Pontificia Universidad Javeriana

Facultad de Ingeniería
Departamento de Electrónica
Pregrado en Ingeniería Electrónica

Extracción automatizada de tapas de botellas plásticas

Steven Forero Giraldo
David Armando Rodríguez Sarmiento

Bajo la dirección de:
Diego Alejandro Patiño Guevara Ph.D
Carol Viviana Martínez Luna Ph.D

Bogotá DC, Colombia
Noviembre 2019

Resumen

El reciclaje de botellas de plástico es uno de los temas fundamentales en la actualidad para permitir un crecimiento sostenible de la sociedad junto con el medio ambiente. Este proceso debe seguir varios pasos para hacerlo de manera correcta, donde uno de ellos es extraer su tapa. Por esta razón, en este trabajo de grado titulado "Extracción automatizada de tapas de botellas de plástico" se desarrolla la automatización de este proceso a través de un robot Motoman Yaskawa SDA10F y un sensor Kinect V1. La detección de las botellas se realiza mediante un proceso de segmentación de color, por el cual el Kinect identifica las tapas de las botellas, con esta información el sistema calcula la posición de la tapa y la posición de la botella en el espacio. Posteriormente, las posiciones se envían a un entorno de simulación en el que el robot realiza la secuencia de extracción de la tapa de acuerdo con la información suministrada.

La solución se desarrolla a través de ROS (*Robot Operating System*) que está soportado en el software *Ubuntu 16.04*. Por un lado, el diseño y la simulación del algoritmo de control de alto nivel para el robot se realizan con las plataformas MoveIt! y RVIZ. Por otro lado, para el módulo de visión se utiliza la biblioteca OpenCV, que proporciona varias herramientas para el desarrollo de proyectos de visión por computadora. Las botellas de prueba que se utilizan en el proyecto son botellas plásticas de bebidas gaseosas o jugos de 300 mililitros a 3 litros con tapas de color rojo, azul o amarillo.

Agradecimientos

Agradezco a mi familia y amigos, especialmente a mi mamá y mi hermana, quienes siempre han estado a mi lado, apoyándome y ayudándome en todo lo que pueden. Esta experiencia se la debo totalmente a mi madre ya que fue el motor para salir adelante y por todo su esfuerzo y dedicación le agradezco inmensamente.

Steven Forero Giraldo

Al culminar esta gran etapa de mi vida quiero agradecer a mi familia, en especial a mi hermana y a mi abuela por brindarme todo su amor y entera confianza y a mi padre por ser el principal apoyo en este arduo proceso que pronto llegará a su fin. Agradezco a mis amigos, porque siempre estuvieron para mi en los buenos y malos momentos, porque sin ellos esta experiencia no hubiese sido igual. Por ultimo agradezco a mis compañeros de carrera, integrantes del laboratorio de electrónica y mis directores por brindarme su apoyo a lo largo de mi carrera y de este proceso.

Dedicado a la memoria de María Margarita Sarmiento Melo.

David Armando Rodríguez Sarmiento

Índice general

Índice de tablas

Índice de figuras

1. Introducción

Mundialmente se ha evidenciado la necesidad de reciclar correctamente los desechos que tienen la posibilidad de ser reutilizados, por ejemplo, las botellas plásticas. Premisas como "El mundo compra un millón de botellas de plástico por minuto que acaban en vertederos o en el mar"**[1]** o "Hay tantos residuos de plástico en el mundo que podrían cubrir un país como Argentina" **[2]** hacen que sea de vital importancia buscar la forma para reciclar estos elementos de una manera eficiente y que permita reducir en un gran porcentaje la contaminación del medio ambiente. Hay que enfatizar en que esta problemática no es pasajera, estudios científicos aseguraron que para el año 2017 cada minuto se compraban 1 millón de botellas plásticas en todo el mundo y que esta cifra podría aumentar en un 20 % para el año 2021 **[1]**, evidenciando así que esta es una problemática fundamental para el futuro del planeta y sus ecosistemas.

Aunque claramente este problema podría ser abordado desde la cultura del reciclaje empezando desde el hogar, es difícil que la sociedad adquiera conciencia sobre el tema y las consecuencias que este trae, una prueba de ello es que: "El hogar promedio del Reino Unido arroja aproximadamente 500 botellas de plástico por año y un gran porcentaje de ellas terminan en vertederos" **y** esto puede ocurrir con una proporción igual o mayor en cualquier otro país del mundo. Así que partiendo del hecho de que las botellas serán desechadas en cierto momento, lo mejor es buscar una forma de reutilizar su material, para así crear materia prima para futuros productos. Debido a lo anterior, es importante conocer el proceso correcto del reciclaje de botellas plásticas, el cual consta de 6 fases: Recolección, clasificación del tipo de plástico, extracción de las tapas y los adherentes plásticos que pone cada marca en sus botellas; donde estos últimos, poseen uno de los materiales más contaminantes; lavado y triturado, procesado del plástico; con el fin de generar una especie de "cinta" que se enrolla en tubos, para finalmente, en la sexta fase, realizar la fabricación de nuevos productos plásticos, como son botellas nuevas, cubiertos plásticos, entre otros **[4]**.

