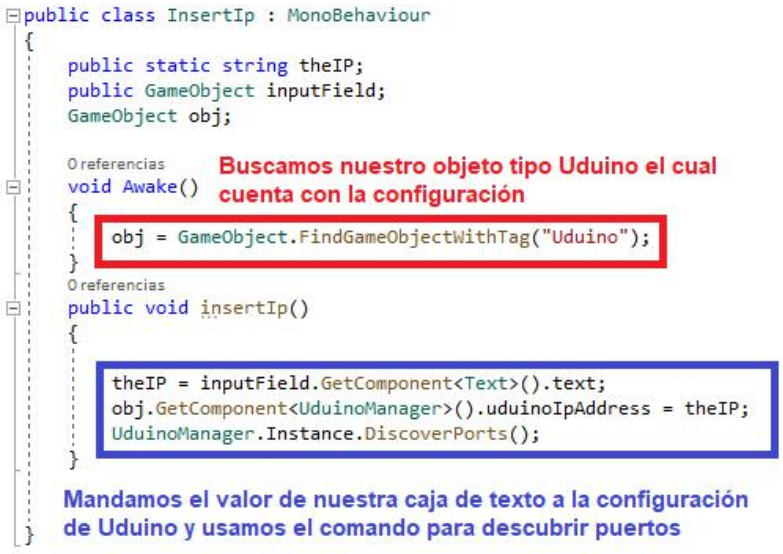


Imagen 68. Código para la conexión IP (Fuente: Propia)

Para el código que nos permitirá controlar los servomotores usamos los mismos **Sliders** que usamos para el control del gemelo digital ya que se busca que tanto el gemelo como el real se muevan simultáneamente.

Para enviar éstos valores al robot se establece el PIN como modo **Servo**, luego tomamos el valor del brazo y lo convertimos a un valor entre 0 y 255 que son los valores que toman los pulsos PWM en Arduino y finalmente le mandamos ese valor a nuestra placa. Se realizaron dos códigos, uno para cada brazo, pero debido a que el código es prácticamente el mismo, solo se muestra el código del brazo inferior, para mover el brazo superior lo único que se debe cambiar es el número de PIN en el que se encuentra el servomotor y también se asigna el código al otro Slider, o sea el que controla el segundo brazo.

```
0 referencias
void Start()
{
    UduinoManager.Instance.pinMode(12, PinMode.Servo);
}
```
Se indica el pin donde se encuentra conectado el servomotor

```
0 referencias
public void Slider_Changed(float brazo_inf)
{
    thetanice = (int)brazo_inf;
    servoValue = (int)(thetanice * 0.8555) + 6;
    UduinoManager.Instance.analogWrite(12, servoValue);
}
// Update is called once per frame
0 referencias
void Update()
{

}
```
Se transforman los valores del Slider al valor del pulso PWM que se enviará al servomotor

Imagen 69. Código para el control de los brazos (Fuente: Propia)

A continuación se ve como controlamos ambos de manera simultanea. (Se ha ocultado momentáneamente los cálculos de cinemática para que se pueda apreciar con más detalle el movimiento de los brazos).

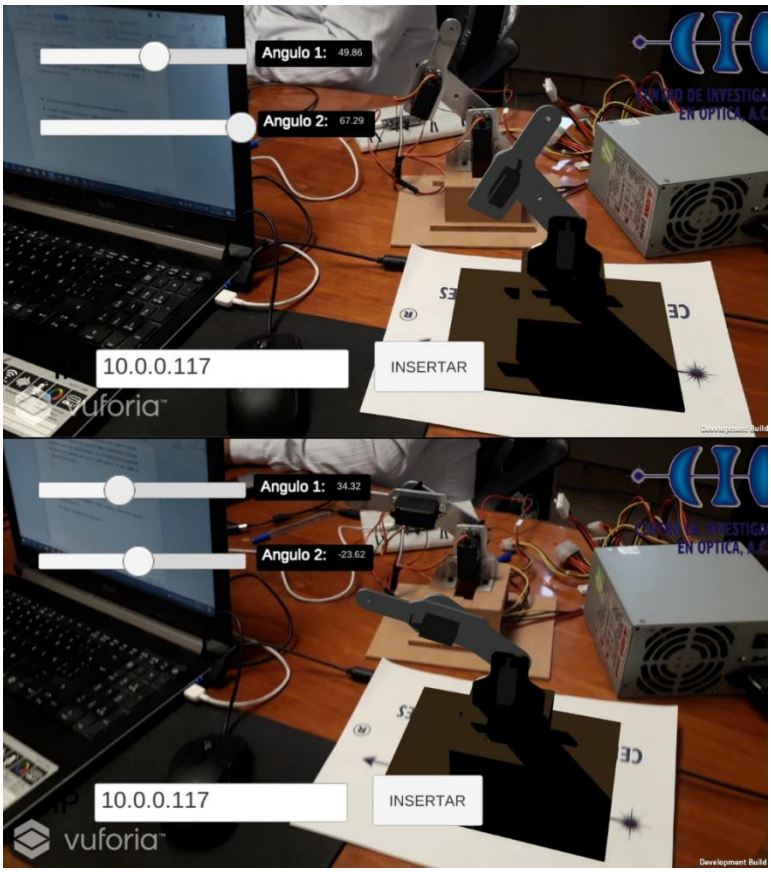


Imagen 70. Movimiento simultaneo del robot y su gemelo digital (Fuente: Propia)

Desarrollar una base de datos para almacenar rutinas del robot

Ya que podemos controlar nuestro robot haciendo uso de los Sliders es momento de agregarle a nuestra aplicación rutinas que podamos guardar para ejecutar posteriormente.

Para poder guardar éstas rutinas se necesitará una **base de datos**. En la base de datos guardaremos nuestras **rutinas** y cada rutina contará con todos los pasos de la rutina. Cada paso guardado nos debe indicar **el número de paso**, **las coordenadas xy del efector final**, **las posiciones de los ángulos de los brazos** y finalmente **el tiempo que tardará en llegar a dicha posición**.

Para poder realizar nuestra base de datos se usa **SQLite**. SQLite es una biblioteca en lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta confiabilidad y con todas las funciones. SQLite es el motor de base de datos más utilizado en el mundo. SQLite está integrado en todos los teléfonos móviles y la mayoría de las computadoras y viene incluido en innumerables aplicaciones que la gente usa todos los días.

El formato de archivo SQLite es estable, multiplataforma y compatible con versiones anteriores. Hay más de 1 billón de bases de datos SQLite en uso activo. El código fuente de SQLite está en el dominio público y es gratuito para que todos lo utilicen para cualquier propósito. Lo primero que tenemos que hacer para desarrollar nuestra base de datos en SQLite es importar todos los archivos que le permiten operar dentro de Unity a nuestra carpeta de Assets.

Imagen 71. Archivos de SQLite para Unity (Fuente: Propia)

Una vez que contamos con los archivos que nos permiten programar bases de datos dentro de nuestro programa procedemos a desarrollar la interfaz gráfica que nos permita controlar las rutinas. Primero empezamos con la interfaz principal de nuestro programa que es la que se ejecuta cuando iniciamos la aplicación.

