Herramienta didáctica de Realidad Aumentada con Python y OpenCV, para la visualización y despiece en 3D de un brazo robótico

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Autor
Sebastián Fernando Ruiz León
Gregorio Esteban Suarez Mahecha

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
Facultad Tecnológica
Ingeniería en Control y Automatización

Bogotá D.C. Junio, 2021

Herramienta didáctica de Realidad Aumentada con Python y OpenCV, para la visualización y despiece en 3D de un brazo robótico

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Autor
Sebastián Fernando Ruiz León- 20181383006
sfruizl@correo.udistrital.edu.co

Gregorio Esteban Suarez Mahecha-20182383001
gsuarezm@correo.udistrital.edu.co

Modalidad
Monografía

Presentado para optar al título de: Ingeniero en Control y Automatización

Director
Frank Giraldo Ramos

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
Facultad Tecnológica
Ingeniería en Control y Automatización
Bogotá D.C. Junio, 2021

Dedicatoria

Los autores de este proyecto agradecen a Oscar Gabriel Espejo Mojica docente de la Universidad Distrital Francisco José de Caldas y Frank Giraldo Ramos como tutor, por toda la colaboración, el tiempo y el espacio que nos brindaron durante la formulación y desarrollo de todo este proyecto.

Índice

Índice de Figuras

Índice de Tablas

Índice de Anexos

Lista de Abreviaturas y Siglas

Sigla/Abreviatura	Significado
AR	Augmented reality (Realidad aumentada)
OVA	Objeto virtual de aprendizaje
TIC	Tecnologías de la información y la comunicación

Resumen

La evolución de la tecnología y su presencia en nuestro entorno, afecta de forma directa diferentes aspectos, ámbitos y sectores, uno de ellos, la educación. Resulta entonces imprescindible, aprovechar el acceso que se tiene a ellas e implementarlas, en búsqueda de aportes y beneficios que promuevan el conocimiento y la interacción en los procesos educativos. Un tipo de estas tecnologías son las denominadas multiexperiencia, categorizada como una de las 10 principales tendencias tecnológicas en el mundo, la cual hace referencia a la realidad virtual (VR), la realidad mixta (MR) y la realidad aumentada (AR), (Panetta, 2020)

La Implementación de este tipo de herramientas multiexperiencia, en entornos de enseñanza y aprendizaje, suponen un material de apoyo didáctico en las clases presenciales y de presencialidad remota, presentando esta última un auge debido a la pandemia causada por el Covid-19, la cual implica un acceso limitado a instalaciones educativas, y supone el planteamiento de nuevas metodologías para clases teórico-practicas, al no tener en físico diferentes elementos o máquinas, como lo es un brazo robótico industrial para carreras enfocadas a la mecánica, electrónica, mecatrónica y control. En estas áreas, es importante que los dispositivos de estudio analizados, cuenten con estructuras complejas y gran variedad de componentes, pues se pueden plantear diferentes ejercicios en base a ello.

Como solución se desarrolló una herramienta de realidad aumentada basada en el lenguaje de programación Python, usando como librerías principales OpenCV y OpenGL, capaz de visualizar elementos que componen un brazo robótico. Para la selección de los modelos fue necesario determinar ciertos parámetros y requerimientos que el Brazo Robótico debía tener para ser implementado en la herramienta, pensando a su vez en su aplicabilidad en entornos de enseñanza/aprendizaje y el desarrollo de actividades.

En base a esto se obtuvo un aplicativo portátil de realidad aumentada que permite visualizar el despiece del brazo robótico KUKA KR360-2, el cual cuenta además con una interfaz de usuario, en la que se hallan diversas funciones enfocadas al desarrollo de actividades como lo es el apartado de animación, la información de elementos visualizados, etc. Por último, al aplicar la herramienta en un entorno de practica de laboratorio, los estudiantes dieron a conocer, que el uso de estas tecnologías, debe tener más presencia en diferentes sectores académicos.

Palabras claves: Brazo robótico, despiece, educación, realidad aumentada (AR), tecnología multiexperiencia.

1. Introducción

Las tecnologías de la información y la comunicación (TIC), han tomado relevancia en el diario vivir debido a sus múltiples ventajas y facilidad de incorporación. Es importante entonces, incluir las nuevas tecnologías en los ámbitos académicos, para explotar los beneficios que puedan brindar y con ello generar nuevas metodologías en las clases favoreciendo a profesores y estudiantes; es por ello, que se debe aprovechar estas nuevas alternativas como lo es la realidad aumentada, teniendo en cuenta la cantidad de recursos tecnológicos con los que se cuentan hoy en día.

La realidad aumentada ofrece un sinfín de nuevas posibilidades de interacción y visualización que combina, de manera funcional, la virtualidad con la realidad misma. (Alvarez et al., 2017). Es por lo anterior, que su implementación en diversos campos del conocimiento favorece el aprendizaje de una manera interactiva y enriquecedora.

Este proyecto tiene como objetivo el desarrollo e implementación de una herramienta didáctica de realidad aumentada con el lenguaje de programación Python. Con el fin de introducir esta nueva tecnología en entornos académicos de la Universidad Distrital Francisco José de Caldas, por medio de la cual se pretende visualizar el despiece en 3D de un brazo robótico, su estructura mecánica y el funcionamiento de los grados de libertad.

A lo largo del documento se encontrarán antecedentes de aplicaciones o herramientas de realidad aumentada en la educación, un marco de referencia sobre los temas que se abarcan en la elaboración de la herramienta, se especifica la metodología que se llevó a cabo durante el desarrollo, donde se observa la implementación de un programa básico de realidad aumentada usando las librerías openCV y OpenGL, se determinan y evalúan los requerimientos y parámetros del brazo robótico, se muestra el proceso de acondicionamiento de los modelos para su carga en la herramienta, el diseño de la interfaz y la creación de funciones, con el fin de darle aplicabilidad y poder generar en base a ello diversas actividades. Al finalizar se observa la validación de la herramienta en la clase de sistemas mecatrónicos I, con el fin de probar el software y observar la percepción de los estudiantes a su interacción.

2. Planteamiento del problema

En la actualidad en Colombia y en la Universidad Distrital, la realidad aumentada es un tema que aún no ha presentado demasiados avances con respecto a su investigación, aplicabilidad educativa y el estudio de los posibles beneficios que podría aportar en los procesos de enseñanza/aprendizaje (Cupitra et al., 2018), esto, pese a que su desarrollo viene cobrando cada vez más fuerza a nivel internacional y mucho más desde el auge de la presencialidad remota en la educación causada por la pandemia, la cual ha dejado ver las barreras que deben afrontar los educadores y estudiantes, un caso específico sería el hecho de que estos, no tienen acceso al modelo ya sea estructural o físico de elementos, máquinas o dispositivos necesarios en carreras como Ingeniería Electrónica y Mecatrónica, como por ejemplo un brazo robótico.

Además, el realizar un análisis poco profundo de estas máquinas, puede generar problemas al momento de identificar piezas, conocer la cantidad de elementos que componen un dispositivo e inclusive pueden entorpecer la metodología a seguir, al realizar el ensamble, reparación e instalación de estos. En general se destina el 50% del tiempo en la práctica a ajustes, dado que no se sigue una guía o no se atiende de forma adecuada (López, 1994). para esto una herramienta como la RA que presenta la información ya sea por medio de gráficos en 3d, sistemas de posicionamiento global (GPS), manipulación de datos mediante códigos gráficos y otras alternativas, hacen de este tipo de tecnologías, una opción potencial para el sector educativo, en entornos presenciales y remotos (Jaguandoy y Puchana, 2014).

3. Justificación

Actualmente en las universidades e instituciones educativas es necesario hacer uso alternado de los laboratorios, equipos y maquinarias indispensables para la formación del estudiante, pues muchas carreras enfocan su pedagogía en el estudio y análisis de estos elementos, los cuales a su vez están compuestos por una gran variedad de piezas y componentes complejos ya sea por su forma, funcionamiento, diseño, o inclusive por poseer una estructura que dificulte el acceso para el usuario, para estos casos existen diversas herramientas de realidad aumentada, Objetos Virtuales de aprendizaje y estrategias didácticas apoyadas en ilustraciones 3D que se pueden utilizar en el aula de clase e inclusive en la educación con presencialidad remota, permitiendo estudiar el dispositivo interactivamente, favoreciendo asi la participación, curiosidad y deseo de aprender acerca de estas tecnologías.

4. Objetivos

4.1. Objetivo General

- Desarrollar una herramienta didáctica de Realidad Aumentada en Python utilizando la librería OpenCV, para la visualización y despiece en 3D de un brazo robótico

4.2. Objetivos específicos

- Establecer los requerimientos y parámetros del brazo robótico a desarrollar, que permita su instrucción en un entorno de enseñanza/aprendizaje
- Aplicar la herramienta de realidad aumentada en un entorno académico de presencialidad remota del laboratorio especializado de ingeniería en control.
- Realizar una guía práctica de laboratorio para el manejo de la herramienta de realidad aumentada en la materia de sistemas mecatrónicos I.

5. Marco de referencia

5.1. Antecedentes

La realidad aumentada ha tenido un auge muy significativo en los últimos años, en especial en ámbitos de entretenimiento, y teniendo en cuenta las tecnologías que se poseen hoy en día la realidad aumentada tiene un gran potencial que se ha venido explotando. Su implementación en la educación es una ayuda en la enseñanza y aprendizaje, que de una manera didáctica e interactiva genera una herramienta para fortalecer los conocimientos teóricos.

5.1.1 Prototipo de realidad aumentada para el proceso de alfabetización en la población de adultos mayores.

En este proyecto se desarrolla una aplicación móvil de realidad aumentada, el objetivo se enfatiza en incrementar los niveles educativos de la población adulta y eliminar la brecha existente entre esta comunidad; además de la enseñanza respecto al manejo de las nuevas tecnologías. Para su ejecución se decidió trabajar en dispositivos Android que permitan, a través de la cámara, prestar atención a la realidad y tomar como punto de referencia la funcionalidad del acelerómetro/giroscopio del dispositivo para generar una visión tridimensional del alfabeto.

Esta fue implementada en la Universidad Distrital, en la elaboración de la aplicación se usó Unity como entorno de desarrollo integrado para la interfaz, este trabaja con la librería Vuforia que permite la captura de la cámara y el giroscopio y se usó OpenGL como motor gráfico. El resultado se observa en la figura 1, con ayuda de la realidad aumenta se muestran las letras o números y se asocian con imágenes para que el aprendizaje sea más fácil y amigable. (Pertuz y Rojas, 2018)

Figura 1. Interfaz gráfica aplicación ALFABETISOFT, visualización de letras y números en 3D. (Pertuz y Rojas, 2018)

5.1.2 Realidad aumentada como estrategia didáctica en curso de ciencias naturales de estudiantes de quinto grado de primaria de la institución educativa Campo Valdés.

En esta tesis realizada en la Universidad de Medellín, se pretende tratar el problema de atención en la formación básica, ya que los estudiantes se dispersan con facilidad en las clases, no alcanzan niveles de concentración prolongados, no realizan las actividades propuestas y en muchos casos se percibe que asisten a las instituciones educativas por que los obligan sus padres (Romero Carranza & Rubiano González).

La utilización de nuevas tecnologías en los procesos de aprendizaje, permite aprovechar el hecho de que los estudiantes de hoy en día gustan y disfrutan la tecnología. Este proyecto pretende explotar esto con la ayuda de la realidad aumentada enfocada en las ciencias naturales (figura 2) en un curso de quinto de primaria.

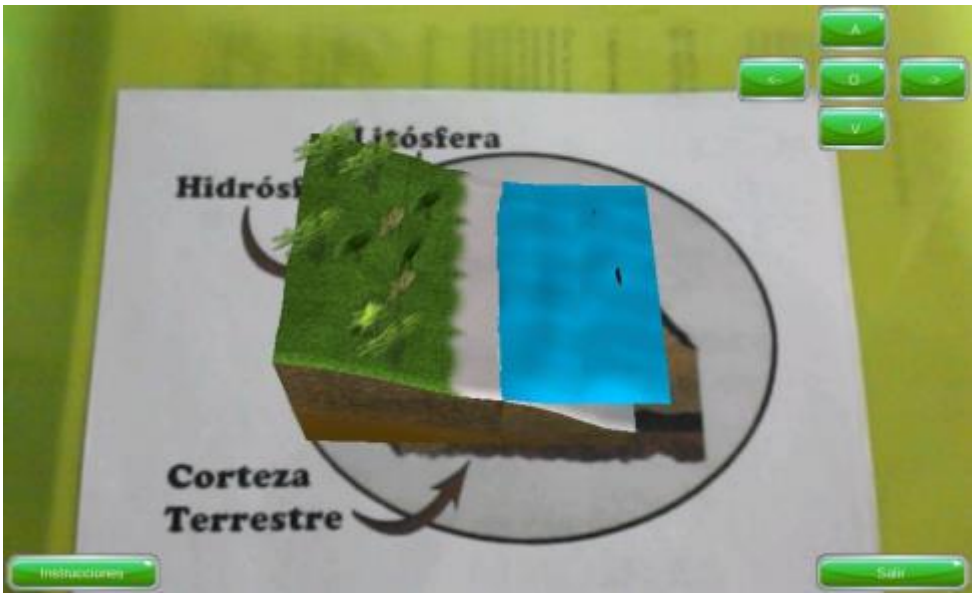


Figura 2. Aplicación de AR ciencias básicas. (Buenaventura, 2014)

La aplicación se desarrolló para Tablet después de un análisis de los recursos del colegio. Esta se implementó obteniendo resultados positivos, al llamar la atención de los niños como se observa en la figura 3. Generando interacción y dinámica en la clase; de la misma manera enseñando a los docentes a usar estas herramientas para generar nuevas metodologías en las clases y fortalecer el aprendizaje. (Buenaventura, 2014)

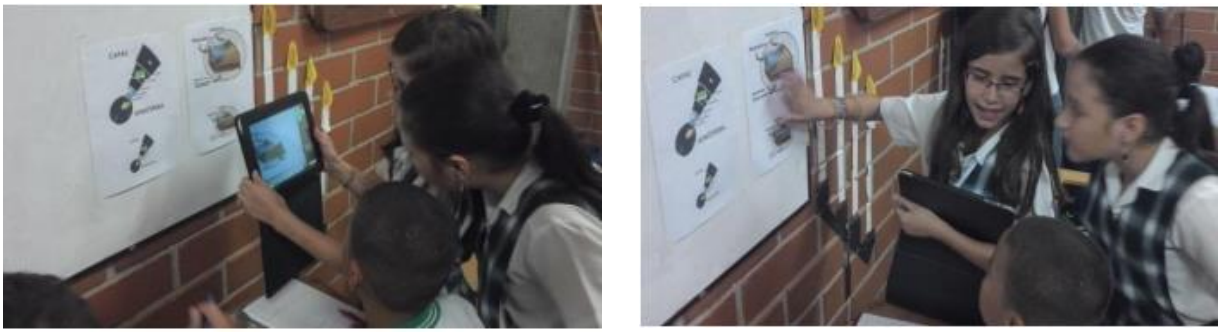


Figura 3. Estudiantes probando aplicación de realidad aumentada en ciencias naturales. (Buenaventura, 2014)

5.1.3 Realidad Aumentada como Apoyo a la Formación de Ingenieros Industriales.

Este trabajo tiene como objetivo desarrollar una experiencia de realidad aumentada durante la formación de ingenieros civiles en el campo de la mecánica de fluidos. Analiza el impacto de las nuevas tecnologías en el campo de la educación superior y cómo las características de la tecnología de realidad aumentada pueden traer mejores resultados en términos de aprendizaje de una manera efectiva y significativa.

El proyecto se desarrolló en la plataforma Unity, usando la librería Vuforia, funciona basado en un código AR. Este se implementó en el ejercicio de la clase como se observa en la figura 4. La apreciación final es que esta tecnología permite a los estudiantes tener una comprensión visual más adecuada de las preguntas planteadas, debido a que estos ejercicios se presentan de manera tradicional, por lo que muchos aspectos no se pueden visualizar fácilmente. (Alvarez et al., 2017)

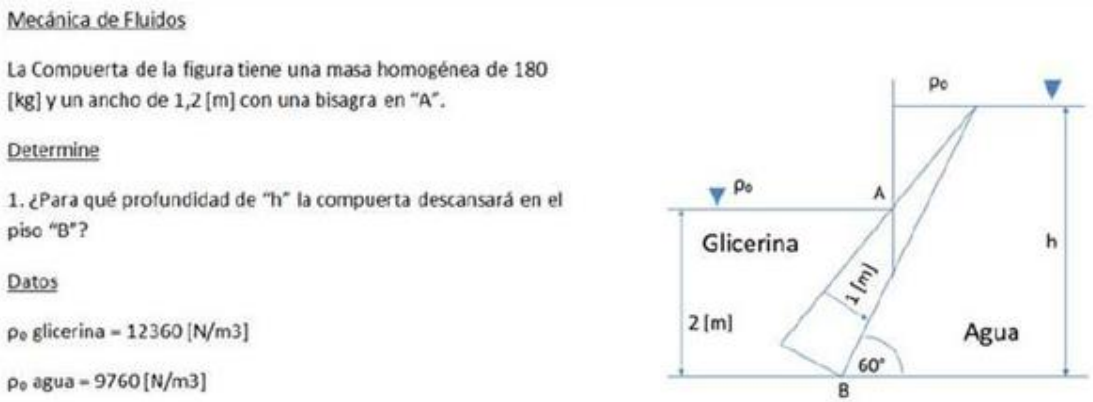


Figura 4. Ejercicio de mecánica de fluidos. (Alvarez et al., 2017)

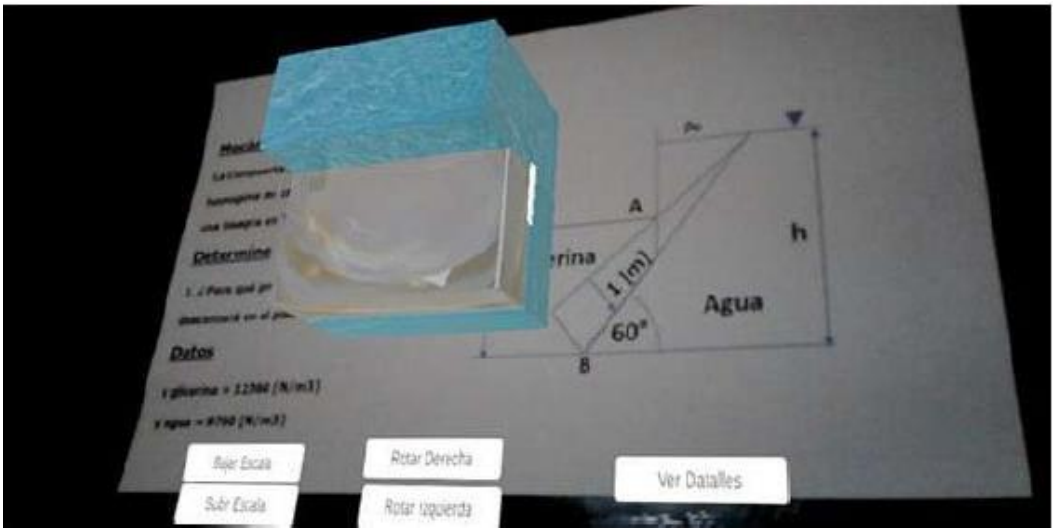


Figura 5. Ejercicio de mecánica de fluidos con realidad aumentada. (Alvarez et al., 2017)

5.1.4 Desarrollo de una aplicación de realidad aumentada como herramienta de capacitación, diseño y planeación en una línea de extrusión de tubería PVC, con base en la librería ARtoolkit.

Este proyecto se enfoca en el problema de la falta de capacitación del personal en las empresas; lo cual genera erros de manejo de la maquinaria, carencia de conocimiento de los componentes, fallas en la producción. Por ende, se desarrolla una herramienta de realidad aumentada enfocada en la industria de tuberías PVC, específicamente en la práctica de extrusión de tubos PVC.

La herramienta se basa en el funcionamiento de la librería ARtoolkit, la cual permite el funcionamiento básico de la realidad aumentada, capturar la cámara y sobrepone imágenes virtuales en el mundo real sobre marcadores. En el proceso de la herramienta se trabaja con el programa Blender que permite tartar las texturas de las imágenes 3D para mejorar problemas de luces y sombras para la carga en la aplicación como se observa en la figura 6.

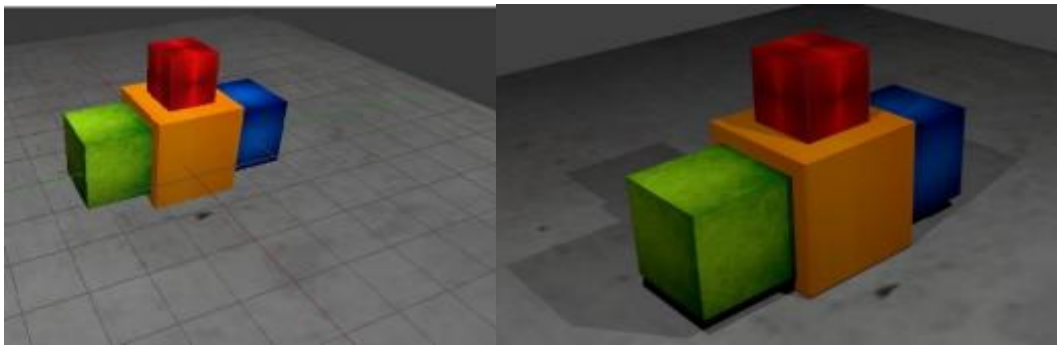


Figura 6. Comparación entre imagen 3D e imagen 3D renderizada. (Uintaco et al., 2016)

La experiencia que genera el resultado de la herramienta es generar la visión de un proceso de extrusión, permitiendo interactuar con los distintos marcadores de las partes necesarias para el procedimiento, logrando fortalecer el conocimiento del proceso, sus componentes. (Uintaco et al., 2016)

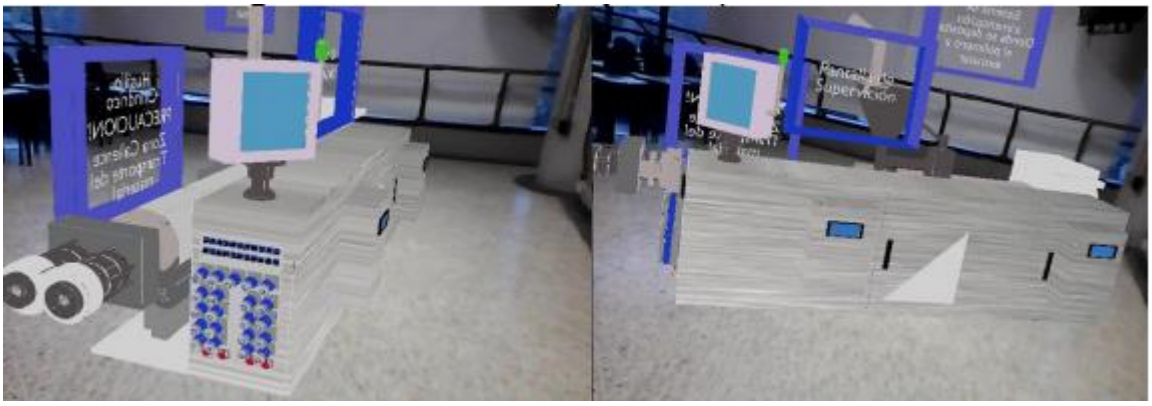


Figura 7. Proyección 1 realidad aumentada extrusora. (Uintaco et al., 2016)

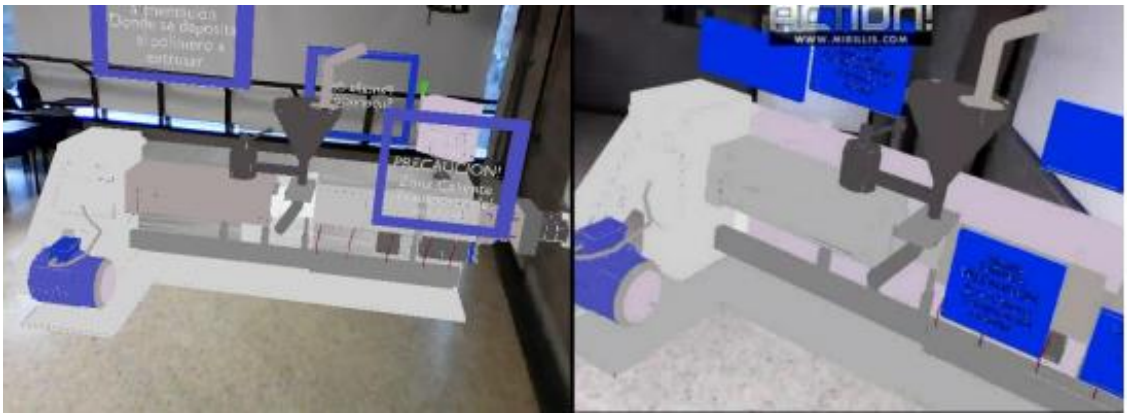


Figura 8. Proyección 2 realidad aumentada extrusora. (Uintaco et al., 2016)

5.2. Marco teórico

5.2.1 Realidad Aumentada.

La realidad aumentada (AR) es una variación de entornos virtuales (VE), o realidad virtual, como se le llama más comúnmente. Las tecnologías VE sumergen completamente a un usuario dentro de un entorno sintético. Mientras está inmerso, el usuario no puede ver el mundo real a su alrededor. Por el contrario, AR permite al usuario ver el mundo real, con objetos virtuales superpuestos o compuestos con el mundo real. Por lo tanto, AR complementa la realidad, en lugar de reemplazarla por completo (Azuma, 1997) lo cual permite aprovechar la información adicional para potenciar el conocimiento sobre los objetos tangibles (Melo, 2018). Idealmente, le parecería al usuario que los objetos virtuales y reales coexisten en el mismo espacio. La Figura 14 muestra un ejemplo de cómo se vería esto. Muestra un escritorio real con un teléfono real. Dentro de esta sala también hay una lámpara virtual y dos sillas virtuales. Tenga en cuenta que los objetos se combinan en 3-D, de modo que la lámpara virtual cubre la mesa real y la mesa real cubre partes de las dos sillas virtuales. AR puede considerarse como el "punto medio" entre VE (completamente sintético) y telepresencia (completamente real) (Azuma, 1997).

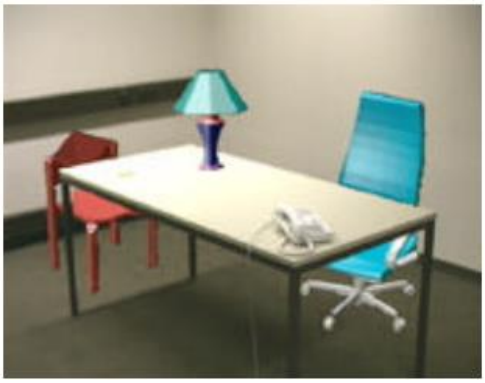


Figura 9. Escritorio real con lámpara y dos sillas virtuales. (Azuma, 1997)

Para evitar limitar la Realidad Aumentada a tecnologías específicas, se define AR como sistemas que tienen las siguientes tres características (Azuma, 1997):

- Combinación de lo real y virtual.
- Interacción en tiempo real.
- Modelos en 3D.

Para apreciar la transición entre ambas tecnologías, es muy útil mostrar la figura de Paul Milgram y Fumio Kishino que explica el concepto “Reality-Virtuality Coninuum”. Este concepto muestra la transición entre el mundo real y el mundo virtual y cómo, el espacio dominado por una realidad mixta donde están ambos presentes (Lacueva et al., 2015).

Figura 10. Reality-Virtuality Coninuum. (Lacueva et al., 2015).