En la actualidad, en un contexto local este proceso de reciclaje de las botellas se realiza de manera manual, un sector de la población se hace cargo de la recolección de las botellas y otros residuos que posteriormente son llevados a centros de reciclaje. En ese lugar varios operarios son los encargados de realizar la clasificación, adecuación (extracción de tapas y etiquetas) y lavado de las botellas, para así de esta forma lograr procesarlas. Este proceso además de ser repetitivo y tedioso para los operarios, los cuales a futuro pueden llegar a tener problemas de salud, presenta altos costos por los requisitos legales y salariales que maneja la contratación de un operario. EL objetivo de este trabajo de grado es automatizar de manera parcial unas de las fases del reciclado de las botellas, la cual es la extracción de sus tapas. Entre las ventajas que presenta la automatización de este proceso, es la reducción de costos al momento de desarrollarlo, dado que los costos de un trabajo manual contra los costos de un proceso automatizado son mayores. Además al realizar un trabajo colaborativo entre humano y maquina se pueden obtener resultados aún mejores, todo lo anterior se puede ver reflejado en la figura 1.1 en donde se compara la relación costos y unidades por año para un trabajo manual, un trabajo automatizado y un trabajo híbrido.

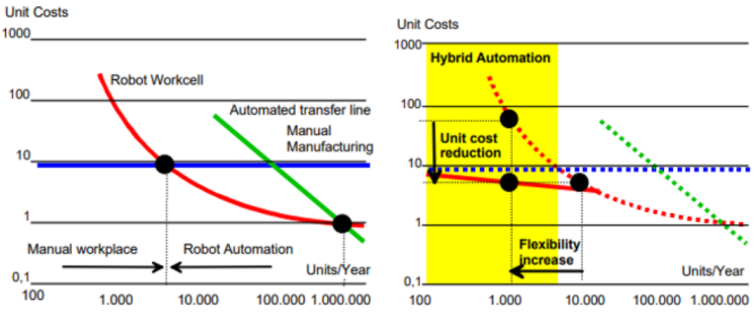


Figura 1.1: Reducción de los costos de un sistema robótico comparado con el trabajo manual y observación de una potencial reducción de costos gracias a una automatización híbrida

La solución propuesta consta de 3 bloques (ver figura 1.2) los cuales son **" identificación del objeto y accesorio "**, **" secuencia de control"** y **"robot actuador"**. El bloque *identificación del objeto y accesorio* es el encargado de entregar la posición en el mundo real de las botellas y de sus tapas. Para la detección del objeto en cuestión, se utiliza un sensor *Kinect* con el cual se obtiene una imagen de color y otra de profundidad; este bloque entrega al bloque secuencia de control tres variables: Posición de la botella (amarillo), posición de la tapa (azul) y confirmación (rojo). El bloque de *secuencia de control* es el encargado de la programación de las extremidades de robot para cumplir con el objetivo de destapar la botella; este bloque recibe las posiciones tanto de la botella como de la tapa, las cuales utiliza para planificar las trayectorias del robot. Dicho proceso se realiza internamente en el software de simulación. Por último, el bloque **robot actuador** es el encargado de realizar el destapado de la botella; para ello se cuenta con el paquete de simulación de un robot Motoman Yaskawa SDA10F, el cual cuenta con 15 grados de libertad. El paquete, además de tener cada detalle del robot original, cuenta con un entorno de simulación con el cual se realizan las pruebas. Este paquete fue entregado por el Centro de Tecnológico de Automatización Industrial (CTAI) de la Pontificia Universidad Javeriana.

