DISEÑO DE UN SISTEMA AUTOMÁTICO DE EVASIÓN DE OBSTÁCULOS BASADO EN VISIÓN ARTIFICIAL PARA EL ROBOT COLABORATIVO UR3

NAIFER DAVID BLANCO VACCA
ALEX JULIAN BUITRAGO RANGEL

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA
INGENIERÍA MECATRÓNICA
BUCARAMANGA
2022

DISEÑO DE UN SISTEMA AUTOMÁTICO DE EVASIÓN DE OBSTÁCULOS BASADO EN VISIÓN ARTIFICIAL PARA EL ROBOT COLABORATIVO UR3

NAIFER DAVID BLANCO VACCA
ALEX JULIAN BUITRAGO RANGEL

Tesis de grado para optar al título de Ingeniero Mecatrónico

Director: M. Sc. Hernando González Acevedo
Ingeniero Electrónico

Co-Director: Ph. D. Carlos Julio Arizmendi Pereira
Ingeniero Electrónico

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA
INGENIERÍA MECATRÓNICA
BUCARAMANGA
2022

AGRADECIMIENTOS

Gracias a todas las personas que nos apoyaron durante el pregrado, especialmente a nuestros padres, que son el principal apoyo para lograr lo que día a día nos proponemos y así poder cumplir nuestras metas, gracias por la confianza y apoyo que siempre nos brindan. Gracias a todas las personas que nos ayudaron a construir este trabajo, especialmente a nuestro director de grado Hernando González Acevedo.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

1. INTRODUCCIÓN

La automatización industrial y robótica son áreas en permanente auge debido a las soluciones que puede proponer hacia diferentes sectores de la economía. Esto hizo que este proyecto de grado se quisiera desarrollar en esta área. Se pudo evidenciar que los brazos robóticos son la aplicación más utilizada de la robótica en el campo industrial. Por eso se decidió trabajar en este proyecto, con el fin de poder conocer y mejorar este campo de aplicación de la robótica.

El robot colaborativo UR3 es un robot de la empresa Universal Robots, el cual ha sido implementado principalmente por PyMES las cuales han podido hacer más óptimos los procesos donde se aplican. En Francia la empresa Pazzi utiliza esta familia de robots para hacer pizzas. Los robots hacen todo el proceso en la fabricación de la pizza hasta llevarla al mostrador donde finalmente la recibe el cliente. La finalidad de los robots colaborativos es poder llegar a ser manipulados por el usuario final y que estos puedan obtener la mayor eficiencia de los robots.

En el presente proyecto tiene como objetivo buscar una forma alternativa de obtener la mayor eficiencia de este tipo de robots. Durante el proyecto se implementa un algoritmo de visión artificial, específicamente de segmentación semántica el cual con ayuda de la red neuronal convolucional ResNet-18 crean una DeepLab para que el robot pueda concebir el entorno donde esta y tomar decisiones a partir de un algoritmo de generación de trayectorias el cual se implementa para determinar la mejor forma de llegar de un punto inicial a uno final evitando los obstáculos que estén en la escena o entorno. Este algoritmo es el RRT. Para la implementación de ambos algoritmos fue necesario establecer la comunicación entre el entorno de desarrollo de los algoritmos con el controlador del robot UR3 y esto se realizó a través del protocolo de comunicación TCP/IP.

1. OBJETIVOS

1.1. Objetivo General

- Diseñar un sistema automático de evasión de obstáculos basado en visión artificial para el robot colaborativo UR3.

1.2. Objetivos específicos

- Determinar el modelo cinemático para evaluar la posición y orientación del efector final del robot.
- Establecer una comunicación bidireccional entre la unidad de control del robot UR3 y el software MATLAB con el objetivo de controlar el manipulador desde este software.
- Desarrollar un algoritmo de visión artificial para determinar la ubicación del obstáculo en el espacio de trabajo del robot UR3.
- Desarrollar un algoritmo para la planeación de trayectorias que permitan desplazar el gripper de un punto inicial a otro evitando la colisión con obstáculos.
- Validar los algoritmos de planeación de trayectorias realizando la tarea de transporte de piezas en un ambiente controlado con obstáculos.

2. ESTADO DEL ARTE

Para el desarrollo del presente proyecto de grado se tuvieron en cuenta trabajos de investigación, donde para el desarrollo del algoritmo de visión artificial se revisó la literatura [1], ahí se trabaja el procesamiento de imágenes por enmascaramiento, el cual se basa en la separación de los colores rgb, de esta manera se puede obtener una detección de colores rgb con mayor exactitud y posteriormente unir nuevamente los colores rgb. En [2] demuestran que las redes totalmente convolucionales entrenadas de extremo a extremo en segmentación semántica es un método eficiente de visión artificial, ya que demuestran como se mejora la precisión de la red al transferir pesos clasificadores previamente entrenados fusionando diferentes capas para aprender de extremo a extremo en imágenes completas. Para el entrenamiento de la red neuronal convolucional se utilizó ResNet18 porque en [3] evalúan esta red con un conjunto de datos de ImageNet y comparan el entrenamiento con la red VGG la cual tiene un nivel de complejidad mayor y menor profundidad, demostrando de manera empírica que las redes residuales son más fáciles de optimizar y pueden tener mayor precisión al tener una mayor profundidad. Para la construcción del ambiente virtual, el cual es fundamental en el momento de pasar a la planeación de trayectorias, en [4] se utiliza el Kinect v2 para reconstruir 3D la escena y adicional lo conectan con el robot UR3 para posteriormente formar un gemelo digital. En [5] se hace una planificación y simulación de trayectorias para poder hacer trabajos de soldadura con el robot UR3. En este trabajo se pudo conocer a detalle el modelado y análisis cinemático en Matlab. En la investigación sobre planeación de trayectorias en [6] se estudia el comportamiento de un manipulador robótico con un humano, donde el comportamiento del robot se genera por medio del algoritmo de árbol aleatorio de exploración rápida (RRT). De igual forma se revisó la forma de realizar las trayectorias suaves, ya que estas por defecto vienen dadas por movimientos discontinuos, en [7] se diseña un control de movimiento suave a la trayectoria generada. En cuanto a la comunicación entre el robot UR3 y un pc externo se tuvo en cuenta [8] donde comunican el controlador del robot y un pc a través de la red Ethernet. En él se utiliza el protocolo de comunicación TCP/IP, donde este algoritmo de comunicación está escrito en el entorno de Matlab y utilizan el puerto 300004 para recibir datos del controlador del robot. En [9] trabajan en un ambiente colaborativo un humano y un robot UR3 donde desarrollan un algoritmo para la interconexión entre el robot y el entorno de Matlab. [8-9] permitió decidir el protocolo de comunicación a utilizar en el trabajo presentado.

3. VISIÓN ARTIFICIAL

La visión artificial tiene como objetivo generar descripciones inteligentes y útiles de escenas y secuencias visuales, así como de los objetos que aparecen en ellas, mediante la realización de operaciones sobre imágenes y vídeos.

Entre las tareas más típicas de la visión artificial están:

- Detección, reconocimiento, identificación y seguimiento de objetos.
- Análisis de movimiento.
- Reconstrucción de escenas.
- Restauración de imágenes.
- Calibración de cámaras.
- Visión estéreo.

Un sistema de visión artificial incluye una computadora y un hardware necesario para proporcionar una función visual similar a la existente en los humanos y poder realizar la ubicación de objetos en espacios 3D.

3.1. Software

Para el desarrollo de este proyecto se decidió usar el software MATLAB R2020a el cual ofrece un amplio portafolio de librerías para la adquisición, procesamiento y estudio de visión artificial. La Computer Vision System Toolbox proporciona algoritmos, funciones y apps para el diseño y la realización de pruebas de sistemas de procesamiento de vídeo, visión artificial y visión 3D. Es posible llevar a cabo la detección y el seguimiento de objetos, así como la detección, extracción y coincidencia de características. En el caso de la visión 3D, la toolbox soporta la calibración de cámaras simples, estéreo y ojo de pez, la visión en estéreo, la reconstrucción 3D y el procesamiento de nubes de puntos lidar y 3D. Las apps de visión artificial automatizan los flujos de trabajo de etiquetado de validación (ground-truth) y de calibración de cámaras.

3.2. Hardware

Se hace necesario determinar el tipo de proceso que se va a realizar para poder realizar una buena elección de los objetos y dispositivos que van a conformar el espacio de trabajo. Para este proyecto es necesario contar con una cámara que cuente con un sensor de profundidad para poder determinar la distancia a la que se encuentran los objetos. Entre las cámaras que se encuentran disponibles en el mercado están la línea RealSense de Intel y Kinect de Microsoft cuyas características se presentan en la tabla 1 y tabla 2.

- **INTEL REALSENSE**

La tecnología RealSense es una gama de productos de tecnologías de seguimiento y profundidad diseñadas para dar a las máquinas y dispositivos capacidades de percepción de profundidad. La tecnología, propiedad de Intel, se utiliza en drones autónomos, robots, AR / VR, dispositivos domésticos inteligentes, entre muchos otros productos de amplio mercado. En la tabla 1 se extraen las propiedades técnicas de las cámaras RealSense obtenida de [10], donde se busca tener claridad sobre las especificaciones técnicas de la cámara.

Tabla 1 Especificaciones técnicas de las cámaras RealSense de Intel.

	D435i / D435	**D415**
Tecnología de profundidad	IR estéreo activo	IR estéreo activo
Resolución de la profundidad	Hasta 1280 x 720	Hasta 1280 x 720
Velocidad fotogramas profundidad	Hasta 90 fps	Hasta 90 fps
Campo de visión profundidad (H x V)	87° ± 3° x 58° ± 1°	69° ± 2° x 40° ± 1°
Resolución y velocidad RGB	1920 x 1080 a 30 fps	1920 x 180 a 30 fps
Campo de visión RGB (H x V)	69° ± 1x 42° ± 1°	69° ± 1° x 42° ± 1°
Rango máximo	Aprox. 10 m	Aprox. 10 m
Conector	USB 3.1	USB 3.1

- **MICROSOFT KINECT**

El Kinect cuenta con una cámara RGB o de video y un sensor que es capaz de medir la profundidad, este sensor empleó tecnología basada en luz estructurada infrarroja en su primera versión y en las siguientes dos versiones utilizó la tecnología de tiempo de vuelo,

la cual mide el tiempo que se demora en viajar la luz IR y llegar hasta el sensor; además cuenta con un arreglo de micrófonos que permite saber en qué dirección se está produciendo un sonido dependiendo si la señal llega primero al micrófono de la izquierda o de la derecha. En la tabla 2 se compara de manera explícita las propiedades que cada una de las cámaras de la familia Kinect obtenidas de [11].

Tabla 2 Especificaciones técnicas familia Kinect de Microsoft.

	Kinect V1	**Kinect v2**	**Azure Kinect**
Tecnología de profundidad	Luz estructurada codificada infrarroja	Tiempo de vuelo	Tiempo de vuelo
Resolución de la profundidad	320 x 240	512 x 424	640 x 576 512 x 512
Velocidad fotogramas profundidad	30 fps	30 fps	30 fps
Campo de visión profundidad (V)	45°	60°	65° 120°
Resolución y velocidad RGB	640 x 480 @ 30 fps 1280 x 960 @ 12 fps	1920 x 1080 @ 30 fps	1920 x 1080 @ 30 fps
Campo de visión RGB (H x V)	57° x 43°	70° x 60°	90° x 59°
Rango máximo	3 m	4,5 m	5,4 m
Conector	USB 2.0	USB 3.0	USB 3.0

La tabla 1 y 2 se tomaron en cuenta a la hora de escoger la cámara con la que se trabajó el proyecto, al final se decidió trabajar con la cámara Microsoft Kinect V 2.0 porque es una herramienta conveniente, fácil de usar y económica para la captura de imágenes RGB y cuenta con una buena tecnología de profundidad. Esta cámara TOF(time-of-flight) en [12] la utilizan para capturar el movimiento y determinar el espacio trabajo, a partir del número de errores que se generar pueden determinar un espacio de trabajo optimo en las coordenadas X,Z por medio del software de Matlab el cual permite calcular y mapear el espacio de trabajo. Acá se llega a indicar que un buen espacio para trabajar con el kinect V2 es a una distancia de 0,8 a 4 metros del dispositivo con un ángulo de visión aproximado de 57 grados. En [13] lo utilizan para el reconocimiento de gestos mediante la herramienta EasyGR (Easy Gesture Recognition) basada en algoritmos de aprendizaje automático. En [14] se utiliza más recientemente para alimentar una base de datos con la que

posteriormente se entrena una red neuronal convolucional 3D para detectar problemas relacionados con marcha anormal. Estos trabajos se tomaron en cuenta para poder escoger el kinect V2.

- **Entorno de trabajo**

El área de trabajo es un entorno donde se puede trabajar. En la figura 1 se observa el área de trabajo del robot UR3, el cual tiene un alcance recomendado de 1m en cualquier dirección, desde su base hasta el gripper.

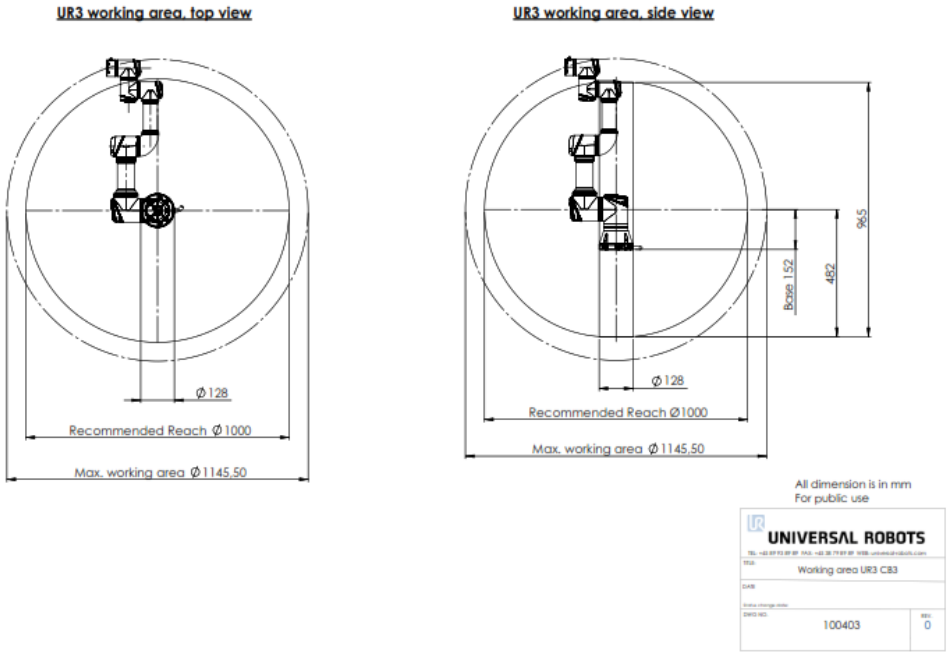

Figura 1 Area de trabajo maxima del robot UR3

Para garantizar esta área de trabajo al robot UR3, se definió una superficie de 1.5 m x 2m, donde el robot se puede trabajar en el 50% de su alcance. De igual forma se diseñaron y construyeron dos objetos cilíndricos, el primero es de 20cm de altura con un radio de 8 cm y el segundo tiene una altura de 8cm con un radio de 4cm.

3.3. Calibración de Kinect V2

Con el fin de saber el error y si se debía calibrar ajustes al Kinect v2 se realizaron tomas con el sensor de profundidad y poder realizar comparaciones con medidas tomadas mediante un flexómetro digital. A continuación, se muestra el código utilizado para realizar una toma con el sensor de profundidad del kinect y como se extraen estas posiciones.

```
% Cree el objeto para el sensor de profundidad
vid2 = videoinput('kinect', 2);
% Inicie el objeto de profundidad
start(vid2);
% Acceda a los datos de seguimiento del cuerpo como metadatos en el objeto de
profundidad.
[frame, ts, metaData] = getdata(vid2);
% Con la propiedad metaData.JointPositions obtiene las ubicaciones de los puntos
en las posiciones X, Y y Z
metaData.JointPositions(:,:,:);
```

Para la nube de puntos que tiene como imagen rgb la figura 2 se hizo una toma con el sensor de profundidad, las coordenadas reales del objeto eran de 0.49, 0.1 y 0.915 en metros en el eje X, Y y Z respectivamente y los datos que genero el Kinect fueron de 0,4889, 0.0952 y 0,9132 en metros en el eje X, Y y Z respectivamente. Estos datos se obtuvieron realizando un promedio con todos los datos que se tienen en la propiedad JointPositions.

Figura 2 Escena inical

Con el fin de saber si el sensor funcionaba bien se realizaron más tomas, en total son 120 muestras, las cuales se agruparon en lotes de 10, ya que se tomaron al menos 10 muestras de la escena con los objetos u obstáculos en la misma posición. En la tabla 3 se muestran unos datos obtenidos con el sensor de proximidad en el eje x. La primera columna corresponde a datos que se tomaron y midieron con el flexómetro mientras la segunda columna corresponde a los datos que se lograron obtener con el sensor de proximidad del Kinect en dos dimensiones.

Tabla 3 Datos bidimensionales

Matriz x (metros)	
Reales	Ambiente virtual
-0,5	-0,4860
-0,5	-0,4622
-0,5	-0,4983
0	-0,0112
0	-0,0028
0	-0,0153
0	0,0156
0	0,0108
0,16	0,1751
0,49	0,4874
0,49	0,4915
0,49	0,5065

También se registraron los 120 datos de profundidad o eje z, los cuales se observan en la tabla 4. En esta tabla se muestra en la primera columna corresponde a datos que se tomaron y midieron con el flexómetro mientras la segunda columna corresponde a los datos que se lograron obtener con el sensor de proximidad del Kinect.

Tabla 4 Datos profundidad

Profundidad (metros)	
Reales	Ambiente virtual
0,915	0,9120
0,915	0,9100
0,915	0,9190
0,915	0,9070
0,915	0,9080
0,915	0,9150
1,275	1,2290
1,53	1,5430
1,53	1,5460
1,53	1,5340
1,575	1,5260
1,575	1,5320

Con los componentes de hardware definidos y revisado que sean preciso se puede hacer el montaje como se observa en la figura 3 para poder comenzar a tomar pruebas. Basados en la información obtenida en [12] se estableció que la cámara debe estar a la misma altura y alineado con la base del robot a una distancia de 1,75 m. El Kinect v2 por defecto trabaja con un cable de comunicación USB 3.0. Al controlador hay varias formas mediante las cuales se puede acceder, una es por medio del cable de red RJ45 a través del protocolo de comunicación TCP/IP. En la sección 4.4 se explica de manera específica la manera en que se establece la comunicación entre el controlador del robot y computador portátil. Por último, se tiene el computador portátil, donde se alojan los algoritmos que permitirían recibir y procesar la información que genera el Kinect v2 y con base en esta enviarle ordenes al robot UR3 a través del controlador del mismo.