Un sistema de Realidad Aumentada consiste en tres simples fases. Una primera fase de reconocimiento, una segunda de seguimiento y, por último, una última fase de mezclar/alinear la información del mundo o del objeto virtual debe estar correctamente alineada con la imagen del mundo real (Lacueva et al., 2015). teniendo en cuenta esto a continuación se definen los elementos básicos que componen la realidad aumentada (Silva y Olivera, 2003):

- Generador de escena.

El generador de escena es el dispositivo o software responsable de representar la escena (Silva y Olivera, 2003), que están viendo los usuarios. Basta para ello una sencilla cámara presente en el teléfono, tableta o gafas de RA (Lacueva et al., 2015).

- Elemento Visualizador.

Se necesita un elemento sobre el que proyectar la mezcla de las imágenes reales con las imágenes sintetizadas. Este elemento puede ser una pantalla de ordenador, de móvil o el display de unas gafas de realidad aumentada (Lacueva et al., 2015).

- Unidad de Procesamiento

Un dispositivo de procesamiento para interpretar la información del mundo real y generar la información virtual para combinarla (Rigueros, 2017).

- Trigger.

Un disparador, conocido también como activador de Realidad Aumentada, dentro de los cuales a continuación se desglosan los más comúnmente usados: (Blázquez, 2017)

- Imagen.
- Entorno Físico.
- Marcador.
- Objeto.
- Código QR.

5.2.2 Educación y realidad aumentada

La realidad aumentada ha sido aplicada en numerosos proyectos educativos, tales como el proyecto Magic Book del grupo HIT de Nueva Zelanda, en el cual los usuarios ven las páginas de un libro real a través de una pantalla de mano, lo que se puede ver en la pantalla es contenido virtual superpuesto sobre las páginas reales (Buenaventura, 2014). Ver figura 11.

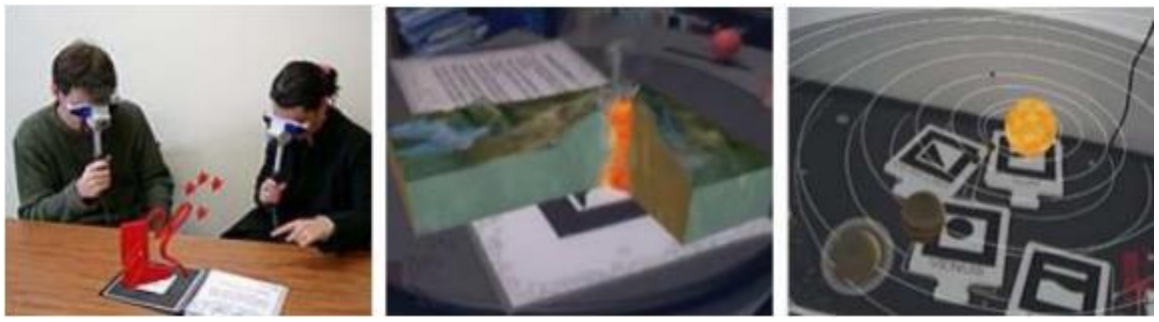


Figura 11. El magic book y su aplicación en el área de ciencias sociales. (Buenaventura, 2014)

También se destaca el proyecto CREATE, en el cual los usuarios reconstruyen un sitio arqueológico, comenzando con un modelo foto-realista del sitio como lo es en la actualidad y añadiendo elementos, pieza por pieza, con el fin de explorar, hacer juicios, examinar escenarios alternativos o experimentar con diferentes posibilidades y completar la reconstrucción final (Buenaventura, 2014). Ver figura 12.

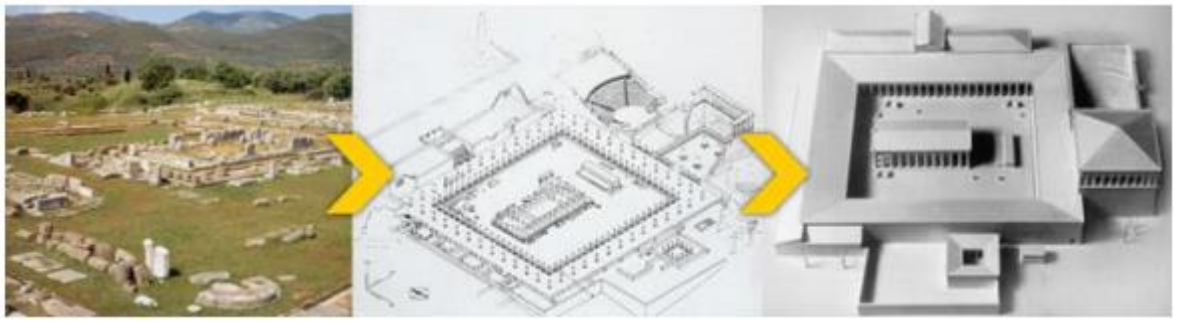


Figura 12. Fases de construcción en el proyecto CREATE. (Buenaventura, 2014)

Otro ejemplo es el proyecto adelantado por la Universidad de Sussex, en el cual se utiliza la realidad aumentada y las tecnologías Web para apoyar la enseñanza de la Ingeniería (Buenaventura, 2014). Ver figura 13.

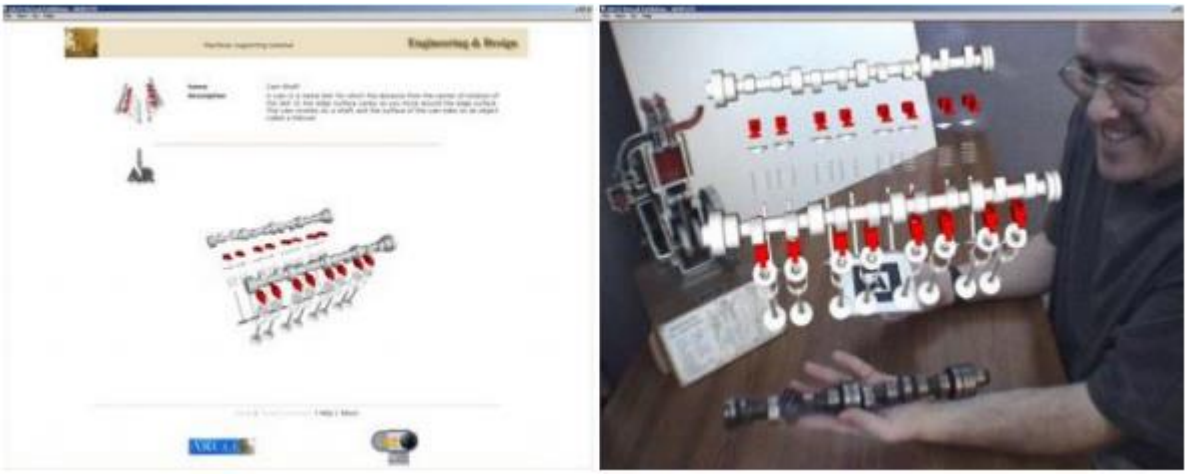


Figura 13. Visualización AR de un eje de levas. (Buenaventura, 2014)

En Colombia, podemos encontrar el proyecto adelantado en la Universidad EAFIT llamado: La realidad aumentada: un espacio para la comprensión de conceptos del cálculo en varias variables, el cual permite la visualización de conceptos matemáticos a partir de la creación de un objeto virtual, que se puede comparar con objetos reales, potenciando las posibilidades de comprensión de los conceptos matemáticos estudiados (Buenaventura, 2014). Ver figura 14.

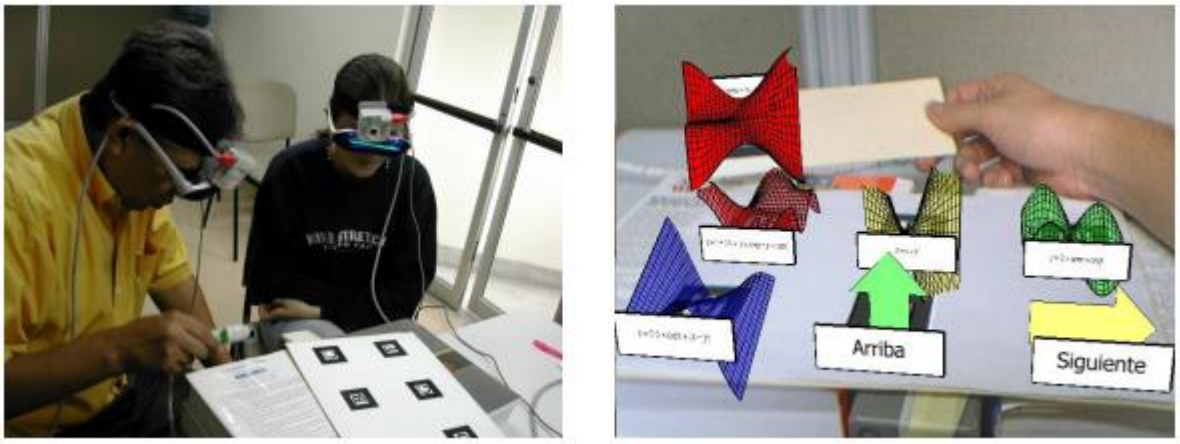


Figura 14. Realidad Aumentada para el estudio de conceptos del cálculo en varias variables aplicados a superficies de la forma z = f (x, y). (Buenaventura, 2014)

5.2.3 Python

El software libre se ha convertido en uno de los movimientos tecnológicos de mayor auge en el siglo XXI. Para su desarrollo ha sido necesario contar con un grupo de herramientas que hagan óptima su utilización y sean fáciles de aprender. Python es un lenguaje de programación que

cumple con lo planteado y se viene perfilando como una opción recomendada para el desarrollo de software libre**.**

Python fue creado por Guido van Rossum, un programador holandés a finales de los 80 y principio de los 90 cuando se encontraba trabajando en el sistema operativo Amoeba. Primariamente se concibe para manejar excepciones y tener interfaces con Amoeba como sucesor del lenguaje ABC. (Challenger, 2014)

Las características principales por las que destaca python son:

- Sintaxis simple.
- Alta legibilidad (sangrado obligatorio).
- Entorno amigable de desarrollo (intérprete interactivo).
- Abstracciones de más alto nivel (mayor nivel de expresividad).
- Potente librería estándar y gran cantidad de módulos de terceros (actualmente son más de 100.000).
- Multi-paradigma (imperativo, POO y funcional).
- Disponibilidad de recursos educativos abiertos.
- Software libre y comunidad entusiasta.

No en vano es el lenguaje más popular en los cursos introductorios de programación de las universidades en Estados Unidos. (GARCÍA, 2017)

5.2.4 OpenCV y OpenGL.

OpenCV:

OpenCV es una biblioteca de visión por computadora de código abierto (Kaehler y Bradski, 2008) que incluye varios cientos de algoritmos de visión por computadora. Una de las principales ventajas de OpenCV es que está altamente optimizado y disponible en casi todas las plataformas (Joshi y Escrivá, 2016), ya que posee un desarrollo activo en interfaces para Python, Ruby, Matlab y otros (Kaehler y Bradski, 2008).

Fue diseñado para la eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real. ya que está escrito en C optimizado y puede aprovechar los procesadores multinúcleo (Kaehler y Bradski, 2008). Estando dirigida fundamentalmente a la visión por computador en tiempo real, entre sus muchas áreas de aplicación destacaría las siguientes (Arévalo et al., 2002):

- Interacción hombre-máquina.
- Segmentación y reconocimiento de objetos.
- Reconocimiento de gestos.
- Seguimiento del movimiento.
- Estructura del movimiento.
- Robots móviles.

OpenCV está ampliamente estructurado en cinco componentes principales, cuatro de los cuales se muestran en la Figura 15 (Kaehler y Bradski, 2008).

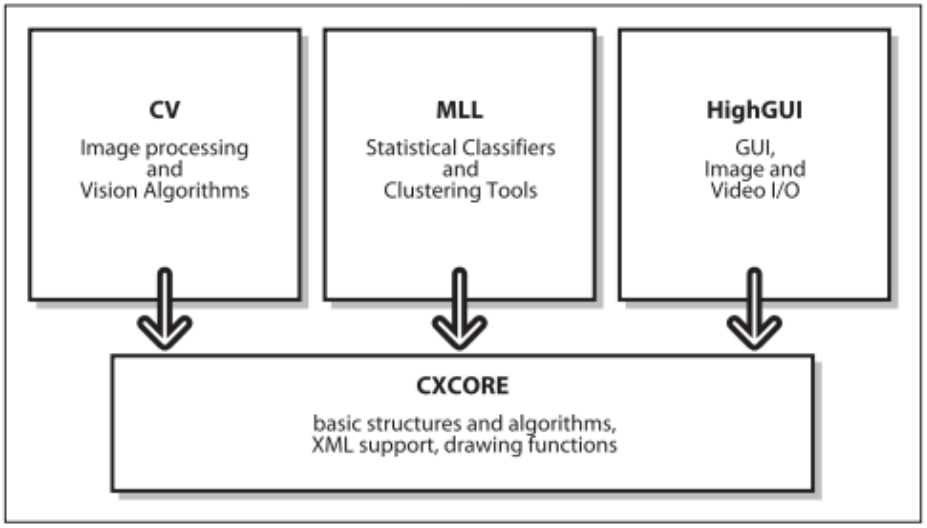


Figura 15. Estructura básica de OpenCV. (Kaehler y Bradski, 2008)

El componente CV contiene el procesamiento básico de imágenes y algoritmos de visión por computadora de nivel superior; ML es la biblioteca de aprendizaje automático, que incluye muchos clasificadores estadísticos y herramientas de agrupación. HighGUI contiene rutinas y funciones de E / S para almacenar y cargar vídeos e imágenes, y CXCore contiene las estructuras de datos básicos y el contenido (Kaehler y Bradski, 2008).

OpenGL:

OpenGL (Open Graphics Library) es una API multiplataforma para el despliegue de primitivas gráficas en 2D y 3D. Con OpenGL es posible crear aplicaciones interactivas en computación gráfica para el despliegue de imágenes a color en alta calidad que representa a objetos geométricos, primitivas, escenas 3D, datos multidimensionales, imágenes, entre otros. La API de OpenGL es independiente del sistema operativo y del sistema de ventanas, parte del código se ejecuta independiente de la plataforma de ejecución y otra es controlada por el sistema de ventanas de la plataforma donde se ejecuta (e.g. X Windows, API Win32, etc.) (Ramírez, 2014), La librería se ejecuta a la par con el programa independientemente de la capacidad gráfica de la máquina que se esté usando. Esto significa que la ejecución se dará por software a no ser que se cuente con hardware gráfico específico en la computadora, así esta librería puede usarse bajo todo tipo de sistemas operativos e incluso usando una gran variedad de lenguajes de programación (García y Guevara, 2004).

5.3. Marco Legal

5.3.1 Ministerio de TICS decreto Número 1412

Adiciona el título al Decreto Único Reglamentario del sector TIC, Decreto 1078 de 2015, para reglamentar el Estatuto Tributario, en relación con el Contenido digital y los softwares para el desarrollo de éstos.

5.3.2 Ley N° 1341 de 2009.

Por la cual se definen Principios y conceptos sobre la sociedad de la información y la organización de las Tecnologías de la Información y las Comunicaciones -TIC-, se crea la Agencia Nacional del Espectro y se dictan otras disposiciones.

6. Metodología

Para el desarrollo de la herramienta y su posterior aplicación en la materia de sistemas mecatrónicos I, se planteó un total de cuatro fases de trabajo principales, con subdivisiones de diferentes actividades en cada una.

FASES	ACTIVIDADES
Programa básico realidad aumentada con interfaz	• Detección de marcador ArUco. • Obtener matriz de cámara y coeficiente de distorsión. • Estimación de pose del marcador ArUco. • Carga de modelo 3D de prueba sobre el marcador ArUco. • Diseño de interfaz con funciones básicas.
Modelos del brazo robótico	• Determinar parámetros y requerimientos del brazo robótico. • Selección del modelo 3D del brazo robótico.
Acondicionamiento y carga de modelos	• Acondicionamiento de los modelos. • Cargar modelos del brazo robótico en la herramienta.
Guía práctica y funciones complementarias en la herramienta	• Animación de modelos. • Información sobre elementos visualizados. • Posicionamiento de modelos. • Generación de ejecutable. • Componentes y estructura de guía práctica.

Tabla 1. Fases de trabajo para el desarrollo de la herramienta de AR.

La primera fase y sus actividades hace referencia al desarrollo de la herramienta básica de realidad aumentada y su interfaz, y es a partir de la segunda fase, en donde se establecen las características que esta va a tener, con el objetivo de darle aplicabilidad en el laboratorio especializado de ingeniería en control.

6.1 Programa básico de realidad aumentada.

La primera fase fue dedicada a la indagación del tema de realidad aumentada con Python, esto, llevó a escoger uno de los tipos de manera de realidad aumenta basada en marcadores, en la que nos enfocaremos en los marcadores ArUco.

Un marcador ArUco es un marcador cuadrado sintético compuesto por un borde negro ancho y una matriz binaria interna que determina su identificador (id). El borde negro facilita su rápida

detección en la imagen y la codificación binaria permite su identificación y la aplicación de técnicas de detección y corrección de errores.

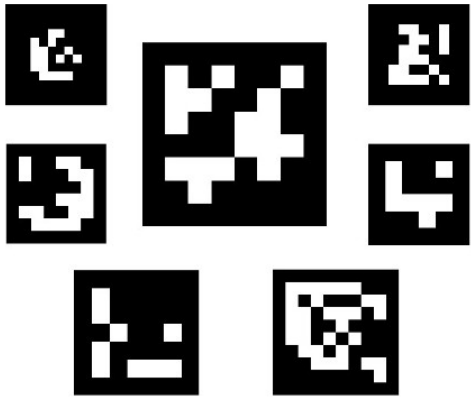


Figura 16. Marcador ArUco. (Jiménez, 2018)

Para entrar en contexto del funcionamiento de la realidad aumentada a base de marcadores, se procedió a investigar e implementar un programa básico con marcadores. Como se menciona en el título del proyecto se hará uso de la librería openCV, ya que esta posee una gran cantidad de algoritmos, que permiten identificar objetos, caras, marcadores, imágenes, hacer rastreo de movimiento de objetos; y muchas otras cosas mediante el uso de la cámara.

El programa base sirve para familiarizarnos con la librería y los marcadores ArUco comprobando su funcionamiento y tener una idea de cómo desarrollar la herramienta; el primer paso es realizar la detección del marcador ArUco. Para esto es necesario instalar Python, en este caso se trabajó con la versión 3.6, y se instaló la librería openCV.

6.1.1 Detección del marcador ArUco.

Es importante conocer el procedimiento que sigue ArUco para detectar estos marcadores:

- Primero, se aplica un umbral adaptivo hasta obtener los bordes del marcador.
- Tras encontrar los bordes se realiza la búsqueda de los contornos del marcador. Una vez encontrados, se detectan también bordes no deseados.
- El primer paso para eliminar los bordes no deseados, es eliminar los bordes con un pequeño número de puntos.
- Luego, se realiza una aproximación poligonal de los contornos encontrados y se mantienen aquellos con cuatro esquinas. Se enumeran las esquinas en sentido de las agujas del reloj.
- Se eliminan los rectángulos demasiados cercanos.
- Se Identifica el marcador.

Para realizar este proceso, en primer lugar, se importan las librerías necesarias: openCV y numpy (necesaria para opencv, ya que realiza múltiples operaciones matemáticas con las imágenes que se van procesando). Por otro lado, se crea un objeto denominado **cap** al que asociaremos la fuente de la cámara conectada (0 por defecto).

```
import cv2
import cv2.ArUco as ArUco
import numpy

cap =cv2.VideoCapture(0)
```

El proceso se trabaja dentro de un bucle el cual solo se romperá al presionar la tecla ‘q’, en este definimos dos variables **ret** y **frame**; ret ira capturando las imágenes del video mediante el procedimiento **read ()** en el objeto **cap**. Por otro lado, creamos una ventana a partir de **frame** para visualizar la captura y este se asocia con la variable **gray** para definir el formato de color de la ventana.

```
while (True):

        ret, frame =cap.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        if cv2.waitKey(1) & 0xff == ord ('q'):
                break

cap.release()
cv2.destroyAllWindows()
```

Hasta acá esto solo permite visualizar la captura de la cámara. Dentro de este mismo bucle se tiene la parte que nos permite identificar los marcadores ArUco, para este proceso es necesario definir ciertas funciones, una es el diccionario ArUco, la cual contiene información de los distintos marcadores ArUco en este caso se trabajó con **ArUco. DICT_6X6_250**.

La detección de los marcadores se hace con la función **detectMarkers**, esta contiene los parámetros que permiten identificar los marcadores, estos parámetros son:

- **markerCornes:** es la lista de esquinas de los marcadores detectados. Para cada marcador, sus cuatro esquinas se devuelven en su orden original. Entonces, la primera esquina es la esquina superior izquierda, seguida de la esquina superior derecha, inferior derecha e inferior izquierda como se observa en la figura 19.

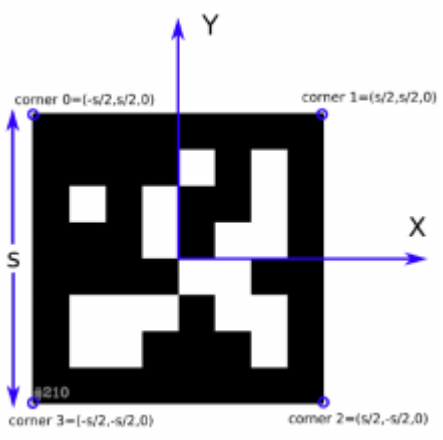


Figura 17. Detección de las esquinas del marcador ArUco. (Jiménez, 2018)

- **markerIds:** es la lista de identificadores de cada uno de los marcadores detectados en **markerCornes.**
- **rejectedCandidates**: es una lista devuelta de candidatos a marcadores, es decir, formas que se encontraron y consideraron pero que no contenían un marcador válido.
- **DetectorParameters:** este incluye todos los parámetros que se pueden personalizar durante el proceso de detección.

Una vez detectado, podemos realizar ciertas operaciones sobre la imagen capturada, pintando los bordes de las esquinas del marcador, el sistema de ejes de coordenadas o pintando la información del identificador con el fin de visualmente identificar qué marcador es. Para ello se utiliza la función **drawDetectedMarkes** (Jiménez, 2018).

```
        ArUco_dict = ArUco.Dictionary_get(ArUco.DICT_6X6_250)
        parameters =  ArUco.DetectorParameters_create()
        markerCornes, markerIds, rejectedCandidates =
cv2.ArUco.detectMarkers(frame, ArUco_dict, parameters = parameters)

        if markerIds is not None:
                for i, corner in zip(markerIds, markerCornes):
                        print('ID: {}; corners :{}'.format(i,corner))
                        frame=
cv2.ArUco.drawDetectedMarkers(frame,markerCornes,borderColor=(0,255,0))

        cv2.imshow('frame',frame)
```

Con la implementación de este código se realiza la primera prueba que es detectar el marcador ArUco, el resultado se puedo observar en la figura 20, en la cual detallamos como identifica el marcador señalando en un cuadrado verde y como se explica, una de sus funciones es también identificar la primera esquina (superior izquierda), con esto nos hacemos a la idea de que detecta el movimiento del marcador.

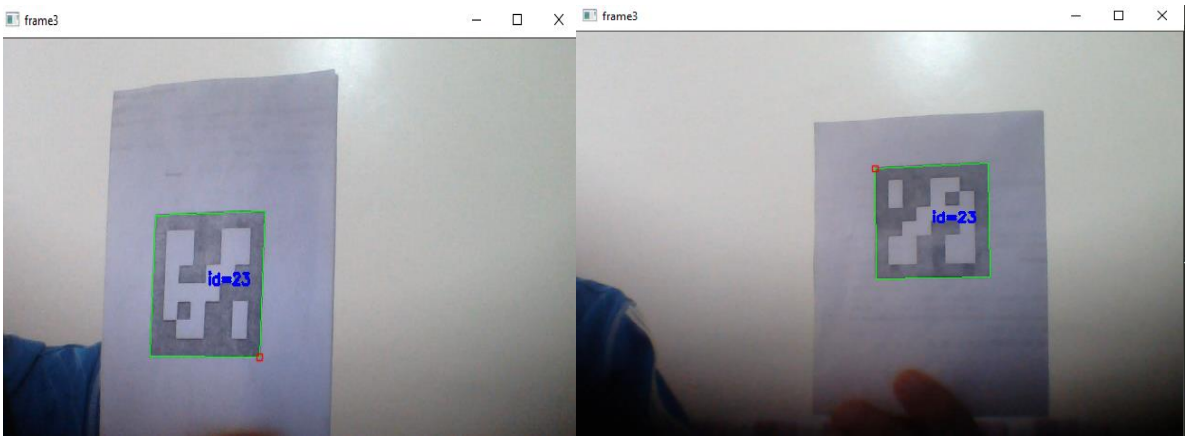


Figura 18. Detención marcador ArUco.

La detección del movimiento del marcador, nos lleva a realizar la estimación de pose, pero para hacer esto es necesario realizar la calibración de la cámara y conocer sus parámetros. Estos son la matriz de la cámara y el coeficiente de distorsión.

6.1.2 Matriz de cámara y coeficiente de distorsión

Las cámaras fotográficas sin lente introducen mucha distorsión en las imágenes. Dos distorsiones principales son la distorsión radial y la distorsión tangencial. (Mordvintse, 2020)

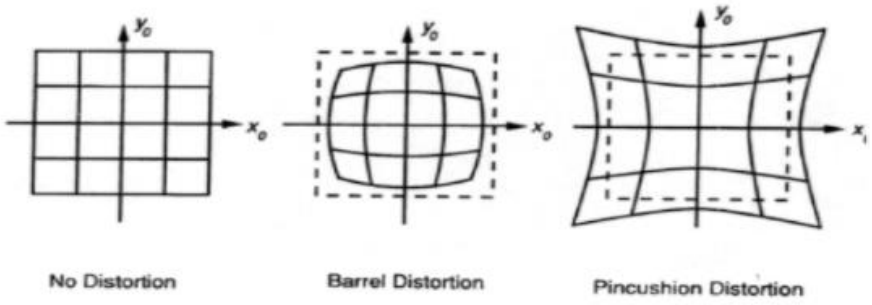


Figura 19. Distorsión de cámara. (Open Source, 2020)

Se conoce como distorsión radial a las aberraciones producidas por defectos de construcción en la lente. Estos defectos suelen causar que cuanto más se aleje el punto de impacto del rayo de luz incidente del centro de la imagen, mayor será la desviación a la que será sometido. Esta clase de errores llevan a una visión de ojo de pez, en la que los píxeles situados en el centro de la imagen sufren pequeñas modificaciones, y los más alejados sufren aberraciones cada vez mayores. (Torrontegi, 2010)

Esta distorsión se resuelve de la siguiente manera:

$$Xcorregida = x(1 + k1r^2 + k2r^4 + k3r^6) \quad (1)$$

$$Ycorregida = y(1 + k1r^2 + k2r^4 + k3r^6) \quad (2)$$

En donde:

r =distancia desde el centro hasta la circunferencia de aberración donde se encuentra el punto a corregir

k1, k2, k3= coeficientes de distorsión.

Así mismo se conoce como distorsión tangencial, a la distorsión que se produce por la falta de paralelismo entre la lente de la cámara, y el sensor de la cámara, y genera imágenes trapezoidales. (Torrontegi, 2010) en otras palabras esta distorsión se produce cuando el lente que captura las imágenes no está alineado perfectamente en paralelo al plano de imagen, lo cual causa que determinadas áreas de la imagen capturada parezcan estar más cerca de lo esperado.
Esta distorsión se resuelve de la siguiente manera:

$$Xcorregida = x + [2p1xy + p2(r^2 + 2x^2)] \quad (3)$$

$$Ycorregida = y + [p1(r^2 + 2y^2) + 2p2xy] \quad (4)$$

En donde:

p1, p2= coeficientes de distorsión.