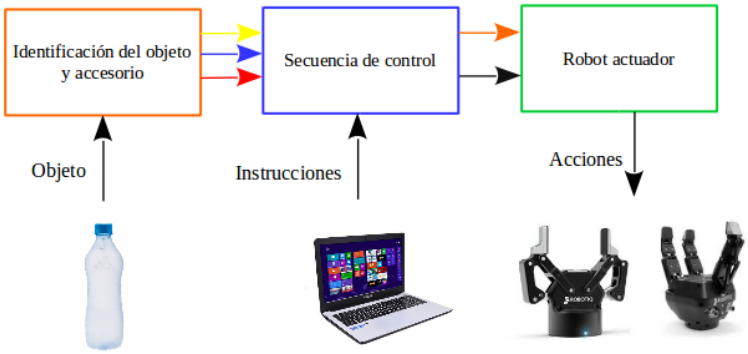


Figura 1.2: Diagrama de bloques del sistema

Entre las especificaciones del proyecto se tienen: El kinect estará ubicado a 1.20m aproximadamente sobre el área de trabajo y su ubicación será completamente perpendicular a ésta. Las botellas que se utilizarán para el desarrollo del proyecto son botellas plásticas de bebidas entre los 300ml y los 3L. El sistema de detección estará diseñado para detectar tapas de color rojo, amarillo y azul, estos colores se determinaron gracias a que la gran mayoría de productos embotellados que cumplen con el volumen determinado son sellados con tapas de dichos colores.

2. Objetivos

2.1 Objetivo General

- Diseñar un control de alto nivel para realizar trabajo cooperativo entre las extremidades de un robot Motoman Yaskawa SDA10F

2.2 Objetivos Específicos

- Identificar la posición de una botella o varias botellas plásticas por medio de un sistema Kinect con una precisión mayor al 90 %.
- Diseñar el algoritmo para el control de alto nivel de los dos brazos.
- Diseñar el algoritmo de alto nivel para remover la tapa de la botella en un tiempo menor a 2 minutos.
- Implementar los algoritmos de control para el control cooperativo de los dos brazos y la tarea de extracción de la tapa de la botella en la plataforma ROS e integrarlo en simulación.
- Diseñar un protocolo de pruebas que permita validar los resultados en simulación.

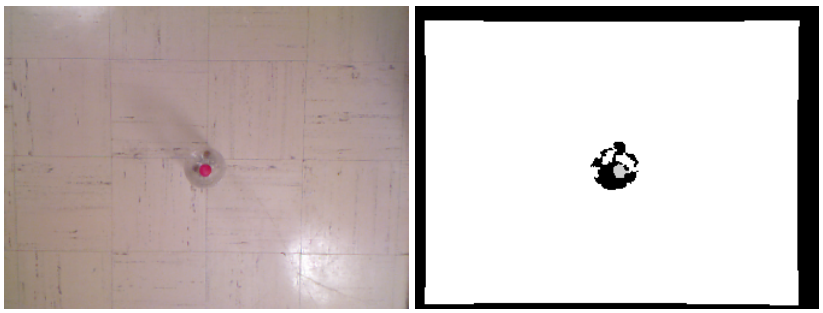
3. Visión

Este capítulo presenta todo el desarrollo teórico y práctico del algoritmo para la detección de las tapas de las botellas y la estrategia para obtener la posición de la botella. De igual forma, se presenta el protocolo de pruebas y sus resultados correspondientes.

Para empezar, el algoritmo para detectar la tapa de la botella está dividido en cinco fases: *Captura de imagen*, *Segmentación de color*, *Contornos*, *Momentos* y *Posición de la tapa de la botella*. Posteriormente, con base en los resultados obtenidos de las 5 fases, se emplea el modelo Pin Hole, con el cual se realiza la conversión de una posición en píxeles a una posición en metros. Para realizar las diferentes tareas del algoritmo se emplea OpenCV, que corresponde a una biblioteca libre que provee una gran cantidad de funciones para tareas en el ámbito de visión por computadora.

3.1 Captura de imagen

La función de esta fase se centra en la obtención correcta de la imagen del mundo real y su posterior adaptación para poder utilizar a libertad la información que esta ofrece. Para esto se emplea un sensor Kinect V1 desarrollado por la compañía Microsoft, el cual entrega dos tipos de imagen; una imagen RGB en un formato VGA (640*480) de 8 bits y una imagen de profundidad de la misma resolución pero de 16 bits. Para lograr utilizar el *Kinect* en conjunto con ROS, se emplea el driver openNI, el cual permite la comunicación con los sensores de audio, vídeo y profundidad del *kinect*, mientras que proporciona una API que sirve de vínculo entre el hardware del dispositivo, NITE Middleware y las aplicaciones e interfaces del S.0 [5].