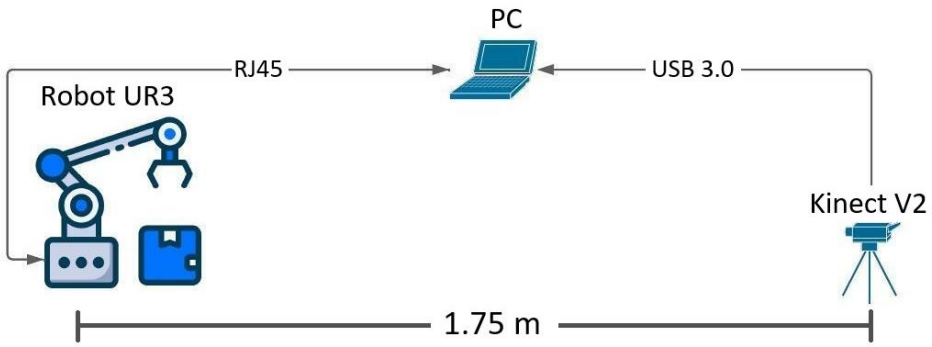

Figura 3 Esquema general de hardware.

3.4. Reconocimiento basado en RBG-D

A continuación, se explican los pasos que siguieron para realizar el reconocimiento por medio de los colores RGB, trabajando con el sensor de profundidad D.

- **Adquisición de imágenes**

La etapa de adquisición es un importante paso porque se debe buscar la imagen más adecuada posible para poder continuar con las siguientes etapas. Para lograr esto se debe tener en cuenta varios factores que van directamente relacionados con el proceso de captura de la imagen, como el hardware, software, el entorno y posicionamiento de los elementos.
Para la adquisición de imagen en el software de Matlab desde la cámara Kinect V2 a una velocidad de 30 fps que es la velocidad estándar que permite el conector USB 3.0 se utilizó Image Acquisition Toolbox el cual proporciona funciones y bloques para conectar cámaras

a MATLAB. Con la función imaq.VideoDevice se puede adquirir un fotograma a la vez desde el dispositivo de vídeo. Se adquieren dos fotogramas, uno de la cámara RGB y uno del sensor de profundidad y con la función pcfromkinect se logra retornar una nube de puntos de una imagen de profundidad de Kinect agregándole al final el componente color a cada punto de la nube. Para logra la reconstrucción de la escena el siguiente código:

```
colorDevice = imaq.VideoDevice('kinect',1) % Cree un objeto para el fotograma de
la camara RGB
depthDevice = imaq.VideoDevice('kinect',2) % Cree un objeto para el sensor de
proximidad
% Inicialice la cámara
colorDevice();
depthDevice();
% Cargue un fotograma desde el dispositivo.
colorImage = colorDevice();
depthImage = depthDevice();
% Extraiga la nube de puntos
ptCloudRef = pcfromkinect(depthDevice, depthImage, colorImage);
% Visualizar la nube de puntos
pcshow(ptCloudRef)
```

En la figura 4 se puede ver la escena reconstruida desde una vista isométrica. En la escena está el brazo robótico UR3, una plataforma de color caoba con un cuadro rojo y un fondo. La cámara está en un punto estático, por tanto, hay partes que la cámara no alcanza a registrar. En la figura se ve al menos 3 zonas sin reconstruir u oscuras, que son el borde de la figura, al fondo del robot y al lado de la plataforma color caoba. Estas zonas sin reconstruir son las partes que la cámara no alcanza a registra y por tanto no tienen puntos o registros.

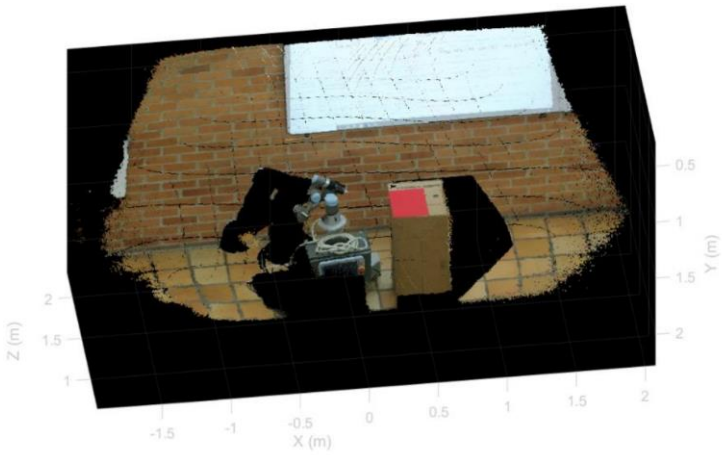

Figura 4 Nube de puntos

- **Preprocesado**

Una imagen real es continua con respecto a las coordenadas x & y, y también en amplitud. La conversión de dicha imagen a forma digital requiere que tanto las coordenadas como la amplitud se digitalicen. La digitalización de los valores de coordenadas se conoce como muestreo y a la digitalización de los valores de amplitud se le denomina cuantización. Cuando x & y, y los valores de amplitud de la función f son cantidades finitas y discretas, se puede decir que se tiene una imagen digital.
En el caso de las imágenes a color están formadas por una combinación de imágenes individuales en el sistema RGB, una imagen en color está conformada por tres imágenes individuales denominadas imágenes primarias (o componentes) rojas (R), verdes (G) y azules (B). El resultado del muestreo y cuantización es una matriz de números reales donde e trabajo con imágenes de 1920x1080 pixeles.
Para el caso de la Toolbox de procesamiento de imágenes de MATLAB se usa la notación (r, c) para indicar las filas, las columnas y el origen está en (r, c) = (1,1), donde r varía entre 0 y 1080 & c varía entre 0 y 1920. En otros casos de menor uso se emplean las coordenadas espaciales.
Para el preprocesamiento desde matlab se procede a crear la escena 3D con la función pcfromkinect la cual permite crear una nube de puntos con los fotogramas que se adquieren desde la cámara Kinect V2.
Existen diversos tipos de ruido que se producen por múltiples circunstancias: sensores, ruidos eléctricos, perturbaciones en el medio de transmisión, etc. La mayoría de las implementaciones de filtros se realiza en dos dominios: el dominio espacial y el dominio frecuencial. En este caso se usarán filtros del dominio espacial. La toolbox de Matlab permite reducir la resolución de una nube de puntos de diferentes formas, dentro de la cual está la reducción de muestreo mediante un filtro de cuadricula de cuadro. Los puntos dentro del mismo cuadro se fusionan en un solo punto de la salida. Su color y propiedades normales se promedian en consecuencia.
La cantidad de datos que genera la cámara KinectV2 son muchos, por tanto, se deben reducir para poder trabajarlos con menos coste computacional. A través de la función pcdownsample se puede reducir la resolución de la nube de puntos 3D. En la figura 5 se presenta la misma escena de la figura 4 pero pasándola por la función pcdownsample para reducir el coste computacional sin afectar la escena

```
FptCloudRef = pcdownsample(ptCloudRef, 'gridAverage', gridSize);
```

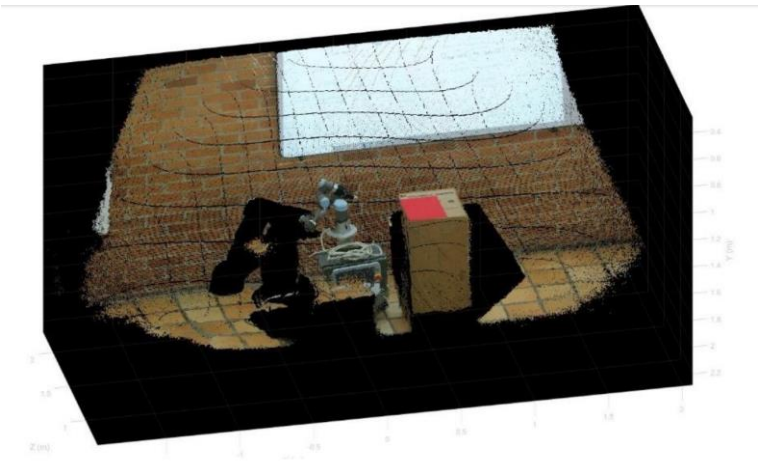

Figura 5 Nube de puntos filtrada

La función pcdownsample permite aplicar un filtro de cuadricula de caja mediante la propiedad gridSize. En esta función se tiene como entrada la nube de puntos original, que es la que genera el Kinect v2 mediante la función pcfromkinect anteriormente indicada, como segundo parámetro de entrada es el método por el cual se va a escoger el color del punto encontrado, en este caso se hizo mediante el promedio de cuadricula el cual promedia los colores rgb y obtiene un único color y por último se establece la resolución 3D de cada caja, gridSize; recordar que al ser cuadriculas solo se ingresa un parámetro. Con este filtro se redujo el coste computacional ya que el filtro gridSize permite eliminar datos redundantes. Por ejemplo, en la tabla 5 se observan los datos que arroja la nube de puntos presentada en la escena figura 4, que en este caso si se contabilizan son de 2073600, los cuales corresponden a número de pixeles de la imagen, cuando se le aplica el filtro a la nube de puntos, que son los datos que se presentan en la table 6, los datos se reducen a 76125, que es la escena que se presenta en la figura 5. Esto permite poder tener una percepción del coste computacional que se evita.

Tabla 5 Datos de la nube de puntos inicial

```
pointCloud with properties:

   Location: [1080×1920×3 single]
      Count: 2073600
     XLimits: [-5.5553 5.5934]
     YLimits: [-3.8942 3.1233]
     ZLimits: [1.0430 7.9890]
      Color: [1080×1920×3 uint8]
     Normal: []
  Intensity: []
```

Tabla 6 Datos de la nube de puntos filtrados

```
pointCloud with properties:

   Location: [76125×3 single]
      Count: 76125
     XLimits: [-5.5553 5.5934]
     YLimits: [-3.8942 3.1233]
     ZLimits: [1.0430 7.9890]
      Color: [76125×3 uint8]
     Normal: [0×3 single]
  Intensity: [0×1 single]
```

- **Reconocimiento**

La etapa de reconocimiento permite saber los colores que componen a cada píxel, y a partir de ahí se pueden clasificar por medio del algoritmo de búsqueda secuencial, los que cumplen con las características deseadas.

En este paso se toma el tensor de color que genera la cámara RGB en la nube de puntos, que es de 1920x1080x3, se clasifican los puntos y con el tensor que se genera a partir del sensor de proximidad que de igual forma es de 1920x1080x3 se puede saber la ubicación exacta de cada punto clasificado.

Se tomaron pruebas en el laboratorio de automatización, donde en diferentes escenas y en objetos con diferentes propiedades se validó el funcionamiento del algoritmo de búsqueda paso a paso de colores RGB. Se trabajo la escena mostrada en la figura 6.

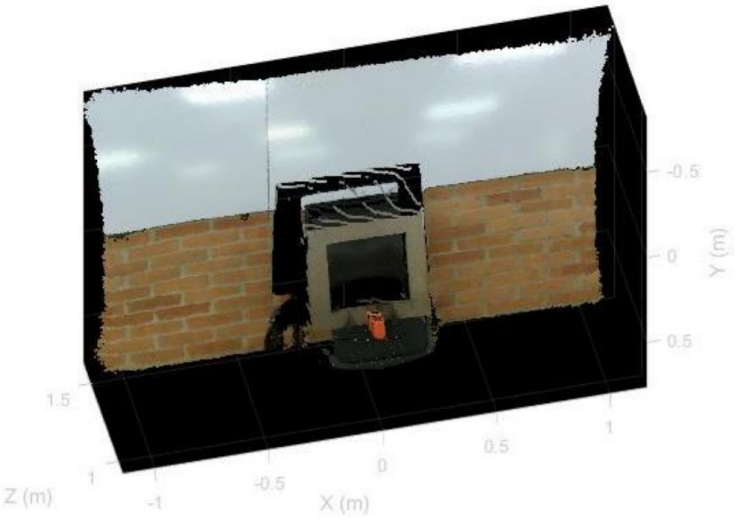

Figura 6 Prueba RBG 1

Donde se busca identificar los objetos de color RGB [250 120 20] que es el objeto de color naranja y [130 70 0] que es el de color caoba, donde son el objetivo y el obstáculo para identificar respectivamente.

Para llegar a encontrar los puntos en el espacio se implementó el siguiente algoritmo:

1. Obtener nube de puntos.
2. Dividir la nube en un tensor RGB y otro de proximidad. Ambos tensores serán de NxMx3, en la primera tendrá tres colores (rojo, verde y azul) y la de proximidad tendrá tres longitudes (X, Y & Z).
3. Calcular el tamaño de la matriz NxMX3, donde N puede tener un rango de 0-1920 y M de 0-1080. Los valores máximos son el tamaño máximo de pixeles que permite la cámara RGB del Kinect V2.
4. Extraer los colores RGB de los objetos a identificar.
5. Dividir el tensor de color NxMx3 en tres matrices NxM donde la matriz uno es el color rojo(R), la matriz dos el color verde(G) y la matriz tres el color azul(B) de la cámara RGB.
6. Filtrar las tres matrices que se obtuvieron el paso cinco con los parámetros que se obtuvieron en el paso cuatro.
7. Las tres matrices tienen el mismo tamaño, pero no todos los pixeles que las conforman pasaron el filtro en 6. Ahora se buscan las coordenadas de los pixeles que pasaron el filtro y por medio de iteración se encuentras las coordenadas coincidentes en las tres matrices.
8. Los tensores de colores y de profundidad tienen el mismo tamaño, cada una de sus matrices tienen las mismas dimensiones. Con las coordenadas XY obtenidas en 7 se

busca iterando esas mismas coordenadas en las matrices que conforman el tensor de profundidad. Este último tensor permite saber exactamente la ubicación en el espacio de cada una de las coordenadas clasificadas, que es la información que ingresara en el algoritmo de generación de trayectorias

Al aplicar los pasos anteriores a los objetos de interés, se obtuvieron las clasificaciones que se ven en las figuras 7 y 8 para la escena presentada en la figura 6.
Obstáculo

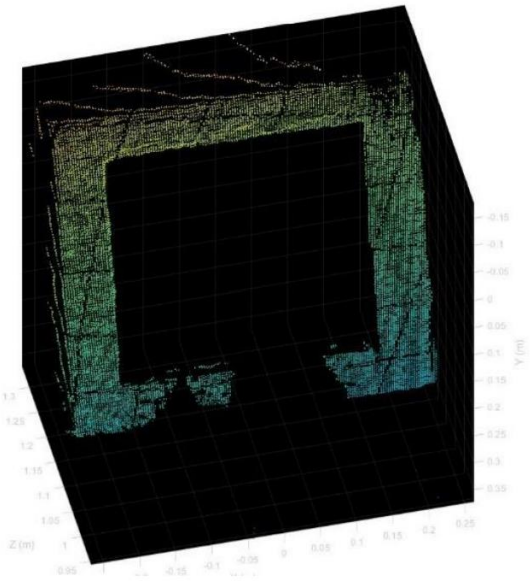

Figura 7 Obstáculo prueba 1

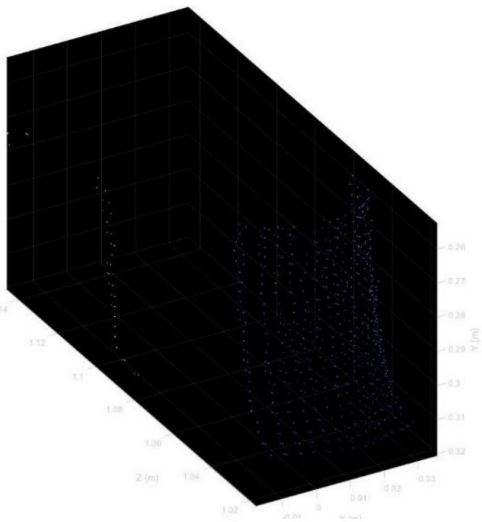

Figura 8 Objetivo prueba 1

Los datos obtenidos para el objetivo en la figura 8 son los siguientes:
Largo: 0.4747 m
Ancho: 0.0498 m
Alto: 0.4052 m
Centroide: [-0.0420 0.0825 1.2203]
El largo, ancho y alto corresponden a las medidas del objeto. El centroide permite conocer la ubicación en metros del objeto en el espacio tridimensional.

3.5. Redes neuronales convolucionales

Las redes neuronales convolucionales se distinguen de otras redes neuronales por su rendimiento superior con entradas de señales de imagen, voz o audio. Tienen tres tipos principales de capas, que son:

- Capa convolucional
- Capa de agrupación
- Capa completamente conectada (FC)

Capa convolucional

Es el bloque de construcción central de una CNN, y es donde ocurre la mayor parte de los cálculos. Requiere algunos componentes, que son datos de entrada, un filtro y un mapa de características. Por ejemplo, una imagen RGB, está formada por una matriz de píxeles en

3D. Esto significa que la entrada tendrá tres dimensiones (altura, ancho y profundidad) que corresponden a RGB en esa imagen. También está un detector de características, también conocido como kernel o filtro, que se moverá a través de los campos receptivos de la imagen, verificando si la característica está presente. Este proceso se conoce como convolución.
En última instancia, la capa convolucional convierte la imagen en valores numéricos, lo que permite que la red neuronal interprete y extraiga patrones relevantes.

Capa de agrupación

La agrupación de capas, también conocida como submuestreo, lleva a cabo una reducción de dimensionalidad, lo que reduce el número de parámetros en la entrada. Similar a la capa convolucional, la operación de agrupación barre un filtro a través de toda la entrada, pero la diferencia es que este filtro no tiene ningún peso. En cambio, el kernel aplica una función de agregación a los valores dentro del campo receptivo, llenando la matriz de salida. Hay dos tipos principales de agrupación:

- Agrupación máxima
- Agrupación promedio

Si bien se pierde mucha información en la capa de agrupación, también tiene una serie de beneficios para la CNN. Ayudan a reducir la complejidad, mejorar la eficiencia y limitar el riesgo de sobreajuste.