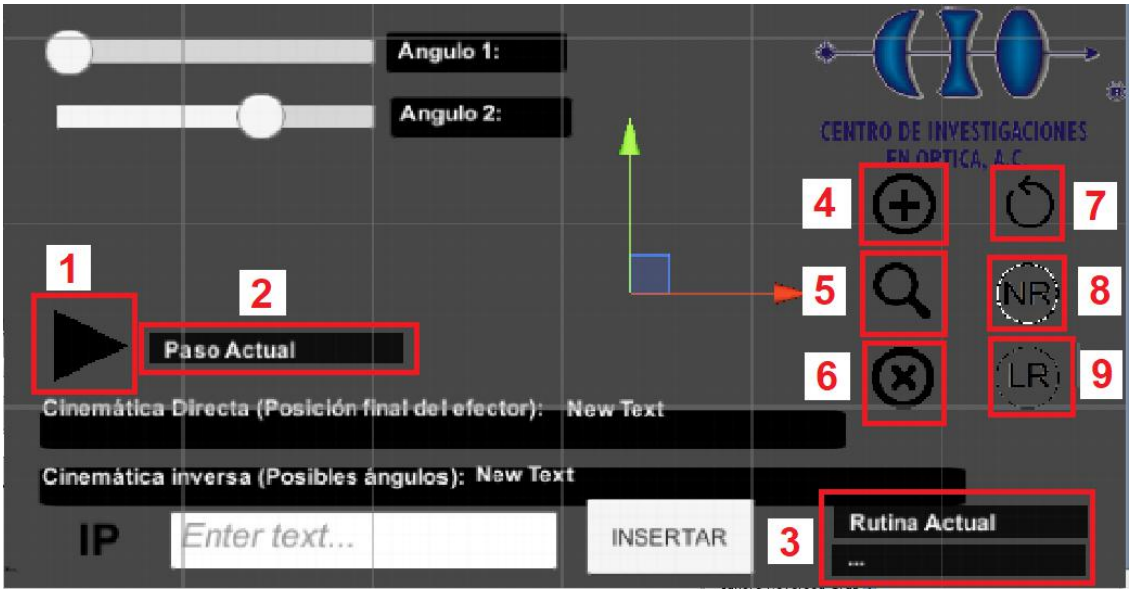


Imagen 72. Interfaz principal de la aplicación (Fuente: Propia)

1. Este botón nos deja reproducir la rutina actual con la que se está trabajando.

2. Cuando se da click en el botón de reproducir rutina aquí se nos muestra el paso que se está ejecutando en el momento.

3. Acá se muestra la rutina con la que estamos trabajando.

4. Este botón nos lleva al menú **“Insertar Pasos”**

5. Este botón nos lleva al menú **“Buscar por Paso”**

6. Este botón nos lleva al menú **“Borrar Pasos”**

7. Este botón nos lleva al menú **“Actualizar Pasos”**

8. Este botón nos lleva al menú **“Crear nueva rutina”**

9. Este botón nos lleva al menú **“Cargar Rutina”**

Una vez que contamos con todos los botones que nos llevarán a todos los distintos menús, se crean cada uno de los menús que nos permitirán trabajar con las rutinas. A su vez se irá explicando que es lo que hace el código para cada función que nos proporciona el menú.

Primero se explica la primera parte del código que nos permite crear la base de datos en SQLite y que se ejecuta en cuanto se inicia la aplicación.

Cuando iniciamos la aplicación lo primero que busca el código de la base de datos es la existencia de el archivo de formato **.s3db** que es en el cual se guardan las rutinas. Si éste archivo no existe nuestra aplicación crea el archivo y lo guarda dentro de la carpeta de la aplicación dentro de nuestro dispositivo Android. Una vez que contamos con el archivo podemos realizar la conexión con la base de datos para poder editarla.

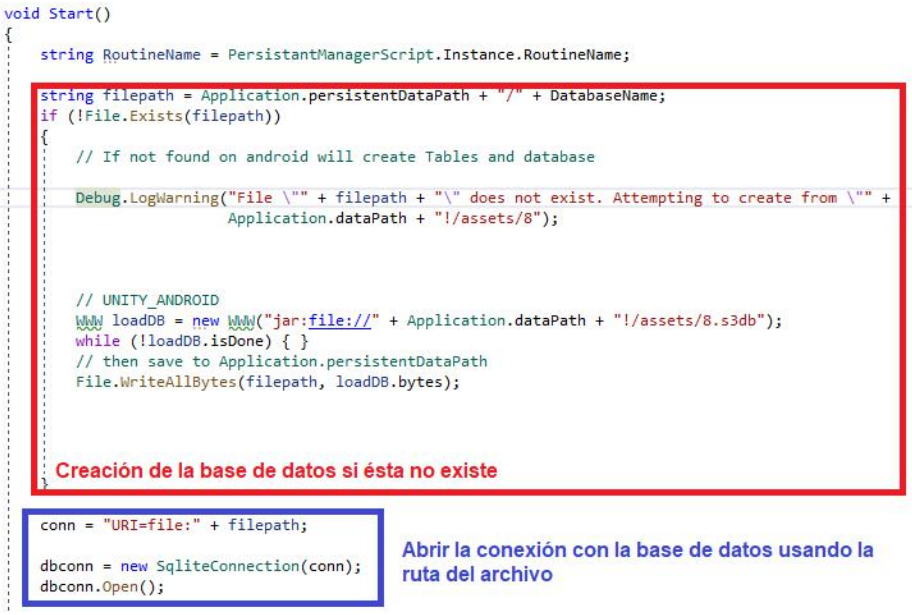


Imagen 73. Creación de la base de datos y conexión con ésta (Fuente: Propia)

Posteriormente automáticamente se crea una primera tabla, en la que se encuentra una primera rutina vacía que tenemos por defecto. Esta rutina cuenta con todos los parámetros de pasos que se indicaron anteriormente (número de paso, ángulo 1, ángulo 2, posiciones x y y del efector final y el tiempo en el que debe tardarse para llegar a esa posición.

```
string query;
query = "CREATE TABLE Rutina1 (Paso INTEGER PRIMARY KEY   AUTOINCREMENT, A1 FLOAT, A2 FLOAT, x FLOAT, y FLOAT, t INTEGER)";
try
{
    dbcmd = dbconn.CreateCommand(); // create empty command
    dbcmd.CommandText = query; // fill the command
    reader = dbcmd.ExecuteReader(); // execute command which returns a reader
```

Imagen 74. Creación de tabla de Rutina por defecto (Fuente: Propia)

Una vez contando con esto podemos mostrar como funciona cada uno de lo menús.

INSERTAR PASOS

Imagen 75. Interfaz del menú "Insertar Pasos" (Fuente: Propia)

1. Aquí se muestran los valores de los ángulos y la posición del efector final en los que se encuentra actualmente nuestro robot.

2. Aquí se agrega el tiempo que el robot tardará en llegar a ésta posición.

3. Con éste botón agregamos este paso a la rutina y la guardamos en la tabla.

4. Luego de presionar el botón de guardar, acá se muestran los valores guardados.

5. Este botón nos permite regresar a la interfaz principal y se encuentra en todos los menús de la aplicación, así que solo se mencionará su funcionamiento una vez.

Teniendo esta interfaz podemos desarrollar el código que nos permitirá guardar pasos en nuestra rutina. Primero creamos las variables en las cuales se guardarán nuestros valores.

Siempre que queremos comunicarnos con la base de datos debemos de **abrir la conexión** y luego crear un **comando tipo String** mediante el cual le indicamos a nuestra base de datos que es lo que debe de hacer. En el caso de insertar una linea a una tabla usamos el comando **"insert into"** seguido del nombre de la tabla, que en nuestro caso es el nombre de la rutina con la que estemos trabajando y posteriormente se indica **que valores ingresaremos y en que columnas de la tabla se ingresarán**. De ahí se le indica al programa que hay que enviar ésta linea de comando y finalmente se cierra la conexión con la base de datos después de cada comando. Una vez que agregamos el paso ejecutamos la función **reader_function**. Esta función nos permite mostrar el paso que acabamos de agregar en el cuadro de texto de abajo (**4** en la figura anterior).

```
private void insert_function(string t)
{
    float A1;
    float A2;
    float x;
    float y;

    A1 = PersistantManagerScript.Instance.ANGULO1;
    A2 = PersistantManagerScript.Instance.ANGULO2;
    x = PersistantManagerScript.Instance.EQUIS;
    y = PersistantManagerScript.Instance.YE;

    using (dbconn = new SqliteConnection(conn))
    {
        dbconn.Open(); //Open connection to the database.
        dbcmd = dbconn.CreateCommand();
        sqlQuery = string.Format("insert into " + PersistantManagerScript.Instance.RoutineName + " (A1, A2, x, y, t) values (\"{0}\
        dbcmd.CommandText = sqlQuery;
        dbcmd.ExecuteScalar();
        dbconn.Close();
    }
    data_staff.text = "";

    reader_function();
}
```

Creamos las variables en las cuales guardaremos los valores actuales y del tiempo que ingresemos

Se envía el comando que permite guardar éstos datos en la tabla

Ejecutamos la funcion reader_function

Imagen 76. Código para insertar pasos (Fuente: Propia)