(a) Imagen de RGB (b) Imagen de profundidad

Figura 3.1: Imagenes entregadas por el kinect

El controlador openNI ofrece una gran variedad de tópicos con los cuales se puede obtener diversos tipos de imagen a partir del kinect. Para hacer uso de estos tópicos, se debe realizar un proceso de subscripción por parte del usuario, con el fin de obtener información del kinect solo cuando el usuario así lo desee. Para este caso, como se quiere obtener una imagen RGB y una imagen de profundidad, openNi ofrece los tópicos *camera/color/image* y *camera/depth_registered/image* para obtener datos de color y profundidad respectivamente. Cuando se realiza la obtención de la imagen, tanto de color como de profundidad, se debe realizar la conversión de una imagen de ROS a una imagen que se pueda utilizar con OpenCV, dado que ROS envía las imágenes en su propio formato *sensor_msgs*, el cual corresponde a un mensaje de imagen, se debe utilizar una biblioteca de ROS denominada CvBridge, que proporciona una interfaz entre ROS y OpenCv

[6]. (ver figura 3.2).

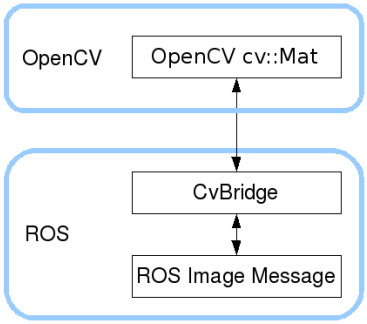


Figura 3.2: Interfaz entre ROS y OpenCv empleando CvBridge

de las funciones es similar, se tiene como parámetro de entrada el mensaje proveniente de la subscripción al tópico de openNI y se crea una variable "*cv_ptr*" de tipo "*cv_bridge* :: *CvImageConstPtr*" con la cual se guarda la salida de la función "***cv_bridge::toCvcopy***()", que tiene como finalidad realizar la conversión del mensaje recibido a un formato que se pueda usar con OpenCV.

3.2 Segmentación de color

La función de esta fase es la creación de una máscara con el objetivo de diferenciar colores en específico dentro de una imagen. Como punto de partida se tiene la información entregada por la función *colorcallback* para la captura de imagen RGB. Para realizar la segmentación por color de la imagen se realiza primero una conversión del formato RGB de la imagen a HSV; de sus siglas en ingles Hue, Saturation, Value; con la finalidad de tener una detección más precisa, puesto que al realizar este proceso con el primer modelo se pueden presentar dificultades con la iluminación y/o sombras que pueda tener la imagen. El modelo de color HSV se define en términos de sus componentes y se trata de una transformación no lineal del espació de color RGB. Este modelo se puede representar por medio de un cono, en el cual el matiz corresponde a la región circular de este, mientras que la saturación y valor de los colores se representan a través de los catetos de un triángulo que se puede obtener realizando un corte al cono. El eje horizontal de dicho triángulo denota la saturación mientras que el eje vertical corresponde al valor. Los rangos para los valores del modelo varían de 0°-360°, 0°-100° y 0°-100° para el matiz (H), la saturación (S) y el valor (V) respectivamente [7].

Figura 3.3: Modelo de color HSV [3]

Para realizar el cambio de espacio de color OpenCV ofrece la función ***CvtColor()***, esta función permite cambiar la imagen de RGB a HSV. Para especificar el tipo de conversión que se desea se utiliza el parámetro de codificación, en el cual se establece el cambio de espacio de color [8]. Para esta aplicación se emplea el código "CV_BGR2HSV"dada la transformación deseada. Después de realizar dicha conversión se crean las máscaras necesarias para la detección de un color en específico, para esta labor se emplea la función ***inrange()***, la cual comprueba si los valores de una matriz se encuentran dentro de un rango determinado por dos arreglos y en caso de cumplir la relación, los bits de salida se establecen como 1 y en caso contrario como 0, es decir, la imagen de salida será de color blanca en la región de interés y negra en su área restante [9]. En la tabla 3.1 se presentan los rangos establecidos para detectar los tonos colores de interés, en este caso rojo, azul y amarillo:

Color	Limite inferior	Limite superior	Salida
Rojo	(0, 120, 70)	(10, 255, 255)	mask1
Rojo	(170, 120, 70)	(180, 255, 255)	mask2
Amarillo	(20, 100, 100)	(30, 255, 255)	mask3
Azul	(100, 65, 75)	(130, 255, 255)	mask4

Tabla 3.1: Rangos del modelo HSV para los colores de interés

Como se puede apreciar en la tabla 3.1 para el color rojo se establecen dos máscaras, ya que este tono está presente en los dos extremos de la paleta de colores del modelo. En consecuencia, se deben tener dos rangos diferentes para abarcar todo el espectro de este. Por último, se realiza una suma de las máscaras generadas para que esta forma se logren detectar los tres colores a la vez.