Capa completamente conectada

Cada nodo de la capa de salida se conecta directamente a un nodo de la capa anterior. Esta capa realiza la tarea de clasificación en base a las características extraídas a través de las capas anteriores y sus diferentes filtros. Mientras que las capas convolucionales y agrupadas tienden a usar funciones ReLu, las capas FC (capa completamente conectada) generalmente aprovechan una función de activación softmax para clasificar las entradas de manera apropiada, produciendo una probabilidad de 0 a 1.

3.6. Residual Networks

Una red neuronal residual es una red neuronal artificial de un tipo que se basa en construcciones conocidas de células piramidales en la corteza cerebral. Las redes neuronales residuales hacen esto mediante el uso de conexiones de salto o atajos para

saltar sobre algunas capas. Los modelos típicos de ResNet se implementan con saltos de doble o triple capa que contienen no linealidades (ReLU) y normalización de lotes en el medio. Se puede usar una matriz de peso adicional para aprender los pesos de salto; estos modelos se conocen como HighwayNets. Los modelos con varios saltos en paralelo se denominan DenseNets. En el contexto de las redes neuronales residuales, una red no residual puede describirse como una red simple.

Hay dos razones principales para agregar conexiones de salto: para evitar el problema de la desaparición de gradientes o para mitigar el problema de degradación; donde agregar más capas a un modelo adecuadamente profundo conduce a un mayor error de entrenamiento. Durante el entrenamiento, los pesos se adaptan para silenciar la capa ascendente y amplificar la capa previamente omitida. En el caso más simple, solo se adaptan los pesos para la conexión de la capa adyacente, sin pesos explícitos para la capa aguas arriba. Esto funciona mejor cuando se pasa por encima de una sola capa no lineal o cuando las capas intermedias son todas lineales. De lo contrario, se debe aprender una matriz de peso explícita para la conexión omitida.

Omitir simplifica eficazmente la red, utilizando menos capas en las etapas iniciales de formación. Esto acelera el aprendizaje al reducir el impacto de los gradientes que desaparecen, ya que hay menos capas por las que propagarse. Luego, la red restaura gradualmente las capas omitidas a medida que aprende el espacio de características. Hacia el final del entrenamiento, cuando se expanden todas las capas, permanece más cerca de la variedad y, por lo tanto, aprende más rápido. Una red neuronal sin partes residuales explora más el espacio de características. Esto lo hace más vulnerable a las perturbaciones que hacen que salga del colector y necesita datos de entrenamiento adicionales para recuperarse.

ResNet-18

Para el desarrollo del algoritmo se trabajó con una red convolucional preentrenada ResNet-18, en la figura 9 se muestra la arquitectura de esta red, la cual al estar preentrenada con más de un millón de imágenes de la base de datos ImagenNet permite extraer característica eficaces e informáticas de imágenes naturales para poder usarlas como punto de partida y poder aprender a extraer estas características de imágenes nuevas.

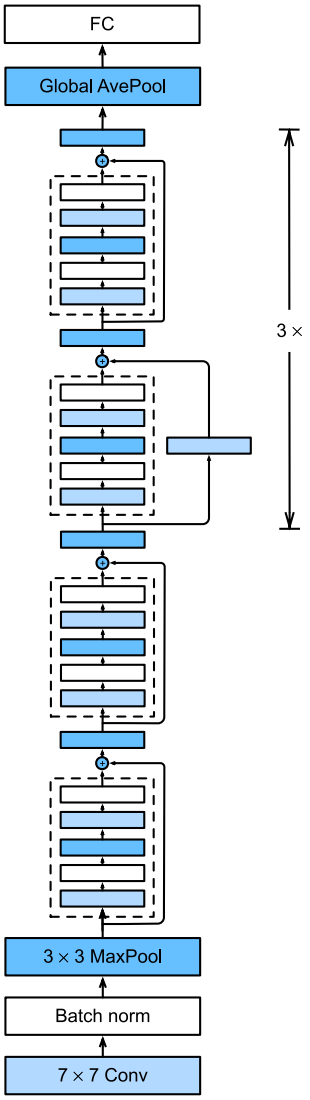

Figura 9 La arquitectura ResNet-18 [15]

La red puede clasificar múltiples objetos, dependiendo de la red previamente entrenada. En la siguiente sección se explica el proceso de construcción de la base de datos para completar el entrenamiento de la CNN (convolutional neural network).

3.7. Segmentación semántica

La segmentación semántica es un algoritmo de deep learning que asocia una etiqueta o categoría a cada píxel presente en una imagen. Se utiliza para reconocer un conjunto de píxeles que conforman distintas categorías. Por ejemplo, un vehículo de conducción

autónoma necesita identificar vehículos, peatones, señales de tráfico, aceras y otros elementos de la carretera.
Un sencillo ejemplo de segmentación semántica es la separación de las imágenes en dos clases distintas. Por ejemplo, en la figura 10 se muestra una persona en la playa que se empareja con otra versión en la que se muestran los píxeles de la imagen segmentados en dos clases distintas: la persona y el fondo.

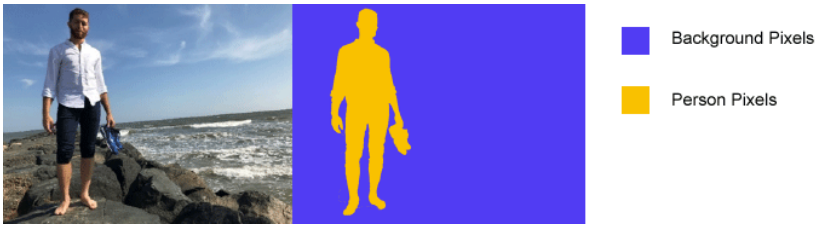

Figura 10 Imagen y píxeles etiquetados [16]

La segmentación semántica no se limita a solo dos categorías. Es posible modificar el número de categorías para clasificar el contenido de la imagen. Por ejemplo, esta misma imagen se podría segmentar en cuatro clases: persona, cielo, agua y fondo.
El proceso de entrenamiento de una red de segmentación semántica para clasificar imágenes consta de los siguientes pasos:

1. Analizar un conjunto de imágenes con píxeles etiquetados.
2. Crear una red de segmentación semántica.
3. Entrenar la red para clasificar imágenes en categorías de píxeles.
4. Evaluar la precisión de la red.

Para crear la base de datos con las imágenes etiquetadas, que se utilizó para completar el entrenamiento de la CNN ResNet-18 se utilizó la aplicación Image Labeler, la cual al igual que Matlab pertenece a The MathWorks, Inc, la cual se utiliza principalmente para aplicaciones de inteligencia artificial. Esta proporciona una manera fácil de crear de forma interactiva una variedad de formas para marcar como etiquetas de región de interés. Puede crear etiquetas rectangulares, polilíneas, de píxeles y polígonos y etiquetas de escena en una imagen o secuencia de imágenes.
Para generar la base de datos con Image Labeler se debe realizar lo siguiente:

1. Importar los datos que desea etiquetar.
2. Cree una o las etiquetas deseadas con el tipo de etiqueta requerida.
3. Seleccione cada una de las imágenes y etiquete todos y cada uno de los objetos que hay en la imagen.
4. Exporte el groundTruth.

El groundTruth contiene información sobre el origen de datos, las definiciones de etiqueta y las anotaciones de etiqueta marcadas para un conjunto de etiquetas de verdad básica.
Con esta red pre-entrenada (ResNet-18) y la base de datos que se construyó ya se tiene la base para entrenar la red y poder aplicarla a un algoritmo especifico de visión artificial.

3.8. Implementación del algoritmo basado en segmentación semántica.

El algoritmo de segmentación semántica permitió obtener una respuesta más robusta gracias a que este algoritmo etiqueta los píxeles de una imagen es más precisa que las otras formas de detección de objetos. En el proceso de entrenamiento para clasificar las imágenes se está haciendo uso de los siguientes tres pasos:

- Analizar las imágenes etiquetadas.
- Crear una red de segmentación semántica.
- Entrenar la red para clasificar imágenes de píxeles

- **IMPLEMENTACIÓN EN EL SOFTWARE DE MATLAB**

En MATLAB, el flujo de trabajo para realizar segmentación semántica consta de estos cinco pasos:

PASO 1: Etiquetar los datos: Se utiliza la aplicación Image Labeler de MATLAB. Esta app permite etiquetar cada píxel de la imagen y agregarle una categoría.

PASO 2: Crear un datastore para las imágenes originales y otro para las imágenes etiquetadas: Cuando se trabaja con enormes cantidades de datos, se debe gestionar el conjunto de datos, para utilizarlos solo cuando sea necesario. Se crea una datastore, la cual contiene la ubicación de los archivos a los que desea acceder.
Para crear una red ResNet-18, se necesitan dos datastore, una que contenga las imágenes originales y otra con las imágenes etiquetadas.

PASO 3: Dividir los datastore: Se deben dividir el datastore en dos partes: Un conjunto de entrenamiento, para entrenar la red, y otro conjunto de prueba, para evaluar la precisión de la red.

PASO 4: Importar una CNN y modificarla para convertirla en una deepLab v3+: Cargando una red previamente entrenada, y utilizando la función deeplabv3plusLayers se crea la arquitectura de codificador-decodificador necesaria para el etiquetado a nivel de píxel. Por

último, se modifica la última capa de la deepLab para que la salida la red genere la clasificación con el número de clases que se le ingresaron a la red pre-entrenada.

PASO 5: Entrenar y evaluar la red: Definir los hiperparámetros de la red y entrenarla.

A continuación, se presenta el código implementado en MATLAB para entrenar la red:

```
% Red pre-entrenada
resnet18(); % Se carga red preentrenada a trabajar
load('gTruth.mat'); % Se cargan datos etiquetados
imds = imageDatastore('folder_addess_img','FileExtensions','img_format'); % Se
cargan imágenes originales
pxds = pixelLabelDatastore(gTruth); % crear un objeto para leer datos de
etiquetas de píxeles

% Ahora se debe tener claro la recurrencia de cada clase o dato etiquetado
% Frecuencia de pixeles
tbl = countEachLabel(pxds)
frequency = tbl.PixelCount/sum(tbl.PixelCount);
```

El 100% de las imágenes utilizadas se dividieron en dos grupos, el de entrenamiento y el de evaluación de la red, el de entrenamiento fue el 80% y el de test es el 20% restante. Del 80% de entrenamiento se utilizó un 75% para entrenar la red y el 25% para realizar la validación al final de cada época.

```
rng(0);
numFiles = numel(imds.Files);
indicesrand = randperm(numFiles);
numTrain = round(0.6*numFiles);
trainingIdx = indicesrand(1:numTrain);
numVal = round(0.2*numFiles);
valIdx = indicesrand(numTrain+1:numTrain+numVal);
testIdx = indicesrand(numTrain+numVal+1:end);

% Images datastores para entrenamiento
trainingImages = imds.Files(trainingIdx);
valImages = imds.Files(valIdx);
testImages = imds.Files(testIdx);

imdsTrain = imageDatastore(trainingImages);
imdsVal = imageDatastore(valImages);
imdsTest = imageDatastore(testImages);

% % Extraer info clases & IDs
classes = pxds.ClassNames;
labelIDs = 1:4;

% Pixel label datastores para entrenamiento
```

```
trainingLabels = pxds.Files(trainingIdx);
valLabels = pxds.Files(valIdx);
testLabels = pxds.Files(testIdx);

pxdsTrain = pixelLabelDatastore(trainingLabels, classes, labelIDs);
pxdsVal = pixelLabelDatastore(valLabels, classes, labelIDs);
pxdsTest = pixelLabelDatastore(testLabels, classes, labelIDs);

numTrainingImages = numel(imdsTrain.Files);
numValImages = numel(imdsVal.Files);
numTestingImages = numel(imdsTest.Files);

% Creación de DeepLab v3+
```

Se utilizó la función deeplabv3plusLayers para crear una red neuronal convolucional deepLab v3+ para la segmentación semántica de imágenes. La función deeplabv3plusLayers tiene como entrada el tamaño de las imágenes de entrada a la red, el número de clases para que la red clasifique y por último tendrá la red base.

```
% Se especifica el tamaño de las imagenes de la red
imageSize = size(I);
% Se especifica el número de clases de la red
numClasses = numel(pxds.ClassNames);
% Por último se establece la red sobre la que se basara, que en este caso
% es resnet18

lgraph = deeplabv3plusLayers(imageSize, numClasses, "resnet18");

imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount; % Se especifica la frecuencia
con que aparecen las clases en las imagenes
classWeights = median(imageFreq) ./ imageFreq;
```

Ahora se procede a la creación de la última capa de la red, para esto se le transfiere los pesos que están en classWeights, de igual forma se le transfiere el nombre de las clases a segmentar, que se almacenaron en tbl.Name, esto lo que permite es que la salida de la red tenga una imagen segmentada con las clases que se desea.

```
pxLayer =
pixelClassificationLayer('Name','labels','Classes',tbl.Name,'ClassWeights',class
Weights);
% Ahora se remplaza ultima capa de la red por la que se acaba de crear
lgraph = replaceLayer(lgraph,"classification",pxLayer);

% Definir los datos de validación
dsVal = combine(imdsVal,pxdsVal);

% Definir opciones de entrenamiento
options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise',...
```

```
    'LearnRateDropPeriod',10,...
    'LearnRateDropFactor',0.3,...
    'Momentum',0.9, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'ValidationData',dsVal,...
    'MaxEpochs',30, ...
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2,...
    'Plots','training-progress',...
    'ValidationPatience', 4);

% Datos de entrenamiento
dsTrain = combine(imdsTrain, pxdsTrain);
```

Finalmente, para iniciar el entrenamiento se utiliza la función trainNetwork, la cual tiene como entrada los datos de entrenamientos que están se definieron en dsTrain, la segunda entrada es lgraph que es la red neuronal deepLab v3+ que se modificó anteriormente en su última capa y por último tiene como entrada las opciones de entrenamiento que se definieron y almacenaron en options

```
% Finalmente para iniciar el entrenamiento
% se le transfieren options que son las opciones de entrenamiento
% especificadas anteriormente
% se le transfiere dsTrain que son los datos de entrenamientos
% y lgraph que es la red neuronal deepLab v3+ que se le modifico la ultima
% capa
[net, info] = trainNetwork(dsTrain,lgraph,options)
```

Las gráficas finales del entrenamiento se muestran en la figura 11, donde la precisión de validación fue del 99.59%

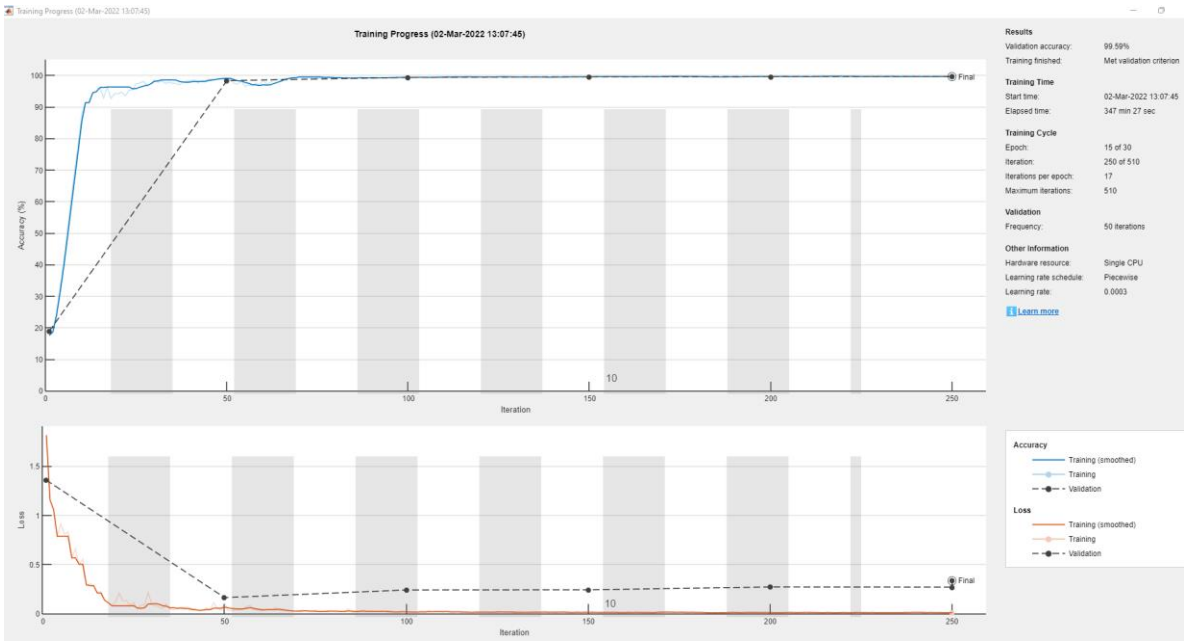

Figura 11 Resultados del entrenamiento

Ahora se procede a evaluar la precisión del entrenamiento con las imágenes que no ingresaron al entrenamiento y que corresponde al 20% de las imágenes iniciales.

```
pxdsResults = semanticseg(imdsTest,net, ...
    'MiniBatchSize',4, ...
    'WriteLocation',tempdir, ...
    'Verbose',false);
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTest,'Verbose',false);
```

La función evaluateSemanticSegmentation tiene como entradas fundamentales dos, una con imágenes etiquetadas a partir de la red que se han entrenado, donde se establece un lote pequeño para reducir el coste computacional, y otra entrada con los mismos datos, pero sin pasar por la red, lo cuales se han segmentado de manera manual y a partir de las diferencias poder evaluar la precisión de la red. Una de las propiedades de la función evaluateSemanticSegmentation es NormalizedConfusionMatrix que muestra la matriz de confusión normalizada. La matriz de confusión normalizada se puede ver en la tabla 7, la cual permite analizar el desempeño de la red.