La función reader_function lo que hace es una consulta a la base de datos, solicitando los datos de la tabla haciendo uso del comando "**SELECT** *datos a solicitar* **FROM** *nombre de la tabla*" y guardando cada uno de estos en una variable string. Finalmente junta todos los strings en un string largo que es el que se muestra en el cuadro de texto.

```
using (dbconn = new SqliteConnection(conn))
{
    dbconn.Open(); //Open connection to the database.
    IDbCommand dbcmd = dbconn.CreateCommand();
    string sqlQuery = "SELECT  Paso, A1, A2, x, y, t " + "FROM " + PersistantManagerScript.Instance.RoutineName;// table name
    dbcmd.CommandText = sqlQuery;
    IDataReader reader = dbcmd.ExecuteReader();
    while (reader.Read())
    {
        // idreaders = reader.GetString(1);
        Pasoreaders = reader.GetInt32(0).ToString();
        A1readers = reader.GetDouble(1).ToString();
        A2readers = reader.GetDouble(2).ToString();
        xreaders = reader.GetDouble(3).ToString();
        yreaders = reader.GetDouble(4).ToString();
        treaders = reader.GetInt32(5).ToString();

        data_staff.text += Pasoreaders + " - " + A1readers + " - " + A2readers + " - " + xreaders + " - " + yreaders + " - " + treaders + "\n";
    }
    reader.Close();
    reader = null;
    dbcmd.Dispose();
    dbcmd = null;
    dbconn.Close();
```

Comando para solicitar los datos

Guardamos estos datos y los convertimos en strings

Mostramos los datos en el cuadro de texto

Imagen 77. Código para mostrar el paso en el cuadro de texto (Fuente: Propia)

BUSCAR POR PASO

Imagen 78. Interfaz del menú "Buscar por paso" (Fuente: Propia)

1. Aquí seleccionamos el paso que queremos buscar de nuestra rutina.

2. Este botón busca el paso que ingresamos en la caja de texto.

3. Todos los datos del paso buscado nos aparecerán en ésta caja de texto.

El código que usamos para buscar un paso es muy similar al que usa la función reader_function, con la diferencia de que al final de nuestro comando de consulta colocamos un **where**, es aquí que le indicamos al programa el número de paso que estamos solicitando a la base de datos. La linea de comando para SQLite sería "**SELECT** *datos a solicitar* **FROM** *nombre de la tabla* **WHERE Paso =** *número de paso*". Posteriormente guardamos los valores en strings y lo mostramos en el cuadro de texto inferior.

```
using (dbconn = new SqliteConnection(conn))
{
    string Paso_readers_Search, A1_readers_Search, A2_readers_Search, x_readers_Search, y_readers_Search, t_readers_Search;
    dbconn.Open(); //Open connection to the database.
    IDbCommand dbcmd = dbconn.CreateCommand();
    string sqlQuery = "SELECT Paso,A1,A2,x,y,t " + "FROM " + PersistantManagerScript.Instance.RoutineName + " where Paso =" + Search_by_Paso;
    dbcmd.CommandText = sqlQuery;
    IDataReader reader = dbcmd.ExecuteReader();
    while (reader.Read())
    {
        //  string id = reader.GetString(0);
        Paso_readers_Search = reader.GetInt32(0).ToString();
        A1_readers_Search = reader.GetDouble(1).ToString();
        A2_readers_Search = reader.GetDouble(2).ToString();
        x_readers_Search = reader.GetDouble(3).ToString();
        y_readers_Search = reader.GetDouble(4).ToString();
        t_readers_Search = reader.GetInt32(5).ToString();

        data_staff.text += Paso_readers_Search + " - " + A1_readers_Search + " - " + A2_readers_Search + " - " + x_readers_Search + " - " + y_



    }
    reader.Close();
```

Indicamos el paso del que solicitamos los datos

Imagen 79. Código para buscar pasos (Fuente: Propia)

BORRAR PASOS

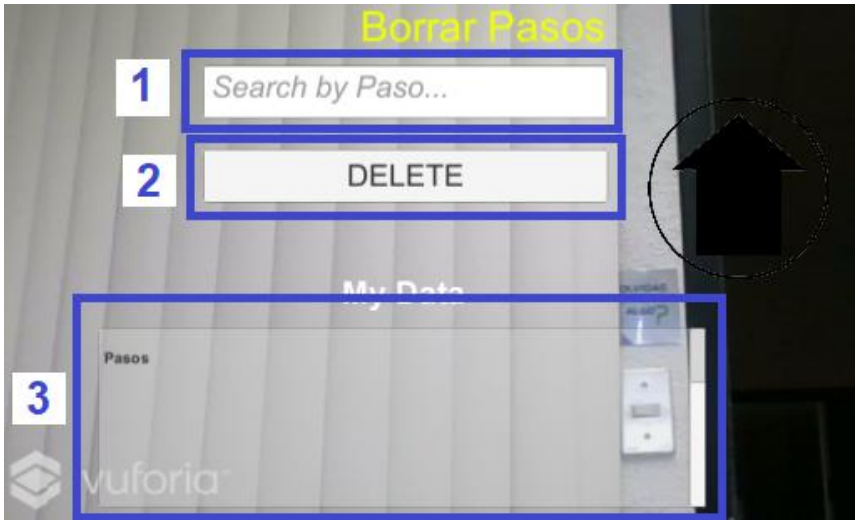


Imagen 80. Interfaz del menú "Borrar Pasos" (Fuente: Propia)

1. Aquí indicamos el paso que queremos borrar.

2. Este botón borra el paso que ingresamos en la caja de texto.

3. Esta caja de texto nos muestra un texto que indica que se el paso se ha borrado exitósamente.

El código para borrar pasos es bastante sencillo, lo único que hay que hacer es hacer uso del comando de SQLite, así que la linea de comando que mandamos es "**DELECT FROM** *nombre de la tabla* **WHERE Paso =** *número de paso*". Al final se envía un string al cuadro de texto inferior que nos indica que el paso se ha borrado.

```
private void Delete_function(string Delete_by_Paso)
{
    using (dbconn = new SqliteConnection(conn))
    {

        dbconn.Open(); //Open connection to the database.
        IDbCommand dbcmd = dbconn.CreateCommand();
        string sqlQuery = "DELETE FROM " + PersistantManagerScript.Instance.RoutineName + " where Paso =" + Delete_by_Paso;
        dbcmd.CommandText = sqlQuery;
        IDataReader reader = dbcmd.ExecuteReader();


        dbcmd.Dispose();
        dbcmd = null;
        dbconn.Close();
        data_staff.text = Delete_by_Paso + " Delete  Done ";

    }
```

Comando para borrar pasos

Imagen 81. Código para borrar pasos (Fuente: Propia)

ACTUALIZAR PASOS

Imagen 82. Interfaz del menú "Actualizar Pasos" (Fuente: Propia)

1. Aquí insertamos el paso al cual le queremos cambiar los valores.

2. Luego de presionar *SEARCH* aquí nos apareceran los datos del paso buscado.

3. Aquí aparece la posición actual de nuestro robot y por lo tanto la posición que usaremos para remplazar los viejos valores.

4. Aquí ingresamos el valor de tiempo que queremos que tome el paso actualizado.

5. Este botón nos permite actualizar el viejo paso buscado con los datos actuales.

6. Acá nos aparece el paso que hemos actualizado.

El código cuenta con dos partes, en donde se busca el paso y en donde actualizamos los valores. El código de la primera parte es exactamente igual al de buscar paso, pero se guarda el string en el cuadro que se indica como **2** en la imagen anterior, así como mostrar los valores que tenemos actualmente en el cuadro de texto **3**. Para la segunda parte guardamos los valores nuevos en variables y de ahí usamos una linea de comandos de SQLite. La linea de comando

que nos permite actualizar la base de datos es “**UPDATE** *nombre de la tabla* **set A1 = @A1 ,A2 = @A2 ,x = @x ,y = @y ,t = @t where Paso = @paso**” de esa manera actualizamos los viejos valores con los nuevos valores que se le indican al programa.

```
using (dbconn = new SqliteConnection(conn))
{
    dbconn.Open(); //Open connection to the database.
    dbcmd = dbconn.CreateCommand();
    sqlQuery = string.Format("UPDATE " + PersistantManagerScript.Instance.RoutineName + " set A1 = @A1 ,A2 = @A2 ,x = @x ,y = @y ,t = @t where Paso = @paso ");

    SqliteParameter P_update_A1 = new SqliteParameter("@A1", PersistantManagerScript.Instance.ANGULO1);
    SqliteParameter P_update_A2 = new SqliteParameter("@A2", PersistantManagerScript.Instance.ANGULO2);
    SqliteParameter P_update_x = new SqliteParameter("@x", PersistantManagerScript.Instance.EQUIS);
    SqliteParameter P_update_y = new SqliteParameter("@y", PersistantManagerScript.Instance.YE);
    SqliteParameter P_update_t = new SqliteParameter("@t", update_t);
    SqliteParameter P_update_Paso = new SqliteParameter("@paso", update_Paso);

    dbcmd.Parameters.Add(P_update_A1);
    dbcmd.Parameters.Add(P_update_A2);
    dbcmd.Parameters.Add(P_update_x);
    dbcmd.Parameters.Add(P_update_y);
    dbcmd.Parameters.Add(P_update_t);
    dbcmd.Parameters.Add(P_update_Paso);
```