Cabe recalcar que los coeficientes de distorsión tanto de la distorsión radial como de la distorsión tangencial, varían de una cámara a otra, por lo que lo ideal sería, que cuando se utilice una cámara diferente, se realice su respectiva calibración.

Así mismo, las cámaras CCD son las cámaras digitales usadas hoy en día. Bajo este modelo, un punto en el espacio 3D, de coordenadas X= (X, Y, Z) es proyectado al plano imagen por medio de una línea que une el punto X con el centro de proyección como se muestra en la siguiente figura. (Witt, 2014)

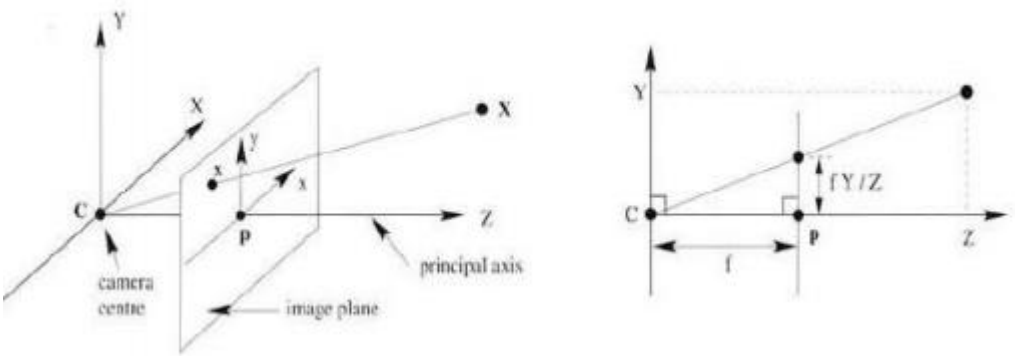


Figura 20. Geometría de una cámara CCD. (Witt, 2014)

Razón por la cual es primordial encontrar los parámetros intrínsecos y extrínsecos de la cámara a utilizar.
Los parámetros intrínsecos Incluyen información como distancia focal (fx, fy), centros ópticos (cx, cy), etc. (Mordvintse, 2020), estos parámetros también se conocen como matriz de cámara.

$$Matriz\ de\ Camara = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Por otra parte, Los parámetros extrínsecos corresponden a vectores de rotación y traslación que traducen las coordenadas de un punto 3D a un sistema de coordenadas.

Para herramientas de realidad aumentada es de vital importancia corregir primero este tipo de distorsiones mencionadas anteriormente, para esto es necesario tener imágenes de un patrón bien definido como por ejemplo un tablero de ajedrez.

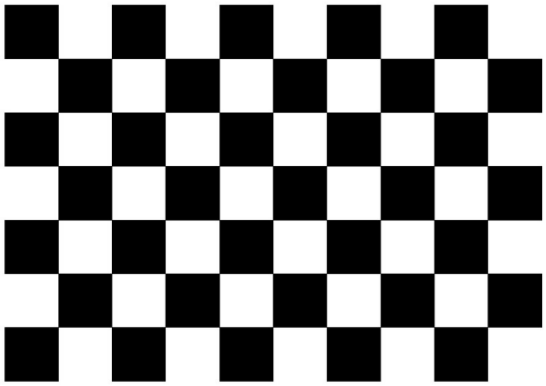


Figura 21. patrón de tablero de ajedrez utilizado para calibrar la cámara. (Open Source, 2020)

Es necesario tener al menos 10 imágenes desde ángulos diferentes del patrón, para mejorar así los resultados de la calibración de la cámara.

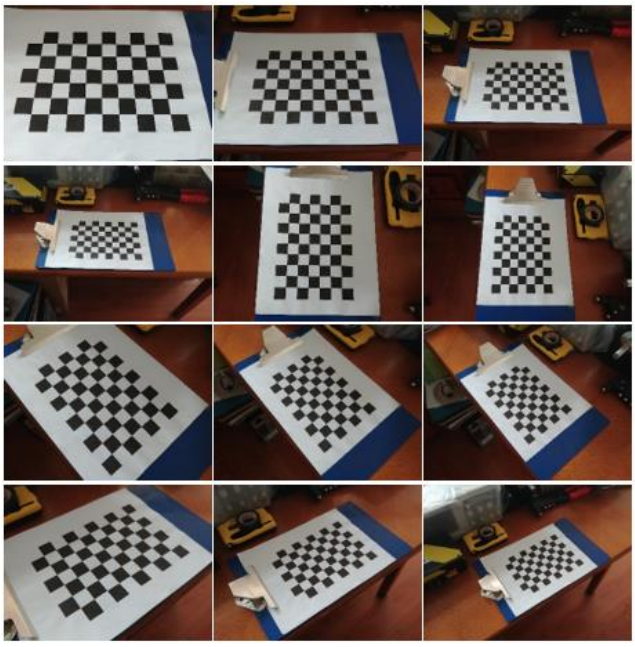


Figura 22. Imágenes de muestra tomadas al patrón de ajedrez.

Los datos de entrada importantes necesarios para la calibración de la cámara son un conjunto de puntos 3D del mundo real (puntos de objeto) y sus correspondientes puntos de imagen en 2D (puntos de imagen), Estos puntos de imagen 2D son ubicaciones donde dos cuadrados negros se tocan en tableros de ajedrez. (Mordvintse, 2020), además es necesario saber qué tipo de cuadricula se está usando, para este caso es de 9x6.

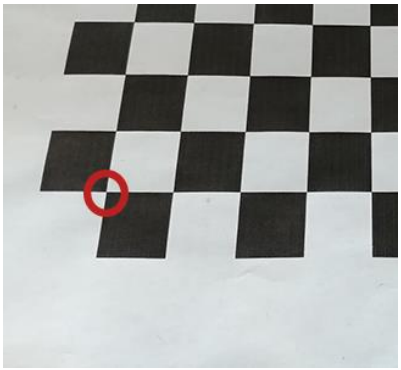


Figura 23. Ejemplo de puntos de imagen.

Para la implementación de la calibración de la cámara, lo primero es necesario utilizar las siguientes bibliotecas:

```
import numpy as np
import cv2
import glob
import yaml
```

Se crea además un criterio de terminación de la iteración. Cuando se satisface este criterio, la iteración del algoritmo se detiene.

```
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
```

Luego se determina el tamaño del tablero de ajedrez, como ya se había especificado anteriormente es de 9x6.

```
width, height = 9, 6
objp = np.zeros((width * height,3), np.float32)
objp[:,:2] = np.mgrid[0:width,0:height].T.reshape(-1,2)
```

Y se crean las variables para el conjunto de puntos 3D (objpoints), los puntos de imagen 2D (imgpoints) y la variable que almacenara todos los datos posteriormente (data), además se cargan las imágenes de muestra tomadas al patrón de ajedrez.

```
objpoints = []
imgpoints = []
data=None
images = glob.glob('./ChessBoardSet/*.jpg')
```

Para encontrar un patrón en el tablero de ajedrez se usa la función **cv2.findChessboardCorners()** la cual se encarga de encontrar las posiciones de las esquinas internas del tablero de ajedrez. La función intenta determinar si la imagen de entrada es una vista del patrón del tablero de ajedrez y ubica las esquinas internas del tablero. La función devuelve un valor distinto de cero si se encuentran todas las esquinas y se colocan en un orden determinado (fila por fila, de izquierda a derecha en cada fila). De lo contrario, si la función no encuentra todas las esquinas o las reordena, devuelve 0. (Open Source, 2020)

```
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    w, h = img.shape[:2]
    ret, corners = cv2.findChessboardCorners(gray, (width, height ),None)
```

Si la función devuelve un valor diferente de cero entonces, se puede mejorar su precisión utilizando **cv2.cornerSubPix ()**, y también se pueden dibujar y guardar los patrones sobre las imágenes tomadas con **cv2.drawChessboardCorners ().**

```
if ret == True:
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,
        -1),criteria)
        imgpoints.append(corners2)
        imgName = fname[fname.rindex('\\') + 1 : ]
        img = cv2.drawChessboardCorners(img, (width ,height),
```

```
        corners2,ret)
        cv2.imwrite('./ChessBoardMarkedSet/' + imgName, img)
```

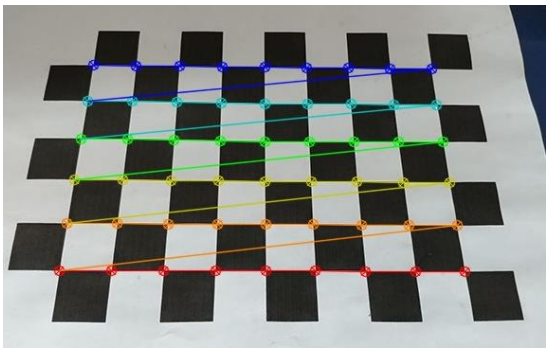


Figura 24. Ejemplo de imagen con patrón dibujado.

Una vez teniendo listo los patrones, puntos de imagen y puntos de objeto, el siguiente paso es realizar la calibración con la función, **cv2.calibrateCamera ().** La cual devuelve los parámetros intrínsecos y extrínsecos de la cámara, como, matriz de la cámara, coeficientes de distorsión, los vectores de rotación y traslación, etc. (Open Source, 2020)

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
```

Con lo cual ya se obtienen los parámetros que se estaban buscando, sin embargo, antes de guardarlos es posible refinar la matriz de la cámara basándose en un parámetro de escala libre usando **cv2.getOptimalNewCameraMatrix().**

```
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist, (w,h), 1, (w,h))
```

Por último, solo resta guardar las constantes obtenidas en el archivo ***data1.yaml*.**

```
if ret:
     data = {'camera_matrix': np.asarray(newcameramtx).tolist(),
             'dist_coeff': np.asarray(dist).tolist(),
             'rvecs': np.asarray(rvecs).tolist(),
             'tvecs': np.asarray(tvecs).tolist()}
with open('data1.yaml', 'w') as f:
    yaml.dump(data,f)

print(data)
cv2.destroyAllWindows()
```

Los códigos descritos anteriormente muestran el procedimiento para obtener los datos que permitan realizar la calibración de la cámara, ahora solo resta cargar el archivo generado ***data1.yaml*** en el programa principal. Para lo cual es necesario crear una sentencia ejecutable definida por el usuario, en donde se obtendrá la matriz de cámara, coeficientes de distorsión, vectores de rotación y traslación.

```
self.cam_matrix,self.dist_coefs,rvecs,tvecs=self.get_cam_matrix("data1.yaml")
```

```
    def get_cam_matrix(self,file):
        with open(file) as f:
            loadeddict = yaml.load(f)
            cam_matrix = np.array(loadeddict.get('camera_matrix'))
            dist_coeff = np.array(loadeddict.get('dist_coeff'))
            rvecs = np.array(loadeddict.get('rvecs'))
            tvecs = np.array(loadeddict.get('tvecs'))
            return cam_matrix,dist_coeff,rvecs,tvecs
```

De esta forma se obtienen los siguientes valores:

$$Matriz\ de\ Camara = \begin{bmatrix} 551.8930053710938 & 0 & 314.8915481815857 \\ 0 & 631.5000610351562 & 269.7444676101404 \\ 0 & 0 & 1 \end{bmatrix} \quad (6)$$

En donde los valores de distancia focal (expresada en pixeles) son:

$$fx = 551.8930053710938 \quad (7)$$

$$fy = 631.5000610351562 \quad (8)$$

Y el valor de los centros ópticos son:

$$cx = 314.8915481815857 \quad (9)$$

$$cy = 269.7444676101404 \quad (10)$$

Los coeficientes de distorsión son los siguientes:

$$\text{Coeficientes de distorsion} = [0.2948087746425963, -2.2163429624355064, -0.001336180122392878, -0.0003835332010734688, 5.500882789601218] \quad (11)$$

En donde:

$$k1 = 0.2948087746425963 \quad (12)$$

$$k2 = -2.2163429624355064 \quad (13)$$

$$k3 = 5.500882789601218 \quad (14)$$

$$p1 = -0.001336180122392878 \quad (15)$$

$$p2 = -0.0003835332010734688 \quad (16)$$

Una vez se han obtenido estos datos es posible reemplazar (12), (13), (14) en (1) y (2), solucionando así la distorsión radial:

$$Xcorregida = x(1 + 0.2948087746425963r^2 + -2.2163429624355064r^4 + 5.500882789601218r^6) \quad (17)$$

$$Ycorregida = y(1 + 0.2948087746425963r^2 + -2.2163429624355064r^4 + 5.500882789601218r^6) \quad (18)$$

En cuanto a la distorsión tangencial, también se procede a reemplazar (15) y (16) en (3) y (4), con lo que se obtiene:

$$Xcorregida = x + [2(-0.001336180122392878)xy + (-0.0003835332010734688)(r^2 + 2x^2)] \quad (19)$$

$$Ycorregida = y + [(-0.001336180122392878)(r^2 + 2y^2) + 2(-0.0003835332010734688)xy] \quad (20)$$

6.1.3 Estimación de pose

ArUco cuándo realiza una captura de un marcador, devuelve la pose del marcador con un vector de rotación rvecs, un vector de traslación tvecs y el identificador del marcador que ha detectado.

El sistema de ejes utilizado por Aruco y OpenCv en general, es el siguiente:

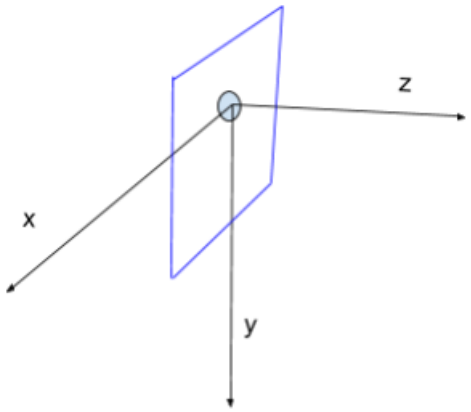


Figura 25. Sistema de ejes utilizado por ArUco. (Villacorta, 2020)

Para indicar la pose del marcador, lo que hace Aruco es considerar que originalmente el marcador está en el origen de coordenadas del sistema de la propia cámara, con la imagen apuntando hacia afuera. Con el dibujo al revés apuntando hacia abajo, ya que el eje Y se encuentra apuntando también hacia abajo (Villacorta, 2020).

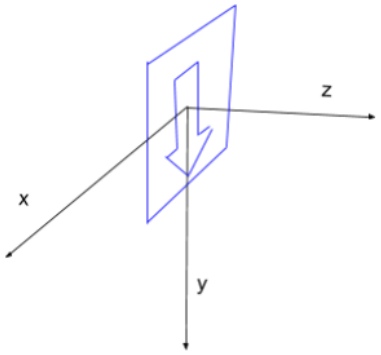


Figura 26. Sistema de ejes. (Villacorta, 2020)

El sistema de referencias real de cómo se visualiza el marcador es:

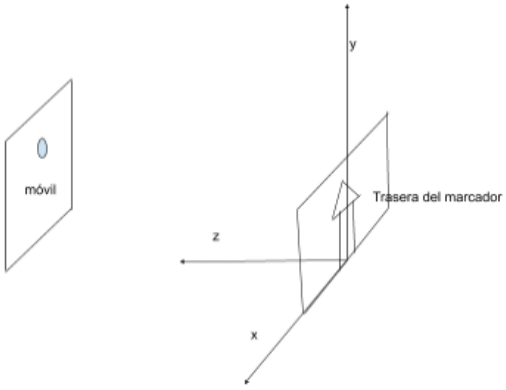


Figura 27. Sistema de referencia. (Villacorta, 2020)

La librería de Aruco procesa esa información de dos pasos:

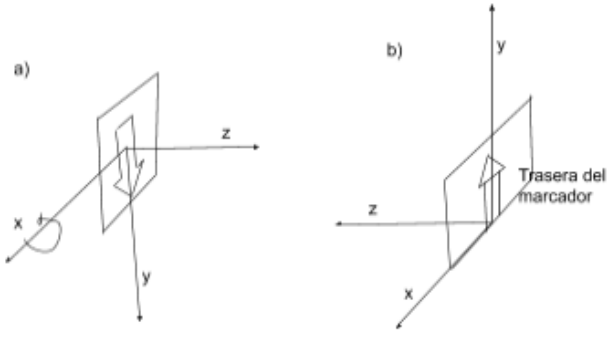


Figura 28. Procesamiento de información. (Villacorta, 2020)

Tenemos que pasar de la posición **a** hacia **b** y modificar la orientación de la flecha de abajo hacia arriba. Para pasar desde **a** hasta **b** se realiza dos fases:

- Una primera fase, donde se realiza una rotación in situ, en el propio sistema de la cámara. Rotamos el eje x en sentido contrario a las agujas del reloj un ángulo de 180º
- Una segunda fase, donde una vez rotado se realiza una traslación de una cantidad determinada en el eje z, dando como resultado **b**.

ArUco devuelve la información cómo:

$$rotación\ x = \pi \tag{21}$$

$$traslación\ z = \Delta z \tag{22}$$

Para este proceso era necesario haber realizado el proceso de calibración de la cámara. ya teniendo el archivo **data1.yaml**, con los parámetros de la cámara tras la calibración se procede a implementar el código para la estimación de pose.

Este código es la continuidad del programa de detección del marcador, con las siguientes líneas de código:

```
        if ids is not None and corners is not None:
            rvecs, tvecs, _= aruco.estimatePoseSingleMarkers(corners, 0.05 ,
self.cam_matrix, self.dist_coefs)
```

Si se cumple la condición, detectar un id o esquinas de un marcador, en general si detecta un marcador, se ejecuta el código dentro de la sentencia. En esta parte de código es donde se realiza la estimación de pose con la función **aruco.estimatePoseSingleMarkes**, la cual necesita 4 parámetros:

- Las esquinas del marcador (corners).
- El segundo parámetro es la definición de la longitud de los vectores. Normalmente, la unidad son metros.
- Matriz de la cámara.
- Coeficiente de distorsión.

Esta función nos retorna como se menciona anteriormente dos vectores, rotación (rvecs) y translación (tvecs).

```
        for i in range (0, ids.size):

            aruco.drawAxis(frame, self.cam_matrix , self.dist_coefs , rvec[i],
tvec[i], 0.1)
```

Con la función **aruco.drawAxis** se dibuja los ejes x, y, z sobre el marcador, esta recibe los parámetros generados de rotación y translación con los cuales se realiza la estimación de pose.

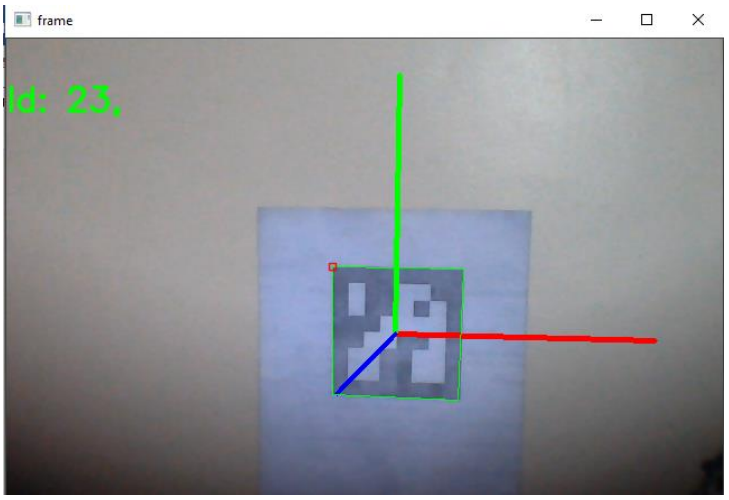


Figura 29. Estimación de pose marcador ArUco.

En la figura 29 se puede observar la estimación de pose cuando el marcador esta de frente a la cámara en una posición donde la esquina 0 se encuentra en la parte superior izquierda, siendo esta la posición que se observa en el sistema de referencia. Donde el vector de color verde representa el eje Y, el vector de color rojo representa el eje X y el vector de color azul representa el eje Z.

Para comprobar la estimación de pose, se mueve el marcador en diferentes ángulos, observando cómo se refleja en el movimiento de los ejes, como se observa en la figura 30.

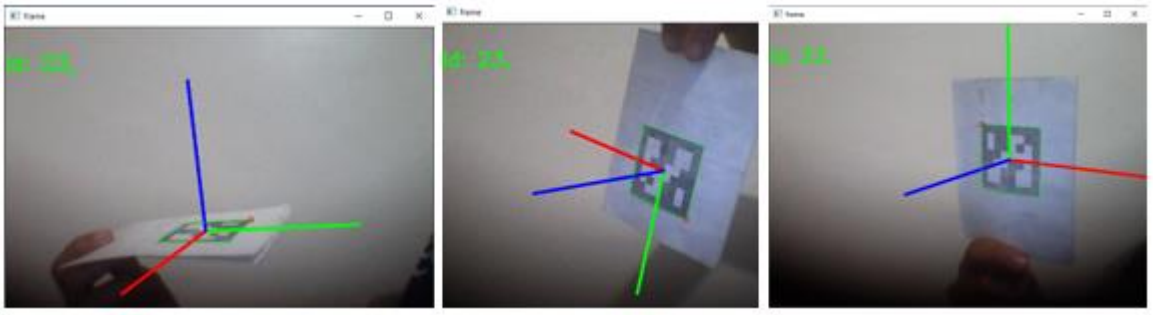


Figura 30. Estimación de pose diferentes ángulos.

6.1.4 Carga de Modelo 3D.

Para la carga y visualización de un modelo 3D sobre el marcador ArUco es necesario el uso de un motor gráfico, para esto se implementa openGL.

Para ello el primer paso es importar las librerías que implementan las funciones de OpenGL, cada una representa un nivel de abstracción de operaciones relacionadas con el desarrollo de aplicaciones gráficas.

```
from OpenGL.GL import *
from OpenGL.GLUT import *
```

```
from OpenGL.GLU import *
```

- GL es el nivel de operaciones más básico.
- GLU (OpenGL Utility Library), operaciones de más alto nivel para crear objetos (desde un cubo hasta una tetera, p.ej.), mipmaps con texturas, etc.
- GLUT (OpenGL Utility Toolkit), agrupa a las funciones encargadas de la gestión de eventos del sistema de ventanas de forma transparente al sistema operativo sobre el que se ejecuta. Estrictamente hablando no es parte de OpenGL, pero suele acompañarlo en muchos ejemplos ya que permite el desarrollo de un interfaz de usuario solvente y portable.

Las funciones con el prefijo "glut" se utilizan en el programa principal para crear la ventana de trabajo y establecer las funciones que van a responder a los eventos, tarea que es gestionada por glutMainLoop (Agustí, 2020).

```
    def main(self):
        glutInit()
        glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
        glutInitWindowSize(900, 680)
        glutInitWindowPosition(380, 0)
        self.window_id = glutCreateWindow(b"AR Didactic Tool")
        glutDisplayFunc(self.draw_scene)
        glutIdleFunc(self.draw_scene)
        self.initOpengl(640, 480)
        glutMainLoop()
```

con estas funciones se define la ventana:

- Inicializar el modo de visualización. Esto se hace llamando a la función: glutInitDisplayMode. En este caso, estamos activando el buffer de profundidad (GLUT_DEPTH), el doble buffering (GLUT_DOUBLE) y la renderización en RGB (GLUT_RGB).
- Inicializar el tamaño de la ventana. Esto se hace mediante la función: glutInitWindowSize (ancho, altura).
- Inicializar la posición de la ventana en el escritorio. La función correspondiente es: glutInitWindowPosition (x, y)
- Crear la ventana. Llamaremos a la función: glutCreateWindow(char* titulo) Esta función muestra una ventana del tamaño y posición definidas y con el título que le hayamos puesto.
- Activar el test del buffer de profundidad: glEnable(GL_DEPTH_TEST)
- Definir el callback para redibujar la ventana. Esto lo hacemos mediante la función glutDisplayFunc(). Así, la función que se llamará cada vez que haya que redibujar la ventana.

En el main se llama a la función **initOpengl**; en esta se inicializa openGL con parámetros definidos (Jorge, 2003) (Morales, 2020).

```
glClearColor(0.0, 0.0, 0.0, 0.0)
glClearDepth(1.0)
glDepthFunc(GL_LESS)
glEnable(GL_DEPTH_TEST)
glShadeModel(GL_SMOOTH)
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
gluPerspective(37, 1.3, 0.1, 100.0)
glMatrixMode(GL_MODELVIEW)
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
glDisable(GL_BLEND)
glEnable(GL_TEXTURE_2D)
self.texid = glGenTextures(1)
```

- glclearcolor estamos configuramos el color de fondo con R = 0, G = 0, B = 0 (negro).
- glClearDepth especifica el valor claro para el búfer de profundidad.
- glDepthFunc realiza la comparación entre el pixel entrante con el del buffer de profundidad, se inicializa con GL_LESS la cual permitirá el paso si el pixel entrante si es menor que el almacenado.
- Gl_DEPTH_TEST habilitamos la comprobación de la profundidad en el dibujado.
- glShademodel seleccionamos una técnica de sombreado se escoge GL_SMOOTH, con este seleccionamos una técnica de sombreado suave.
- glMatrixMode, esta específica cual es la matriz actual se utilizan dos funciones:

    - GL_PROJECTION: la matriz de proyección, es en la que se guarda la información relativa a la "cámara" a través de la cual vamos a visualizar el mundo.
    - GL_MODELVIEW: La matriz de visualización/modelado es la que se encarga de la transformación y visualización de los objetos (traslación, rotación y escala) en el marco de coordenadas.

    Este proceso se inicializa primero la proyección para crear el volumen de visualización y después se deja activa la matriz visualización/modelado de con esta se trabaja los objetos que se visualizan.

- glLoadIdentity, esta función carga a la matriz identidad a la matriz actual.
- gluPerspective establece el modo de visualización en perspectiva (el ángulo de visualización es de 37 grados, la razón ancho/alto es 1.3, la distancia mínima es z = 0.1 y la distancia máxima es z = 100).
- GL_BLEND es la función para implementar transparencia dentro de los objetos, y se configura los tipos de transparencia con glBlendFunc, en el inicio del programa se deshabilita la transparencia.

- Se habilita las texturas 2D con la función GL_TEXTURES_2D.
- glGenTextures retorna el identificador de las texturas.

En el *initOpengl* solo se inicializan las funciones, pero en el main hay una función que se utiliza para redefinir la ventana esta se asocia a una definición llamada *draw_scene*.

En la definición de *draw_scene* se captura la imagen de la cámara como se mostró anterior mente con openCV para hacerle el tratamiento y convertir la imagen al formato de texturas openGL.

```
        bg_image = cv2.flip(image, 0)
        bg_image = Image.fromarray(bg_image)
        ix = bg_image.size[0]
        iy = bg_image.size[1]
        bg_image = bg_image.tobytes("raw", "BGRX", 0, -1)
```

después de convertir la imagen, se define el formato de texturas ya inicializado.

```
        glBindTexture(GL_TEXTURE_2D, self.texid)
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
        glTexImage2D(GL_TEXTURE_2D,    0,    3,    ix,    iy,    0,    GL_RGBA,
GL_UNSIGNED_BYTE, bg_image)
```

con la función **glBindTexture** estamos activando la textura como textura activa, texid es el identificador de la textura que vamos a activar; con **glTextParameterf se** fijan los parámetros para el mapeo de texturas; esto es una técnica que aplica una imagen a la superficie de un objeto como si la imagen fuera una calcomanía o una envoltura de celofán. La imagen se crea en el espacio de textura, con un sistema de coordenadas (x, y) (Overvoorde, 2020).

se especifican qué tipo de interpolación se va utilizar para dos casos separados: reducir la escala de la imagen y aumentar la escala de la imagen. Estos dos casos se identifican con **GL_TEXTURE_MAG-FILTER** y **GL_TEXTURE_MIN_FLITER.**

Lo siguiente es cargar la imagen de textura esto se hace con la función **glTextImage2D**. El primer parámetro después del objetivo de texturas es el nivel de detalle, donde "0" es la imagen base. El segundo parámetro especifica el formato del pixel interno, el formato en el que se debe almacenar los píxeles en la tarjeta gráfica. los parámetros tercero y cuarto especifican el ancho y el alto de la imagen "ix, iy". El quinto parámetro siempre debe tener un valor de 0 según la especificación. Los siguientes dos parámetros describen el formato de los pixeles de la matriz que se cargaran y el parámetro final especifica la matriz "bg_image" que es la matriz que se transformó de la captura de la cámara.