Tabla 7 Matriz de confusión normalizada

True Class	**Fondo**	0,9965	0,0007	0,0010	0,0018
	Obj1	0,0023	0,9776	0,0199	0,0002
	Obj2	0,0021	0,0034	0,9943	0,0002
	Base	0,0088	0,0003	0,0046	0,9863
		Fondo	**Obj1**	**Obj2**	**Base**
		Predited Class			

A continuación, se muestra el código para validar el funcionamiento de la red con imágenes, donde el objetivo es segmentar tres objetos: la base del robot, un objeto rojo y otro naranja. En las figuras 12 está la imagen ingresada a la red y en la figura 13 se muestra la imagen segmentada que genera la red.

```
%% Cargar red entrenada
load('net.mat');
%% Segmentar imagen
image = ptCloudRef; % cargar nube de puntos a segmentar
I = image.Color; % especificar la ubicación de la imagen rgb
I = imresize(I,[1080 1920]); % redimensionar la imagen al tamaño de la red
C = semanticseg(I, net); % matriz con la clase de cada pixel de la imagen rgb
B = labeloverlay(I,C); % fusionar imagen rgb con matriz de clases segmentada
```

Figura 12 Input 2. Segmentación semántica

Figura 13 Output 2. Segmentación semántica

4. PLANEACIÓN DE TRAYECTORIAS

La planeación de trayectorias permite conocer las posiciones a las cuales el robot debe llegar para trasladarse de un punto inicial a uno final. Para esto se debe conocer la geometría del robot y los obstáculos que estarán distribuidos a lo largo del espacio de trabajo. Para esto primero se definió la geometría del robot y posteriormente la forma en como el robot va del punto inicial al final.

4.1. Robot UR3

El robot UR3 (figura 14) es un robot de Universal Robots con una carga efectiva de 3 kg y un peso de 11 kg. Permite que todas las articulaciones de la muñeca giren 360 grados y las articulaciones finales pueden rotarse infinitamente, así como un dispositivo de detección de fuerza único para garantizar la seguridad.

Figura 14 UR3

- **Dimensiones del robot UR3**

En la figura 15 se pueden ver la geometría del robot proporcionada por el fabricante [17]. Esa es la posición inicial del robot.

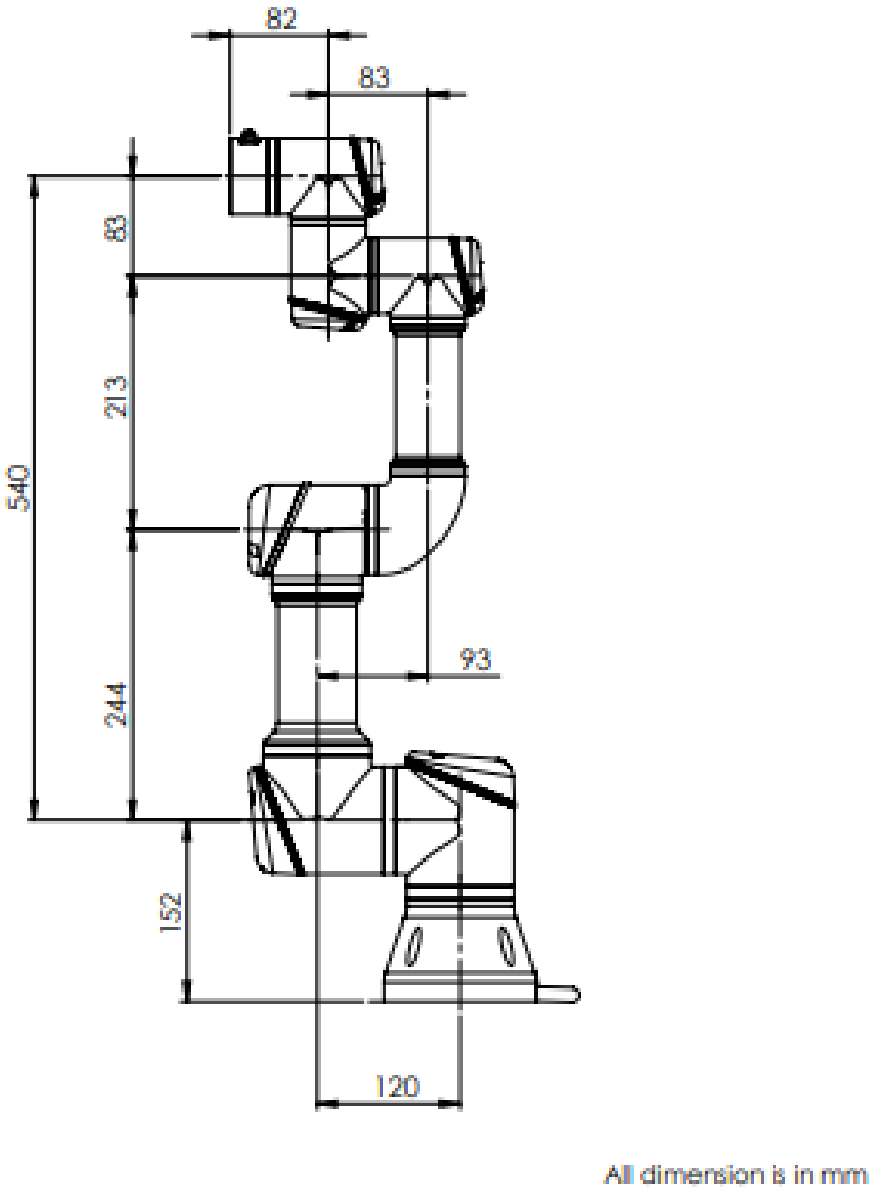

Figura 15 Medidas de UR3

4.2. Cinemática

La cinemática del robot estudia el movimiento del mismo con respecto a un sistema de referencia. Así, la cinemática se interesa por la descripción analítica del movimiento espacial del robot como una función del tiempo, y en particular por las relaciones entre la posición y la orientación del extremo final del robot con los valores que toman sus coordenadas articulares.

En la cinemática de un robot existen dos problemas fundamentales a resolver; el problema cinemático directo y el problema cinemático inverso.

- **Cinemática directa**

Permite encontrar una matriz homogénea de transformación T que relacione la posición y orientación del extremo del robot respecto del sistema de referencia fijo situado en la base de este. Para esto se han escogido coordenadas cartesianas y ángulos de Euler. Para un robot de seis grados de libertad, la solución al problema cinemático directo vendrá dada por las relaciones:

$$\begin{aligned} x &= f_x(q_1, q_2, q_3, q_4, q_5, q_6) \\ y &= f_y(q_1, q_2, q_3, q_4, q_5, q_6) \\ z &= f_z(q_1, q_2, q_3, q_4, q_5, q_6) \\ \alpha &= f_\alpha(q_1, q_2, q_3, q_4, q_5, q_6) \\ \beta &= f_\beta(q_1, q_2, q_3, q_4, q_5, q_6) \\ \gamma &= f_\gamma(q_1, q_2, q_3, q_4, q_5, q_6) \end{aligned}$$

x, y & z representan las coordenadas en el plano cartesiano y $\alpha, \beta \,\&\, \gamma$ representan los ángulos de Euler que se forman al girar con respecto a los ejes cartesianos x, y & z respectivamente. Para robots de más grados de libertad es mejor utilizar las matrices de transformación homogénea. Al momento de la obtención de las matrices de transformación homogénea se necesitan 4 transformaciones básicas, las cuales son una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento i con el sistema del elemento i-Estas 4 transformaciones son:

- Rotación alrededor del eje z_{i-1} un ángulo θ_i.
- Traslación a lo largo del z_{i-1} una distancia d_i; vector $d_i(0,0,d_i)$.
- Traslación a lo largo del x_i una distancia a_i; vector $a_i(0,0,a_i)$.
- Rotación alrededor del eje x_i un ángulo α_i.

Dado que el producto de matrices no es conmutativo, las transformaciones se deben realizar en el orden indicado. De este modo se tiene que:

$$^{i-1}A_i = T(z,\theta_i)T(0,0,d_i)T(a_i,0,0)T(x,\alpha_i) \qquad (3)$$

La ecuación 4 se tomó de [13] y de [14] donde se aplicó. De igual forma en este trabajo se evalúo su funcionamiento.

$$ {}^{i-1}A_i = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & a_{i-1} \\ S\theta_i C\alpha_{i-1} & C\theta_i C\alpha_{i-1} & -S\alpha_{i-1} & -S\alpha_{i-1} d_i \\ S\theta_i S\alpha_{i-1} & C\theta_i S\alpha_{i-1} & C\alpha_{i-1} & C\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4) $$

Así, para el UR3 que es un robot de 6 GDL, se tiene que la posición y orientación del eslabón final vendrá dada por la matriz T:

$$ {}^{0}A_6 = T = {}^{0}A_1 \, {}^{1}A_2 \, {}^{2}A_3 \, {}^{3}A_4 \, {}^{4}A_5 \, {}^{5}A_6 \quad (5) $$

Para simplificar la ecuación anterior se hace uso del algoritmo Denavit-Hartenberg, el cual permite cancelar operaciones redundantes, y estas simplificaciones se dan debido a una postura determinada que toma el mecanismo, en este caso en la figura 16 ve se la postura y las coordenadas que toma el robot UR3.

Los parámetros que conforman el algoritmo son los siguientes:

θ_i: Es el ángulo que forman los ejes x_{i-1} y x_i medido en un plano perpendicular al eje z_{i-1} utilizando la regla de la mano derecha. Se trata de un parámetro variable en articulaciones giratorias.
d_1: Es la distancia a lo largo del eje z_{i-1} desde el origen del sistema de coordenadas (i-1)-ésimo hasta la intersección del eje z_{i-1}con el eje z_i. Se trata de un parámetro variable en articulaciones prismáticas.
a_1: Es la distancia a lo largo del eje x_i que va desde la intersección del eje z_{i-1} con el eje x_i hasta el origen del sistema i-ésimo en el caso de articulaciones giratorias. En el caso de articulaciones prismáticas, se calcula como la distancia más corta entre los ejes z_{i-1} y z_i.
α_1: Es el ángulo de separación del eje z_{i-1} y el eje z_i, medido en un plano perpendicular al eje x_i, utilizando la regla de la mano derecha.

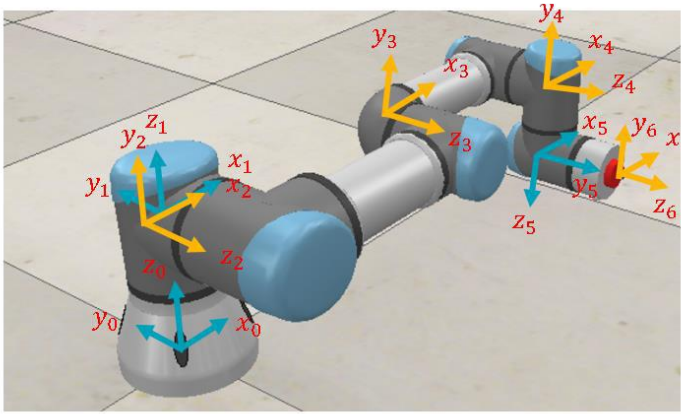

Figura 16 Sistema de coordenadas conjuntas del robot UR3 [16]

Los valores de los parámetros Denavit-Hartenberg son los siguientes:

Tabla 8 parámetros Denavit-Hartenberg

i	a_i/mm	α_i/rad	d_i/mm	θ_i/rad
1	0	0	0.1519	θ_1
2	0	$\pi/2$	0	θ_2
3	-0.24365	0	0	θ_3
4	-0.21325	0	0.11235	θ_4
5	0	$\pi/2$	0.08535	θ_5
6	0	-$\pi/2$	0.0819	θ_6

Por tanto, las matrices de transformación homogénea se simplifican a las siguientes matrices:

$$^0T_6 = T = {}^0T_1(\theta_1) * {}^1T_2(\theta_2) * {}^2T_3(\theta_3) * {}^3T_4(\theta_4) * {}^4T_5(\theta_5) * {}^5T_6(\theta_6) \quad (12)$$

$$0_{T_6} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

En el cual al realizar las operaciones se obtiene el siguiente resultado:

$$n_x = C\theta_6(S\theta_1 S\theta_5 + C\theta_{234} * C\theta_1 C\theta_5) - S\theta_{234} C\theta_1 S\theta_6 \ (14)$$

$$n_y = -C\theta_6(C\theta_1 S\theta_5 - C\theta_{234} C\theta_5 S\theta_1) - S\theta_{234} S\theta_1 S\theta_6 \ (15)$$

$$n_y = C\theta_{234} S\theta_6 + S\theta_{234} C\theta_5 C\theta_6 \ (16)$$

$$o_x = -S\theta_6(S\theta_1 S\theta_5 + C\theta_{234} C\theta_1 C\theta_5) - S\theta_{234} C\theta_1 C\theta_6 \ (17)$$

$$o_y = S\theta_6(C\theta_1 S\theta_5 - C\theta_{234} C\theta_5 S\theta_1) - S\theta_{234} C\theta_6 S\theta_1 \ (18)$$

$$o_z = C\theta_{345} C\theta_6 - S\theta_{234} C\theta_5 S\theta_6 \quad (19)$$

$$a_x = C\theta_5 S\theta_1 - C\theta_{234} C\theta_1 S\theta_5 \quad (20)$$

$$a_y = -C\theta_1 C\theta_5 - C\theta_{234} S\theta_1 S\theta_5 \quad (21)$$

$$a_z = -S\theta_{345}S\theta_5 \quad (22)$$

$$p_x = d_6(C\theta_5 S\theta_1 - C\theta_{234}C\theta_1 S\theta_5) + d_2 S\theta_1 + d_3 S\theta_1 + d_4 S\theta_1 - a_3 C\theta_{23}C\theta_1 - a_2 C\theta_1 C\theta_2 + d_5 S\theta_{234}C\theta_1 \quad (23)$$

$$p_y = d_5 S\theta_{234}S\theta_1 - d_2 C\theta_1 - d_3 C\theta_1 - d_4 C\theta_1 - a_3 C\theta_{23}S\theta_1 - a_2 C\theta_2 S\theta_1 - d_6(C\theta_1 C\theta_5 + C\theta_{234}S\theta_1 S\theta_5) \quad (24)$$

$$p_z = d_1 - a_3 S\theta_{23} - a_2 S\theta_2 - d_5(C\theta_{23}C\theta_4 - S\theta_{23}S\theta_4) - d_6 S\theta_5(C\theta_{23}S\theta_4 + S\theta_{23}C\theta_4) \quad (25)$$

La validación de la cinemática directa se utilizó el software matlab para poder tener datos precisos sobre el modelo desarrollado. La cinemática directa al ser una operación que realiza multiplicaciones similares, una por cada grado de libertad, y valiéndose de las operaciones que permite Matlab se construyó una función que se le dio por nombre cdur3 donde esta función tiene como entradas los parámetros de Denavit-Hartenberg, que en este caso en particular están en la tabla 8. La función es la siguiente:

```
function [A] = cdur3(q,d,a,alfa)
% q = Rotacion alrederdor del eje z
% d = Traslación a lo largo de z
% a = Traslación a lo largo de x
% alfa = rotación alrededor de x
    A = [cos(q) -sin(q) 0 a;
        sin(q)*cos(alfa) cos(q)*cos(alfa) -sin(alfa) -sin(alfa)*d;
        sin(q)*sin(alfa) cos(q)*sin(alfa) cos(alfa) cos(alfa)*d;
        0 0 0 1];
End
```

Por tanto, la cinemática directa del UR3 es la siguiente:

```
% Cinematica directa
[A1] = cdur3(q1,d1,a1,alfa1);
[A2] = cdur3(q2,d2,a2,alfa2);
[A3] = cdur3(q3,d3,a3,alfa3);
[A4] = cdur3(q4,d4,a4,alfa4);
[A5] = cdur3(q5,d5,a5,alfa5);
[A6] = cdur3(q6,d6,a6,alfa6);

A06 = A1*A2*A3*A4*A5*A6
```

Para la validación se hicieron 10 pruebas. En la tabla 9 se muestran los ángulos ingresados, los cuales están de grados, en la tabla 10 se muestran los datos generados por el controlador del UR3 para indicar la ubicación y ángulos de rotación del gripper y en la tabla 11 se muestran los datos generados por la cinemática directa desarrollada.

Tabla 9 Ángulos ingresados

Prueba	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
1	90	-20	20	10	90	0
2	-90	-20	60	0	90	0
3	-150	-60	20	25	90	0
4	180	0	-45	70	90	0
5	180	0	-100	45	100	0
6	270	0	-90	-30	100	70
7	200	-70	-120	20	100	0
8	200	-70	150	-90	90	0
9	180	-100	-90	20	90	0
10	120	-145	-90	200	-100	0

Tabla 10 Validación cinemática directa real

	Real					
Prueba	x (mm)	y (mm)	z (mm)	RX(rad)	RY(rad)	RZ(rad)
1	111,33	-661,48	111	1,7450	-0,0060	0,0030
2	-110,34	520,51	-119,08	0,0170	2,8470	1,3290
3	408,09	365,44	480,02	1,3100	0,0000	2,0930
4	572,75	113,66	125,48	2,0130	0,0000	1,5690
5	409,77	75,5	503,97	0,6240	-0,0780	1,4680
6	-74,01	199,94	609,78	0,3940	0,5070	-1,8790
7	-346,86	-45,4	467,49	-1,4050	-0,1140	2,1340
8	-311,45	228,05	128,39	2,0020	0,0340	1,3740
9	-470,31	116,41	479,03	-1,4170	0,0000	-1,3920
10	168,46	438,12	-86,76	2,1540	-0,5200	-2,4320

Tabla 11 Validación modelo cinemático directo desarrollado

	Modelado					
Prueba	x (mm)	y (mm)	z (mm)	RX(rad)	RY(rad)	RZ(rad)
1	112,35	-660,6862	110,0426	1,7453	0,0000	0,0000
2	-112,35	518,9291	-119,4995	0,0174	2,8481	1,3315
3	408,103	365,349	478,854	1,3094	0,0000	2,0944
4	573,0744	112,35	125,2189	2,0071	0,0000	1,5708
5	410,3501	71,2127	504,0645	0,6181	-0,1427	1,4700
6	-71,2127	200,9148	609,8696	0,3881	0,5125	-1,9676

7	-345,3672	-49,9204	468,3913	-1,3989	-0,0302	2,0918
8	307,8372	231,6039	127,9298	1,9199	0,0000	1,3963
9	-470,7997	112,35	480,0085	-1,3963	0,0000	1,5708
10	-170,3558	437,4905	-86,7628	-2,4745	-0,0998	-2,4745

- **Cinemática inversa**

Permite encontrar los valores que deben adoptar las coordenadas articulares del robot para que su extremo se posicione y oriente según una determinada localización espacial. El desarrollo de la cinemática inversa se baso en el modelo desarrollado en la literatura en [19].