Comando para actualizar valores

Guardamos los nuevos valores en variables para agregar a la linea de comando

Imagen 83. Código para actualizar pasos (Fuente: Propia)

CREAR NUEVA RUTINA

Imagen 84. Interfaz del menú "Crear nueva rutina" (Fuente: Propia)

1. Ingresamos el nombre de la rutina que queremos crear.

2. Este botón nos permite crear la rutina con el nombre establecido previamente.

3. En la caja de texto nos aparecerán los nombres de todas la rutinas guardadas incluyendo la recién creada.

Para crear una nueva rutina el código lo que hace es crear una nueva tabla indicándole todos los parámetros que ésta tendrá. Usa el comando de SQLite llamado **CREATE TABLE**. Al terminar de crear la tabla ejecuta la función **actualizar_function**.

Imagen 85. Código para crear nuevas rutinas (Fuente: Propia)

La función actualizar_function lo que hace es solicitar los nombres de todas las tablas en nuestro programa, todas estas tablas están guardadas en una tabla maestra llamada **sqlite_master**, así que lo único que debemos hacer es solicitar los nombres y mostrarlos en el cuadro de texto inferior.

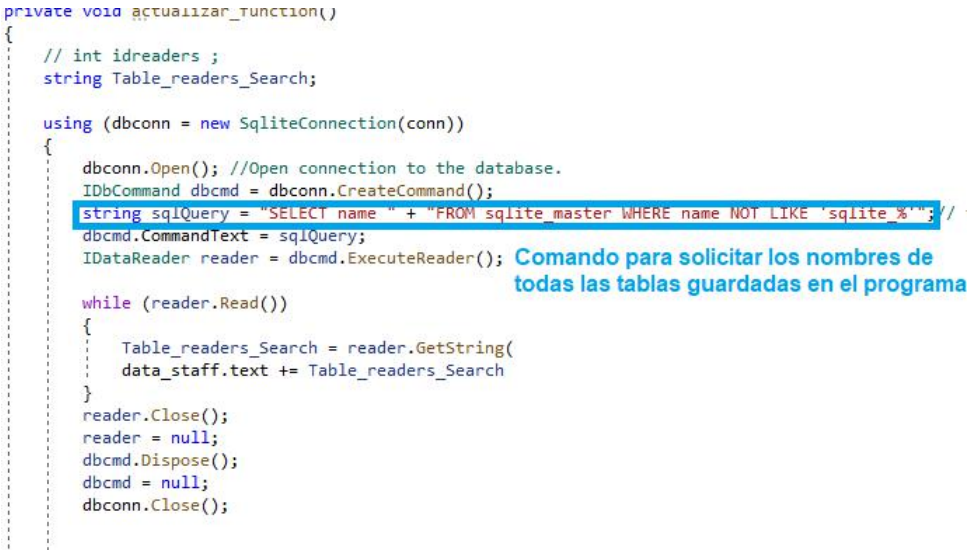


Imagen 86. Código para solicitar rutinas existentes (Fuente: Propia)

CARGAR RUTINA

Imagen 87. Interfaz del menú "Cargar Rutina" (Fuente: Propia)

1. Al hacer click sobre éste botón se nos cargarán todas las rutinas guardadas en el dropdown inferior.

2. Cuando el dropdown tiene las rutinas actualizadas, podemos seleccionar una de estas para trabajar.

El código del botón update lo que hace es solicitar todas las rutinas de la tabla maestra y posteriormente las guarda en una variable tipo lista. Luego de que todas las rutinas se encuentran en ésta lista, las usa para mostrarlas como opciones dentro del dropdown.

```
dbconn.Open(); //Open connection to the database.
IDbCommand dbcmd = dbconn.CreateCommand();
string sqlQuery = "SELECT name " + "FROM sqlite_master WHERE name NOT LIKE 'sqlite_%'";
dbcmd.CommandText = sqlQuery;
IDataReader reader = dbcmd.ExecuteReader();

Rutinas.Clear();
dropR.ClearOptions();
while (reader.Read())
{
    Table_readers_Search = reader.GetString(0);
    Rutinas.Add(Table_readers_Search);
}
dropR.AddOptions(Rutinas);
```

Agregamos las rutinas a una variable tipo lista

Usamos esta lista para llenar las opciones del dropdown

Imagen 88. Código para actualizar la lista de rutinas (Fuente: Propia)

Una vez que la lista se puede desplegar en el dropdown, usamos un pequeño código en el dropdown que no es guarda en una variable la rutina que vamos a usar basándonos en el índice que tiene la rutina en la lista.

```
public void Dropdown_IndexChanged(int index)
{
    PersistantManagerScript.Instance.RoutineName = Rutinas[index];


}
```

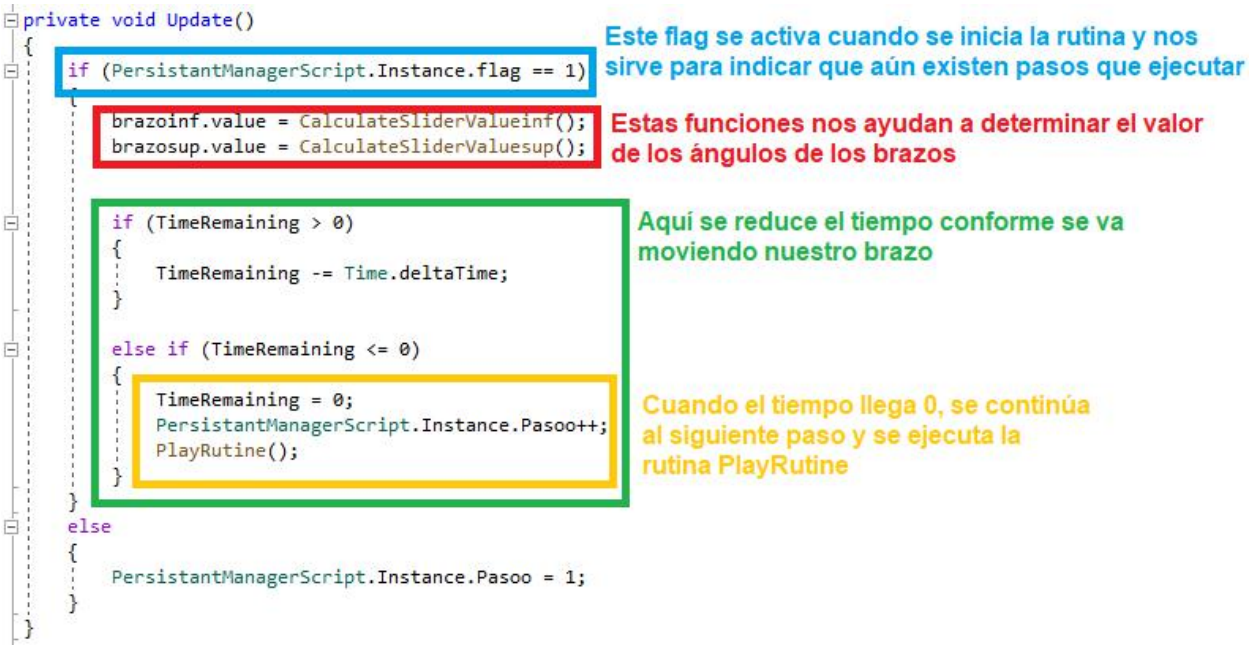
Imagen 89. Código para selección de rutina (Fuente: Propia)

Ya conocemos como funciona cada uno de los menús, la última parte que tenemos que explicar es como se reproducen éstas rutinas, es decir como el programa toma los pasos dentro de la rutina y los ejecuta, haciendo que el robot se mueva de acuerdo a la rutina seleccionada

REPRODUCCIÓN DE RUTINAS

Para el movimiento del robot al ejecutarse una rutina usamos un **void Update**, esto se debe a que las lineas de código que se ponen dentro de un update se ejecutan cada uno de los 60 frames que tiene un segundo. Ésto es muy importante ya que necesitamos una referencia de tiempo que le permita a nuestro código entender los valores de tiempo que guardamos en nuestros pasos.