```
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(33.7, 1.3, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)
```

```
glLoadIdentity()
glPushMatrix()
glTranslatef(0.0,0.0,-10.0)
self.draw_background()
glPopMatrix()
```

Después de cargar las texturas se vuelve a llamar las matrices de proyección y de visualización/modelado esto se hará cada vez que se redefina la ventana, pero en este se con la función glPushMatrix guardará la última matriz en la cima de la pila y con glPopMatrix la saca y la restaura (Jorge, 2003). Esto lo podemos utilizar para dibujar un objeto y antes de dibujar el siguiente, restauramos la transformación inicial. En esta parte llamamos a la definición de draw background.

```
glBegin(GL_QUADS)
glTexCoord2f(0.0, 1.0); glVertex3f(-4.0, -3.0, 0.0)
glTexCoord2f(1.0, 1.0); glVertex3f( 4.0, -3.0, 0.0)
glTexCoord2f(1.0, 0.0); glVertex3f( 4.0,  3.0, 0.0)
glTexCoord2f(0.0, 0.0); glVertex3f(-4.0,  3.0, 0.0)
glEnd()
```

La manera en la que OpenGL dibuja las cosas es en base a primitivas que pueden ser puntos, segmentos de línea y polígonos. Cada vez que queramos hacer que se dibuje una primitiva debemos llamar el comando glBegin, indicando el tipo de primitiva que vamos a dibujar, dar las coordenadas de los vértices y terminar con el comando glEnd. El tipo de primitiva que se escoge fue GL_QUADS, la cual dibuja un cuadrilátero por cada cuatro vértices.

```
image = self.draw_objects(image)
glutSwapBuffers()
```

Retornando a la definición draw scene, después de realizar el llamado al draw background, se realiza el llamado a draw.objects asociándolo a image, esta varible se explicó en OpenCV como el frame que es la encargada de la ventana; y por último se declara la función glutSwapBuffers es la que le indica a la computadora que ha terminado con el marco actual y que los búferes deben intercambiarse para que se muestre ese marco y para que pueda comenzar a trabajar en el siguiente.

En draw objects se implementa la parte de openCV explicada anteriormente, pero se implementa con la visualización en la ventana de openGL. Para realizar esto se tiene que transformar la matriz intrínseca de la cámara por una matriz proyección y la matriz extrínseca por la matriz visualización/modelado, esto se realiza en un programa externo llamado **Matrixtransform** (Straw, 2011).

Tomando la ecuación (5) tenemos la matriz intrínseca de la cámara.

$$Matriz\ de\ Camara = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

- cx: número de pixeles horizontales de la imagen.
- cy: número de pixeles verticales de la imagen.
- fx: longitud focal de la dirección horizontal.
- fy: longitud focal de la dirección vertical.

La matriz de proyección de openGL seria la siguiente:

$$\begin{bmatrix} \frac{2f_x}{c_x} & 0 & 0 & 0 \\ 0 & \frac{2f_y}{c_y} & 0 & 0 \\ 0 & 0 & -\frac{z_f + z_n}{z_f - z_n} & -\frac{2z_f z_n}{z_f - z_n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{23}$$

Donde:

- Zn: distancia del plano de recorte frontal.
- Zf: distancia del plano de recorte posterior.

Esto se realizó con el siguiente código.

```
def intrinsic2Project(MTX, width, height, near_plane=0.01, far_plane=100.0):

    P = np.zeros(shape=(4, 4), dtype=np.float32)
    fx, fy = MTX[0, 0], MTX[1, 1]
    cx, cy = MTX[0, 2], MTX[1, 2]
    P[0, 0] = 2 * fx / width
    P[1, 1] = 2 * fy / height
    P[2, 0] = 1 - 2 * cx / width
    P[2, 1] = 2 * cy / height - 1
    P[2, 2] = -(far_plane + near_plane) / (far_plane - near_plane)
    P[2, 3] = -1.0
    P[3, 2] = - (2 * far_plane * near_plane) / (far_plane - near_plane)

    return P.flatten()
```

En la definición **intrisnsic2Project** se realiza la transformación de la matriz intrínseca a la matriz proyección donde recibimos los valores de MTX (matriz intrínseca de la cámara), el ancho y alto de la ventana, near_plane(Zn), far_plane(Zf).

La matriz extrínseca de la cámara en openCV se ve así:

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \tag{24}$$

Y la matriz de visualización/modelado de openGL es la siguiente:

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ -r_{10} & -r_{11} & -r_{12} & -t_y \\ -r_{20} & -r_{21} & -r_{22} & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{25}$$

Para hacer esta transformación se usa el siguiente código:

```
def extrinsic2ModelView (RVEC, TVEC, R_vector = True):

    R, _ = cv2.Rodrigues(RVEC)

    Rx = np.array([
        [1, 0, 0],
        [0, -1, 0],
        [0, 0, -1]
    ])

    TVEC = TVEC.flatten().reshape((3, 1))
    transform_matrix = Rx @ np.hstack((R, TVEC))
    M = np.eye(4)
    M[:3, :] = transform_matrix
    return M.T.flatten()
```

Esta definición **extrinsic2ModelView**, recibe los parámetros de rotación (rvec) y translación (tvec) de la cámara. Ya teniendo esto solo el llamar este programa externo en el programa principal para hacer uso de estas funciones.

```
        projectMatrix = intrinsic2Project(self.cam_matrix, width, height,
0.01, 100.0)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glMultMatrixf(projectMatrix)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()
```

Se define la matriz proyección, llamando a la matriz generada en el programa, se realiza el tratamiento de matrices declarando la matriz proyección de openGL y esta se multiplica con la matriz proyección de la cámara con la función **glMultMatrixf**.

```
model_matrix = extrinsic2ModelView (rvecs, tvecs)
```

luego se define la matriz de modelo llamando la matriz generada.

```
File = 'Sinbad_4_000001.obj'
self.modelo = OBJ(File,swapyz=True)
glPushMatrix()
glLoadMatrixf(model_matrix)
glCallList(self.modelo)
glPopMatrix()
```

Con **glLoadMatrix** se reemplaza la matriz actual por la matriz **model_matrix**. Se ejecuta la lista de visualización con la variable **glCallList**, donde se llama la variable **self.modelo**, a esta se le asocia el modelo que se va a cargar, y con esto se visualizara el modelo 3D sobre el marcador aruco cuando sea detectado.

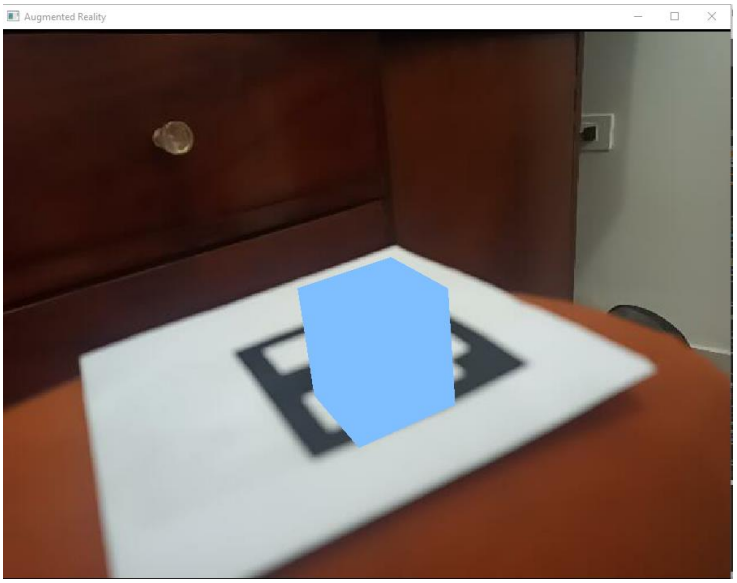


Figura 31. Visualización modelo 3D sobre marcador ArUco.

La herramienta puede llegar a presentar problemas de estabilidad en cuanto a la detección del marcador Aruco, los cuales llegan a ser molestos. Es por esto que se diseñó un código pensado en mejorar la detección del marcador y por ende la visualización de los elementos.

Filtro de Detección de marcador

La Detección del marcador por lo general funciona de forma correcta, sin embargo, si se presentan movimientos bruscos de la cámara o del marcador, puede provocar que no se detecte el marcador aruco, con lo cual no se visualizara los elementos 3D sobre el mismo.

Por consiguiente, se implementó un código capaz de detectar cuanto tiempo pasa entre las detecciones intermitentes del marcador, por lo que fueron creadas 3 variables:

- **self.tiemposi** = captura el último número de segundos obtenido mientras la herramienta detecta la presencia del marcador.
- **self.tiempo** = captura el último número de segundos obtenido mientras la herramienta NO detecta el marcador.
- **self.tiempodif** = es el valor que indicara cuantos segundos a transcurrido entre detección y no detección del marcador por parte de la herramienta.

```
self.tiemposi=0
self.tiempono=0
self.tiempodif=0
```

De esta forma en la segunda línea del siguiente código se captura el número de segundos transcurridos desde la última detección del marcador:

```
if tvecs is not None:
            self.tiemposi=time.time()
            if self.filter.update(tvecs):
                model_matrix = extrinsic2ModelView(rvecs, tvecs)
            else:
                model_matrix = self.pre_extrinsicMatrix
```

En el momento en que la herramienta no detecte el marcador, capturara el número de segundos transcurridos en la variable **self.tiempono,** para luego calcular cuantos segundos a transcurrido entre detección y no detección del marcador (**self.tiempodif**), de forma que si esta cantidad es menor o igual a 0.15 segundos, la matriz del modelo será la última matriz obtenida en el momento de la última detección del marcador (**model_matrix = self.pre_extrinsicMatrix**), por lo cual el modelo seguiría visualizándose en la pantalla, en caso contrario el modelo 3D desaparecerá debido al tiempo excesivo que paso la herramienta sin detectarlo.

```
        else:
            self.tiempono=time.time()
            self.tiempodif=self.tiempono-self.tiemposi
            if self.tiempodif <= 0.15:
                model_matrix = self.pre_extrinsicMatrix
```

A continuación, se mostrarán algunos resultados de la detección del marcador, con y sin filtro.
Para la **primera prueba** se aplica movimiento y rotación en el marcador mientras se mantiene la cámara estática, durante un periodo de 60 segundos.

Conteo de fallos de detección sin filtro	Conteo de fallos de detección con filtro	% de mejora
79	15	81.01%

Tabla 2. Resultados de primera prueba de filtro de detección de marcadores.

Para la **segunda prueba** se movió y rotó la cámara mientras se mantenía el marcador estático, durante un periodo de 60 segundos.

Conteo de fallos de detección sin filtro	Conteo de fallos de detección con filtro	% de mejora
29	5	82.75%

Tabla 3. Resultados de segunda prueba de filtro de detección de marcadores.

Para la **tercera prueba** se aplicaron movimientos y rotaciones tanto en el marcador como en la cámara, durante un periodo de 60 segundos.

Conteo de fallos de detección sin filtro	Conteo de fallos de detección con filtro	% de mejora
122	21	82.78%

Tabla 4. Resultados de tercera prueba de filtro de detección de marcadores.

Para ver Tablas completas ir a Anexo 1. La aplicación de este código corrige la intermitencia de detección del marcador en más del 80%, por lo que el usuario final podrá visualizar de forma más estable los elementos 3D.

6.1.5 Diseño de interfaz con funciones básicas.

Para la elaboración de la interfaz se implementa PyQt5, el cual se define como un conjunto de herramientas para widgets de interfaz gráfica de usuario (GUI). PyQt es el producto de la combinación del lenguaje Python y la biblioteca Qt. Con el apoyo de Qt Designer. Mediante este gadget podemos construir una GUI, y luego obtener el código Python para esa GUI, es compatible con todas las plataformas, incluidas Windows, macOS y UNIX. Se puede utilizar para crear GUI elegantes, un marco de Python moderno y portátil (*Python, 2020*). PyQt ofrece las siguientes características:

- Versatilidad de codificación: la programación GUI con Qt se basa en la idea de señales y ranuras para crear contacto entre objetos. Esto permite versatilidad en el tratamiento de incidentes de GUI, lo que da como resultado una base de código más fluida.
- Más que un marco: Qt utiliza una amplia variedad de API de plataforma nativa para redes, desarrollo de bases de datos y más. Proporciona acceso primario a ellos a través de una API especial.
- Varios componentes de la interfaz de usuario: Qt proporciona múltiples widgets, como botones o menús, todos diseñados con una interfaz básica para todas las plataformas compatibles.
- Varios recursos de aprendizaje: como PyQt es uno de los sistemas de interfaz de usuario más utilizados para Python, puede acceder cómodamente a una amplia variedad de documentación.

En primer lugar, se realizó una prueba para incorporar una interfaz básica con el programa base de openCV.

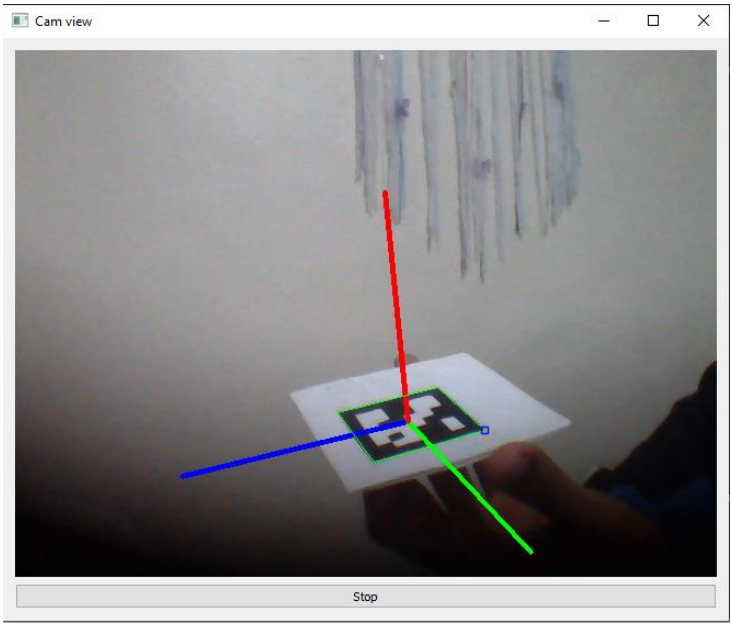


Figura 32. Programa de estimación de pose openCV con interfaz.

Al realizar esta prueba se observó que la fluidez de la captura de video es muy lenta, por lo que se determina, que proyectar la imagen de la captura de la cámara dentro de la interfaz no es funcional para el proyecto. Se toma así la decisión de separar la interfaz en dos ventanas, en la primera se visualizará una barra de herramientas (tool bar) donde estarán todas las funciones y controles para manejar e interactuar con la herramienta y como segunda ventana la visualización de openGL.

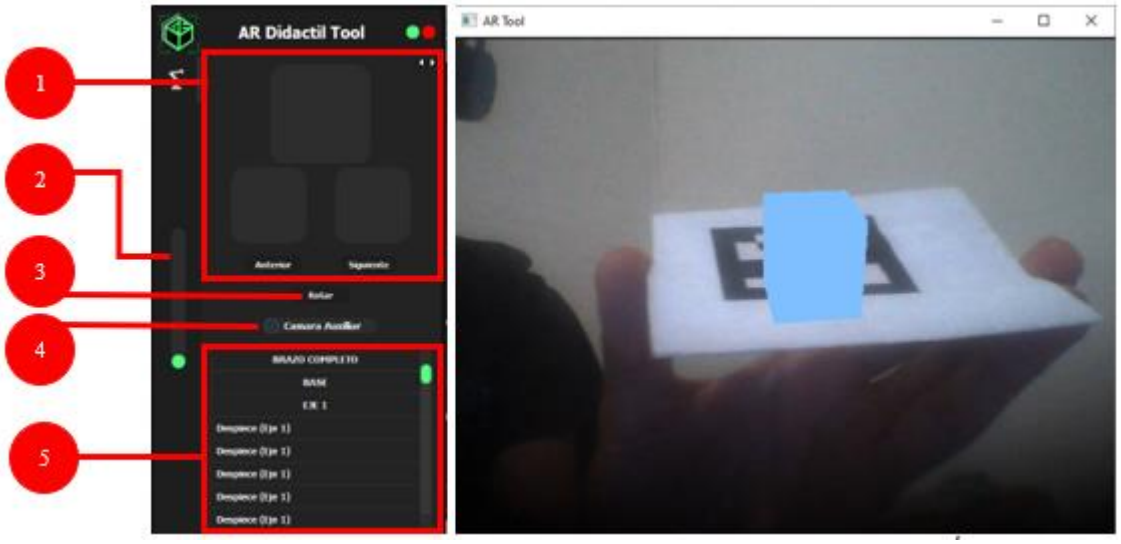


Figura 33. Interfaz Básica.

En la figura anterior se observa la interfaz con dos ventanas, la ventana de visualización y la ventana de herramientas en la cual se encuentran las funciones básicas, en donde:

1. Sección de cambio de modelos mediante botones anterior y siguiente, además de la previsualización de estos.
2. Slider de escala.
3. Botón de rotación.
4. Botón para cambio de cámara.
5. Lista de modelos cargados en la herramienta.

6.2 Modelos del brazo robótico.

6.2.1 Requerimientos y parámetros del brazo robótico.

La implementación de los brazos robóticos ayuda a tener precisión, fuerza y velocidad en labores complicadas para el humano, la Norma Estándar ISO 8373 define el Robot Industrial como un manipulador, reprogramable, multifuncional, con varios grados de libertad, capaz de manipular materiales, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas generalmente en ambientes industriales. (Barrera, 2017)

Estos son una maquina capaz de realizar actualmente actividades propias de la mano humana, el brazo necesita poder hacer al menos 6 movimientos diferentes. Estos movimientos diferentes son denominados grados de libertad. Existen robots con 4 o 5 grados de libertad, pero tienen limitaciones para colocar los objetos en todo su espacio de trabajo. El grado de libertad, o DOF, es el número de parámetros independientes que fija la situación de un efector final. Es un término muy importante de entender. Cada grado de libertad es una articulación del brazo, un lugar donde se puede doblar, rotar o trasladar. Por lo general, puede identificar el número de grados de libertad por el número de actuadores en el brazo del robot, estos grados simulan los movimientos del brazo humano. (Elouafiq, 2020)

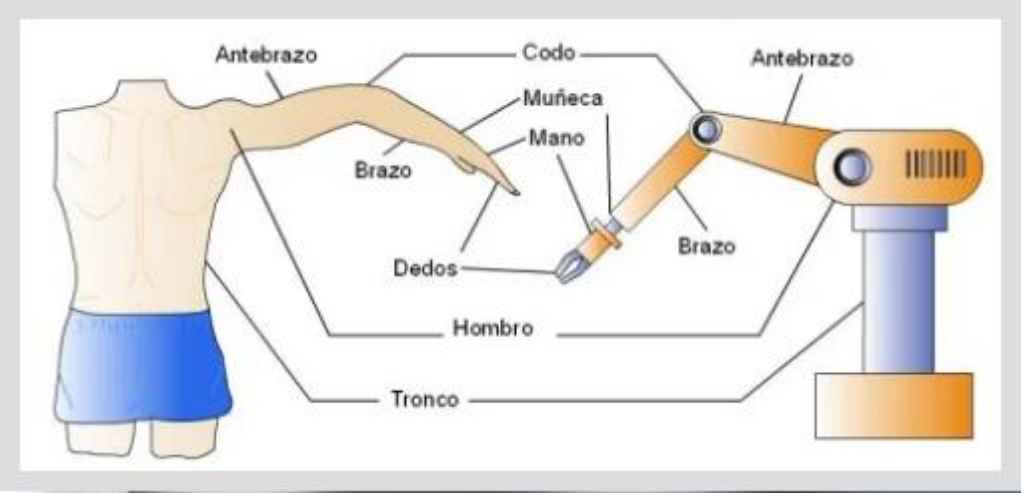


Figura 34. Partes brazo humano y robótico. (Barrera, 2017)

En la figura se observa como las partes del brazo robótico se inspiran en el brazo humano, teniendo en cuenta esto se observa que se tienen 7 articulaciones básicas en un brazo humano, se puede decir que la mayoría de los brazos industriales cuentan como mínimo con 6 grados de libertad, ya que con estos tienen un rango completo de movimiento para realizar las distintas funciones en la industria; el primer parámetro en la búsqueda del brazo robótico a incorporar en la herramienta debe contar como mínimo con 6 grados de libertad siendo un modelo representativo de lo que hay en la industria.

Por otro lado, en el despiece se visualizará la estructura mecánica del brazo y teniendo en cuenta que el proyecto está enfocado como una herramienta que sea de ayuda en la enseñanza/aprendizaje, las otras especificaciones son de nivel académico para poder observar distintos conceptos usando el modelo, como el movimiento de los grados de libertad, trenes transmisión, acoples mecánicos, los tipos de engranaje; permitiendo acercar a los estudiantes a un entorno real lo que es parte del objetivo del proyecto. Es por ello que el brazo robótico debe contar con una estructura mecánica variada que permita observar todo este tipo de factores y están bien fundamental contar con la hoja de especificaciones del brazo robótico.

6.2.2 Selección del modelo 3D del brazo robótico.

Previo al planteamiento del proyecto, se contactó al docente Oscar Gabriel Espejo Mujica de Ingeniería en Control de la Universidad Distrital Francisco José de Caldas, y se le puso en contexto sobre el proyecto y la posible aplicación de este en la materia de sistemas mecatrónicos I, el docente se vio interesado y se mostró dispuesto a colaborar con la puesta a punto de la herramienta para poder ser aplicada en un espacio de laboratorio de ingeniería en control para la materia de sistemas mecatrónicos I.

Fue necesario entonces, realizar varias sesiones virtuales en conjunto con el profesor, para establecer las características que la herramienta y los modelos debían tener. En primer lugar, fue esencial el realizar la selección del modelo 3D que cumpliera en su conjunto con los parámetros especificados para su instrucción en un entorno de enseñanza aprendizaje.

Inicialmente se desarrolló un proceso investigativo en torno al brazo robótico Mitsubishi Move Master RV-M1, el cual se encuentra en la universidad en la dependencia de laboratorios y talleres de mecánica, específicamente en el laboratorio de robótica y CNC (Laboratorios y talleres de mecánica, 2018) por lo que se solicitó información acerca de este a los laboratoristas.

Figura 35. Brazo robótico Mitsubishi Move Master RV-M1 ubicado en el laboratorio de Robótica y CNC de la facultad tecnológica en la Universidad Distrital Francisco José de Caldas. (Laboratorios y talleres de mecánica, 2018)

El brazo robótico RV M1, es un elemento capaz de realizar e interpolar 5 grados de libertad (más la pinza) para desempeñar movimientos automatizados buscando desarrollar procesos conforme a los sistemas de manufactura flexible (Laboratorios y talleres de mecánica, 2018).

Este modelo inicialmente resulto atractivo pues se encuentra disponible en físico en la universidad y el aplicativo pudo servir como soporte para la explicación de este en los laboratorios. Sin embargo, no fue posible incluirlo en la herramienta pues se encontraron inconvenientes:

- Debido a que las instalaciones de la Universidad Distrital se encontraban cerradas como medida de prevención frente a la pandemia provocada por el Covid-19, no fue posible generar los modelos 3D, tanto del brazo robótico como de los elementos internos directamente, dejando abierta la posibilidad de obtenerlos mediante observación basada en fotografías de diversos documentos, lo cual no era viable, o, obtener los modelos en línea, sin embargo solo fue posible tener acceso a modelos básicos, en donde se representaba la carcasa externa y no sus elementos internos ni trenes de transmisión, factores importantes para la selección y uso del brazo robótico en el aplicativo desarrollado.

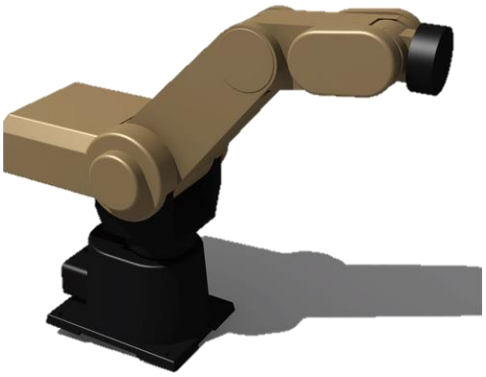


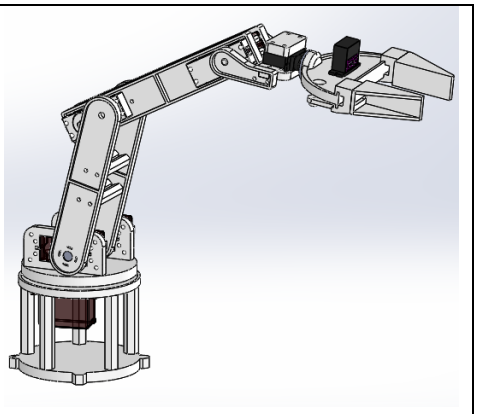
Figura 36. Modelo 3D del brazo robótico Mitsubishi MOVEMASTER RV-M1. (Przemyslaw, 2018)

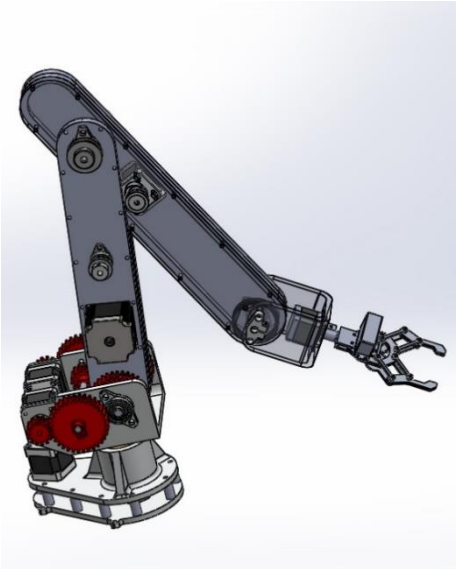
- Limitación en cuanto a características y parámetros que el brazo robótico ofrece, pues, posee únicamente 5 grados de libertad, los trenes de transmisión están compuestos en mayor parte por poleas, lo cual deja poca variedad de elementos para su análisis mediante el uso de la herramienta.

Figura 37. Distribucion de elementos internos y trenes de engranaje del brazo robótico Mitsubishi MOVEMASTER RV-M1. (Gomez y Bustos, 2018)

Debido a esto fue preciso extender la búsqueda de modelos que cubriera con algunos de los requerimientos planteados anteriormente, obteniendo así seis opciones con características diferentes, de las cuales solo se escogió uno para su uso en la herramienta.

	Descripción General	**Características**	**Imagen de modelo 3D**
1	Este modelo fue realizado en SolidWorks 2018. Los archivos tienen	Posee 6 grados de libertad.	

	diseños como partes, documentos de dibujo, Pdf, JPEG y dibujo electrónico (Ashraf, 2021).	Cuenta con pinza. Se observa distribución de los servomotores que controlan los grados de libertad. Modelo de carcasa simple. Carece de trenes de transmisión. No cuenta con hoja de especificaciones.	
2	El FANUC ArcMate 100iB es un robot servoaccionado de construcción modular de seis ejes. Este brazo robótico fue diseñado para su uso en alta velocidad. Compacto y flexible, el ArcMate 100iB es capaz de soldar y cortar a alta velocidad, pero lo suficientemente preciso como para manejar aplicaciones de pick and place (Baird, 2012).	Posee 6 grados de libertad. No cuenta con pinza. Se observa la distribución de 5 de los 6 servomotores que controlan los grados de libertad. Modelo de carcasa externa complejo, basado en el modelo original. Carece de trenes de transmisión. Cuenta con hoja de especificaciones (RobotWorx, 2020).	
3	El KUKA KR16 es un brazo robótico industrial de 6 grados de libertad. Nota: Cuenta con una pinza de mandíbula integrada (Yagmur, 2020).	Posee 6 grados de libertad. Cuenta con pinza. Se observa la distribución de los servomotores que	

		controlan los grados de libertad. Modelo de carcasa externa complejo, basado en el modelo original. Carece de trenes de transmisión. Cuenta con hoja de especificaciones (Kuka, 2020).	
4	Brazo robótico de uso académico, diseñado por Alper Cengiz con 5 grados de libertad y pinza (Cengiz, 2020).	Posee 5 grados de libertad. Cuenta con pinza. Se observa la distribución de los servomotores que controlan los grados de libertad. Modelo de carcasa simple. Cuenta con trenes de transmisión haciendo uso de engranajes y poleas simples. No cuenta con hoja de especificaciones	

5	El modelo KUKA KR360-2 (Kesici, 2017) Los primeros robots del mundo destinados a elevar cargas pesadas destacan por su corpulencia: gracias a sus potentes motores y sus reductores de alto rendimiento estos robots son capaces de manipular con ligereza piezas de gran peso de hasta 360 kg (KUKA, 2020).	Posee 6 grados de libertad. No cuenta con pinza. Se observa la distribución de los servomotores que controlan los grados de libertad. Modelo de carcasa externa complejo, obtenido de la página oficial de KUKA (*Centro de descargas, 2020*). Cuenta con trenes de transmisión complejos en todos los ejes del brazo robótico. haciendo uso de diferentes tipos de engranajes y poleas. Cuenta con hoja de especificaciones.	
6	Walter es un robot industrial vintage con 6 grados de libertad. Aún se encuentra en desarrollo y cuenta con toda la documentación necesaria para que pueda ser hecho por cualquier persona interesada (*Walter, 2020*).	Posee 6 grados de libertad. Cuenta con pinza. Se observa la distribución de los servomotores que controlan los grados de libertad. Modelo de carcasa externa diseñado en AutoCAD. Cuenta con trenes de transmisión simples en	

		los ejes del brazo robótico. haciendo uso de diferentes tipos de engranajes. No cuenta con hoja de especificaciones.	

Tabla 5. Lista de opciones de modelos 3D de brazo robótico con diferentes características y parámetros.

6.3 Acondicionamiento y carga de modelos.

6.3.1 Acondicionamiento de los modelos.

Los modelos 3D del brazo robótico seleccionado KUKA KR360-2, son muy complejos, pues cuenta con muchos elementos y cada uno de ellos posee una elaboración compleja, lo cual hace que sean muy pesados para la herramienta, por eso para incluir los archivos **OBJ** correctamente es necesario realizarles un acondicionamiento, enfocado a la solución de dos problemas diferentes:

- Tiempo de carga excesivo.

Este problema se debe a la complejidad que posee el brazo robótico escogido y la cantidad de elementos que lo componen. Con el fin de disminuir los tiempos de carga de los modelos 3D se borran piezas que no representen mayor importancia como lo son los tornillos, y a los elementos restantes se les realiza un proceso de reducción de bordes, caras y vértices, con ayuda del programa Blender. Para ilustrar el proceso se toma como ejemplo el modelo de carcasa del eje 3 del brazo robótico, el cual sin aplicar ningún modificador en el programa posee 50.782 vértices y 101.703 caras y bordes, y al exportarlo el archivo OBJ pesa 7.24MB lo cual es excesivo teniendo en cuenta la cantidad de modelos que se planea cargar en la herramienta.

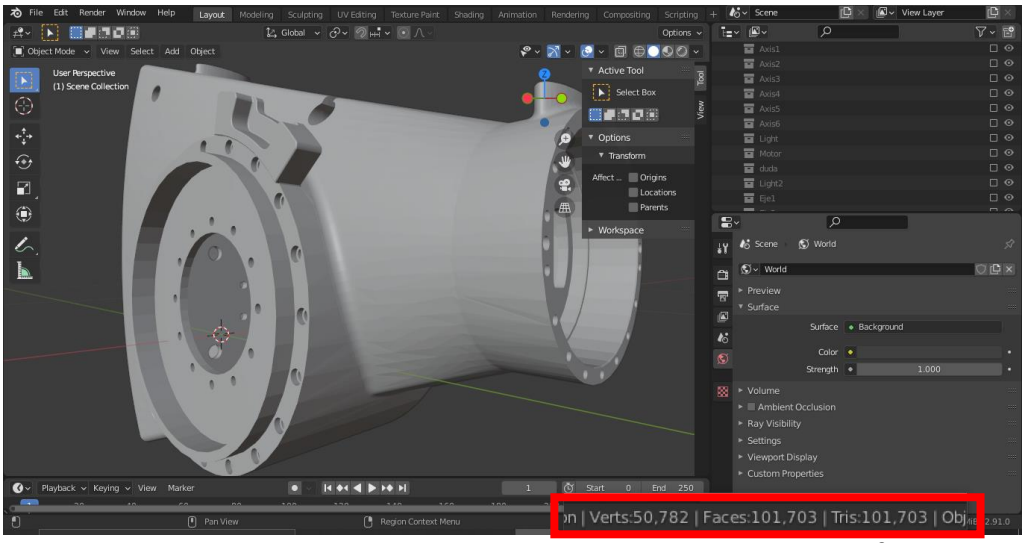


Figura 38. Modelo 3D de carcasa del eje 3 en Blender sin modificador.

Para reducir la cantidad de caras, bordes y vértices que componen los modelos se usa el modificador Diezmar (Decimate), al aplicar este modificador como efecto negativo se tiene cambios en las aristas que componen el modelo, razón por la cual no se debe exceder su uso o de lo contrario los modelos presentarían deformaciones. En el proceso se pasa de 50.782 vértices a 4.462 y 101.703 caras y bordes a solo 9.065, y en cuanto a su peso como archivo OBJ pasa de 7.24MB a 0.98MB. Con estos valores el modelo pierde muy poco detalle.

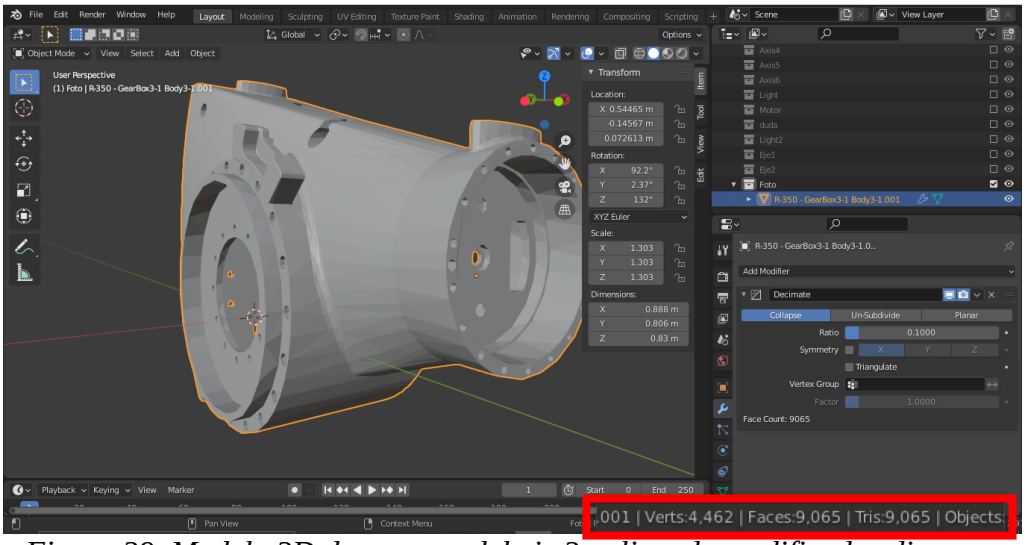


Figura 39. Modelo 3D de carcasa del eje 3 aplicando modificador diezmar.

Al aplicar este modificador sobre los 49 elementos 3D e incluirlos en la herramienta de realidad aumentada, se pasa de 5 minutos y 32 segundos a solo 39 segundos en tiempos de carga lo cual supone una reducción de aproximadamente el 90%.

- Sombreado y materiales.

Por otro lado, al momento de cargar los modelos 3D se observó que los elementos incluidos no generan sombras y los materiales de cada una de las piezas no son representadas adecuadamente en el entorno 3D de OpenGL.

Inicialmente se planteó el generar fuentes de iluminación para corregir esto, mediante el comando ***glLightfv()***, sin embargo, las sombras y su mapeo resultan ser sensibles al movimiento de la cámara y del marcador, generando intermitencias en su visualización, sin mencionar la carga extra que se genera en su procesamiento. Razón por la cual se aplicó una técnica utilizada en el procesamiento de escenas 3D y videojuegos llamada "Bake", este método generalmente se refiere al proceso de grabar en una imagen, aspectos de las características del Material de un modelo, Permitiendo crear mapeos de iluminación, sombreado, colores y texturas (*Render Baking, 2021*). Esto se realizó sobre todos los elementos que se incluyen en la herramienta mediante Blender.

Para ilustrar el antes y después del "**Bakeado**" se toma como ejemplo el modelo de carcasa del eje 1 del brazo robótico.

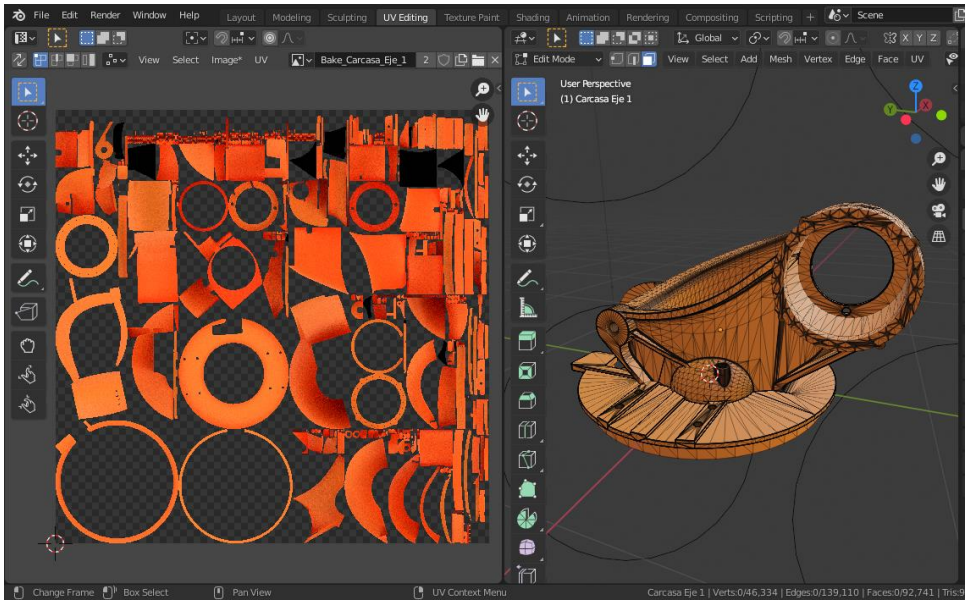


Figura 40. Generación de imagen con Bake y mapeo UV del modelo 3D de la carcasa del eje 1.

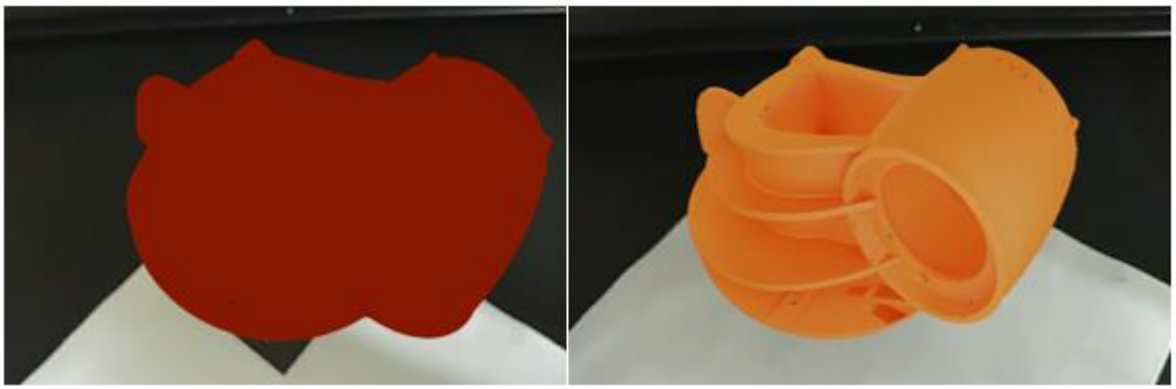


Figura 41. A la izquierda el Modelo 3D de carcasa del eje 1 con error de material y textura y a la derecha el modelo 3D de carcasa del eje 1 con materiales y texturas cargadas en la herramienta.

6.3.2 Carga de modelos del brazo robótico en la herramienta.

La herramienta posee un total de 49 modelos 3D diferentes, los cuales componen el brazo robótico. En donde, al momento de realizar el despiece se va cambiando su visualización sobre el marcador, entre elementos complejos como la estructura de un eje completo, transmisiones y piezas más simples como puede ser un engranaje y su eje.

Para esto se creó la variable **selectmodel**, la cual determina qué conjunto de elementos o pieza del brazo robótico se desea visualizar. Es aquí donde surgen dos posibilidades:

- Cargar un único modelo o pieza sobre el marcador.
- Cargar múltiples modelos sobre el marcador.

6.3.2.1 Carga de un único modelo o pieza.

Este procedimiento es el más simple ya que solo se va a visualizar un modelo a la vez sobre el marcador, haciendo uso de las variables:

- **self.selectmodel**, variable que determina la posición en la lista de elementos.
- **self.model**, Variable que será ejecutada en la lista de visualización.
- **self.model**, Variable en donde se carga cada uno de los modelos 3D, en donde # corresponde a un valor entre 1 y 49.

Y utilizando la función **glCallList(),** la cual se encarga de ejecutar una lista de visualización (*glCallList,2020*).

Esta función maneja el parámetro *list* que Especifica el nombre entero de la lista de visualización del modelo que se ejecutará (*glCallList,2020*).

$$glRCallList(list) \quad (26)$$

Un ejemplo de su implementación sería con el modelo del motor que posee el brazo robótico en sus ejes 1, 2, y 3, el cual está cargado en la variable **self. model3**.

Figura 42. Modelo 3D del motor Kuka Roboter 69-225-464 de 6.6kw.

Primero se establece la estructura condicional para identificar la posición del motor en la lista, si esta se cumple el programa ejecutara la sentencia en la que se asigna la variable **self.model3** a **self.model.**