Primero se toma la definición de la matriz de transformación Homogénea ${}^0T_6$ como:

$$ {}^0T_6 \;=\; T \;=\; {}^0A_1\,{}^1A_2\,{}^2A_3\,{}^3A_4\,{}^4A_5\,{}^5A_6 \qquad (26) $$

Donde como se indica en la ecuación 13

$$ 0_{T_6} \;=\; \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Se toma a ${}^0T_6$ como T por comodidad.

$$ {}^1A_2\,{}^2A_3\,{}^3A_4\,{}^4A_5 = \; {}^0A_1{}^{-1}T\,{}^5A_6{}^{-1} \qquad (27) $$

De la ecuación 27 se obtiene que:

$$ (p_y - d_6 a_y)\cos(\theta_1) - (p_x - d_6 a_x)\sin(\theta 1) = -d4 \; (28) $$

De donde se puede despejar θ_1

$$ \theta_1 = \; Atan2\left(\pm\sqrt{1-\left(\frac{d_4}{\rho}\right)^2}, -\frac{d_4}{\rho}\right) - Atan2\left(\frac{b}{\rho}, \frac{a}{\rho}\right) \qquad (29) $$

Donde

$$ a \;=\; p_y - d_6 a_y;\; b \;=\; p_x - d_6 a_x;\; \rho = \sqrt{a^2 + b^2} \qquad (30) $$

De acuerdo a

$$ {}^1A_2\,{}^2A_3\,{}^3A_4\,{}^4A_5\,{}^5A_6 = \; {}^0A_1{}^{-1}T \qquad (31) $$

Se obtiene la siguiente igualdad:

$$n_x \sin(\theta_1) - n_x \cos(\theta_1) = \cos(\theta_6) \sin(\theta_5) \quad (32)$$
$$o_x \sin(\theta_1) - o_x \cos(\theta_1) = -\sin(\theta_6) \sin(\theta_5) \quad (33)$$

De la ecuación 30 y las ecuaciones 32 y 33 se obtiene:

$$\sin(\theta_5) = \pm\sqrt{\left(n_x \sin(\theta_1) - n_y \cos(\theta_1)\right)^2 + \left(o_x \sin(\theta_1) - o_y \cos(\theta_1)\right)^2} \quad (34)$$
$$\cos(\theta_5) = a_x \sin(\theta_1) - a_y \cos(\theta_1) \quad (35)$$
$$\theta_5 = \text{Atan2}(\sin(\theta_5), \cos(\theta_5)) \quad (36)$$

Del mismo modo se obtienen θ_6

$$\cos(\theta_6) = \frac{n_x \sin(\theta_1) - n_y \cos(\theta_1)}{\sin(\theta_5)} \quad (37)$$
$$\sin(\theta_6) = \frac{o_x \sin(\theta_1) - o_y \cos(\theta_1)}{-\sin(\theta_5)} \quad (38)$$
$$\theta_6 = \text{Atan2}(\sin(\theta_6), \cos(\theta_6)) \quad (39)$$

Al igual que en la cinemática directa, el modelo se ejecutó en el software de Matlab para tener precisión. El siguiente código es el correspondiente a la cinemática inversa hasta θ_6.

```
%% Escalares de la matriz de transformación homogénea total
nx = A06(1,1);
ny = A06(2,1);
nz = A06(3,1);

ox = A06(1,2);
oy = A06(2,2);
oz = A06(3,2);

ax = A06(1,3);
ay = A06(2,3);
az = A06(3,3);

px = A06(1,4);
py = A06(2,4);
pz = A06(3,4);
%%
%% Ecuaciones para hallar los ángulos

% Hay 8 posibles combinaciones de ángulos
% Para th1 hay 2 posibles valores
th1 = zeros(8,1);

% Por cada valor de th1, hay 2 posibles conjuntos de valores de th5 y th6
th5 = zeros(8,1);
th6 = zeros(8,1);
```

```
% Por cada par de valores th5 y th6, hay 2 posibles conjuntos de valores de
% th2, th3 y th4
th2 = zeros(8,1);
th3 = zeros(8,1);
th4 = zeros(8,1);

%% Se hallan los 2 valores de th1
a = py-d6*ay;
b = px-d6*ax;
rho = sqrt(a^2+b^2);

%%
th1(1) = atan2(sqrt(1-(d4/rho)^2),-d4/rho)-atan2(b/rho,a/rho);
for i=2:4
   th1(i)=th1(1);
end

th1(5) = atan2(-sqrt(1-(d4/rho)^2),-d4/rho)-atan2(b/rho,a/rho);
for i=6:8
   th1(i)=th1(5);
end

%% Ahora, se hallan los valores de th5 y th6 dados los valores de th1
S5 = zeros(8,1);
C5 = zeros(8,1);

%%
for i=1:2
    S5(i) = sqrt((nx*sin(th1(1))-ny*cos(th1(1)))^2+(ox*sin(th1(1))-
oy*cos(th1(1)))^2);
End

for i=3:4
    S5(i) = -sqrt((nx*sin(th1(1))-ny*cos(th1(1)))^2+(ox*sin(th1(1))-
oy*cos(th1(1)))^2);
End

for i=5:6
    S5(i) = sqrt((nx*sin(th1(5))-ny*cos(th1(5)))^2+(ox*sin(th1(5))-
oy*cos(th1(5)))^2);
End

for i=7:8
    S5(i) = -sqrt((nx*sin(th1(5))-ny*cos(th1(5)))^2+(ox*sin(th1(5))-
oy*cos(th1(5)))^2);
end

for i=1:4
    C5(i) = ax*sin(th1(1))-ay*cos(th1(1));
End
```

```
for i=5:8
    C5(i) = ax*sin(th1(5))-ay*cos(th1(5));
End

%%
th5 = zeros(5,1);

%%
for i=1:8
    th5(i) = atan2(S5(i),C5(i));
end

%%
S6 = zeros(8,1);
C6 = zeros(8,1);
%%
for i=1:8
    S6(i) = (ox*sin(th1(i))-oy*cos(th1(i)))/(-sin(th5(i)));
    C6(i) = (nx*sin(th1(i))-ny*cos(th1(i)))/(sin(th5(i)));
    th6(i) = atan2(S6(i),C6(i));
end
```

Ahora se procederá a encontrar los ángulos faltantes, para esto se busca la siguiente igualdad

$$ {}^{1}A_2\ {}^{2}A_3\ {}^{3}A_4\ {}^{4}A_5 = \ {}^{0}A_1{}^{-1}\ T\ {}^{5}A_6{}^{-1} \qquad (36) $$

$$ \theta_{234} \ = \ \theta_2 + \theta_3 + \theta_4 \qquad (37) $$

De las ecuaciones 36 y 37 se puede obtener la siguiente ecuación

$$ \cos(\theta_{234}) = \cos(\theta_5)[\cos(\theta_6)(n_x \cos(\theta_1)\ + n_y \sin(\theta_1)\ ) - \sin(\theta_6)(o_x \cos(\theta_1)\ + o_y \sin(\theta_1)\ )] - sin(\theta_5)(a_y \sin(\theta_1)\ + a_x \cos(\theta_1)\ ) \qquad (38) $$

$$ \sin(\theta_{234}) \ = \ \cos(\theta_5)(n_z \cos(\theta_6)\ - o_z \sin(\theta_6)\ ) - \ a_z \sin(\theta_5) \qquad (39) $$

De las ecuaciones 38 y 39 se puede obtener la siguiente igualdad

$$ \theta_{234} \ = \ Atan2(\sin(\theta_{234}), \cos(\theta_{234})) \qquad (40) $$

Ahora, los ejes de rotación de los ángulos $\theta 2$, $\theta 3$ y $\theta 4$ son paralelos para el manipulador robótico UR3, luego esta sección del robot se puede trabajar como un robot manipulador R3 planar.

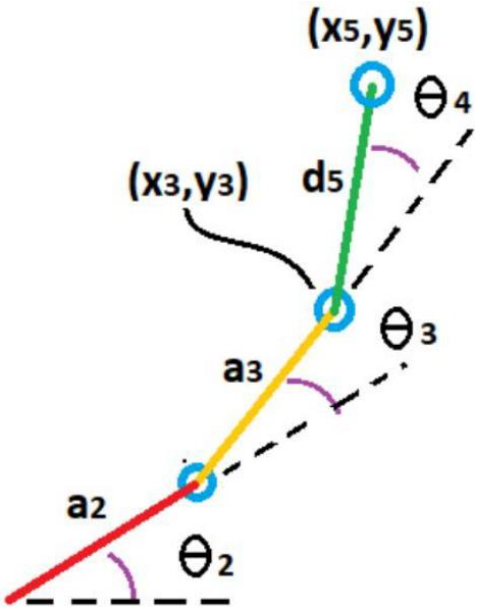

Figura 17 Ángulo auxiliar [20]

De la figura 17 se determina un ángulo auxiliar

$$\beta_4 = \theta_4 - \frac{\pi}{2} \quad (41)$$

$$\beta_{234} = \theta_2 + \theta_3 + \beta_4 = \theta_{234} - \frac{\pi}{2} \quad (42)$$

$$x_3 = x_5 - d_5 \cos(\beta_{234}) \quad (43)$$

$$y_3 = y_5 - d_5 \sin(\beta_{234}) \quad (44)$$

Donde x_5 y y_5 se obtienen de la matriz ${}^0A_1^{-1}{}^0T_6{}^5A_6^{-1}$

$$x_5 = sin(\theta_1)(p_y - a_y d_6) + cos(\theta_1)(p_x - a_x d_6) \quad (45)$$

$$y_5 = p_z - d_1 - a_z d_6$$

Una vez hallados x_3 y y_3, los ángulos θ_2 y θ_3 se hallan aplicando las fórmulas de cinemática inversa de un robot plano de 2 grados de libertad:

$$\theta_3 = \pm 2 * Atan2\left(\sqrt{l_1^2 - (x_3^2 + y_3^2)}, \sqrt{(x_3^2 + y_3^2) - l_2^2}\right) \quad (46)$$

$$l_1 = a_2 + a_3;\ l_2 = a_2 - a_3 \quad (47)$$

$$\theta_2 = Atan2(y_3, x_3) - Atan2(a_3 \sin(\theta_3), a_2 + a_3 \cos(\theta_3)) \quad (48)$$

Teniendo θ_2, θ_3 y θ_{234}, se halla θ_4:

$$\theta_4 = \theta_{234} - \theta_2 - \theta_3 \quad (49)$$

```
%%
S234 = zeros(8,1);
C234 = zeros(8,1);
```

```
th234 = zeros(8,1);

%%
for i =1:8
    S234(i) = cos(th5(i))*(nz*cos(th6(i))-oz*sin(th6(i)))-az*sin(th5(i));
    C234(i) = cos(th5(i))*(cos(th6(i))*(nx*cos(th1(i))+ny*sin(th1(i)))-
sin(th6(i))*(ox*cos(th1(i))+oy*sin(th1(i))))-
sin(th5(i))*(ay*sin(th1(i))+ax*cos(th1(i)));
    th234(i) = atan2(S234(i),C234(i));
end

y5 = pz - d1 - az*d6
%%
x5 = zeros(8,1);
x3 = zeros(8,1);
y3 = zeros(8,1);
%%
for i=1:8
    x5(i) = px*cos(th1(i)) + py*sin(th1(i)) - ay*d6*sin(th1(i)) -
ax*d6*cos(th1(i));
    x3(i) = x5(i)-d5*cos(th234(i)-pi/2);
    y3(i) = y5-d5*sin(th234(i)-pi/2);
end

for i=1:8
    if isreal([sqrt((a2+a3)^2-(x3(i)^2+y3(i)^2)), sqrt((x3(i)^2+y3(i)^2)-(a2-
a3)^2)])
        th3(i) = ((-1)^(1+i))*2*atan2(sqrt((a2+a3)^2-(x3(i)^2+y3(i)^2)),
sqrt((x3(i)^2+y3(i)^2)-(a2-a3)^2));
        th2(i) = atan2(y3(i),x3(i))-atan2(a3*sin(th3(i)),a2+a3*cos(th3(i)));
        th4(i) = th234(i)-th3(i)-th2(i);
    else
        th1(i) = NaN;
        th2(i) = NaN;
        th3(i) = NaN;
        th4(i) = NaN;
        th5(i) = NaN;
        th6(i) = NaN;
    end
end


th = [th1,th2,th3,th4,th5,th6]
%%
for i=1:8
    for j=1:6
        while th(i,j)>pi
            th(i,j) = th(i,j) - 2*pi;
        end
        while th(i,j)<-pi
            th(i,j) = th(i,j) + 2*pi;
```

```
        end
    end
end

th_d = th*180/pi
```

Para la validación del modelo cinemático inverso se hicieron 5 pruebas. En las siguientes matrices se muestra el lugar en el espacio en milímetros y la dirección al que se debe llegar. La notación que se utiliza es P para saber que es una prueba y se van a hacer pruebas desde P_1 hasta P_5. Los resultados se exponen en la tabla 12 que permiten validar que el modelo cinemático inverso se desarrolló bien. En esta tabla se ponen las 8 posibles respuestas, de las cuales 4 son soluciones validas y las otras 4 son soluciones invalidas o NaN. Estos valores NaN se generan porque los ángulos pueden estar en una singularidad, que es donde se causan colisiones o las uniones llegan a su límite, lo que impide llegar a ese ángulo.

$$P_1 \rightarrow \begin{matrix} x = 120 \\ y = 431{,}98 \\ z = 213{,}78 \end{matrix} \quad \begin{matrix} Rx = 1{,}5708 \\ Ry = 0 \\ Rz = -3{,}1416 \end{matrix}$$

$$P_2 \rightarrow \begin{matrix} x = 4{,}1 \\ y = 467{,}98 \\ z = 313{,}78 \end{matrix} \quad \begin{matrix} Rx = 0 \\ Ry = 0 \\ Rz = 2{,}871 \end{matrix}$$

$$P_3 \rightarrow \begin{matrix} x = 0 \\ y = 390{,}04 \\ z = 313{,}78 \end{matrix} \quad \begin{matrix} Rx = 0 \\ Ry = 0 \\ Rz = 0 \end{matrix}$$

$$P_4 \rightarrow \begin{matrix} x = 100 \\ y = 490 \\ z = 180 \end{matrix} \quad \begin{matrix} Rx = 0 \\ Ry = 0 \\ Rz = 0 \end{matrix}$$

$$P_5 \rightarrow \begin{matrix} x = 100 \\ y = 490 \\ z = 180 \end{matrix} \quad \begin{matrix} Rx = 1{,}5708 \\ Ry = 0 \\ Rz = -3{,}1416 \end{matrix}$$

Tabla 12 Validación cinemática inversa prueba 2

Soluciones	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
1	-104,3923	-20,4550	69,1801	-138,7251	90,0000	-1,1115
2	-104,3923	43,4711	-69,1801	-64,2909	90,0000	-1,1115
3	-104,3923	-20,4550	69,1801	-138,7251	90,0000	-1,1115
4	-104,3923	43,4711	-69,1801	-64,2909	90,0000	-1,1115
5	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN

En la figura 18 se muestran los datos generados por la consola del robot ur3 en la prueba 2.

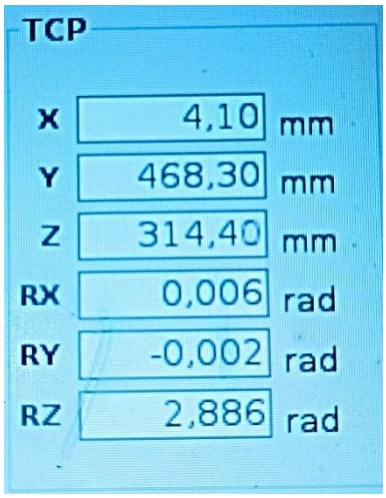

Figura 18 Posición y orientación del efector gripper en la prueba 2

Los ángulos que genera el robot para llegar a la posición de la figura 18 son:

θ_1: -103.89
θ_2: -20.46
θ_3: 69.19
θ_4: -138.73
θ_5: 90
θ_6: 360

En las 6 articulaciones el error entre el ángulo calculado por el controlador del robot y el generado por el modelo es menor al 1%.

4.3. Planeación de trayectorias

La planeación de trayectorias permite encontrar una ruta valida entre un punto inicial y uno final. La ruta puede ser un conjunto de estados (posición y orientación) o waypoints donde se requiere un mapa del entorno junto con los estados de inicio y objetivo como entrada, y este mapa se puede representar de diferentes maneras, como mapas de cuadrícula, espacios de estado y/o hojas de ruta topológicas.
Las técnicas de planificación de rutas incluyen dos tipos principales de algoritmos utilizados para vehículos autónomos:

- Los algoritmos de búsqueda basados en cuadrícula encuentran una ruta basada en el costo mínimo de viaje en un mapa de cuadrícula. Se pueden utilizar para aplicaciones como robots móviles en un entorno 2D. Sin embargo, el requisito de memoria para implementar algoritmos basados en cuadrícula aumenta con el número de dimensiones, como para un manipulador de robots 6 DOF.
- Los algoritmos de búsqueda basados en muestreo crean un árbol de búsqueda mediante el muestreo aleatorio de nuevos nodos o configuraciones de robot en un espacio de estado. Los algoritmos basados en muestreo son adecuados para espacios de búsqueda de baja y alta dimensión.