Cuando presionamos el botón para correr rutinas activamos un **flag**, el valor de este flag se mantiene en uno hasta que todos los pasos de la rutina se hayan ejecutado. Luego determinamos los valores que deben de tener los brazos ejecutando dos pequeñas rutinas. Los valores de los ángulos de los brazos son dependientes del tiempo que haya transcurrido desde que inició la rutina y cuando el tiempo de **cada paso** llega a 0, se ejecuta una rutina llamada **PlayRutine**, lo que hace ésta rutina es pasar al siguiente paso dentro de la rutina, y el ciclo continúa hasta que ya no existen pasos que ejecutar, en ese momento el **flag** inicial se regresa al valor de 0, lo que indica que la rutina ha terminado.

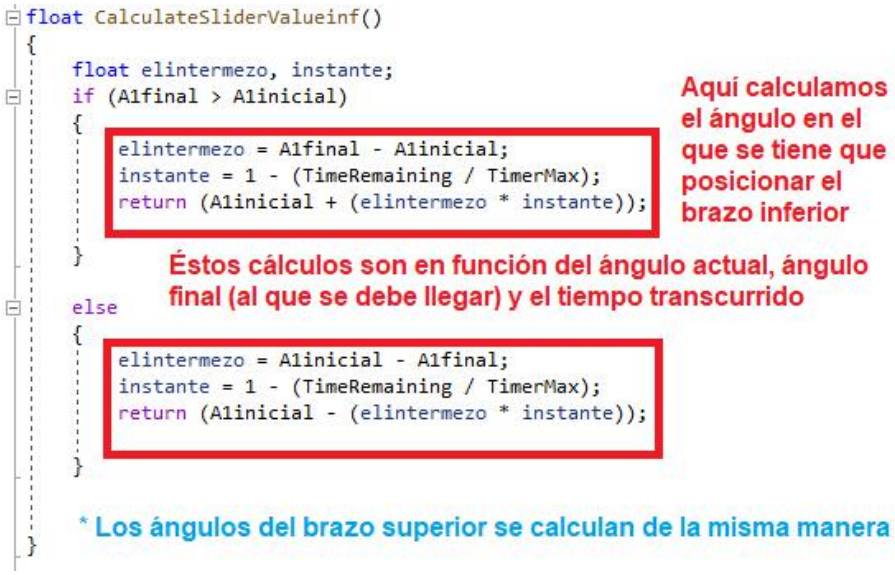



```
private void Update()
{
    if (PersistantManagerScript.Instance.flag == 1)
    {
        brazoinf.value = CalculateSliderValueinf();
        brazosup.value = CalculateSliderValuesup();

        if (TimeRemaining > 0)
        {
            TimeRemaining -= Time.deltaTime;
        }

        else if (TimeRemaining <= 0)
        {
            TimeRemaining = 0;
            PersistantManagerScript.Instance.Pasoo++;
            PlayRutine();
        }
    }
    else
    {
        PersistantManagerScript.Instance.Pasoo = 1;
    }
}
```

Imagen 90. Código para el movimiento del brazo al ejecutar una rutina (Fuente: Propia)




```
float CalculateSliderValueinf()
{
    float elintermezo, instante;
    if (A1final > A1inicial)
    {
        elintermezo = A1final - A1inicial;
        instante = 1 - (TimeRemaining / TimerMax);
        return (A1inicial + (elintermezo * instante));
    }
    else
    {
        elintermezo = A1inicial - A1final;
        instante = 1 - (TimeRemaining / TimerMax);
        return (A1inicial - (elintermezo * instante));
    }
}
```

Imagen 91. Código para calcular el valor del ángulo del brazo inferior (Fuente: Propia)

Imagen 92. Código para determinar los valores del siguiente paso (Fuente: Propia)

Integrando esta base de datos para rutinas a nuestra aplicación, hemos teminado con todas las características con las que esta debe contar. Una vez que ya tenemos toda la programación de nuestra aplicación, lo único que queda hacer es exportar el archivo Android a una APK, instalarla en el dispositivo, y si es necesario hacer ajustes para arreglar errores de compatibilidad.

Exportar aplicación a Android, realizar pruebas y ajustes para encontrar posibles errores e integrar a la plataforma de capacitación

Una vez que contamos con todos los elementos que nuestra aplicación requiere, debemos exportarla. Existen algunas consideraciones que hay que tomar para que nuestra aplicación pueda correr correctamente en nuestro dispositivo Android.

Debido a la variedad de SDKs y herramientas de las que hace uso nuestra aplicación es importante indicar la configuración de las **build settings**. Primero debemos asegurarnos que la escena en la que hemos estado trabajando se encuentre en **Scenes in build**. También es muy importante que la opción de compresión de texturas sea **Don't override**. Finalmente, y esto es muy importante para evitar problemas de compatibilidad y para permitir que se pueda exportar fácilmente nuestra aplicación, en build system debemos seleccionar **Internal**.

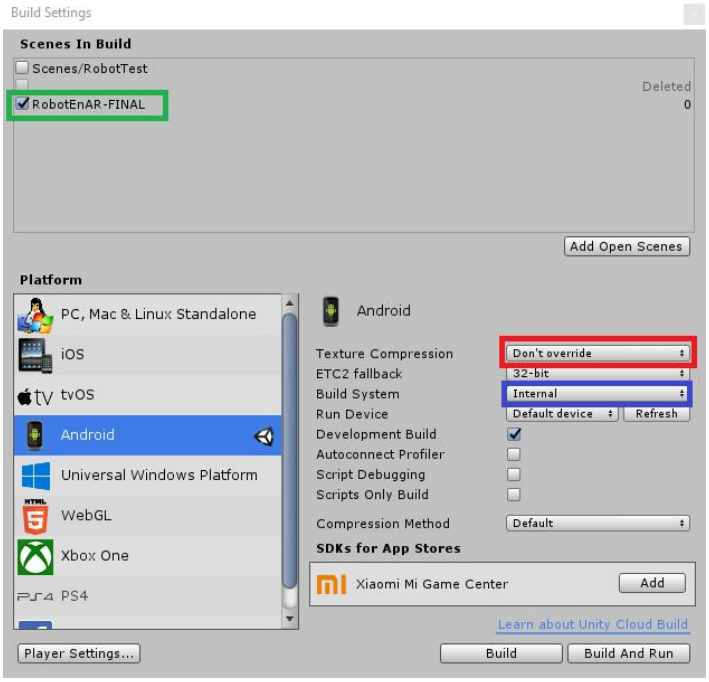


Imagen 93. Build settings de nuestro proyecto (Fuente: Propia)

También debemos tomar en cuenta algunas configuraciones que se encuentran en el **Player settings**. En **Other settings** es importante poner el **nombre de la compañía y del producto**, indicar el **nombre del paquete**, seleccionar el nivel de API mínimo para correr el sistema que en nuestro caso es **Android 8.0 'Oreo' (API level 26)** que es con el que cuenta nuestro celular. Y desactivar la compatibilidad con Android TV, ya que no la vamos a necesitar. Finalmente en **XR settings** tener activada la casilla de **Vuforia Augmented Reality**.