```
if self.selectmodel==5 or self.selectmodel==12 or self.selectmodel==24:
            self.model =self.model3
```

Modelo	Descripción	Posición en la lista
BRAZO COMPLETO	KUKA KR 360-2	0
BASE	Soporte - Conexión al Controlador	1
EJE 1	±185°	2
Carcasa (Eje1)	IP 65	3
Transmisión (Eje 1)	DOF 1	4
Motor (Eje 1)	Kuka Roboter 69-225-464 6.6kw	5
Engranaje 1 (Eje 1)	Dp=48mm	6
Engranaje 2 (Eje 1)	Dp=120mm	7
Engranaje 3 (Eje 1)	Dp=285mm	8
EJE 2	+110°/-40°	9
Carcasa (Eje 2)	IP 65	10
Transmisión (Eje 2)	DOF 2	11

Motor (Eje 2)	Kuka Roboter 69-225-464 6.6kw	12
Carcasa Transmisión (Eje 2)	Soporte Ejes	13
Engranaje 1 (Eje 2)	Dp=61mm	14
….	….	…

Tabla 6. Ejemplo de Posición en lista de Motor de Eje 1 y Eje 2.

Ahora solo resta hacer uso de la función **glCallList** después de que se detecte el marcador Aruco, estableciendo la lista de visualización asignada a la variable **self.model.**

```
glCallList(self.model.gl_list)
```

por consiguiente, se genera el modelo del motor sobre el marcador.

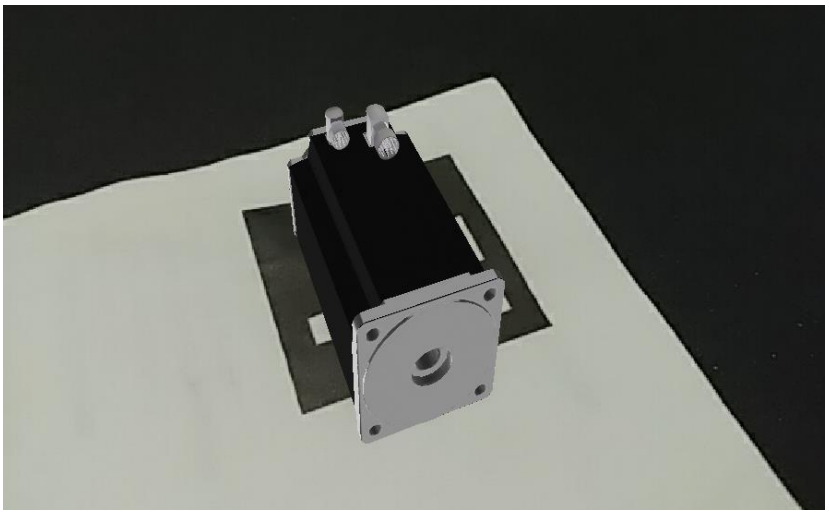


Figura 43. Carga de modelo de Motor del Eje1 sobre marcador Aruco.

6.3.2.2 Carga de múltiples modelos sobre el marcador.

El método de carga múltiple de modelos 3D permite visualizar sobre el marcador diferentes piezas o elementos, los cuales al estar combinados generan los mecanismos y transmisiones que describen los movimientos del brazo robótico en su totalidad o de cada uno de sus ejes.

Para lograr esto, se hace uso de la función (26) y de las siguientes funciones:

- **gltranslatef(),** la cual se encarga de multiplicar la matriz actual por una matriz de translación (*glTranslate, 2020*).

  Esta función maneja los siguientes parámetros:

$$glTranslatef\ (x, y, z) \tag{27}$$

En donde: (x, y, z) son las coordenadas de el vector de traslación al cual se trasladará el modelo (*glTranslate, 2020*).

- **glPushMatrix() - glPopMatrix(),** Insertan la matriz de transformación actual en la pila de matrices (*glPushMatrix, 2020*). La función glPushMatrix() guarda el sistema de coordenadas actual en la pila de matrices y glPopMatrix() restaura el sistema de coordenadas anterior. glPushMatrix() y glPopMatrix() se utilizan junto con las otras funciones de transformación y se pueden incrustar para controlar el alcance de las transformaciones (*pushMatrix, 2021*).

$$glPushMatrix\ (\ ) \tag{28}$$

$$glPopMatrix(\ ) \tag{29}$$

con estas dos funciones combinadas, es posible aplicar diferentes funciones de transformación a diferentes Objetos. Antes de llamar a las funciones de transformación de un determinado elemento, se llama glPushMatrix() Cuando se haya terminado y se desee dejar de aplicar esas transformaciones a las formas dibujadas, se llama glPopMatrix() (*pushMatrix, 2015*).

Al presionar (glPushMatrix()) y hacer estallar (glPopMatrix()) matrices, se puede controlar qué transformaciones se aplican a qué objetos, así como aplicar transformaciones a grupos de objetos y revertir fácilmente las transformaciones para que no afecten a otros objetos (*pushMatrix, 2021*)..

Es posible apreciar la aplicación de las funciones (26), (27), (28) y (29), en el Eje 1, en donde se utilizan 4 modelos diferentes y se emplean transformaciones de traslación sobre cada uno.

Primero se establece la lista de visualización usando la función (26), sobre el motor del eje 1 el cual se encuentra guardado en la variable **self.model3,** puesto que el modelo deber ser reubicado, se le aplica antes una transformación de translación usando la función (27), sin embargo ya que esta transformación solo se desea aplicar al modelo del motor, se hace uso de las funciones (28) y (29).

A cada modelo que se dibuje se le aplicaran transformaciones de traslación diferentes, por lo que cada uno se encapsulara entre (28) y (29) con sus respectivas funciones.

```
def Axis1(self):
    glPushMatrix()
    glTranslatef(0,-0.314,0)
    glCallList(self.model3.gl_list)
    glPopMatrix()
```

Tras esto, es posible continuar con la distribución de los engranajes que componen el eje1, por lo que se establece ahora la lista de visualización para el modelo del primer engranaje del eje 1 (Planeta) guardado en la variable **self.model4.**

Figura 44. Modelo 3D del primer engranaje del eje 1.

```
glPushMatrix();
glCallList(self.model4.gl_list)
glPopMatrix();
```

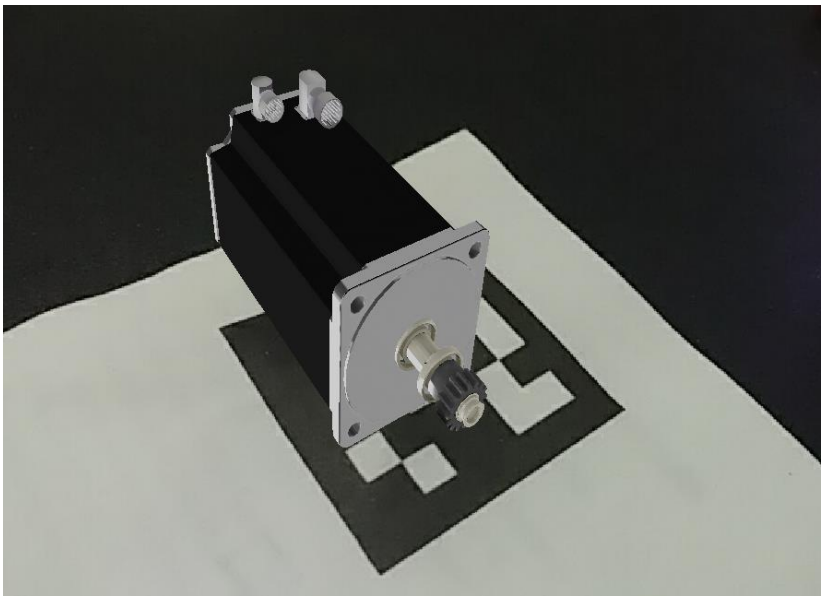


Figura 45. Modelos 3D del motor y planeta de transmisión del eje 1.

Ahora, como satélites se encuentran los modelos del segundo engranaje del eje 1, este está guardado en la variable **self.model5,** el cual será replicado tres veces con diferentes ubicaciones en los ejes **X** y **Z,** haciendo uso de la transformación de translación mediante la función (27).

Figura 46. Modelo 3D del segundo engranaje del eje 1.

```
glPushMatrix()
glTranslatef(0.124,0,0)
glCallList(self.model5.gl_list)
glPopMatrix()

glPushMatrix()
glTranslatef(-0.073,0,0.101)
glCallList(self.model5.gl_list)
glPopMatrix()

glPushMatrix()
glTranslatef(-0.057,0,-0.11)
glCallList(self.model5.gl_list)
glPopMatrix()
```

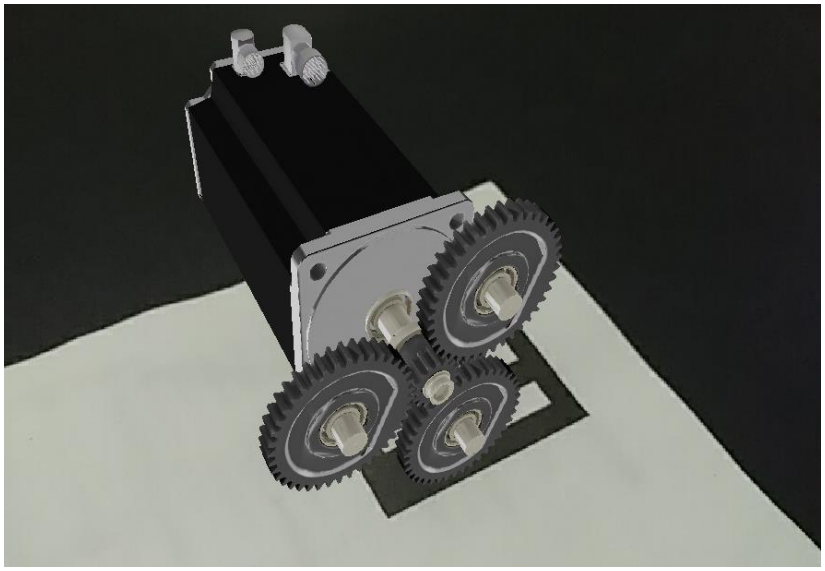


Figura 47. Modelos 3D del motor, planeta, y satélites de transmisión del eje 1.

Por ultimo como corona se establece la lista de visualización para el tercer engranaje del Eje 1, el cual se encuentra guardado en la variable **self.model6.**

Figura 48. Modelo 3D del tercer engranaje del eje 1.

```
glPushMatrix();
glCallList(self.model6.gl_list)
glPopMatrix();
```

Obteniendo como resultado la visualización de la transmisión completa del eje 1 sobre el marcador.

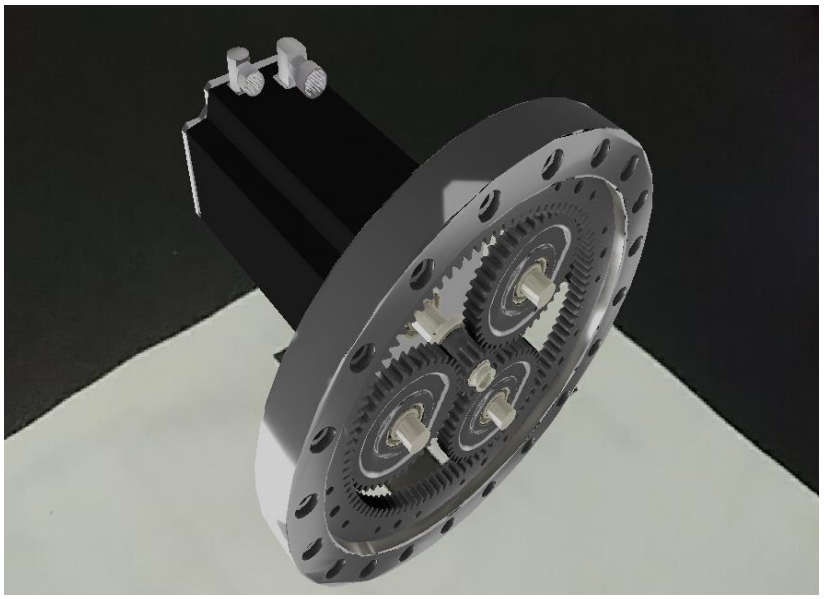


Figura 49. Modelo 3D de transmisión completa del eje 1.

6.4 Guía práctica y funciones complementarias en la herramienta.

El propósito de elaborar una guía práctica es generar actividades por medio de la herramienta. Los elementos considerados en el diseño estuvieron enfocados a la viabilidad y utilidad, todos ellos para facilitar la instrumentación de cada actividad práctica. Tener en cuenta estos factores resultará en un material didáctico que apoyará mejor el proceso enseñanza y aprendizaje. (Alemán y Mata, 2006, p.1)

Por ello, resulta imprescindible considerar los parámetros establecidos con los que debe cumplir el modelo 3D del brazo robótico, pues proporcionan información implícita, la cual es relevante para la materia de sistemas mecatrónicos I, como lo son:

- Variedad de elementos mecánicos.
- Tipos de trenes de transmisión
- Grados de libertad.

Así mismo se busca complementar los recursos que hasta este punto posee la herramienta, con el fin de entender el despiece y el funcionamiento del brazo robótico, mediante la adición de las siguientes características:

- Animación de modelos.
- Información sobre elementos visualizados.
- Posicionamiento de modelos.

La animación y el posicionamiento de los modelos están pensados para facilitar la observación y análisis de los trenes de transmisión y los elementos que lo componen, por otra parte, la información agregada a la interfaz referente a características que posee cada modelo, se incorporó con el objetivo de utilizar esta información para el desarrollo de la guía práctica, y por último la generación del ejecutable pretende facilitar la implementación del aplicativo al usuario final, pues no es necesario instalar ningún archivo adicional, solo se necesita correr el programa.

6.4.1 Animación de modelos.

La mayoría de transmisiones mecánicas funcionan a través de elementos rotativos, de modo que, para realizar la animación de un tren de engranajes, es necesario generar una rotación sobre cada engranaje de forma independiente. Para esto se utilizara la función **glRotatef()** la cual se encarga de multiplicar la matriz actual por una matriz de rotación (*glRotate, 2020*).

Esta función maneja los siguientes parámetros:

$$glRotatef\ (ángulo, x, y, z) \tag{30}$$

En donde: *ángulo* Especifica el ángulo de rotación en grados y (*x*, *y*, *z*) Especifica las coordenadas x, y y z de un vector, respectivamente (*glRotate, 2020*).

Como ejemplo se explicará la animación para el tren de engranajes del eje 1.

Lo primero a considerar es el valor capturado de los sliders de la ventana de animación, en donde para el desarrollo del tren de engranajes del eje 1 solo se utilizara el dato almacenado en la variable **angles[0],** ya que este representa el ángulo de rotación en grados para este eje.