La planificación de rutas, junto con la percepción (visión) y los sistemas de control comprenden los tres bloques de construcción principales de la navegación autónoma para cualquier robot o vehículo que le permite agregar autonomía en sistemas como automóviles autónomos, manipuladores de robots, UGV y UAV.
En este caso se utilizó el algoritmo bidireccional- RRT para establecer la ruta de acceso del gripper a través de un mapa conocido. Éste se basa en la generación simultánea de dos árboles cuyas raíces son las configuraciones inicial y final donde estos crecen explorando el espacio vacío y buscándose entre sí hasta conectar obteniendo entonces una trayectoria como resultado. El algoritmo concluye en cuanto dichos árboles conectan. Si alcanzado el valor de Kmax ambos árboles no han coincidido, se devuelve un mensaje de error.
En cada iteración uno de los árboles agregará una nueva rama si es posible en dirección al punto aleatorio generado. De esta forma dicho árbol tendrá una función de exploración del espacio vacío, dado que qrand es una configuración que puede surgir en cualquier punto del escenario, marcando una dirección de crecimiento para el nodo más próximo.
El pseudocódigo de del algoritmo RRT bidireccional es el siguiente:

```
RRT-Bidirecional (qini, qfin)
{
  ArbolA[0] = qini;
  ArbolB[0] = qfin;
  Para k = 1 hasta Kmax
  {
    qrand = ConfiguraciónAleatoria( );
    Si (Extiende(ArbolA, qrand)≠rechazado)
    {
      Si (Extiende(ArbolB, qrand)= alcanzado Y Extiende(ArbolB, qrand)=alcanzado)
      {
      DevuelveCamino (ArbolA, ArbolB);
```

```
      }
    }
    Intercambiar (ArbolA, ArbolB)
  }
  Devuelve Error
}
```

Si ha habido una nueva rama, entonces existe un vértice nuevo qnew. Ahora le toca el turno al segundo árbol que tomará como meta no el punto aleatorio qrand, sino el nuevo vértice generado qnew. De esta forma dicho árbol no explora el espacio vacío, sino que intentará encontrar a su homólogo si es posible.
En la figura 19 se puede observar cómo se hace uso de la función Intercambiar, que alterna el orden de los árboles de manera que el reparto de ambas funciones sea equilibrado. Así pues, las acciones resultantes de dos iteraciones consecutivas serían:

1. Árbol A crece hacia qrand (1a iteración).
2. Árbol B crece hacia qnew de A.
3. Árbol B crece hacia qrand (2a iteración).
4. Árbol A crece hacia qnew de B.

De esta forma cada árbol gasta la mitad de su tiempo en explorar el espacio libre, y la otra mitad, en buscar a su compañero

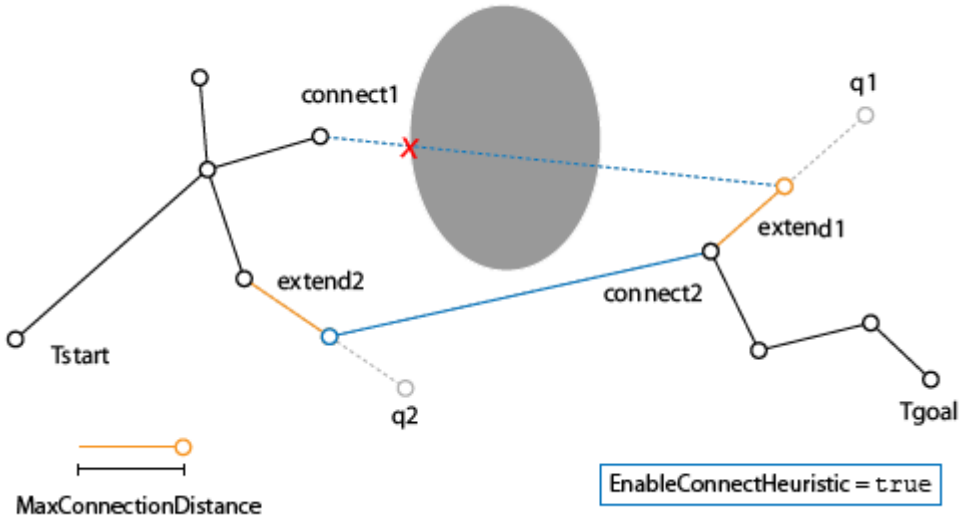

Figura 19 RRT bidireccional [21]

- **Implementación en MATLAB**

1. Cargar un modelo de robot de árbol de cuerpo rígido y se definen objetos de colisión colocados en el entorno de trabajo.
2. Se crean las mallas de colisión del robot, las cuales se usarán para evaluar la colisión del cuerpo del robot con cada objeto que se encuentre en el entorno.
3. Crear el planificador RRT para el modelo de robot y los obstáculos.
4. Especifique una configuración de inicio y meta.
5. Planificar el movimiento desde el principio hasta la configuración de la meta.
6. Se crean interpolaciones entre estados para hallar la mejor ruta entre el inicio y la meta.

Para cargar el modelo de robot se utiliza la función loadrobot, la cual mediante el parámetro "universalUR3" permite cargar el modelo de árbol de cuerpo rígido del robot UR3. De igual forma mediante la función collisionBox y collisionCylinder permite crear objetos rectangulares y cilíndricos respectivamente, donde como entrada tiene las dimensiones de los mismo.

```
robot = loadrobot('universalUR3'); % Modelo árbol de cuerpo rígido del robot UR3
```

Ahora se crean los objetos que se encuentran en la escena. En este caso se definieron tres objetos, primero es la plataforma donde esta el robot y los obstáculos y los otros dos son objetos con forma cilíndrica.

```
plane = collisionBox(1,0.75,0.05); % la plataforma al ser una mesa se puede
definir como un rectángulo con un espesor mínimo. La plataforma se creó con 1m
de ancho, 0.75m de profundidad y 0.05 de grueso.
Cilindro1r = collisionCylinder(radio1, altura1); % Los dos cilindros se crearon
con la función collisionCylinder. El primer parámetro es el radio del cilindro y
el segundo el alto del objeto.
```

Posteriormente estos objetos se almacenan en una matriz

```
env = {plane,Cilindro1r,Cilindro2r};
```

Los objetos de esta matriz están ubicados en el espacio tridimensional, por tanto, la ubicación se le pasa con los datos generados en la sección de visión artificial. Estos datos son la ubicación en X, Y y Z, los cuales son un vector que se deben ingresar en como matriz homogénea. Para esto se utiliza la función trvec2tform. En el siguiente código se le esta pasando la ubicación al objeto n que tiene como ubicación en el espacio la posición X, Y y Z.

```
env{n}.Pose = trvec2tform([x y z]);
```

La función manipulatorRRT tiene como entradas el modelo de árbol de cuerpo rígido del UR3 y los obstáculos creados mediante las funciones collisionBox y collisionCylinder. El objetivo de crear un planificador es dar a conocer al algoritmo de planeación de trayectorias el medio o ecosistema en donde se encuentra.

```
rrt = manipulatorRRT(robot,env);
```

Por último, se define los puntos inicial y final para que mediante la función plan se pueda planificar el movimiento desde el punto inicial al final, evitando los obstáculos en el camino. La función plan tiene entrada el ecosistema en el que se encuentra y las posiciones inicial y final de cada articulación del robot, para esto se utiliza la cinemática inversa del brazo robótico.

```
path = plan(rrt,startConfig,goalConfig);
```

Finalmente, se utiliza la función interpolate con la que se busca encontrar una mejor respuesta sobre la creada con la función plan. La función interpolate tiene como entrada el ecosistema creado con la función manipulatorRRT y la ruta generada.

```
interpPath = interpolate(rrt,path);
```

De esta manera se obtiene la ruta final. Al momento de enviar la trayectoria al controlador del robot se genera un problema debido a que la única articulación con movimiento libre o infinito es la última articulación, donde está situado el gripper, las otras 5 articulaciones tienen un giro máximo de 360°, por tanto, cuando se genera un movimiento en estas

articulaciones que supero el ángulo 360° se va generar un movimiento brusco, donde la articulación va girar 359° en sentido contrario para llegar al ángulo 1° y continuar la trayectoria. Para solventar este problema se desarrollo el siguiente código, el cual permite evaluar cuando un giro es superior a 10°, y si es así se le suman o restan 360° dependiendo si gira o no hacía la derecha.

```
tampath = size(path);
filaspath = tampath(1);
ind = 1;
 while ind<filaspath
 fila1 = (path(ind,:));
 fila2 = (path(ind+1,:));
 gradosdif = 10;
 gradosrad = deg2rad(gradosdif);
 a = fila2(1,:)<fila1(1,:)-gradosrad;
 b = fila2(1,:)>fila1(1,:)+gradosrad;
 c = find(a);
 d = find(b);
    if ~isempty(a(c))==1 | ~isempty(b(d))==1
        path(ind+1,c) = path(ind+1,c)+2*pi();
        path(ind+1,d) = path(ind+1,d)-2*pi();
        disp("Diferentes")
    end
ind = ind+1;
 end
```

En la figura 20 se visualiza una trayectoria generada en el ambiente virtual, donde entre el punto inicial y final se colocaron dos objetos sobre la plataforma, ante este escenario el algoritmo genera una trayectoria sobre los obstáculos.

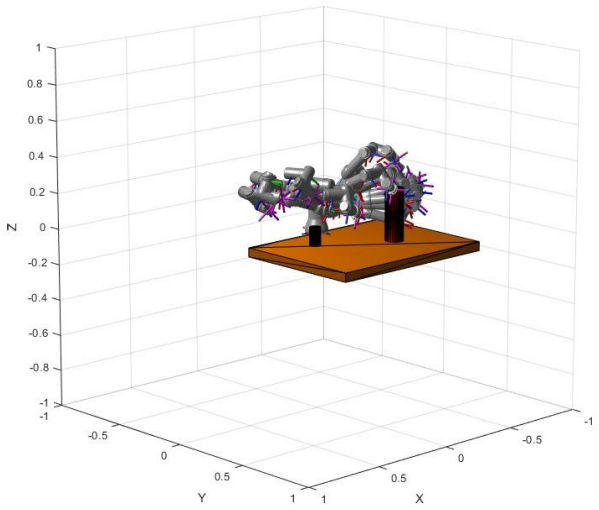

Figura 20 Ambiente virutal. Prueba 1

En la figura 21 se genera un ambiente virtual donde se crea una pieza adicional en la mitad de la escena para revisar la repuesta del algoritmo. En esta trayectoria la articulación 2 pasa de 360° a 1°, donde al aplicar las anteriores líneas de código se corrigió el problema generado y poder realizar una trayectoria sin colisiones.

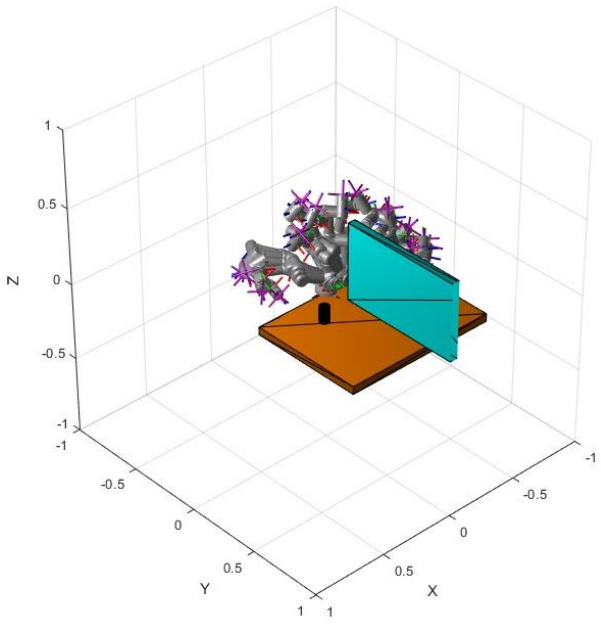

Figura 21 Ambiente virutal. Prueba 2

4.4. Protocolo de comunicación

Sistema de reglas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellas para transmitir información por medio de cualquier tipo de variación de una magnitud física. Se trata de las reglas o el estándar que define la sintaxis, semántica y sincronización de la comunicación, así como también los posibles métodos de recuperación de errores. Los protocolos pueden ser implementados por hardware, por software, o por una combinación de ambos.

- **Arquitectura del protocolo TCP/IP**

El protocolo TCP/IP fue creado antes que el modelo de capas OSI, así que los niveles del protocolo TCP/IP no coinciden exactamente con los siete que establece el OSI.
Existen descripciones del protocolo TCP/IP que definen de tres a cinco niveles. La figura 22 representa un modelo de cuatro capas TCP/IP y su correspondencia con el modelo de referencia OSI.

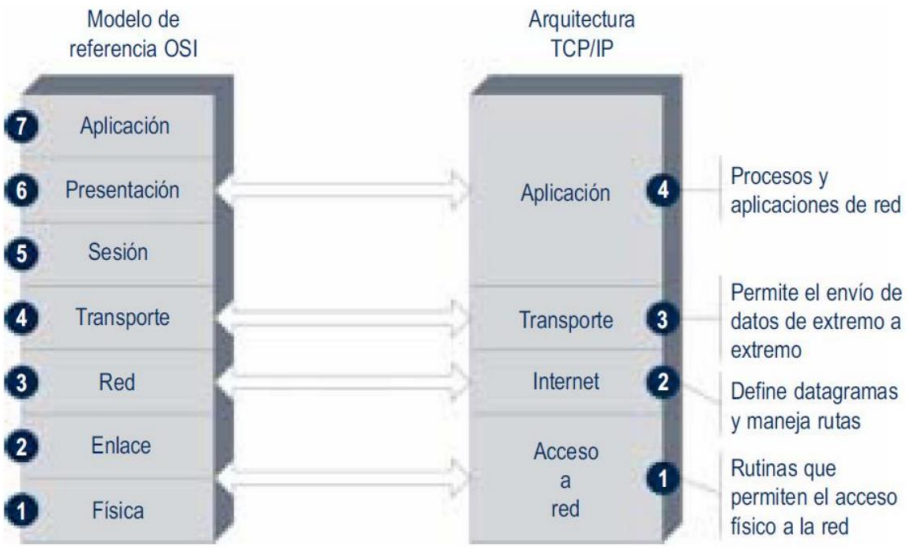

Figura 22 Correspondencia del modelo OSI con TCP/IP

Los datos que son enviados a la red recorren la pila del protocolo TCP/IP desde la capa más alta de aplicación hasta la más baja de acceso a red. Cuando son recibidos, recorren la pila de protocolo en el sentido contrario.
Durante estos recorridos, cada capa añade o sustrae cierta información de control a los datos para garantizar su correcta transmisión. Como esta información de control se sitúa antes de los datos que se transmiten, se llama cabecera (header). La figura 23 se puede ver

cómo cada capa añade una cabecera a los datos que se envían a la red. Este proceso se conoce como encapsulado.

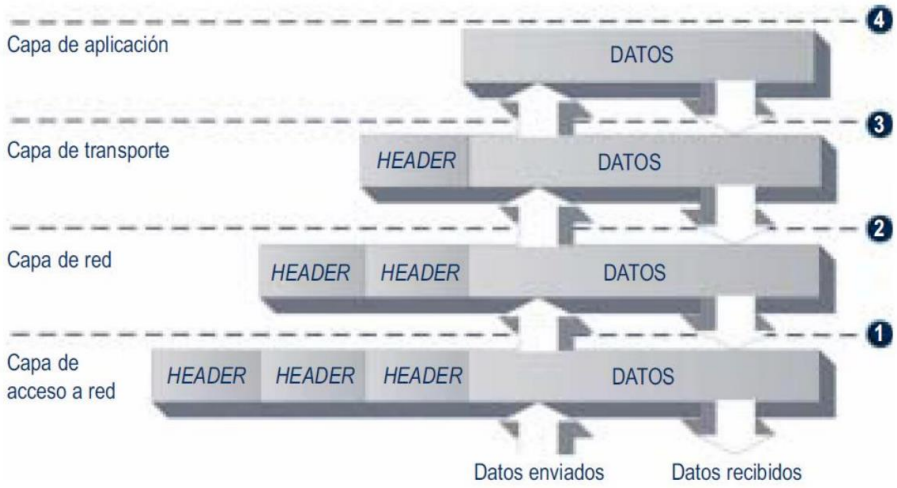

Figura 23 Encapsulado de datos por los niveles TCP/IP

Si en vez de transmitir datos se trata de recibirlos, el proceso sucede al revés. Cada capa elimina su cabecera correspondiente hasta que quedan sólo los datos.
En teoría cada capa maneja una estructura de datos propia, independiente de las demás, aunque en la práctica estas estructuras de datos se diseñan para que sean compatibles con las de las capas adyacentes. Se mejora así la eficiencia global en la transmisión de datos.

Para realizar la conexión entre Matlab y el Robot UR3 se realizó el siguiente código en Python:

```
import socket
import time
def conection(host):
  port = 30004
  print("Starting Program")
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
  s.bind((host, port))
  s.listen(5)
  c, addr = s.accept()
  return c

def enviarPosicion(c,pos):
    c.send(bytes(pos, "ascii"))

def leerPosicion(c):
    msg = c.recv(1024)
```

```
    return (msg)
```

En el cual se definen las funciones conection, que se encarga de establecer la comunicación inicial entre el Robot y Matlab, enviarPosicion, que envía la posición como un texto en formato ascii, y la función leerPosicion, la cual lee el mensaje que se envía desde el robot, el cual contiene los ángulos actuales de cada articulación del robot, estas posiciones se utilizan para asegurar que el robot está siguiendo la posición que se le envió.
Para configurar la dirección IP del robot, primero se debe saber en qué dirección se encuentra el ordenador a utilizar, en este caso la configuración de red se puede ver en la figura 24.

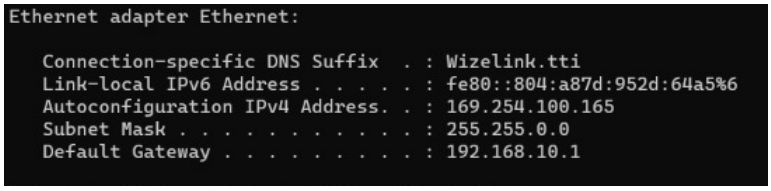

Figura 24 Dirección IP del ordenador.

Teniendo esta dirección IP, se procede a ir a la pantalla de configuración del robot y a la pestaña de red que es la que se muestra en la figura 25.

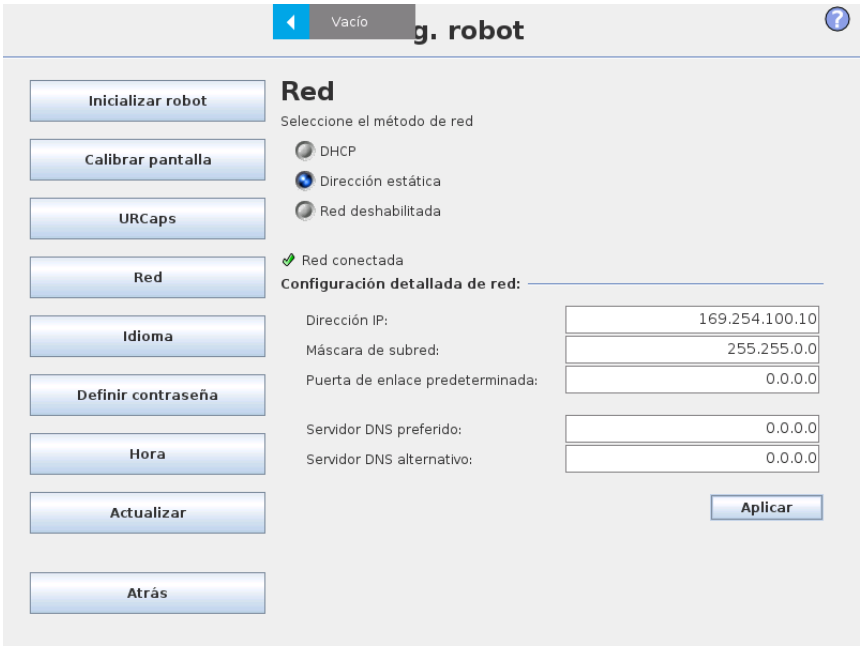

Figura 25 Configuración de red del UR3.