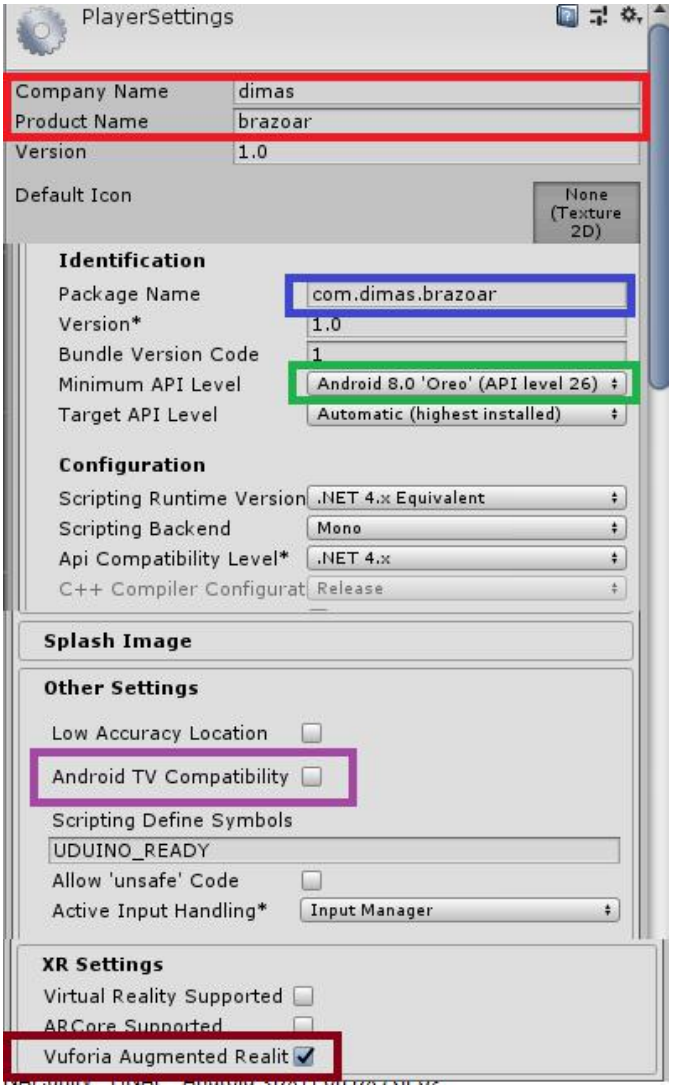


Imagen 94. Player settings de nuestro proyecto (Fuente: Propia)

Una vez que contamos con toda la configuración necesaria exportamos nuestra aplicación como APK, la cual se puede instalar en nuestro dispositivo Android y la cual se puede empezar a usar inmediatamente.

INTEGRACIÓN A LA PLATAFORMA DE CAPACITACIÓN

Imagen 95. Alumnos interactuando con la aplicación (Fuente: Propia)

Ya que contamos con nuestra aplicación instalada es momento de integrarla a la plataforma de capacitación. La aplicación fue utilizada para la clase de mecatrónica y los alumnos interactuaron con el robot y la aplicación. Además de pequeños detalles con el apartado visual de la interfaz, los cuales fueron arreglados inmediatamente debido a que eran meros detalles estéticos, los alumnos mostraron gran satisfacción con el uso de la aplicación.

Posteriormente se usó la aplicación como una herramienta interactiva en un diplomado de robótica y en una platica de difusión del CIO. En ambas se mostro de gran utilidad para los usuarios.

Resultados

El principal producto resultado del desarrollo del proyecto es una aplicación de realidad aumentada controlada desde un dispositivo Android que nos permite visualizar el gemelo digital de un brazo robótico de 2 grados de libertad.

La aplicación cuenta con las siguientes funcionalidades:

- Nos permite visualizar el gemelo virtual de un brazo robótico de 2 grados de libertad generado a partir de un marcador tipo imagen.
- Hace uso de controles deslizantes para el control del ángulo de las articulaciones del gemelo digital.
- Nos permite conectarnos vía WiFi al robot ingresando la dirección IP de la tarjeta ESP8266 que lo controla y nos permite controlar simultáneamente el robot y a su gemelo digital.
- Nos muestra los resultados de los cálculos de cinemática directa e inversa usando para los cálculos las medidas reales del robot.
- Cuenta con una base de datos que nos permite almacenar rutinas que podemos programar directamente desde la aplicación.

A continuación se pueden ver varias capturas tomadas desde la aplicación:

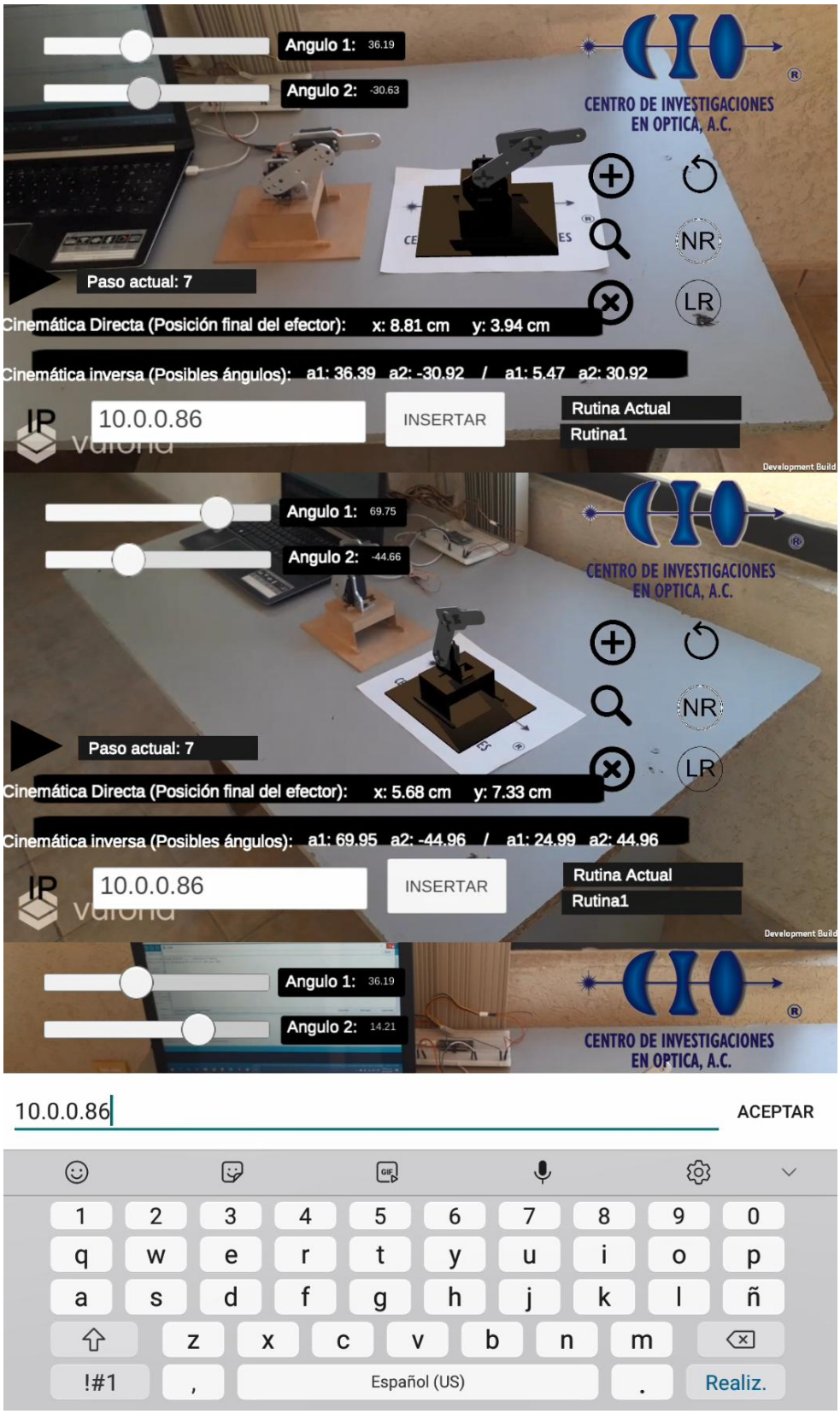


Imagen 96. Capturas tomas desde la app (1) (Fuente: Propia)