```
angles[0] = -(self.ui.S1.value()/100)
angles[1] = -(self.ui.S2.value()/100)
angles[2] = -(self.ui.S3.value()/100)
angles[3] = -(self.ui.S4.value()/100)
angles[4] = -(self.ui.S5.value()/100)
angles[5] = -(self.ui.S6.value()/100)
```

Con el dato almacenado en la variable **angles[0]** se hace uso de la función de rotación (30), aplicando así la transformación de rotación sobre cada uno de los engranajes de forma independiente.

```
glPushMatrix()
glRotatef(angles[0], 0, 1, 0);
glCallList(self.model4.gl_list)
glPopMatrix()

glPushMatrix()
glTranslatef(0.124,0,0)
glRotatef(angles[0], 0, 1, 0);
glCallList(self.model5.gl_list)
glPopMatrix()

glPushMatrix()
glTranslatef(-0.073,0,0.101)
glRotatef(angles[0], 0, 1, 0);
glCallList(self.model5.gl_list)
glPopMatrix()

glPushMatrix()
glTranslatef(-0.057,0,-0.11)
glRotatef(angles[0], 0, 1, 0);
glCallList(self.model5.gl_list)
glPopMatrix()

glPushMatrix()
glTranslatef(0,0,0)
glRotatef(angles[0], 0, 1, 0);
glCallList(self.model6.gl_list)
glPopMatrix()
```

El código anterior especifica que se va a establecer una rotación sobre el eje **Y** en cada uno de los modelos 3D con un ángulo de rotación igual al valor de la variable **angles[0],** sin embargo el

movimiento que describe esta animación no es correcto, pues todos los engranajes se mueven a la misma velocidad y poseen el mismo sentido de giro.

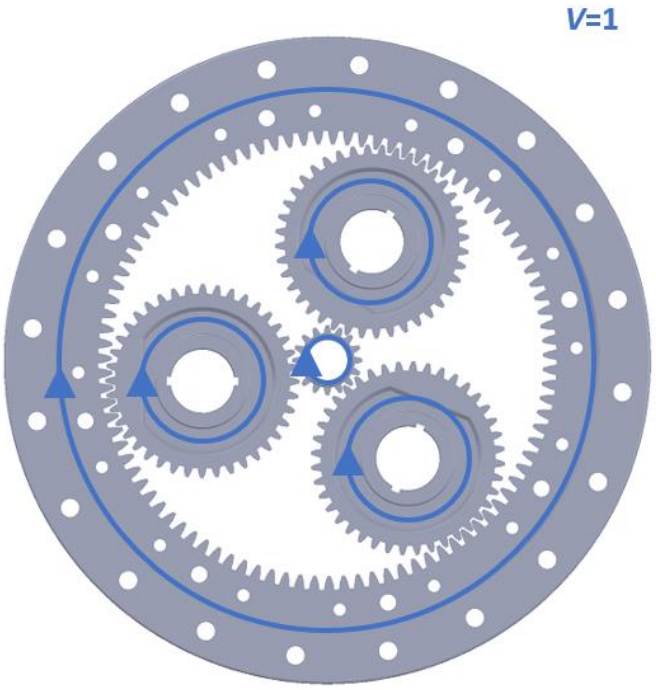


Figura 50. Ilustración de tren de engranajes del eje 1 con sentidos de giro incorrectos y relación de velocidad idéntica.

Para invertir el sentido de giro basta con multiplicar la variable **angles[0]** por **-1**, esto solo se realizara con el engranaje planeta (**self.model4**) pues los satélites y la corona del tren de transmisión poseen el sentido de giro correcto.

```
glPushMatrix()
glRotatef(-angles[0], 0, 1, 0);
glCallList(self.model4.gl_list)
glPopMatrix()
```

Para establecer la velocidad correcta es necesario calcular la relación de velocidad de los engranajes. El tren de engranajes del eje 1 es de tipo epicicloidal y está conformado por 3 tipos diferentes de engranajes rectos.

Nombre de elemento	Variable en programa	Diámetro primitivo (mm)	Numero de Dientes	Modelo 3D

Engranaje 1 (planeta)	self.model4	48	16	
Engranaje 2 (Satélite)	self.model5	120	40	
Engranaje 3 (Corona)	self.model6	285	95	

Tabla 7. Elementos que componen el tren de engranajes del eje 1.

Con los datos de la tabla superior es posible determinar la relación de velocidad en el tren de engranajes. Para ello se utiliza la siguiente ecuación (Mott et al., 2017):

$$Rv = \frac{Z2}{Z1} = \frac{D2}{D1} = \frac{W1}{W2} \tag{31}$$

Donde:

- Rv es la relación de velocidad.
- *Z1* es la cantidad de dientes del engranaje conductor.
- *Z2* es la cantidad de dientes del engranaje conducido.
- *D1* es el diámetro primitivo del engranaje conductor.
- *D2* es el diámetro primitivo del engranaje conducido.
- *W1* velocidad en rpm de engranaje conductor.
- *W2* velocidad en rpm de engranaje conducido.

Para hallar las velocidades angulares de cada uno de los engranajes es necesario usar la formula (31) así como tener en cuenta Los diámetros primitivos, los cuales se encuentran en la tabla 7. Se asigna como velocidad angular de salida 1rpm.

$$w_3 = 1 \text{ rpm} \tag{32}$$

$$w_2 = w_3 * \frac{D_3}{D_2} = 1rpm * \frac{285mm}{120mm} = 2.375 \text{ rpm} \tag{33}$$

$$w_1 = w_2 * \frac{D_2}{D_1} = 2.375rpm * \frac{120mm}{48mm} = 5.9375\ rpm \quad (34)$$

Donde w_3 es la velocidad angular de la corona, w_2 es velocidad angular de los satélites y w_1 es la velocidad angular del planeta. Los valores de w_2 y w_1 se usan en el código final de animación para el eje 1, al multiplicarlos por la variable **angles[0]** en su respectiva lista de dibujo.

```
def Axis1(self):

        glPushMatrix();
        glTranslatef(0,-0.314,0)
        glCallList(self.model3.gl_list)
        glPopMatrix();

        glPushMatrix();
        glRotatef(angles[0]*5.9375, 0, 1, 0);
        glCallList(self.model4.gl_list)
        glPopMatrix();

        glPushMatrix();
        glTranslatef(0.124,0,0)
        glRotatef(-angles[0]*2.375-1, 0, 1, 0);#3.61
        glCallList(self.model5.gl_list)
        glPopMatrix();

        glPushMatrix();
        glTranslatef(-0.073,0,0.101)
        glRotatef(-angles[0]*2.375+2, 0, 1, 0);
        glCallList(self.model5.gl_list)
        glPopMatrix();

        glPushMatrix();
        glTranslatef(-0.057,0,-0.11)
        glRotatef(-angles[0]*2.375+1, 0, 1, 0);
        glCallList(self.model5.gl_list)
        glPopMatrix();