Estando en esa pestaña, se configura la dirección IP del Robot dentro de la misma red del ordenador, con su respectiva máscara de subred. El robot UR3 está configurado con la dirección IP de 169.254.100.10. Posteriormente se puede realizar la petición desde Matlab para conectarse con las siguientes líneas de código:

```
host = "169.254.63.211"
port = 30004;
port = int16(port);
c = py.connect_to_ur3.conection(host);
```

- **Descripción del programa**

En la rutina AntesDeIniciar, el robot se mueve al punto inicial de la trayectoria y a la variable socket_1 se le asigna una dirección IP y un puerto a un servidor correspondiente. En este caso, hay un PC con la dirección IP 169.254.100.165 y un programa de servidor escuchando en el puerto 30004. Socket_1 será falso si no hay un servidor escuchando o verdadero si lo hay. El script que se utiliza se muestra en la figura 26.

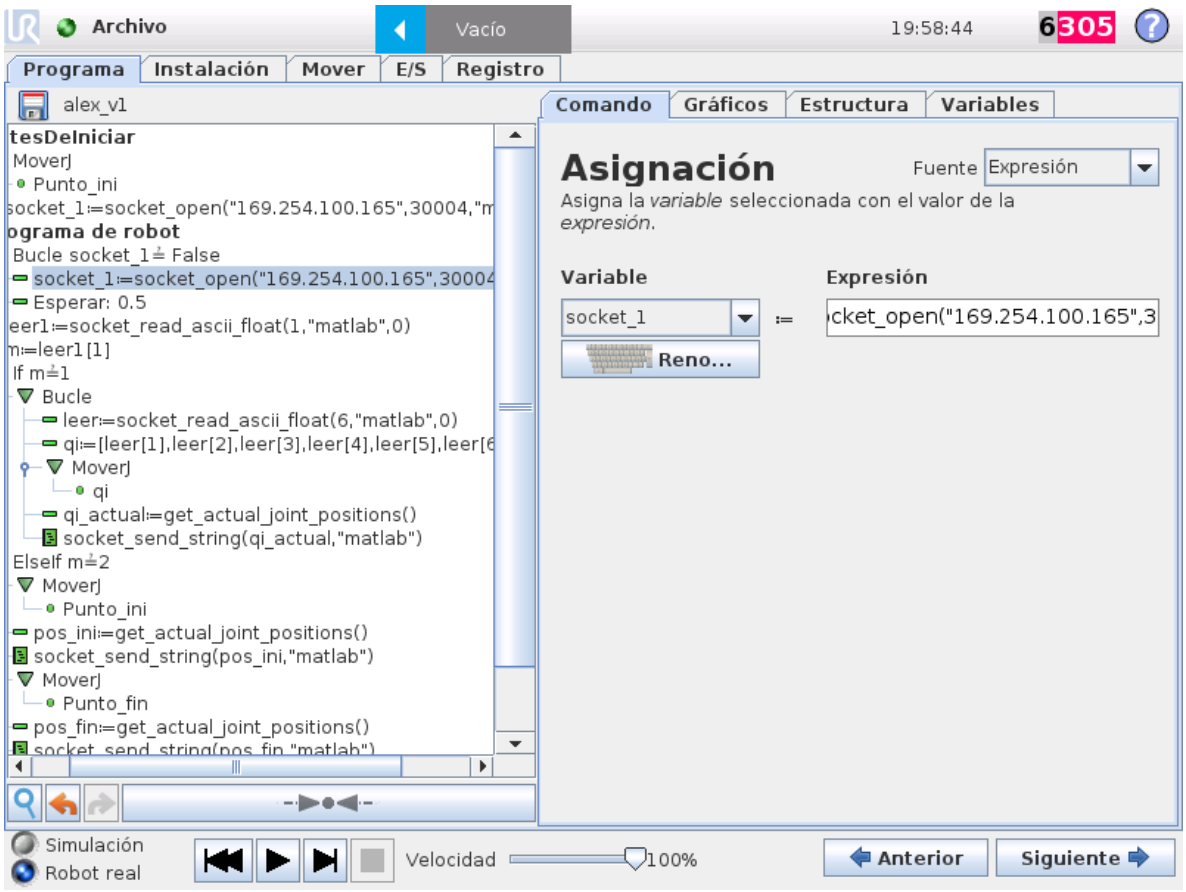

Figura 26 Programa del robot UR3.

En el programa principal se verifica si el servidor está listo para conectarse. De lo contrario, el bucle seguirá repitiéndose hasta que haya un servidor con un puerto activo. Cuando el robot encuentra un servidor con un puerto activo, espera recibir 1 dato, el cual le indica en qué parte del programa entrar.

Si el robot recibe un “1” entra a un bucle de seguimiento, en el que primero espera recibir una lista de datos que contiene las 6 posiciones de los ángulos, cuando el robot recibe la lista, esta contiene 7 datos, donde el primer elemento representa la cantidad de datos recibidos, por ejemplo [6,q1,q2,q3,q4,q5,q6]. Una vez ha leído los datos de entrada, separa sólo los ángulos en la variable qi para luego por medio de un movimiento de juntas (MoveJ) ir hasta esa posición. Apenas el robo llega a la posición que se le indicó en la variable qi_actual, por medio de la cinemática inversa del robot, se obtienen los ángulos actuales que luego se envían al PC, para ser procesados en Matlab, donde se comparan con la posición que se envió y se comprueba que el error no sea superior al 10%. El robot sigue este loop para cada una de las posiciones que hayan sido creadas en la planeación por medio del RRT.

Si el robot recibe un “2” el robot va hasta la posición inicial, que se ha configurado previamente y estando en esta posición calcula los ángulos que corresponden a esa posición con la orientación dada y se envían a Matlab, donde se guardará como Punto_ini y será el punto de partida de nuestra trayectoria. Luego se desplazará a la posición que se haya configurado como Punto_fin, como se observa en la figura 27. Esta posición se enviará a Matlab donde pasará a ser pos_fin, la cual representará la posición objetivo de la trayectoria.

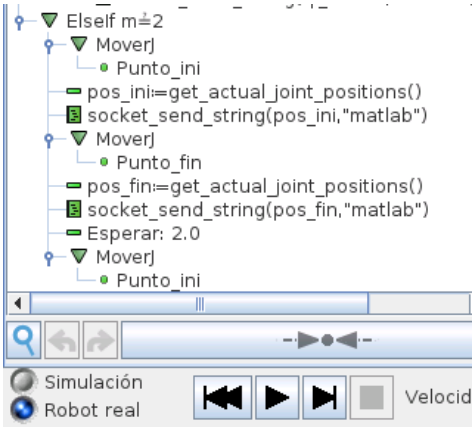

Figura 27 Set de posición inicial y final.

5. VALIDACIONES

Para realizar las validaciones experimentales se unieron los algoritmos de visión artificial y planeación de trayectorias. Se hicieron 3 pruebas donde se validan 3 escenarios distintos. A continuación, se revisa un escenario donde no hay obstáculos y se le pide ir al robot de un punto inicial a uno final. En la figura 28 está el robot en la posición inicial, sin obstáculos.

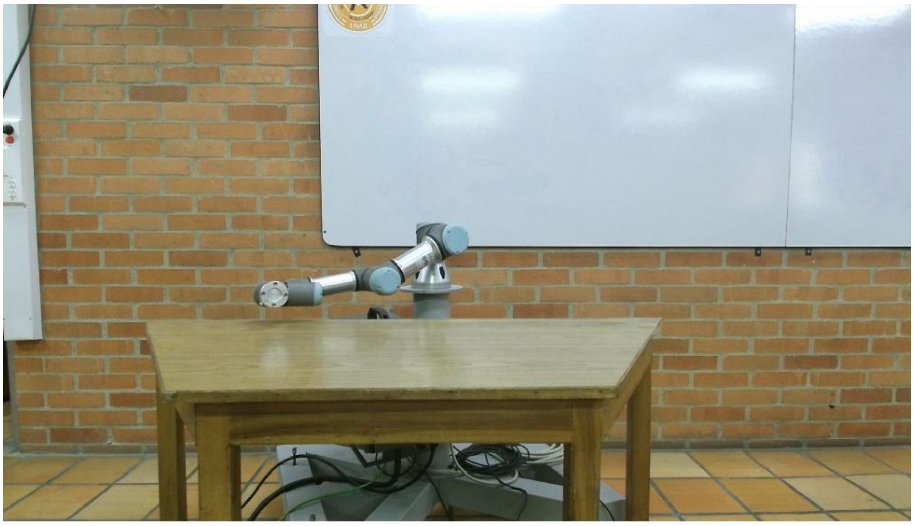

Figura 28 Escena inicial. Validación 1.

En la figura 29 se puede evidenciar el funcionamiento del algoritmo de visión artificial, el cual al encontrar en la escena solo al robot, segmenta la base del robot y el resto de la imagen la segmenta como fondo de imagen porque no hay mas objetos conocidos para el algoritmo en la escena.

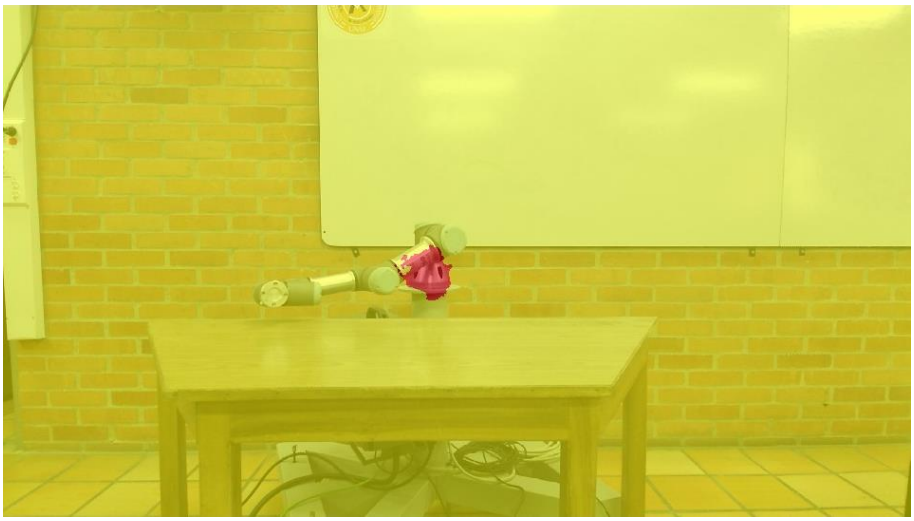

Figura 29 Escena segmentada. Validación 1.

En esta prueba, que no se tiene obstáculos se genera una trayectoria como la que se puede observar en la figura 30, donde la mayor parte del movimiento se genera en la primera articulación, ya que no debe evitar obstáculos.

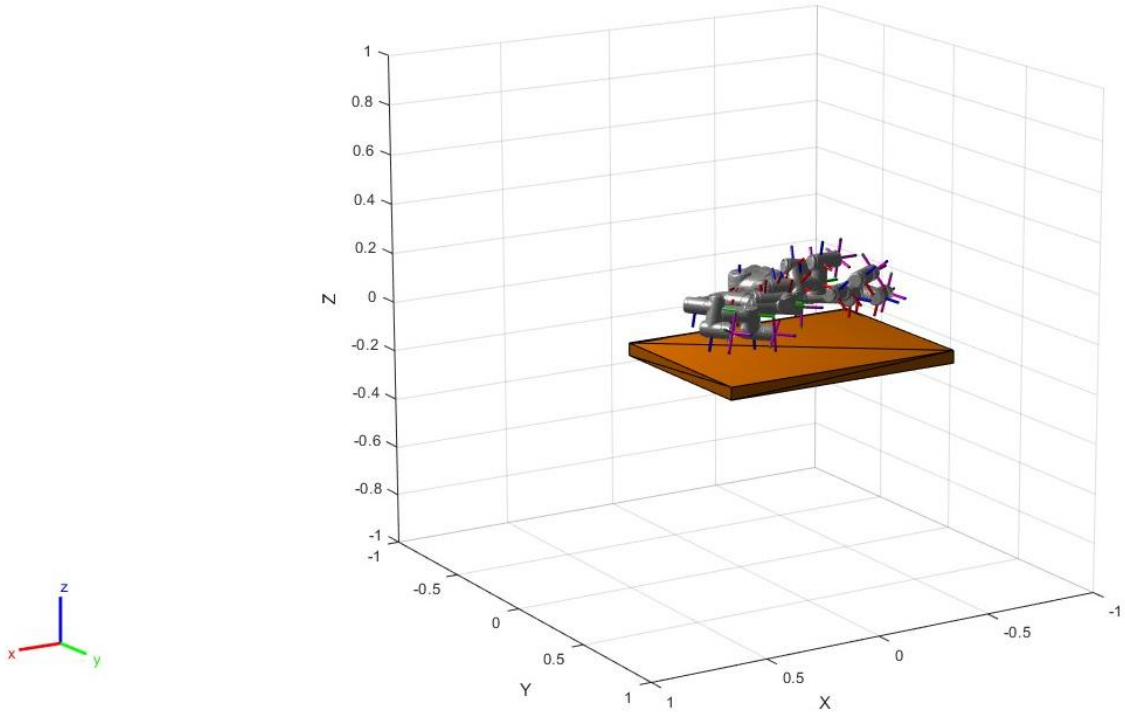

Figura 30 Trayectoria generada. Validación 1.

Una segunda validación se realizo con la escena presentada en la figura 31, donde hay 2 obstáculos, que se incluyeron en el algoritmo de visión artificial. En este caso se busca que el robot pueda evitar los obstáculos. El algoritmo logra segmentar la escena como se presentan en la figura 32, donde la segmentación es buena debido la mayoría de los pixeles segmentados son correctos.

Figura 31 Escena inicial. Validación 2.

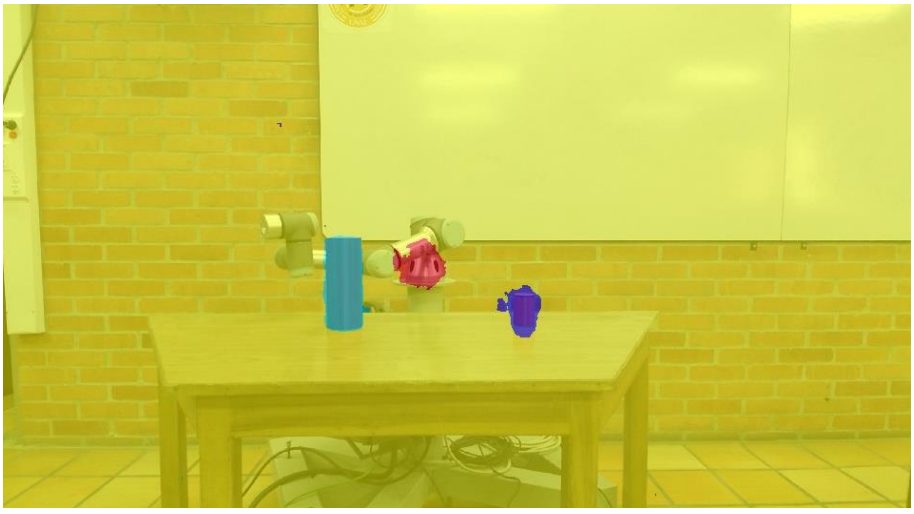

Figura 32 Escena segmentada. Validación 2.

En esta prueba, a diferencia de la anterior al tener obstáculos el algoritmo debe calcular múltiples cambios en todas las articulaciones para lograr superar los obstáculos. La trayectoria generada se muestra en la figura 33, donde se genera un movimiento del robot sin colisionar con los obstáculos.

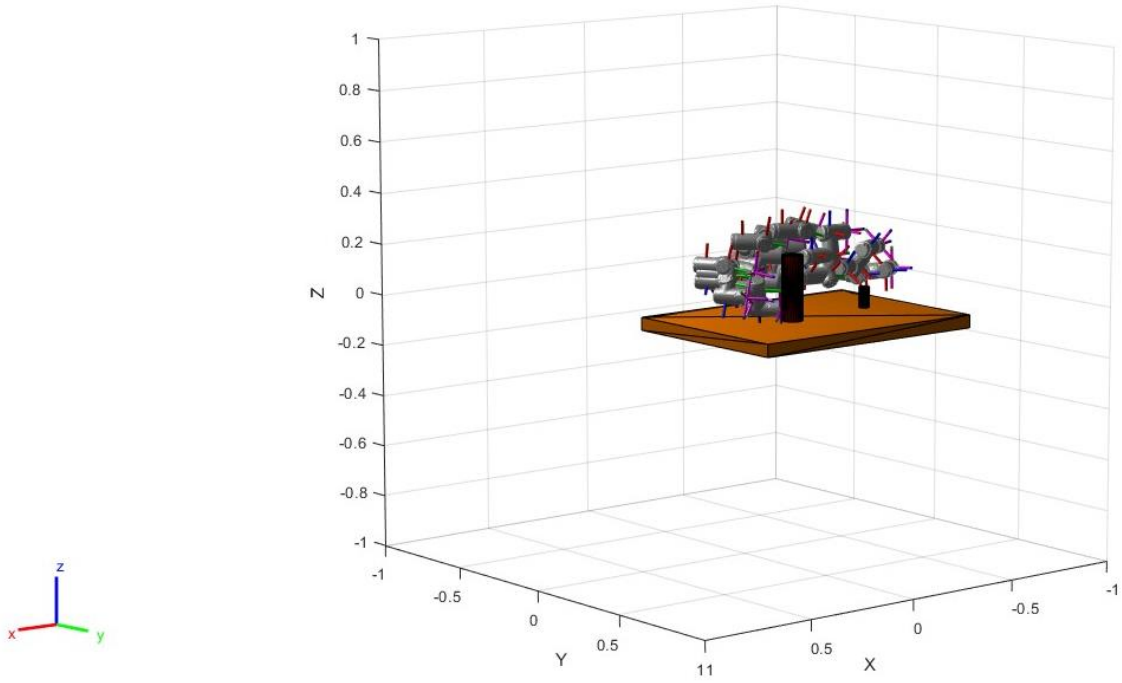

Figura 33 Trayectoria generada. Validación 2.