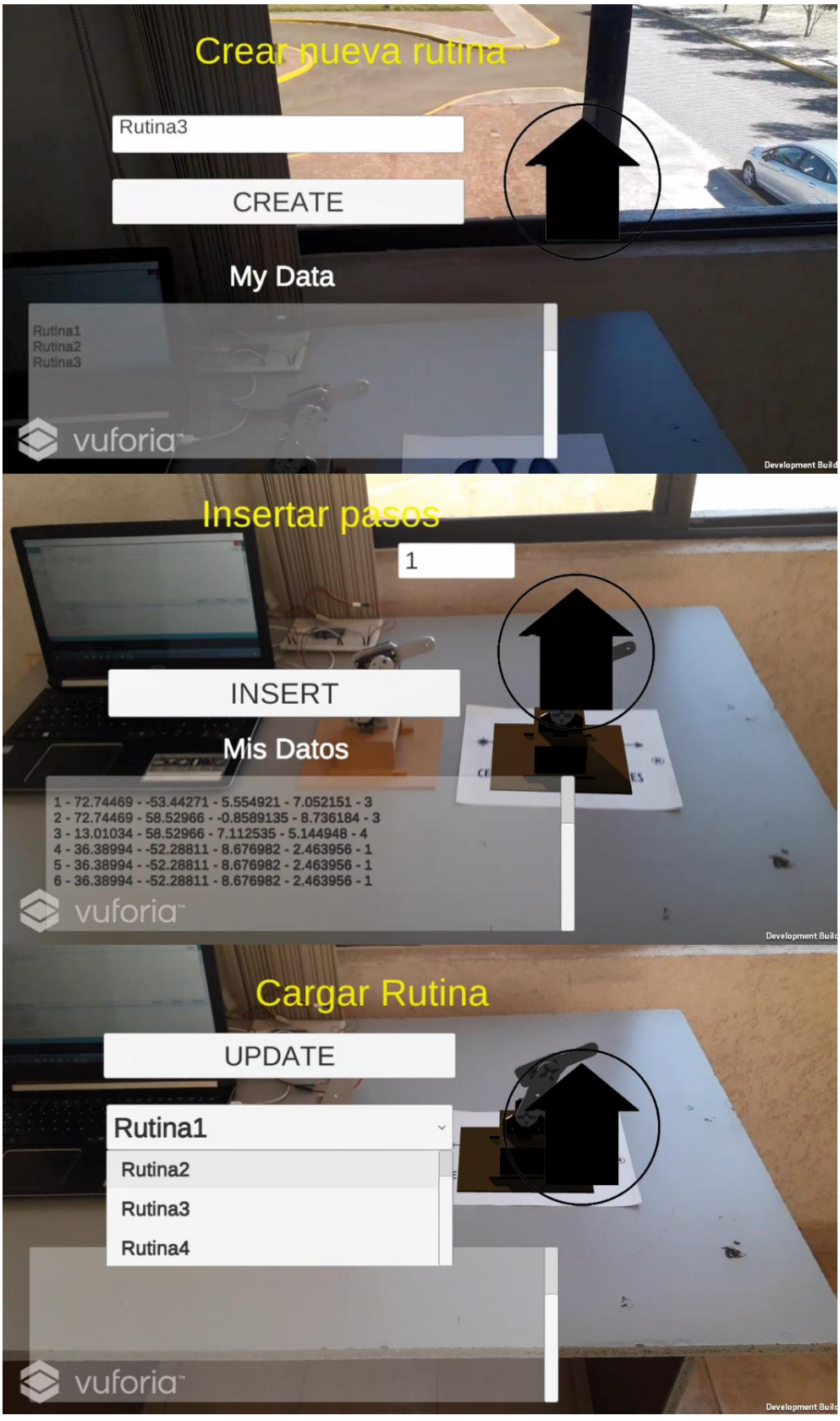


Imagen 97. Capturas tomas desde la app (2) (Fuente: Propia)

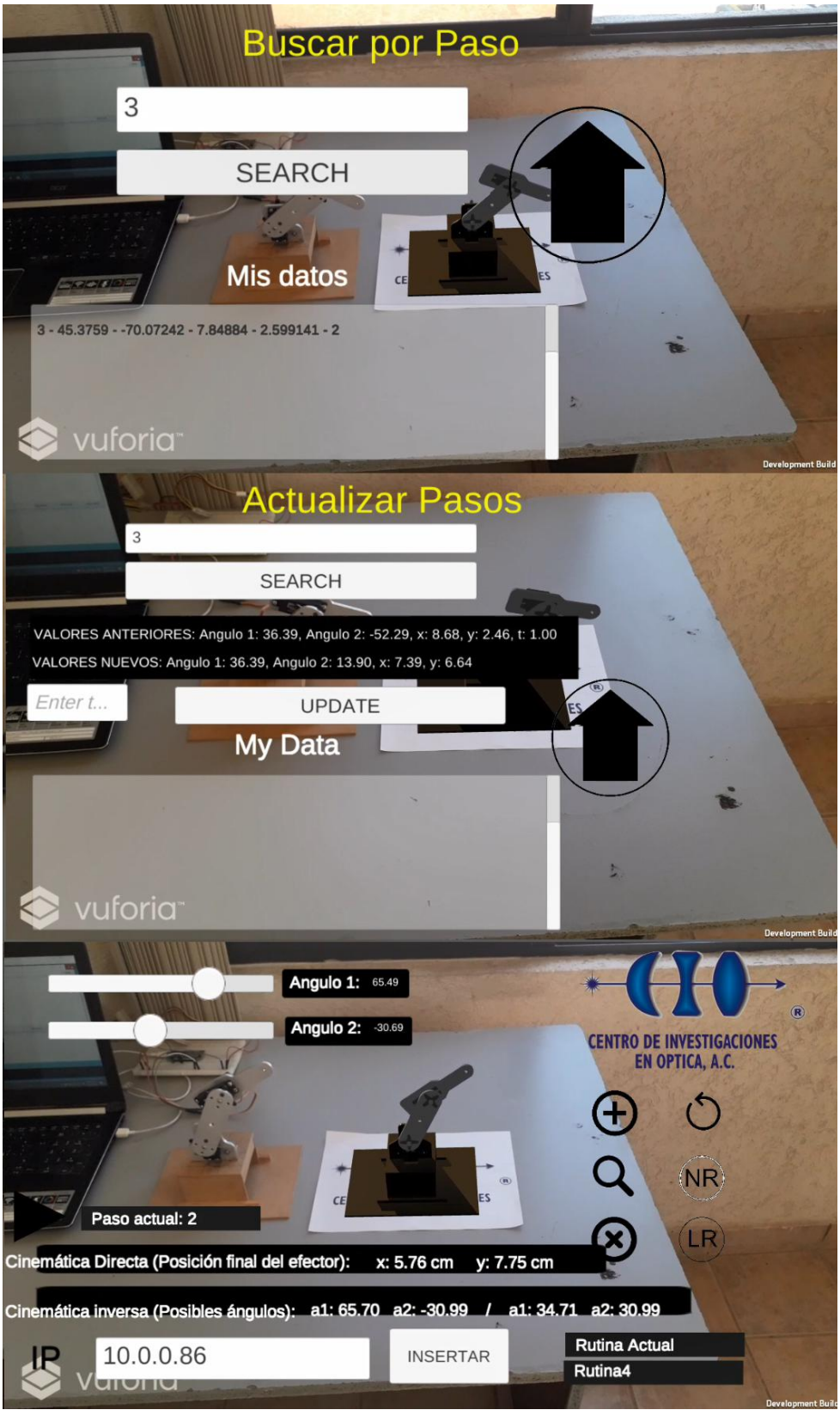


Imagen 98. Capturas tomas desde la app (3) (Fuente: Propia)

Otro de los productos resultado del desarrollo del proyecto es el robot mismo. El robot cuenta con dos servomotores que son controlados por una placa ESP8266 que funciona para la comunicación vía WiFi entre la aplicación y el robot, y a la vez como microcontrolador que nos permite establecer los pulsos PWM de los servomotores, cuyo valor nos determina el ángulo de las articulaciones del robot. El robot es alimentado por su propia fuente de voltaje.