        glPushMatrix();
        glTranslatef(0,0,0)
        glRotatef(-angles[0], 0, 1, 0);
        glCallList(self.model6.gl_list)
        glPopMatrix();
```

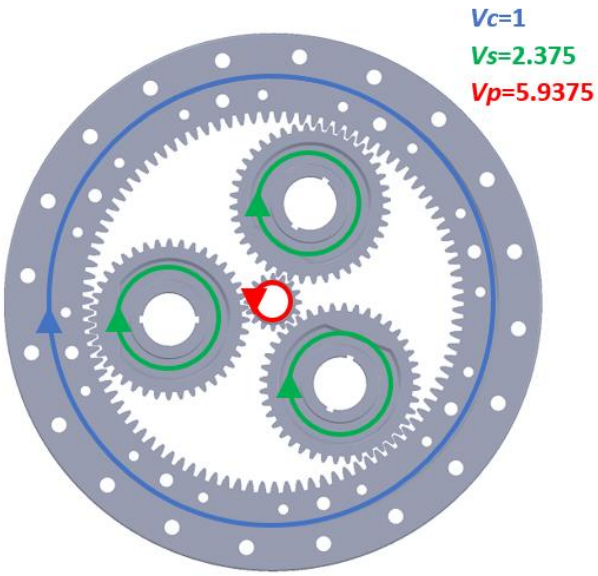


Figura 51. Ilustración de tren de engranajes del eje 1 con sentidos de giro y relaciones de velocidad corregidos.

El cálculo de relaciones de velocidad y velocidad angular para cada uno de los ejes se encuentra en el anexo 2, así como cálculos complementarios del módulo de los engranajes.

6.4.2 Información sobre elementos visualizados.

Inicialmente la herramienta y su interfaz únicamente permitía visualizar el despiece de los modelos en 3D, y no incluía ningún tipo de información referente a características que posee cada elemento del brazo robótico, como lo es el valor de diámetro primitivo para el caso de los engranajes. Razón por la cual fue imprescindible el agregar esta información, añadiendo una segunda columna a la tabla que posee la interfaz de despiece, estas descripciones de los elementos permiten, además, efectuar la guía práctica planteada en este documento ya que posee información relevante para su desarrollo.

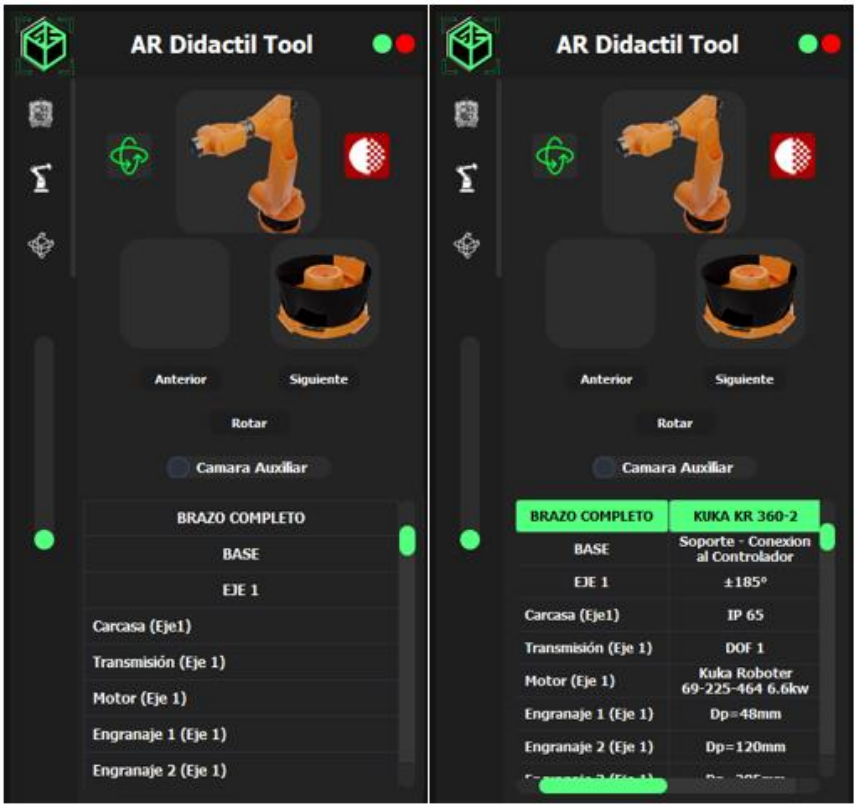


Figura 52. Cambio en tabla de interfaz para añadir descripción de modelos, engranajes cuentan con dato de Diámetro primitivo.

6.4.3 Posicionamiento de modelos

Algunos modelos 3D o agrupaciones formadas con múltiples modelos, pueden poseer una longitud considerable, por lo que al querer analizar cada pieza que los compone, o las animaciones que poseen, puede significar un problema, ya que no es posible apreciarlos en una escala o ubicación ideal, como es el caso de las transmisiones de los ejes 4, 5 y 6.
Un ejemplo es el caso del modelo del Eje 6, en donde a pesar de que el modelo se encuentra en su escala mínima (ver cuadro rojo *Figura 53*), posee un tamaño extenso, debido a la distribución de sus componentes. Desde esta perspectiva es posible apreciar el comportamiento y la ubicación de todos sus elementos en general, sin embargo, no se aprecia detalladamente la geometría o animación individual de los componentes de forma clara.

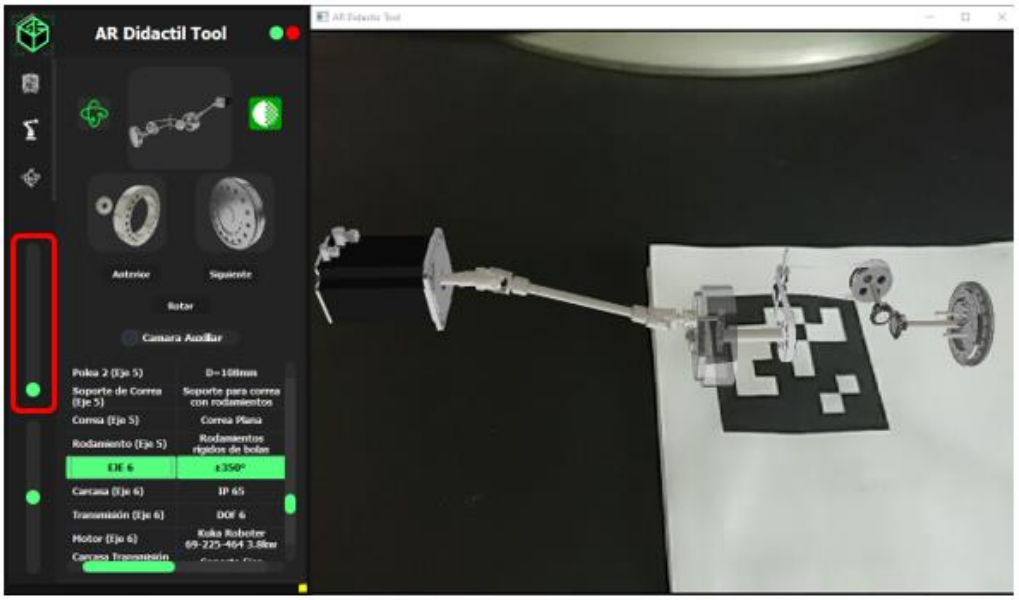


Figura 53. Conjunto de modelos 3D que componen el Eje 6 con escala mínima.

De tal forma que si se desea visualizar mejor los componentes que conforman este eje, es posible aumentar la escala al modelo (ver cuadro rojo *Figura 54*).

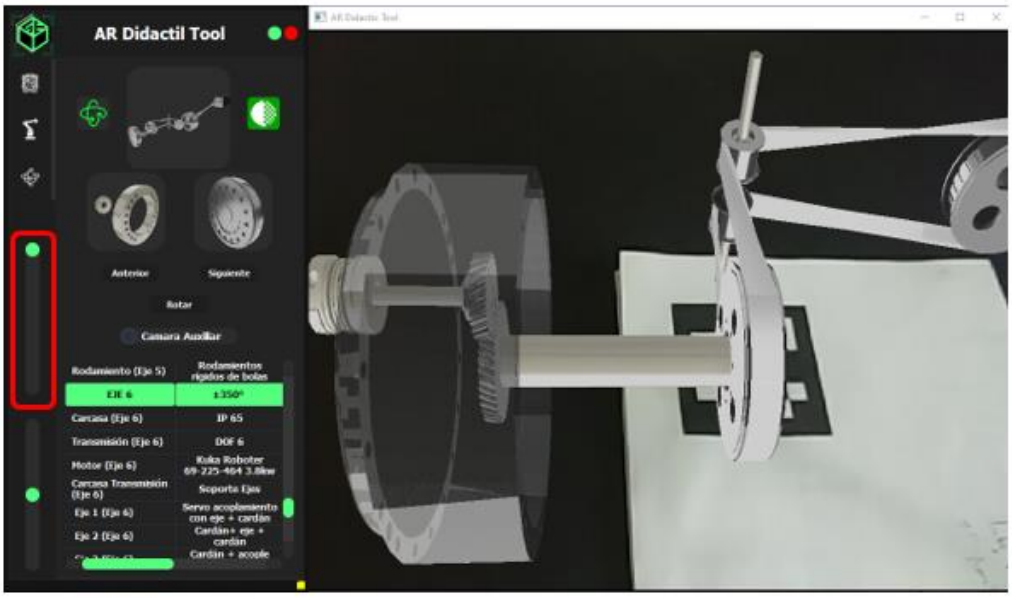


Figura 54. Conjunto de modelos 3D que componen el Eje 6 con escala Máxima.

Sin embargo, al hacer esto, gran parte del modelo queda fuera de la imagen proyectada en la pantalla, para esto se creó un segundo slider en la barra lateral izquierda (ver cuadro azul *Figura 55*) el cual, al moverlo, permite reubicar el modelo 3D que se esté visualizando sobre el marcador. Permitiendo así, observar partes del modelo que antes no era posible analizar.

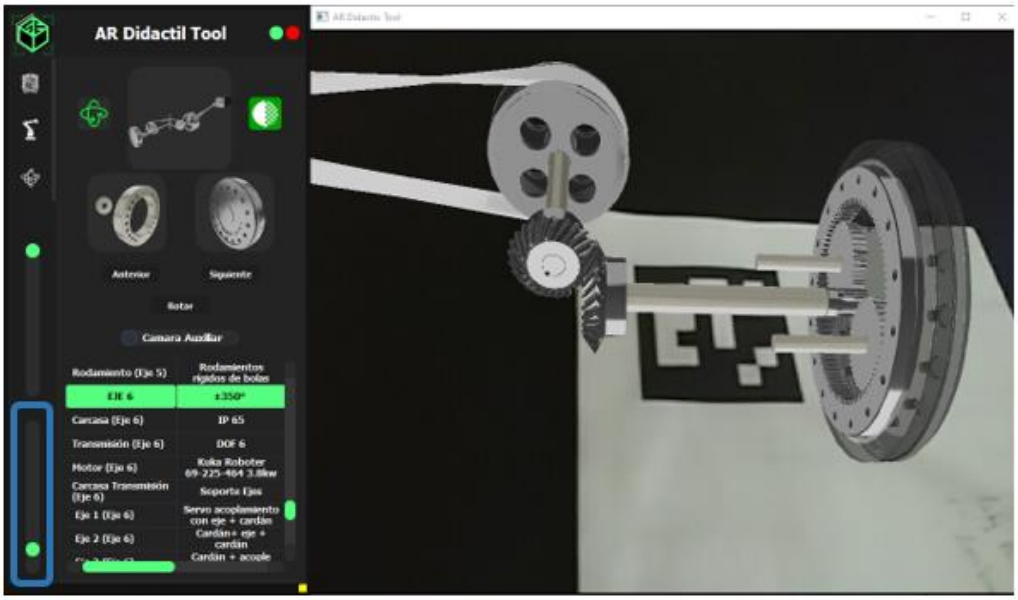


Figura 55. Conjunto de modelos 3D que componen el Eje 6 reubicado y con escala Máxima.

6.4.4 Generación de ejecutable.

Para generar el ejecutable se utiliza la librería **Auto-py-to-exe,** esta librería funciona en base a **PyInstaller**, es un complemento que brinda facilidad al programador, pues se llama desde el símbolo del sistema y mediante una interfaz de usuario se realiza la generación del ejecutable. Esta interfaz está diseñada para facilitar el proceso, donde se busca el archivo principal de la aplicación que se va generar, además de esto se puede adicionar la imagen del icono de la aplicación, algo importante para este proyecto que son archivos adicionales, donde se cargó la carpeta donde se encuentran los modelos 3D e imágenes del brazo robótico y genera una carpeta donde se encuentran los archivos necesarios para la funcionalidad del ejecutable y el ejecutable, el resultado se observa en la siguiente figura.

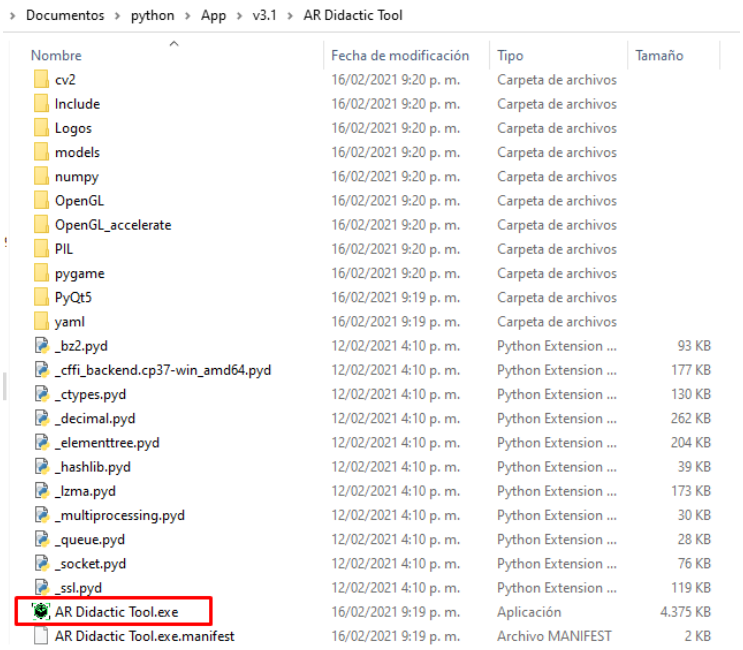


Figura 56. Carpeta contenedora de la herramienta y su ejecutable .exe.

6.4.5 Componentes y estructura de guía práctica.

Tomando como referencia la metodología citada en el documento "Guía de elaboración de un manual de prácticas de laboratorio, taller o campo: asignaturas teórico prácticas (Alemán y Mata, 2006)" para el desarrollo de actividades prácticas, es posible identificar los siguientes componentes:

- Reglamento: Guía practica
- Metodología:
- Recurso Humano: Estudiantes y docente.
- Recursos asociados: Computador, Herramienta didáctica de realidad aumentada para el despiece y visualización en 3D de un brazo robótico (AR Didactic Tool), Manual de usuario, Marcador ArUco y Guía Práctica.

Así mismo, se estipula los apartados implementados en la estructura de la guía práctica:

- Introducción: Descripción general el orden de la guía, se menciona el objetivo para el que fue diseñada y la importancia y situación en la que toma contexto para su desarrollo en la materia. (Guía para la elaboración de un manual de prácticas, 2002)
- Título de actividad.
    - Objetivo: Señala la finalidad del experimento o actividad específica. Está directamente relacionado con la demostración o comprobación práctica que se va a llevar a cabo. (Alemán y Mata, 2006)
    - Actividad.

- Metodología: Serie de indicaciones o pasos secuenciados para poder realizar la práctica de la mejor manera posible. (Guía para la elaboración de un manual de prácticas, 2002)

- Ecuaciones: Serie de fórmulas necesarias para poder resolver las actividades.
- Referencias: En este apartado se indica la bibliografía de las ecuaciones. También incluye lista de documentos sugeridos a consultar por los usuarios como complemento para el desarrollo de la guía.

6.5 Prueba de aplicación

Llegado a este punto de desarrollo, se buscó la validación previa por parte del docente, por lo que se puso a disposición del mismo para pruebas (v2.1), durante las cuales fue posible apreciar errores de rendimiento y estabilidad, pues el aplicativo tendía a bloquearse y cerrarse después de cierto tiempo de actividad.

Durante la monitorización de recursos de los equipos que corren el aplicativo fue posible concluir que estas anomalías ocurrían debido al uso excesivo de memoria RAM, esto debido a la repetición del subproceso encargado de crear la ventana de realidad aumentada en el código Python principal.

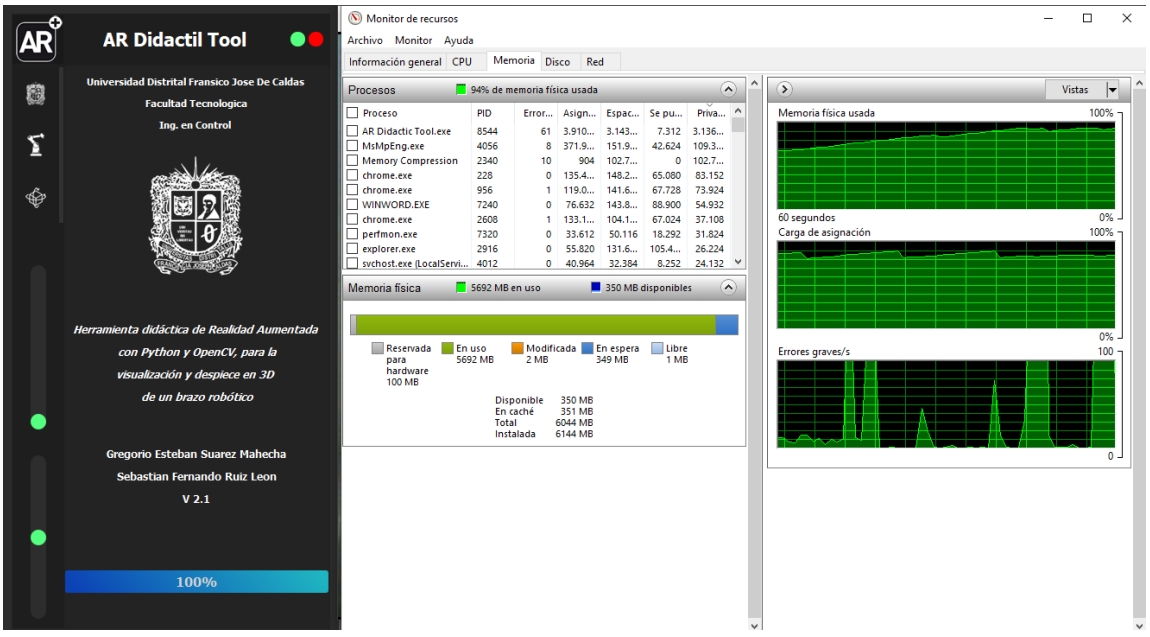


Figura 57. Acumulación de carga de asignación y memoria RAM física por ejecución de la herramienta en su versión 2.1.

Para la siguiente versión del aplicativo (v3.1) se optimizo el código referente a la función definida llamada *draw_scene*, pasando de tener múltiples ejecuciones desde el inicio a una única llamada una vez este corriendo.

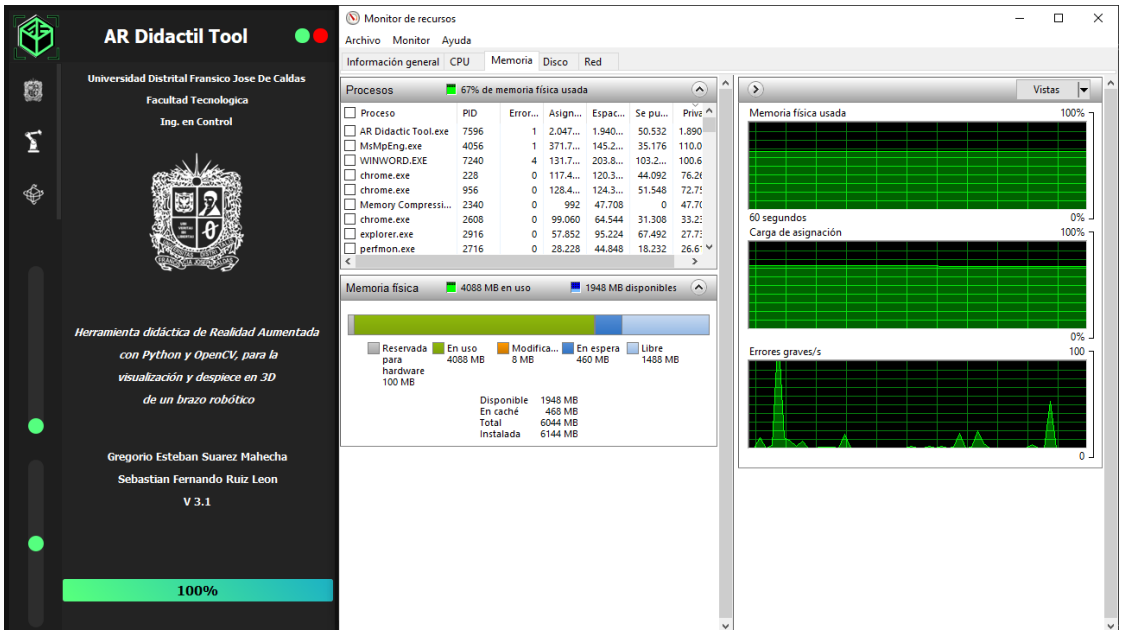


Figura 58. Uso de memoria RAM con optimización de función definida draw_scene.

7. Resultados

Se logro una herramienta didáctica de realidad aumentada, para el sistema operativo Windows, desarrollada mediante el lenguaje de programación Python y la librería OpenCV, que permite el despiece y visualización de un brazo robótico. A lo largo de este documento se muestra y se explica códigos necesarios para el desarrollo de la herramienta, desarrollados 100% en Python y utilizando OpenCV.

La herramienta posibilita la visualización y despiece de trenes de transmisión y sus componentes mecánicos, junto con las respectivas animaciones de grados de libertad de un brazo robótico, que se selecciona al evaluar y cumplir los parámetros establecidos previamente en este documento, los cuales básicamente son: debe ser un brazo robótico industrial con hoja de especificaciones y poseer 6 grados libertad, con una estructura mecánica compleja y variada en cada uno de sus componentes, trenes transmisión, acoples mecánicos y tipos de engranajes; permitiendo acercar así, a los estudiantes a un entorno real.

Es por esto que se elige el brazo KUKA KR360-2 de 6 opciones posibles indagadas, el cual es un modelo industrial destinado a trabajo de cargas pesadas, por lo mismo tiene una estructura compleja que contiene una amplia variedad de acoples, trenes de transmisión y partes mecánicas que hacen de este un modelo ejemplar para su introducción en la enseñanza.

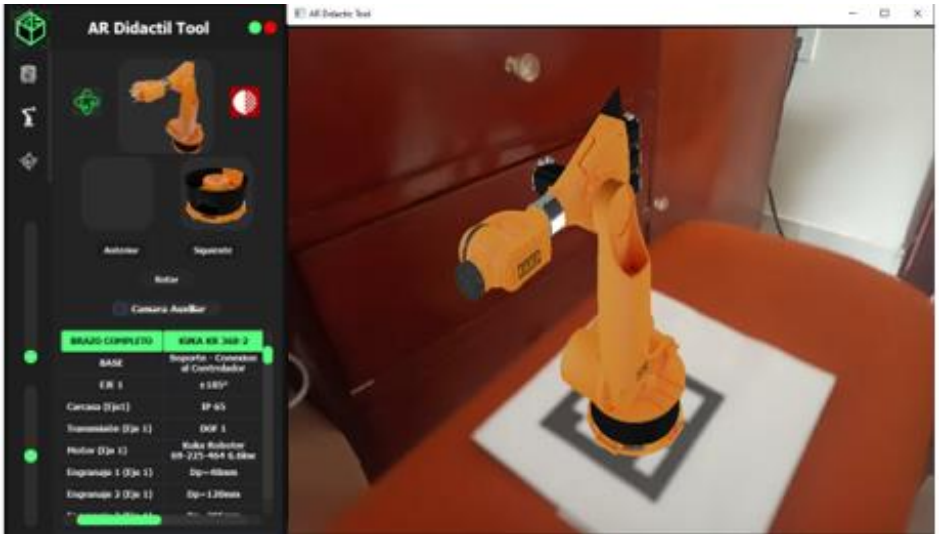


Figura 59. Visualización del brazo robótico KUKA KR 360-2 en la herramienta AR Didactil Tool.

La incorporación para su visualización y despiece en la herramienta se realiza en un orden determinado previamente, arrancando desde el brazo robótico general (este modelo es el primero cargara la herramienta), seguido de este se observa los 6 ejes que forman los grados de libertad, empezando desde la base hasta la muñeca. Cada eje cumple el siguiente orden establecido para su despiece, visualización y animación:

- Eje completo.
- Carcasa del eje.
- Transmisión.
- Motor del eje.
- Carcasa de transmisión (no todos los ejes cuentan con una).
- Engranajes.

A continuación, se presentan diversas imágenes en donde se toma como ejemplo el despiece del eje 3 del brazo robótico.

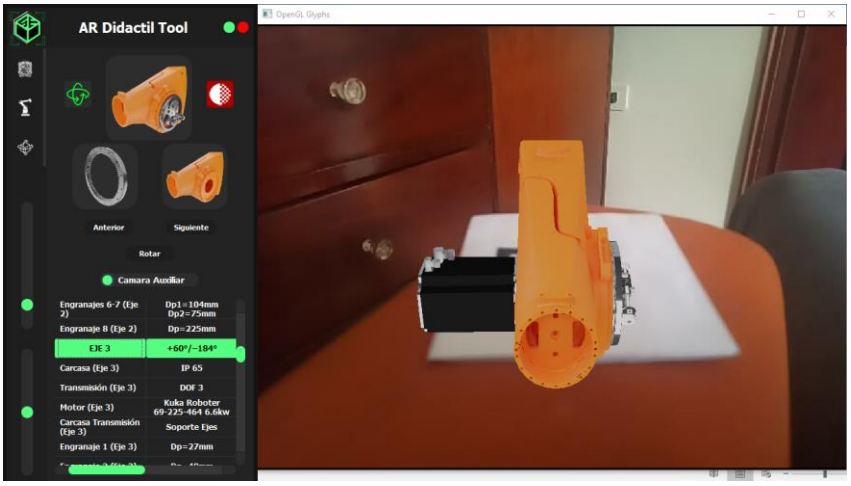


Figura 60. Visualización eje 3 completo herramienta AR Didactil Tool.

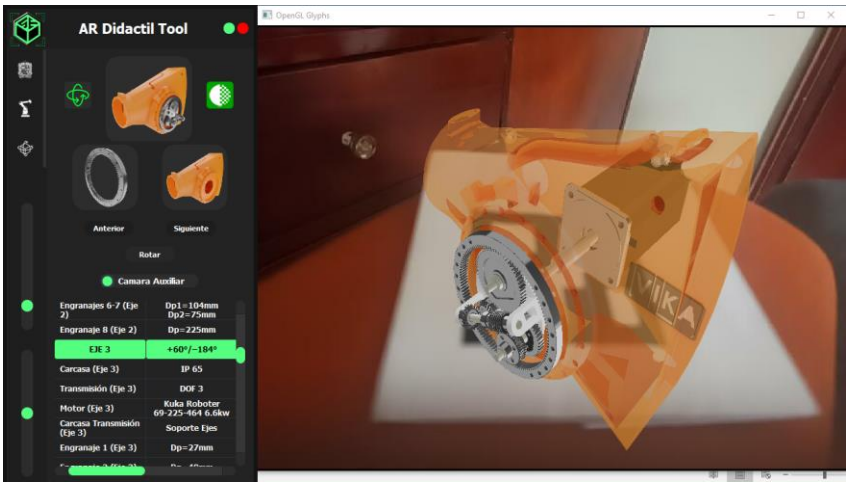


Figura 61. Visualización eje 3 con transparencia herramienta AR Didactil Tool.

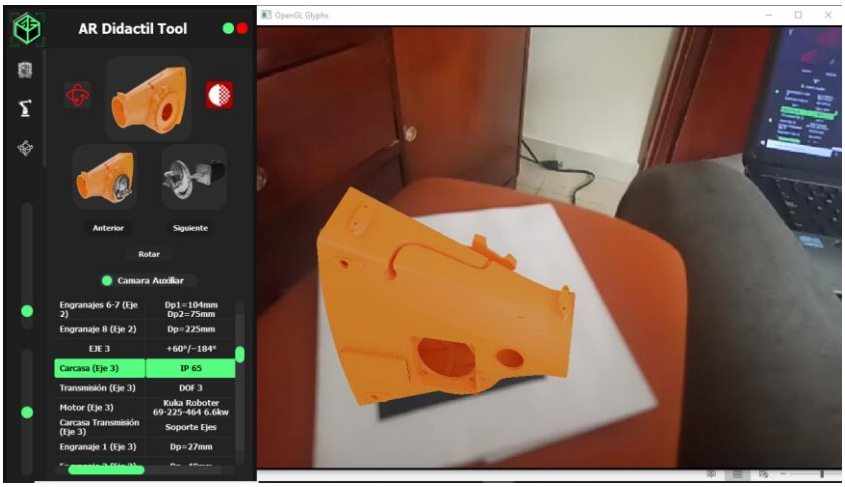


Figura 62. visualización Carcasa eje 3 herramienta AR Didactic Tool.

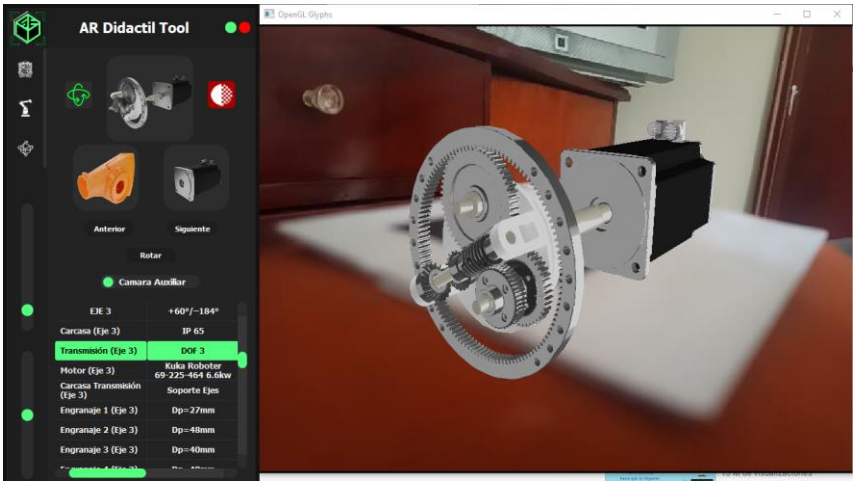


Figura 63. Visualización transmisión eje 3 herramienta AR Didactic Tool.

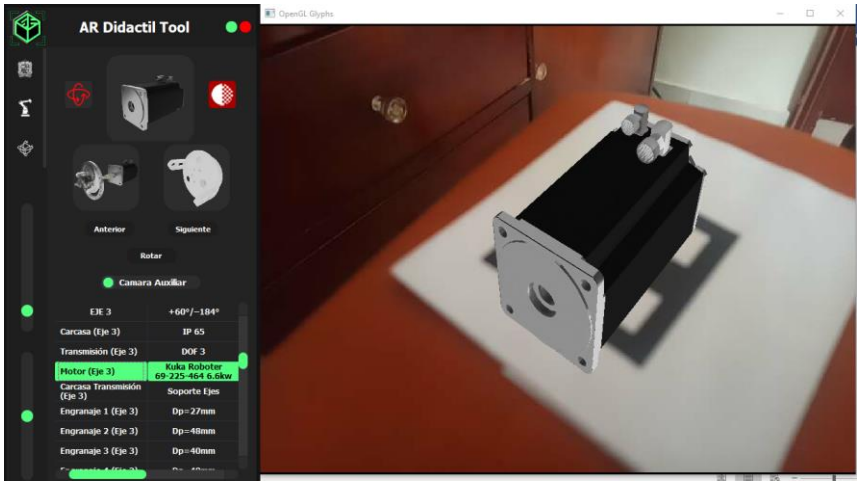


Figura 64. Visualización motor eje 3 herramienta AR Didactic Tool.

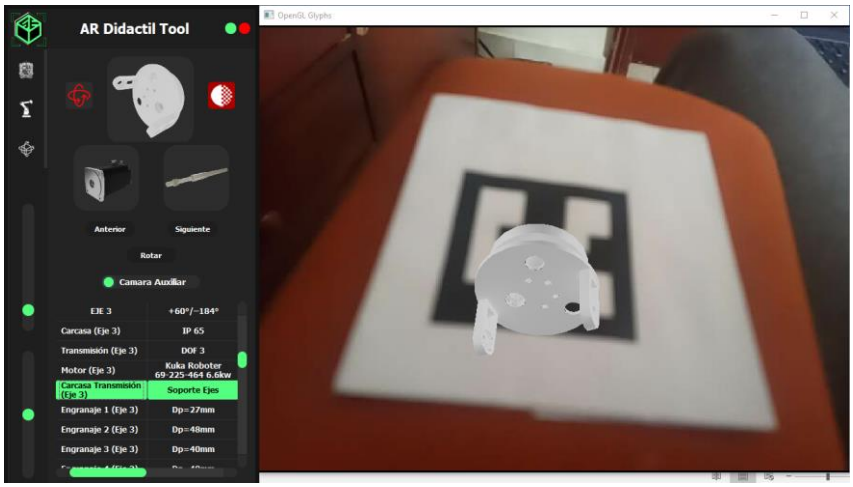


Figura 65. Visualización carcasa de transmisión eje 3 herramienta AR Didactic Tool.

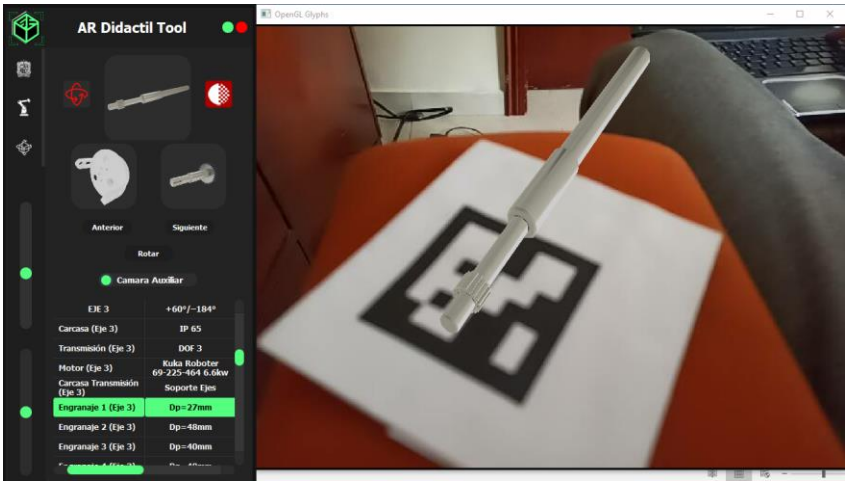


Figura 66. Visualización engranaje 1 eje 3 herramienta AR Didactic Tool.

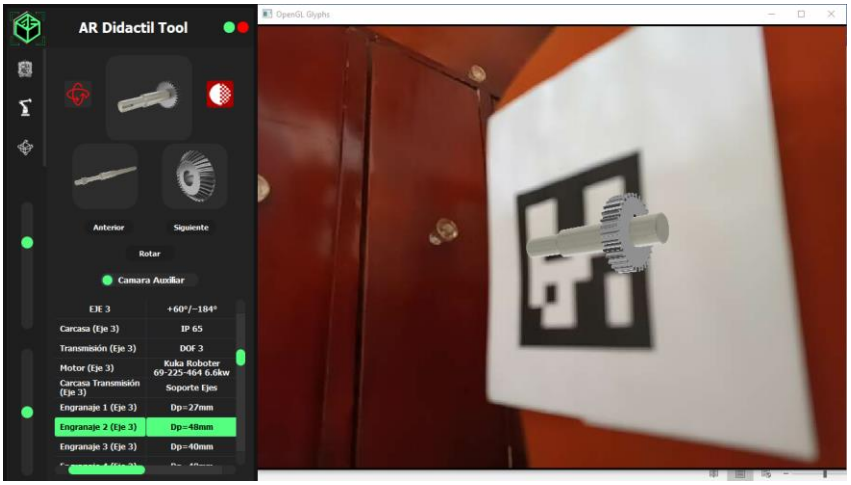


Figura 67. Visualización engranaje 2 eje 3 herramienta AR Didactic Tool.

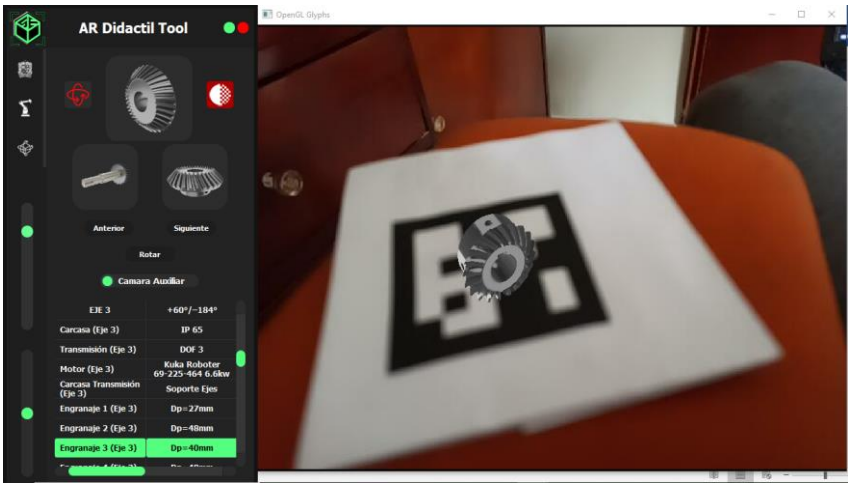


Figura 68. Visualización engranaje 3 eje 3 herramienta AR Didactic Tool.

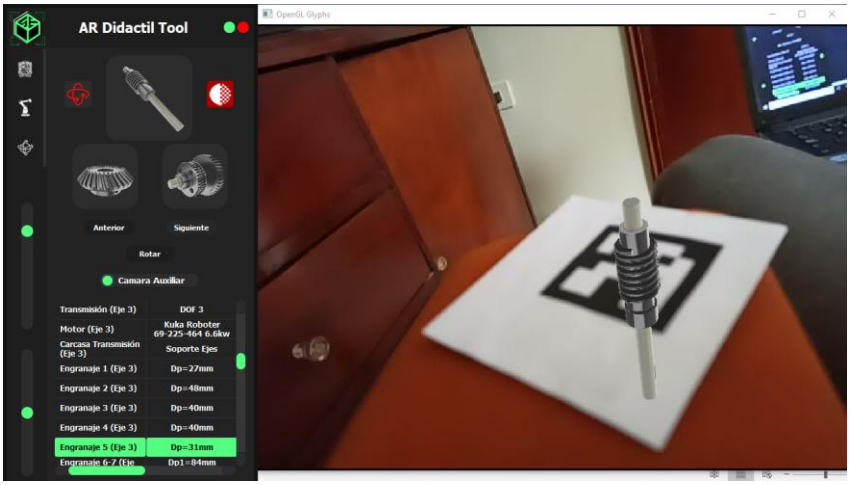


Figura 69. Visualización engranaje 5 eje 3 herramienta AR Didactic Tool.

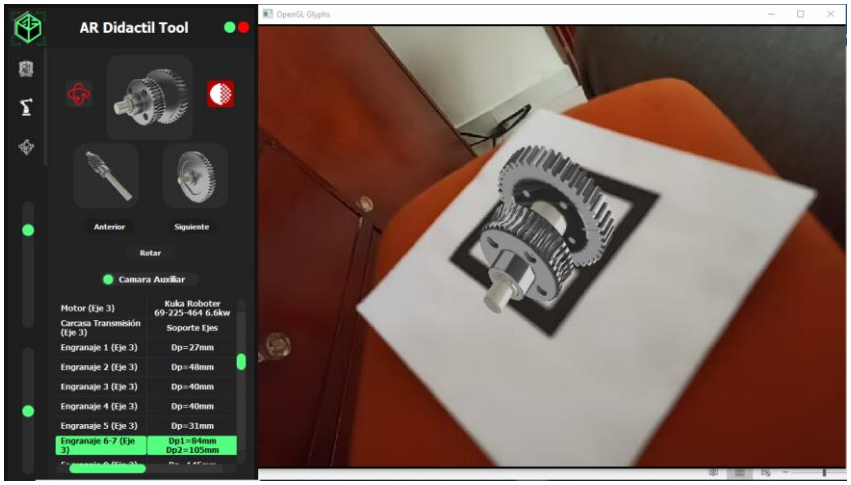


Figura 70. Visualización engranajes 6 y 7 eje 3 herramienta AR Didactic Tool.

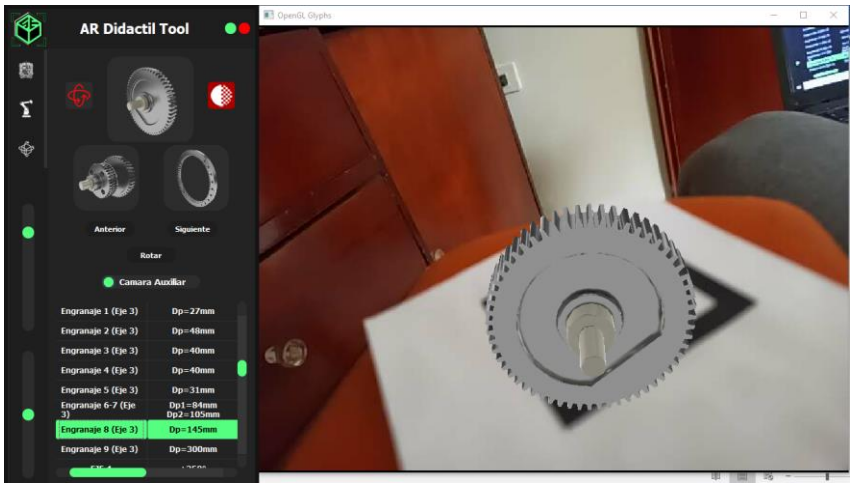


Figura 71. Visualización engranaje 8 eje 3 herramienta AR Didactic Tool.

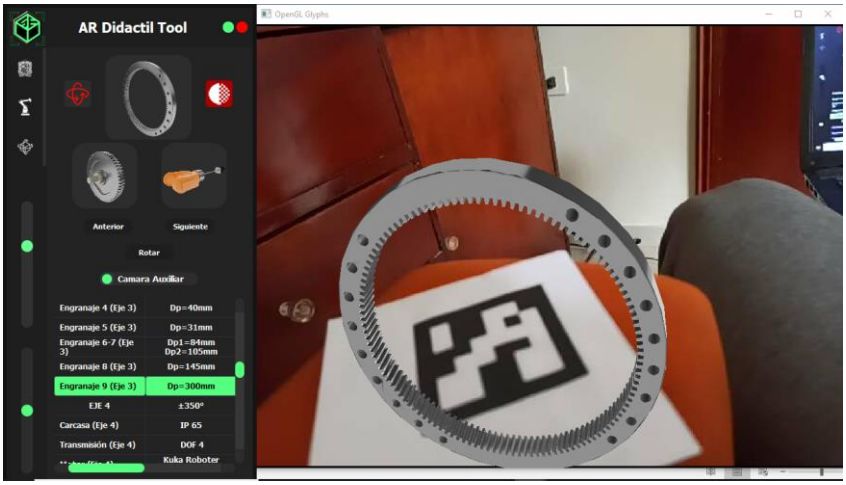


Figura 72. Visualización engranaje 9 eje 3 herramienta AR Didactic Tool.

Inicialmente se visualiza el eje completo, luego el tren de engranajes, en cuanto al orden de despiece, este inicia desde el motor y termina en el engranaje que representa el movimiento motriz acoplado a la estructura externa, de esta forma se detalla cómo está incorporado la parte mecánica y la carcasa.

La herramienta cuenta, además, con funciones enfocadas al desarrollo de actividades de la guía práctica, permitiendo obtener información relevante del despiece y el funcionamiento de los grados de libertad, como lo es la opción de transparencia (figura 61), el apartado de animación de trenes de engranajes o la lista de información de los elementos en la interfaz. En cuanto al enfoque temático dado, se basó en el trabajó en conjunto con el docente Oscar Gabriel Espejo Mojica, buscando la empleabilidad general de la guía en la asignatura de sistemas mecatrónicos I. Con la combinación de recursos que ofrece la herramienta y competencias a desarrollar en la materia se establecieron actividades, dando como resultado una guía práctica con los siguientes temas:

- Identificación de grados de libertad.

- Identificación de piezas.
- Cálculo de parámetros geométricos de engranajes rectos.
- Cálculo de velocidad angular y relaciones de velocidad.

En cada tema, se pone en juego un conjunto de conceptos, procedimientos y métodos para su desarrollo, mediante el uso de la tecnología de realidad aumentada, para ver la guía práctica completa ir a anexo 4.

Por último, la herramienta de realidad aumentada fue aplicada como actividad el día 16 de febrero de 2021 en la clase de sistemas mecatrónicos I dictada por el docente Oscar Gabriel Espejo Mojica de Ingeniería en control, en la Universidad Distrital Francisco José de Caldas, en medio de un entorno académico de presencialidad remota, en un espacio destinado como practica de laboratorio de ingeniería en control, el grupo de clase contó con al menos treinta y cinco (35) estudiantes y el docente.

Inicialmente, fue necesario proveer al docente y sus estudiantes, con el aplicativo de la herramienta y los archivos requeridos para su funcionamiento, los cuales son:

- Ejecutable de la herramienta de realidad aumentada.
- Marcador Aruco, documento PDF listo para ser impreso por estudiantes y docente.
- Manual de usuario, este documento se encuentra referenciado como Anexo 5.

Con el material disponible, el docente realizo una breve introducción, dando a conocer de forma general las principales características y las posibilidades que ofrece la herramienta como soporte en su clase, además planteó un ejercicio utilizando el modelo 3D y el dato del diámetro primitivo del engranaje visualizado, para obtener datos relevantes del engranaje y de el tren de transmisión del cual hace parte (ver Anexo 4).

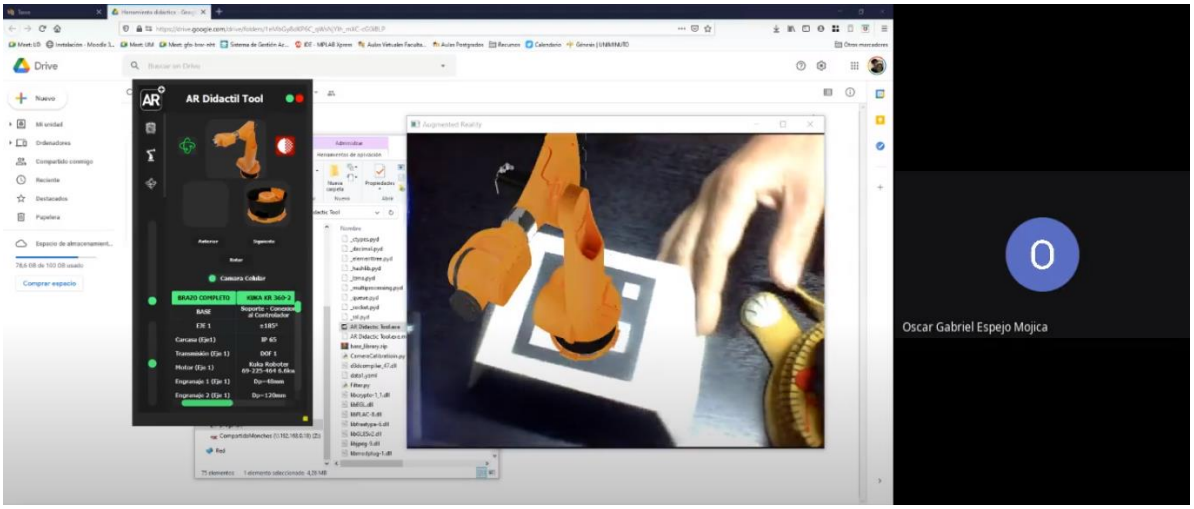


Figura 73. Introducción a la herramienta en la clase mecatrónicos 1 por parte del profesor Oscar Espejo.

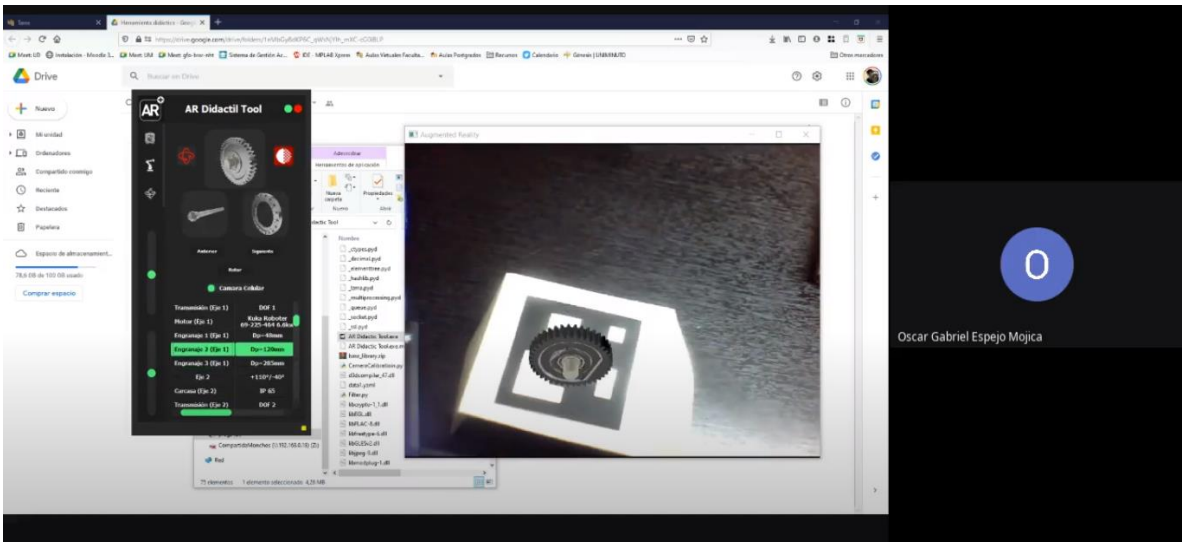


Figura 74. Explicación de ejercicio con el dato del diámetro primitivo por parte del profesor Oscar Espejo.

Tras esto, los autores del proyecto procedieron a exponer detenidamente las funciones que la herramienta ofrece para el despiece y animación del brazo robótico y cada uno de sus trenes de transmisión, como se observa en la figura 75 a 77, esto con el fin de que el docente y alumnos cuenten con las herramientas que le permitan moverse de forma sencilla y no sea un obstáculo el detallar y analizar los modelos que allí se encuentran.

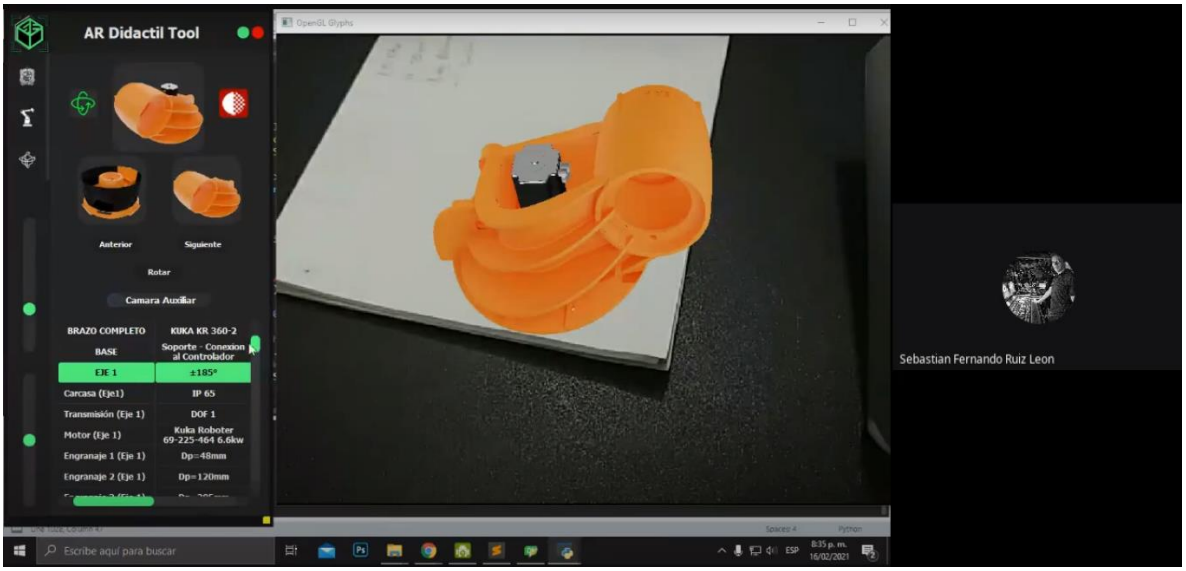


Figura 75. Explicación despiece en la herramienta AR Didactil Tool clase de mecatrónicos 1.

Figura 76. Explicación posicionamiento, escala y transparencia de elementos en la herramienta AR Didactil Tool clase de mecatrónicos 1.

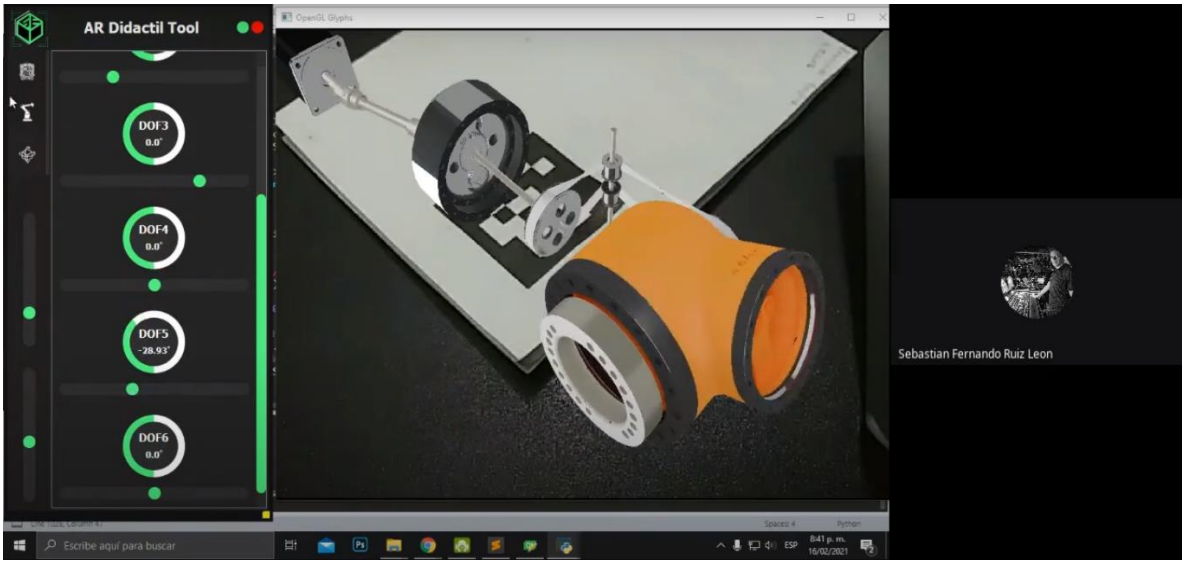


Figura 77. Explicación animación de trenes de transmisión en la herramienta AR Didactil Tool clase de mecatrónicos 1.

Además, se expuso la variedad de componentes y modelos, así como de sistemas disponibles para su análisis en la materia de sistemas mecatrónicos I, ofreciendo la posibilidad de generar diversas actividades ya sea de identificación de piezas, cálculo de variables implícitas en un tren de trasmisión o conocer la distribución de los elementos dentro de un brazo robótico industrial de este modelo.

7.1 Validación de la solución.

Posterior a la utilización del aplicativo por parte de los estudiantes, se realizó una encuesta con el fin de validar diferentes aspectos, como que tan didáctica es, su usabilidad, como les pareció el aporte de la misma en el enfoque académico y si la herramienta sirve en el proceso de enseñanza/aprendizaje. La encuesta fue dejada a disposición de los 35 estudiantes de la materia sistemas mecatrónicos I, en la cual fue aplicada; fue respondida por 15 de ellos.

En cuanto al desarrollo de la encuesta se tomó como base la escala Likert, ya que este tipo de escala permite comprender mejor el nivel de aprobación o desaprobación de los estudiantes con respecto al aplicativo. Para el desarrollo de los ítems se toma como referencia los criterios encontrados en el documento "Construcción de una escala de actitudes tipo Likert (Fernández de pinedo, 1982)", en donde se especifica que cada ítem debe declarar no sólo las dos posturas extremas, sino también graduar las intermedias.