Por último, se hizo una validación incluyendo un objeto adicional, el cual no ha sido incluido en el algoritmo de visión artificial y tiene como objetivo revisar el comportamiento del algoritmo en este caso. En la figura 34 se puede visualizar la escena.

Figura 34 Escena inicial. Validación 3.

La escena segmentada se presenta en la figura 35. En este caso ocurren dos hechos relevantes. Primero al tener un objeto externo el algoritmo de visión artificial no lo segmenta como un obstáculo, esto lleva a un fallo del algoritmo al intentar llevarlo a un ambiente no controlado y segundo la base del robot, que si es un parte con la cual se entreno el algoritmo de visión artificial, esta parcialmente oculta debido a que tanto el obstáculo rojo como el objeto externo están entre la base del robot y la cámara, por tanto el algoritmo no lo percibe como la base y omite la base al segmentar la escena, por lo que no detecta la base del robot.

Figura 35 Escena segmentada. Validación 3.

De manera predeterminada se le indico al algoritmo que al no encontrar la base la situara en la posición XYZ [0 0 1.6]m con respecto al centro del Kinect v2, cuando en realmente la base está en la posición XYZ [0 0.1 1,53]m con respecto al centro del Kinect v2. La trayectoria se genera intentando evitar los obstáculos que el algoritmo segmento, el obstáculo externo no lo tiene en cuenta, por tanto, colisiona con este objeto. También al no ubicarse bien los obstáculos con respecto al centro del robot, porque no se puedo segmentar la base del robot, el primer obstáculo, el objeto de color naranja termina colisionando con la articulación 4 del robot ur3.

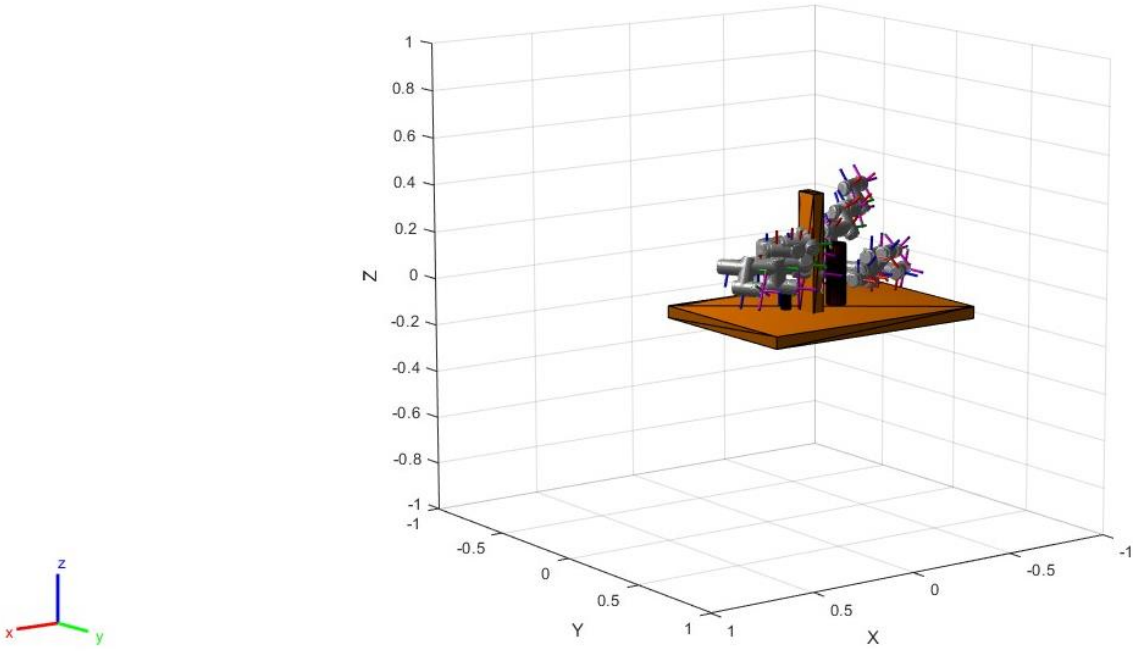

Figura 36 Trayectoria generada. Validación 3.

6. CONCLUSIONES

- La cámara Kinect V2 permite recrear escenarios 3D de un ambiente real, debido a que además de contar con los sensores RGB que nos ayudan a identificar objetos ya sea por color o forma, dispone un sensor de profundidad, los cuales trabajando en conjunto con un algoritmo de visión artificial permiten tener datos con una confiabilidad de 98% de las dimensiones y distancias de los objetos dentro de la escena.
- El algoritmo de segmentación semántica comparado con otros algoritmos, como el de búsqueda secuencial, permite clasificar los objetos con mayor precisión, ya que haciendo uso de las capacidades de las redes neuronales puede identificar un objeto a nivel de píxel, teniendo un dato con mayor precisión en cuanto al tamaño de este, además es más robusto ante situaciones de alto contraste, mala iluminación o ruido en la imagen.
- La cinemática inversa del robot puede tener múltiples soluciones para un punto determinado, dando como resultado cambios grandes en los ángulos de las articulaciones, generando movimientos que causan que el robot entre en falla, para evitar estos cambios abruptos en los ángulos, es necesario evaluar todos los ángulos de la trayectoria, después de generada, en búsqueda de los mismos y poder corregirlos.
- El algoritmo bidireccional RRT permite reducir el tiempo de procesamiento de la generación de la trayectoria del robot, la cuál es compleja de calcular debido a la cantidad de grados de libertad del mismo, haciendo uso de su capacidad de generar dos árboles de exploración, uno desde la posición inicial y otro desde la posición final hasta que se encuentran.
- El robot realiza un seguimiento de la trayectoria que se envía con un porcentaje de error menor al 1%, pero realizar esta comprobación causa un retardo en el envío de datos que genera un movimiento discontinuo en el robot, ya que se tiene que detener en cada una de las posiciones para calcular su desviación entre la posición enviada y la ejecutada. Como se obtenía un porcentaje de error tan pequeño se optó por no realizar este cálculo, dejando que el robot siguiera la trayectoria sin la necesidad de detenerse a enviar la retroalimentación de la posición especificada, esto ayuda a que el movimiento sea más fluido y el tiempo de ejecución de la trayectoria se reduzca.
- Las validaciones realizadas permiten conocer las fortalezas y debilidades del sistema de evasión de obstáculos, el cual en un ambiente controlado funciona bien, pero ante perturbaciones u objetos externos falla por no tomarlos como obstáculos sino como objetos del entorno los cuales omite al aplicar el algoritmo planeación de trayectorias.

7. BIBLIOGRAFÍA

[1]. E. B. Kumar and V. Thiagarasu, "Color channel extraction in RGB images for segmentation," 2017 2nd International Conference on Communication and Electronics Systems (ICCES), 2017, pp. 234-239, doi: 10.1109/CESYS.2017.8321272.

[2]. M. Minos-Stensrud, O. H. Haakstad, O. Sakseid, B. Westby and A. Alcocer, "Towards Automated 3D reconstruction in SME factories and Digital Twin Model generation," 2018 18th International Conference on Control, Automation and Systems (ICCAS), 2018, pp. 1777-1781.

[3]. Z. Shan, X. Xu, Y. Tao and H. Xiong, "A Trajectory Planning and Simulation Method for Welding Robot," 2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), 2017, pp. 510-515, doi: 10.1109/CYBER.2017.8446181.

[4]. E. Shelhamer, J. Long and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 4, pp. 640-651, 1 April 2017, doi: 10.1109/TPAMI.2016.2572683.

[5]. K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[6]. C. Lin and M. Li, "Motion planning with obstacle avoidance of an UR3 robot using charge system search," 2018 18th International Conference on Control, Automation and Systems (ICCAS), 2018, pp. 746-750.

[7]. A. Y. Lee, G. Jang and Y. Choi, "Infinitely differentiable and continuous trajectory planning for mobile robot control," 2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), 2013, pp. 357-361, doi: 10.1109/URAI.2013.6677386.

[8]. L. S. Scimmi, M. Melchiorre, S. Mauro and S. P. Pastorelli, "Implementing a Vision-Based Collision Avoidance Algorithm on a UR3 Robot," 2019 23rd International Conference on Mechatronics Technology (ICMT), 2019, pp. 1-6, doi: 10.1109/ICMECT.2019.8932105.

[9]. L. S. Scimmi, M. Melchiorre, S. Mauro and S. Pastorelli, "Experimental Real-Time Setup for Vision Driven Hand-Over with a Collaborative Robot," 2019 International Conference on Control, Automation and Diagnosis (ICCAD), 2019, pp. 1-5, doi: 10.1109/ICCAD46983.2019.9037961.

[10]. Intel RealSense D400 Series Product Family [En linea]. Avalaible: https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf [Último accedo: 2022].

[11]. Kinect for Windows SDK 2.0. http://www.todokinect.com/

[12]. L. Egorova and A. Lavrov, "Determination of workspace for motion capture using Kinect," 2015 56th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON), 2015, pp. 1-4, doi: 10.1109/RTUCON.2015.7343155.

[13]. Ibañez, R., Soria, Á., Teyseyre, A., & Campo, M. (2014). Easy gesture recognition for Kinect. Advances in Engineering Software, 76, 171–180. doi:10.1016/j.advengsoft.2014
[14]. M. Shoryabi, A. Foroutannia and A. Rowhanimanesh, "A 3D Deep Learning Approach for Classification of Gait Abnormalities Using Microsoft Kinect V2 Sensor," 2021 26th International Computer Conference, Computer Society of Iran (CSICC), 2021, pp. 1-4, doi: 10.1109/CSICC52343.2021.9420611.
[15]. Dive into depp learning, Residual Networks (ResNet). https://classic.d2l.ai/chapter_convolutional-modern/resnet.html#residual-networks-resnet.
[16]. MathWorks, Segmentación semántica. Mathworks. https://la.mathworks.com/solutions/image-video-processing/semantic-segmentation.html.
[17]. "Universal Robot UR3". Universal Robots.
[18]. Romero C. Juan, Paez R. David, Guarnizo M. José (2021). “UR3 Modelo Cinemático Inverso”
[19]. M. Ortiz-Salazar, A. Rodríguez-Liñán, L. M. Torres-Treviño and I. López-Juárez, "IMU-Based Trajectory Generation and Modelling of 6-DOF Robot Manipulators," 2015 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE), 2015, pp. 181-186, doi: 10.1109/ICMEAE.2015.27.
[20]. J. -D. Sun, G. -Z. Cao, W. -B. Li, Y. -X. Liang and S. -D. Huang, "Analytical inverse kinematic solution using the D-H method for a 6-DOF robot," 2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), 2017, pp. 714-716, doi: 10.1109/URAI.2017.7992807.
[21]. Y. Ren, H. Sun, Y. Tang and S. Wang, "Vision Based Object Grasping of Robotic Manipulator," 2018 24th International Conference on Automation and Computing (ICAC), 2018, pp. 1-5, doi: 10.23919/IConAC.2018.8749001.
[22]. L. D. Hanh and C. -Y. Lin, "Combining stereo vision and fuzzy image based visual servoing for autonomous object grasping using a 6-DOF manipulator," 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2012, pp. 1703-1708, doi: 10.1109/ROBIO.2012.6491213.
[23]. MathWorks, bidirectional rapidly exploring random trees. Mathworks. https://la.mathworks.com/help/robotics/ref/manipulator_rrt_true.png
[24]. T. LI and H. ZHU, "Research on model control of binocular robot vision system," 2018 Chinese Automation Congress (CAC), 2018, pp. 1794-1797, doi: 10.1109/CAC.2018.8623756.
[25]. Iglesias García, M., & Lorenzo Prada, A.Sistema de Visión Artificial (Ingeniería).Universidad Carlos III.
[26]. MathWorks, «Computer Vision Toolbox,» [En línea]. Available: https://la.mathworks.com/products/computer-vision.html. [Último acceso: 2022].

8. ANEXOS

8.1. Anexo 1

- Visión artificial

Para extraer la ubicación el espacio XYZ de los obstaculos y la base del robot se utilizó la siguiente línea de código:

```
[centroideObj1 centroideObj2 centroidebase] = Centroides(image,C)
```

Y la función Centroides es la siguiente:

```
function [centroideObj1 centroideObj2 centroidebase] = Centroides(image,C);
[row1, col1] = find(C == 'obj1');% Obj naranja
[row2, col2] = find(C == 'obj2');% Obj rojo
[rowB, colB] = find(C == 'base');% Base de UR3
[r1s r1ss] = size(row1);
[c1s c1ss] = size(col1);
[r2s r2ss] = size(row2);
[c2s c2ss] = size(col2);
[r3s r3ss] = size(rowB);
[c3s c3ss] = size(colB);
    if(r1s>0);
        row1 = row1(end*0.3:end-end*0.3,1);
        col1 = col1(end*0.3:end-end*0.3,1);
        % Fila y columna del pixel promedio (RGB)
        C1 = int16(mean([row1 col1]));
        % Valor X,Y,Z del pixel seleccionado
        x1 = image.Location(C1(1,1),C1(1,2),1);
        y1 = image.Location(C1(1,1),C1(1,2),2);
        z1 = image.Location(C1(1,1),C1(1,2),3);
        centroideObj1 = [x1 y1 z1];
    else
        centroideObj1 = [0 0 0];
    end

    if(r2s>0);
        row2 = row2(end*0.3:end-end*0.3,1);
        col2 = col2(end*0.3:end-end*0.3,1);
        % Fila y columna del pixel promedio (RGB)
        C2 = int16(mean([row2 col2]));
        % Valor X,Y,Z del pixel seleccionado
        x2 = image.Location(C2(1,1),C2(1,2),1);
        y2 = image.Location(C2(1,1),C2(1,2),2);
        z2 = image.Location(C2(1,1),C2(1,2),3);
        centroideObj2 = [x2 y2 z2];
    else
        centroideObj2 = [0 0 0];
    end
```

```
    if(r3s>0);
        rowB = rowB(end*0.3:end-end*0.3,1);
        colB = colB(end*0.3:end-end*0.3,1);
        % Fila y columna del pixel promedio (RGB)
        CB = int16(mean([rowB colB]));
        % Valor X,Y,Z del pixel seleccionado
        xB = image.Location(CB(1,1),CB(1,2),1);
        yB = image.Location(CB(1,1),CB(1,2),2);
        zB = image.Location(CB(1,1),CB(1,2),3);
        centroidebase = [xB yB zB];
    else
        centroidebase = [0 0 1.60];
    end
end
```

Para la construcción del ambiente virtual se utilizó las siguientes líneas de código:

```
figure(2)
show(robot,configSoln,"Collisions","on")
ax = gca;
hold on
plane = collisionBox(1,0.75,0.05);
pared = collisionBox(1, 0.05, 1);
Cilindro1r = collisionCylinder(radio1, altura1);
Cilindro2r = collisionCylinder(radio2, altura2);
env = {plane,pared,Cilindro1r,Cilindro2r};
env{1}.Pose = trvec2tform([0 0.375 -0.035]);
show(env{1},'Parent', ax);
env{2}.Pose = trvec2tform([0 -0.2 0.1]);
[~, patchObj] = show(env{2},'Parent',ax);
patchObj.FaceColor = [1 1 1];
env{3}.Pose = trvec2tform([x1 y1 z1]);
[~, patchObj] = show(env{3},'Parent',ax);
patchObj.FaceColor = [0 0 1];
env{4}.Pose = trvec2tform([x2 y2 z2]);
[~, patchObj] = show(env{4},'Parent',ax);
patchObj.FaceColor = [0 0 1];
```

Posterior a que se genera la trayectoria, se puede visualizar con las siguientes líneas de código:

```
clf
for i = 1:10:size(path,1)
    show(robot,path(i,:));
    ax = gca;hold on
end
show(env{1},'Parent', ax);
[~, patchObj] = show(env{3},'Parent',ax);
patchObj.FaceColor = [1 0 0];
```

```
[~, patchObj] = show(env{4},'Parent',ax);
patchObj.FaceColor = [1 0 0];
```

8.2. Anexo 2

- Envió de trayectoria

Posterior a la generación de la ruta a seguir por el robot y la conexión entre el controlador del robot y el entorno de Matlab, el código para enviarle los datos generados al controlador desde Matlab es el siguiente:

```
enviar = "[1]";
py.connect_to_ur3.enviarPosicion(c,enviar)
posesrad = path;
posini = posesrad(1,:);
posfin = posesrad(end,:);
sizeposes = size(posesrad);
filas = sizeposes(1);
columnas = sizeposes(2);
posam_rad = [];
i = 1;
j = 1;
tic
if (columnas == 6)
    if (i == 1)
    posn = posesrad(i,1:end);
    posenviada(j,:) = posn;
    pos =
"["+posn(1,1)+","+posn(1,2)+","+posn(1,3)+","+posn(1,4)+","+posn(1,5)+","+posn(1
,6)+ "]"
    py.connect_to_ur3.enviarPosicion(c,pos)
    pos_actual = py.connect_to_ur3.leerPosicion(c);
    pos_actual = native2unicode(pos_actual);
    posa = str2num(pos_actual);
    posam_rad(j,:) = posa(1,:);
    i = i+10;
    j = j+1;
    end
    while (i<filas+1)
        if ((posn-0.9<posa)&(posa<posn+0.9))
        posn = posesrad(i,1:end);
        posenviada(j,:) = posn;
        pos =
"["+posn(1,1)+","+posn(1,2)+","+posn(1,3)+","+posn(1,4)+","+posn(1,5)+","+posn(1
,6)+ "]"
        py.connect_to_ur3.enviarPosicion(c,pos)
        pos_actual = py.connect_to_ur3.leerPosicion(c);
        pos_actual = native2unicode(pos_actual);
        posa = str2num(pos_actual);
        posam_rad(j,:) = posa(1,:);
        i = i+10;
```

```
        j = j+1;
         %pause(2)
        else
            msg = 'posn y posa son diferentes'
            error(msg)
        end
    end
else
    msg = 'Número de columnas incorrecto'
    error(msg);
end
time_ejecu = toc
[por_error,ero] = errorm(posenviada,posam_rad,"q")
```