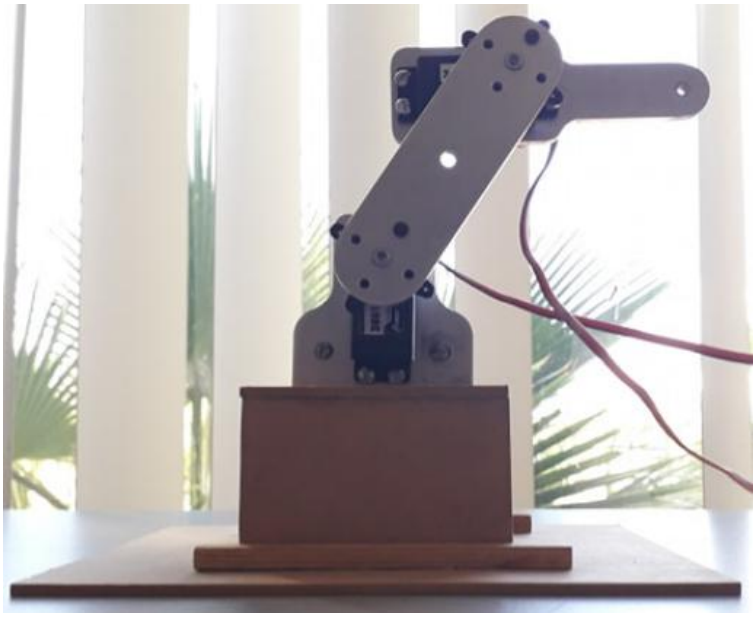


Imagen 99. Robot planar de 2 grados de libertad usado en el proyecto (Fuente: Propia)

Otro producto final del desarrollo del proyecto son todos los códigos que se desarrollaron para la realización de la aplicación, lo que incluye la programación en C#, el programa de Arduino usado para la comunicación vía wifi, así como la interfaz desarrollada en Unity. También se incluye el modelo 3D del robot y la aplicación en formato APK.

Todo el código, modelos 3D, programas, bases de datos y demás archivos desarrollados a lo largo del desarrollo del proyecto se encuentran incluidos en el siguiente enlace de descarga para futuras investigaciones o proyectos:

https://mega.nz/#!ccgGAQZb!nPnFlBh--2iNCSucOHpMJl6wIa-9hIQ6UcOTg_sCQ9E

Conclusiónes y recomendaciones

Al final del desarrollo de éste proyecto se logró desarrollar una aplicación que cuenta con todas las características propuestas al inicio del proyecto. También se construyó el robot en el cual está basado el gemelo digital de la aplicación y finalmente se implementó el sistema de capacitación a la materia de mecatrónica en el CIO. Los alumnos mostraron gran satisfacción con el uso de la herramienta e indicaron que la aplicación les permitió visualizar y comprender mejor los conceptos vistos en la clase. Para lograr todo ésto se tuvo que diseñar el robot físico en un software de diseño asistido por computadora para después ser llevado a la realidad construyéndolo en el taller del CIO. Después se desarrolló la parte electrónica que me permitiera controlar vía WiFi y haciendo uso de Arduino nuestro robot. Se aplicaron muchas herramientas de Unity y se usaron los SDK correctos para desarrollar la aplicación de Android con todas las características solicitadas y finalmente se implementó la herramienta al sistema de aprendizaje y los alumnos interactuaron con ella.

Se lograron todos los objetivos propuestos cuando se empezó a desarrollar el proyecto y a lo largo del desarrollo se generó mucho conocimiento en las areas como electrónica, programación, robótica y modelado 3D, y además se desarrollaron nuevas habilidades y aptitudes como investigador.

Se recomienda a futuras investigaciones expandir el proyecto en dimensiones (incluir mayor número de grados de libertad, incluir articulaciones prismáticas, usar distintos modelos de robot, usar diferentes sistemas de comunicación) y también incluir nuevas características (cálculos de dinámica, sistema de control PID para las articulaciones, diferentes tipos de rutinas), además de que se puede desarrollar la aplicación para plataformas distintas a Android, usando programas diferentes a Unity, o utilizando diferentes sistemas de realidad aumentada. Se invita a seguir explorando el tema, debido a que es un área de investigación con mucho potencial.

Competencias desarrolladas y aplicadas

COMPETENCIAS TÉCNICAS

- Uso de software de diseño asistido por computadora para modelado 3D.

- Construcción de piezas mecánicas previamente diseñadas.

- Conexión electrónica de microcontroladores y sus actuadores.

- Programación en Arduino.

- Dominio de Unity para desarrollo de aplicaciones móviles.

- Desarrollo de aplicaciones de Realidad Aumentada.

- Uso de tarjetas WiFi para control y monitoreo remoto.

- Manejo de bases de datos SQLite para desarrollo de aplicaciones.

- Programación en C#.

- Búsqueda bibliográfica y redacción de textos de investigación.

COMPETENCIAS PROFESIONALES

- Análisis de problemas.

- Capacidad crítica.

- Creatividad.

- Planificación y organización.

- Flexibilidad.

- Colaboración con usuarios y asesor.

Referencias bibliográficas

Adams, M. (2005). *The 10 most important emerging technologies for humanity*. Truth Publishing International. Recuperado a partir de http://www.naturalnews.com/SpecialReports/EmergingTechnologies.pdf

Aguilar, V. (2011). *Diseño y construcción de un robot de 6 grados de libertad con fines educativos para aplicaciones en nivel medio superior*. México D.F.: ESIME.

Andujar, J. M., Mejías, A., & Marquez, M. A. (2011). *Augmented reality for the improvement of remote laboratories: an augmented remote laboratory*. Education, IEEE Transactions on, 54(3), 492–500.

Arias Montiel, M. (2019). *CONSTRUCCIÓN Y ANÁLISIS CINEMÁTICO DE UN PROTOTIPO DE ROBOT PARALELO UPUR DE SEIS GRADOS DE LIBERTAD*. Huajapan de León, Oaxaca: UTM.

Callejas, V. (2006). *Control adaptable para robots manipuladores*. Pachuca de Soto, Hidalgo: UAEH.

Cox, D., Little, J., and O'Shea, D. (2008) *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, Spinger Undergraduate Texts in Mathematics, Springer, New York.

Dede, C. (2009). *Immersive Interfaces for Engagement and Learning*. Science, 323(5910), 66-69.

González, Ó. (2013, marzo). *Educación aumentada*. Boletín del Centro del Conocimiento.

Guaitara, A. (2014). *APLICACIÓN DE REALIDAD AUMENTADA ORIENTADA A LA PUBLICIDAD DE ALTO IMPACTO EN LA EMPRESA VECOVA CÍA. LTDA*.. Ambato, Ecuador: UNIANDES

Huang, H., Schmidt, M., & Gartner, G. (2012). *Spatial Knowledge Acquisition with Mobile Maps, Augmented Reality and Voice in the Context of GPS-based Pedestrian Navigation: Results from a Field Test. Cartography and Geographic Information Science*, 39(2), 107-116.

Landa, H. (2015). *Problems Analysis and Solutions for the Establishment of Augmented Reality Technology in Maintenance and Education*. Tampere University of Technology, Tampere, Finlandia.

Ramírez, J.; Rubiano, A. (2012). *Modelamiento matemático de la cinemática directa e inversa de un robot manipulador de tres grados de libertad*. Ingeniería Solidaria, Vol. 8, 47.

Rodríguez, E.; Mckinley, J.; Ramírez, J. (2017). *Control por par calculado de un robot paralelo planar 2-RR.* Prospect, Vol. 15, 87.

Shelton, B. E., & Stevens, R. R. (2004). *Using Coordination Classes to Interpret Conceptual Change in Astronomical Thinking. En Proceedings of the 6th International Conference on Learning Sciences* (pp. 634–634). Santa Monica, California: International Society of the Learning Sciences.

Solorzano, J. (2012). *DESARROLLO DE PRÁCTICAS AVANZADAS PARA EL LABORATORIO . DE DISEÑO Y MANUFACTURA ASISTIDO POR COMPUTADORA.* México D.F.: UNAM.

Spong, M. (2005). *Robot Modeling and Control*. United States of America: Wiley.

Squire, K., & Klopfer, E. (2007). *Augmented Reality Simulations on Handheld Computers. Journal of the Learning Sciences*, 16(3), 371-413.

Toapanta, E. (2015). *DISEÑO E IMPLEMENTACIÓN DE UN LABORATORIO DE MODELADO Y DISEÑO MECÁNICO ASISTIDO POR COMPUTADORA MEDIANTE SOFTWARE CAD 3D-2D Y SOLIDWORKS*. La Mana, Ecuador: UNIVERSIDAD TÉCNICA DE COTOPAXI.

Torres, D. (2013). *El papel de la realidad aumentada en el ámbito artístico - cultural.* Granada, España: Universidad de Granada.

Vargas, I. (2016). *Diseño y desarrollo de una aplicación para dispositivos moviles de Realidad Aumentada*. México D.F.: IPN.