Figura 78. Escala de Likert. (Muguira, 2020)

Además, se propone una serie de preguntas con un rango de opción de respuesta simple como SI o No para temas más concretos, como lo es la usabilidad de este tipo de herramientas en la universidad o en diferentes sectores.

Generalmente se obtuvieron respuestas satisfactorias a la experiencia que tuvieron los estudiantes con la herramienta ver Anexo 3. Pues cerca del 73% de ellos creyeron que es didáctica y ninguno considero que fuera difícil de usar, ya que los controles de las funciones y su distribución son simples, esto en combinación con el diseño dado a la interfaz, resulto ser atractivo y motivador para más de la mitad de ellos, permitiendo así aumentar su interés en los temas relacionados a la estructura de un brazo robótico según el 80%.

Así mismo, cerca del 93% de los encuestados aseguraron que la aplicación permite fortalecer los conocimientos teóricos y es de ayuda para el análisis y percepción del funcionamiento mecánico de los grados de libertad de un brazo robótico.

Además, el 73% de los estudiantes que resolvieron la encuesta determinaron que este tipo de tecnologías son de apoyo en entornos de presencialidad remota y la mayoría considera que son muy útiles en procesos de enseñanza aprendizaje, esto a su vez se encuentra interrelacionado, con el interés de cerca del 93% de ellos en la posible aplicación de tecnologías similares en la universidad y el potencial de la realidad aumentada en diferentes campos de la educación.

7. 2 Restricciones de la Solución.

El desarrollo de la herramienta AR Didactic Tool, se realizó en computadores que cuentan con el sistema operativo Windows. La Liberia que se usó para generar el ejecutable, realiza este proceso con base en el sistema operativo mediante el cual se desarrolló la herramienta, esto genera la mayor restricción que tiene la herramienta y es que solo está disponible para sistemas operativos de Windows, fuera de eso los requerimientos básicos recomendados es que el procesador sea tercera generación o superior, que cuente con al menos 4 Gb de RAM.

8. Conclusiones y Recomendaciones

El proyecto inicialmente fue planteado de forma simple, pues su fin era el poder realizar el despiece de un brazo robótico haciendo uso de la tecnología de realidad aumentada, sin embargo, en medio de su realización, se fueron agregando cualidades a la herramienta dando como resultado una aplicación con diversas funciones que no se tenían previstas en un inicio, tales como la generación de transparencias sobre elementos, lo cual permite observar la distribución de los componentes al interior de la carcasa y la animación de los grados de libertad de los trenes de transmisión completos y del brazo robótico en general.

La selección del modelo de brazo robótico fue vital para el desarrollo y aplicabilidad de la herramienta en el ámbito académico, pues debido a la variedad de elementos que lo componen fue posible generar diversas actividades y agregar también varias funciones, que estas a su vez permiten la implementación de ejercicios para el desarrollo de la guía práctica.

El desarrollo y aplicación de un código relativamente sencillo de filtro, aporto mejoras sustanciales en el posicionamiento y detección del marcador, lo cual provoco una visualización de los elementos 3D más estable, con menor intermitencia y movimiento de los modelos.

El acondicionamiento y tratamiento de los modelos 3D previo a su inclusión en la herramienta fue esencial pues esto redujo los tiempos de carga inicial y fluidez en el proceso de despiece, además, la inclusión de texturas baqueadas en los modelos permitió reducir potencia de cómputo dedicada a iluminaciones e incorporar un nivel sobresaliente de foto realismo sobre los elementos visualizados.

El funcionamiento de la herramienta no fue optimo al visualizar la totalidad de los modelos que componen el brazo robótico al mismo tiempo sobre el marcador, pues llegó a presentar problemas de carga y fluidez. Esto puede ser solucionado a futuro ya que tanto OpenCV como OpenGL cuentan con desarrollo y soporte constante, por lo que se recomienda esperar actualizaciones que permitan solventar estos problemas.

Durante el planteamiento y desarrollo del proyecto se investigó acerca de la generación del aplicativo de la herramienta para dispositivos móviles, sin embargo, no fue posible encontrar los medios para poder generar una aplicación de este tipo utilizando solo el lenguaje de programación Python, sobre el cual está basado este proyecto, esto deja abierta la posibilidad de desarrollar a futuro un aplicativo de realidad aumentada para estas plataformas y utilizando este lenguaje de programación.

Además, otra de las limitaciones de la herramienta es que solo se generó el ejecutable para el sistema operativo de Windows, esto debido a que el archivo .exe es el resultado de la generación de códigos basado en el sistema operativo en el que se desarrolló la herramienta, esto plantea el hecho de que se pueda alcanzar un uso multiplataforma.

La herramienta y los scripts implementados tiene el potencial de ser utilizados en diferentes áreas, no específicamente en mecatrónica para el despiece de un brazo robótico, pues es posible enfocar este tipo de tecnologías en otros campos educativos como lo puede ser la medicina, la arquitectura u otras ingenierías.

9. Referencias

Agustí i Melchor, M. (2020). Ejemplos de aplicaciones 3D interactivas con OpenGL. *Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València*. https://n9.cl/f6gzh

Alemán Suárez, J. D., & Mata Mendoza, M. A. (2006). Guía de elaboración de un manual de prácticas de laboratorio, taller o campo: asignaturas teórico prácticas. Universidad Autónoma Chapingo. http://www.rivasdaniel.com/Pdfs/GUIAMANUALPRACTICAS.pdf

Alvarez-Marin, A., Castillo-Vergara, M., Pizarro-Guerrero, J., & Espinoza-Vera, E. (2017). Realidad Aumentada como Apoyo a la Formación de Ingenieros Industriales. *Formación universitaria*, *10*(2), 31–42. https://doi.org/10.4067/s0718-50062017000200005

Arévalo, V. M., Gonzáles, J., & Ambrosio, G. (2002). LA LIBRERÍA DE VISIÓN ARTIFICIAL OPENCV APLICACIÓN A LA DOCENCIA E INVESTIGACIÓN. *Dpto. De Ingeniería de Sistemas y Automática, Universidad de Málaga*. http://mapir.isa.uma.es/varevalo/drafts/arevalo2004lva1.pdf

Ashraf, O. (2021, 12 enero). *Robot arm 6 DOF*. GRABCAD. https://grabcad.com/library/robot-arm-6-dof-1

Azuma, R. T. (1997). A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, *6*(4), 355–385. https://doi.org/10.1162/pres.1997.6.4.355

Baird, A. (2012, 3 septiembre). *Requested Fanuc ArcMate 100iB*. GRABCAD. https://grabcad.com/library/requested-fanuc-arcmate-100ib

Barrera, C. (2017, julio). PRESENTE Y FUTURO DE LA ROBÓTICA INDUSTRIAL EN COLOMBIA. *Reportero Industrial*. https://www.reporteroindustrial.com/temas/Presente-y-futuro-de-la-robotica-industrial-en-Colombia+120872?pagina=1

Blázquez Sevilla, A. (2017). Realidad Aumentada en Educación. *GATE*, 2–6. http://oa.upm.es/45985/1/Realidad_Aumentada__Educacion.pdf

Buenaventura Baron, O. (2014). REALIDAD AUMENTADA COMO ESTRATEGIA DIDÁCTICA EN CURSO DE CIENCIAS NATURALES DE ESTUDIANTES DE QUINTO GRADO DE PRIMARIA DE LA INSTITUCIÓN EDUCATIVA CAMPO VALDÉS. *Universidad De Medellin*. https://n9.cl/7os6j

Cengiz, A. (2020, 28 noviembre). *robotic arm*. GRABCAD. https://grabcad.com/library/robotic-arm-241

Centro de descargas. (2020). KUKA. https://www.kuka.com/es-es/servicios/descargas?terms=Language:en:1;Language:es:1;Category:CAD;product_name:KR%20360-2;&q=

Challenger Pérez, I., Díaz Ricardo, Y., & Becerra García, R. A. (2014). El lenguaje de programación Python/The programming language Python. *Centro de Información y Gestión Tecnológica de Santiago de Cuba*, *XX* (2), 1–13. https://www.redalyc.org/pdf/1815/181531232001.pdf

Cupitra García, A., & Duque Bedoya, E. T. (2018). Profesores aumentados en el contexto de la realidad aumentada: una reflexión sobre su uso pedagógico. *Agora U.S.B.*, *18*(1), 245. https://doi.org/10.21500/16578031.3178

Elouafiq, A. (2020). Design and Engineering of a Robot Arm. *Al Akhawayn University in Ifrane School of Science and Engineering*. https://arxiv.org/ftp/arxiv/papers/1204/1204.1649.pdf

Fernández de pinedo, I. (1982). *Construcción de una escala de actitudes tipo Likert*. Ministerio de trabajo y Asuntos sociales de España. https://n9.cl/w5mam

GARCÍA MONSÁLVEZ, J. C. (2017). Python como primer lenguaje de programación textual en la Enseñanza Secundaria. *Education in the Knowledge Society (EKS)*, *18*(2), 147. https://doi.org/10.14201/eks2017182147162

García, O., & Guevara, A. (2004). Introducción a la Programación Gráfica con OpenGL. *Escola Tècnica Superior d'Enginyeria Electrònica i Informàtica La Salle*.

glCallList. (2020). Gl functions. https://www.dei.isep.ipp.pt/~matos/cg/docs/manual/glCallList.3G.html

glPushMatrix. (2020). Khronos. https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glPushMatrix.xml

glRotate. (s. f.). Pyopengl. http://pyopengl.sourceforge.net/documentation/manual-3.0/glRotate.html

glTranslate. (2020). Gl functions. http://pyopengl.sourceforge.net/documentation/manual-3.0/glTranslate.html

Gómez Campos, D. A., & Bustos Quintero, A. E. (2018). DESARROLLO MATEMÁTICO Y APLICACIÓN DE LA CINEMÁTICA INVERSA POR MEDIO DE UN SOFTWARE DE CÁLCULO Y PROGRAMACIÓN, PARA EL ROBOT MITSUBISHI RV-M1. *Universidad Distrital Francisco José de Caldas*. Published. https://n9.cl/4a3p0

Guía para la elaboración de un manual de prácticas. (2002, octubre). Universidad De Colima Dirección General De Educación Superior. https://portal.ucol.mx/content/micrositios/200/file/manual.pdf

Introducción. (2021, 5 abril). Blender. https://docs.blender.org/manual/es/dev/editors/uv/introduction.html

Jaguandoy, H., & Puchana, C. (2014). Estrategia educativa basada en realidad aumentada para el área de tecnología e informática en el grado quinto de primaria. *Universidad de Nariño*.

Jiménez Bravo, R. (2018). Sistema de seguimiento de objetos usando OpenCv, ArUco y Filtro de Kalman extendido. *Dep. Ingeniería de Sistemas y Automática Escuela Técnica Superior de Ingeniería Universidad de Sevilla*. https://n9.cl/72j2

Jorge García. (2003). *Curso de introducción a OpenGL* (v1.0). https://n9.cl/nseejh

Joshi, P., & Escrivá, D. M. (2016). *OpenCV By Example: Enhance your understanding of Computer Vision and image processing by developing real-world projects in OpenCV 3*. Packt Publishing. https://n9.cl/si8jv

Kaehler, A., & Bradski, G. (2008). *Learning OpenCV* (First Edition). O'Reilly Media. https://n9.cl/4cwin

Kesici, B. (2017, 2 junio). *6DOF Vika-350 Industrial Robot with Real Working Mechanisms*. GRABCAD. https://grabcad.com/library/6dof-vika-350-industrial-robot-with-real-working-mechanisms-1

Kuka. (2020). *Los contorsionistas de la clase de cargas bajas*. KUKA ROBOTER GMBH. https://www.kuka.com/-/media/kuka-downloads/imported/6b77eecacfe542d3b736af377562ecaa/pf0035_kr_16_s_es.pdf?rev=634eed17df4f46fda333c8d488b32cd9&hash=B7C1D12CCD7AA59923F3E131FB46E3A1

KUKA. (2020). *Los profesionales para cargas pesadas*. KUKA ROBOTER GMBH. https://n9.cl/xql59

Laboratorios y talleres de mecánica. (2018, septiembre). *MANIPULADOR ROBÓTICO MITSUBISHI MOVEMASTER RV M1* (FT-RC02). Universidad Distrital Francisco José de caldas. https://rita.udistrital.edu.co:23604/Documentos/Fichas_tecnicas/robotica_cnc/FT-RC02.pdf

Lacueva Pérez, F. J., Gracia Bandrés, M. A., & Sanagustín Grasa, L. M. (2015). Realidad Aumentada aplicada a entornos industriales. *TecsMedia*, 3–10. https://n9.cl/fmcxd

López Cuevas, M. (Ed.). (1994). La ausencia del mantenimiento y sus efectos. En *El Mantenimiento y la seguridad industrial en las empresas de clase mundial* (pp. 30–46). División de Ingeniería universidad de Sonora. http://tesis.uson.mx/digital/tesis/docs/5580/Capitulo3.pdf

Melo Bohórquez, I. M. (2018). Realidad aumentada y aplicaciones. *TIA Tecnología Investigación y Academia*, *6*(1), 28–35. https://n9.cl/yueme

Morales Cristina Cañero. (2020). *Apuntes de OpenGL y GLUT*. http://www.ieef.upm.es/webantigua/spain/Asignaturas/InfoInd/teoria/Apuntes%20de%20OpenGL.pdf

Mordvintse, A. (2020). *Camera Calibration*. OpenCV. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html

Mott, R., Vavrek, E., & Wang, J. (2017). *Machine Elements in Mechanical Design* (6th ed.). Pearson.

Muguira, A. (2020, 29 abril). *¿Qué es la escala de Likert y cómo utilizarla?* QuestionPro. https://www.questionpro.com/blog/es/que-es-la-escala-de-likert-y-como-utilizarla/

Open Source Computer Vision. (2020). *Camera Calibration and 3D Reconstruction*. OpenCV. https://docs.opencv.org/master/d9/d0c/group__calib3d.html

Open Source Computer Vision. (2020b). *Camera Calibration and 3D Reconstruction*. OpenCV. https://docs.opencv.org/master/d9/d0c/group__calib3d.html

Overvoorde, A. (2020). *Textures objects and parameters*. OpenGL. https://open.gl/textures

Panetta, K. (2020, 3 diciembre). Gartner Top 10 Strategic Technology Trends for 2020 - Smarter With Gartner. Copyright (C) 2021 Gartner, Inc. All Rights Reserved. https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2020/

Pertuz Egea, J. D., & Rojas Arias, F. A. (2018). PROTOTIPO DE REALIDAD AUMENTADA PARA EL PROCESO DE ALFABETIZACIÓN EN LA POBLACIÓN DE ADULTOS MAYORES. *Universidad Distrital Francisco José De Caldas*. https://n9.cl/f1cwe

Przemyslaw, W. (2018, 10 marzo). *Mitsubishi MOVEMASTER RV-M1*. GrabCAD. https://grabcad.com/library/mitsubishi-movemaster-rv-m1-1

pushMatrix(). (2021, 1 enero). processing. https://processing.org/reference/pushMatrix_.html

pushMatrix()/popMatrix() | *ProcessingJS*. (2015). khanacademy. https://www.khanacademy.org/computer-programming/pushmatrixpopmatrix-processingjs/5505194477486080

Python GUI – PyQt VS TKinter. (2020, 11 diciembre). Geeksforgeeks. https://www.geeksforgeeks.org/python-gui-pyqt-vs-tkinter/

Ramírez, E. (2014). Despliegue Básico en OpenGL Moderno. *Universidad Central de Venezuela Facultad de Ciencias*.

Render Baking. (2021, 25 febrero). Blender. https://docs.blender.org/manual/en/latest/render/cycles/baking.html

Rigueros Bello, C. (2017). La realidad aumentada: lo que debemos conocer. *TIA Tecnología, Investigación y Academia*, *5*(2). https://n9.cl/lfi9e

RobotWorx. (2020). *FANUC ARC Mate 100iB*. FANUC. https://www.robots.com/images/robots/Fanuc/ARC-Mate/FANUC_ARC_Mate_100iB_Datasheet.pdf

Silva, R., & Olivera, J. (2003). Introduction to Augmented Reality. *Introduction to Augmented Reality*, 1–9. https://n9.cl/49vm4

Straw, A. (2011, 5 noviembre). *augmented reality - computing the OpenGL projection matrix from intrinsic camera parameters*. strawlab. https://strawlab.org/2011/11/05/augmented-reality-with-OpenGL/

Torrontegi, U. M. (2010). Reconstrucción densa de modelos tridimensionales utilizando Visión Artificial. *Universidad del País Vasco Departamento de Ciencia de la Computación e Inteligencia Artificial*. https://n9.cl/p4eae

Uintaco Sierra, L. T., Vélez Gutierrez, M. D. P., & Otalora Mancilla, W. C. (2016). Desarrollo de una aplicación de realidad aumentada como herramienta de capacitación, diseño y planeación en una línea de extrusión de tubería PVC, con base en librería Artoolkit. *Universidad Piloto De Colombia*. http://polux.unipiloto.edu.co:8080/00003130.pdf

Villacorta Betancor, A. O. (2020). Prototipo para estimación de la pose de un sistema móvil mediante múltiples sensores basados en visión artificial. *Escuela Superior de Ingeniería y Tecnología Universidad de la Laguna*. https://n9.cl/ut41u

Walter. (2020). Walter. https://walter.readthedocs.io/en/latest/

Witt García, A. M. (2014). Algoritmo de Calibración en MATLAB para Cámaras Digitales. *Universidad San Francisco de Quito*. https://n9.cl/ws0yu

Yagmur, H. (2020, 16 noviembre). *KUKA INDUSTRIAL ROBOTİC ARM*. GRABCAD. https://grabcad.com/library/kuka-industrial-robotic-arm-1