3.3 Contornos

Los contornos son conjuntos de puntos conectados entre sí, que forman una figura que rodea un objeto determinado. La detección de contornos en OpenCv se aplica en imágenes binarias [10]. Para nuestra aplicación, antes de encontrar los contornos se realizan transformaciones morfológicas con el fin de mejorar la máscara de los colores. Las transformaciones morfológicas son operaciones simples que se desarrollan sobre la forma de una imagen y comúnmente se realizan en imágenes binarias, dos de los operadores morfológicos básicos son *Erosión* y *Dilatación* los cuales presentan variaciones como *Apertura* y *Cierre*.[11]

Por un lado, el proceso de dilatación agrega píxeles a los exteriores del objeto, mientras que erosión extrae el exterior del objeto. Por otro lado, el proceso de cierre consiste en aplicar un proceso de dilatación seguido de un proceso de erosión, esto es útil para cerrar pequeños agujeros que pueda tener la imagen. Para el proceso de apertura se aplica primero erosión y posteriormente se realiza el proceso de dilatación, esto ayuda a eliminar el ruido en la imagen (ver figura 3.4).[12]

(a) Proceso de dilatación (b) Proceso de erosión

(c) Proceso de apertura (d) Proceso de cierre

Figura 3.4: Transformaciones morfológicas [11]

Después de realizar las operaciones morfológicas se procede a encontrar los contornos de la imagen, para esto OpenCV presenta la función ***findContours()*** la cual obtiene los contornos de una imagen. Dos de los parámetros de esta función a destacar son el modo y el método. El modo establece de qué forma se van a recuperar los contornos, mientras que el método indica la forma de aproximación del contorno.

La metodología empleada para esta fase es la siguiente: primero se establece una estructura para realizar las operaciones morfológicas con la ayuda de la función ***getStructuringElement()*** y dado que desea detectar las tapas de las botellas, se genera un elemento de referencia, elíptico, de un tamaño de 15x15 píxeles. Posteriormente se realizan los procesos de cierre y apertura empleando los métodos mencionados anteriormente, pero para el proceso de apertura en este caso se utiliza la función ***morphologyEx()***, posteriormente se crean dos vectores con los cuales se guardan los contornos de la imagen y la jerarquía de los mismos. Por ultimo al utilizar la función ***findContours()*** el modo establecido es ”CV_RETR_EXTERNA” con el fin de recuperar solo los contornos exteriores de la imagen, mientras que el método se determina como ”CV_CHAIN_APPROX_SIMPLE ”para que sean comprimidos los segmentos y solo se tengan los puntos finales de los contornos[13].

3.4 Momentos

Los momentos son una medida particular que indica la dispersión de una nube de puntos, con ellos se pueden describir variables geométricas de una imagen, tales como el tamaño, posición y orientación. Los momentos se pueden definir matemáticamente por medio de la ecuación 3.1, en la que x,y son la coordenadas de un píxel y la función I(x,y) corresponde a su intensidad, dado que se está trabajando con una imagen binaria los valores de intensidad solo pueden ser 0 o 1.

$$M_{ij} = \sum_{x}\sum_{y} x^i y^j I(x,y) \tag{3.1}$$

Para encontrar la posición de las tapas de las botellas lo que se hace es calcular la posición del contorno que estas entregan por medio de la segmentación de color. Para este proceso se necesitan tres momentos en particular, los cuales son: M00, M10 y M01. El momento M00 devuelve el área del contorno, dado que al remplazar los valores de "i" y "j" en la ecuación 3.1, la ecuación resultante es equivalente al número de píxeles cuyo valor es 1. El momento M10 equivale a la sumatoria de todas las coordenadas x de los píxeles que posean una intensidad de 1, lo mismo ocurre con el momento M01, pero con este se obtiene la sumatoria para las coordenadas en y. Ahora bien, si se realiza un cociente entre los momentos M10 y M01 con M00 se obtiene el centroide del contorno, es decir, $\bar{x}$ y $\bar{y}$, lo que corresponde a la posición del punto central de este.[14]

Para calcular los momentos OpenCv ofrece la función ***moments()***, la cual calcula los momentos hasta el tercer orden de un polígono. Ya que se quiere encontrar los momentos de los posibles contornos, se debe crear un vector del mismo tamaño del arreglo en el cual se encuentra la información de los contornos con el fin de almacenar el centroide de todos ellos.

3.5 Posición de la tapa de la botella

En esta fase se obtiene la posición en metros de la tapa de la botella; para esto se emplea el modelo Pin Hole sobre las coordenadas de los centroides de los contornos, lo cuales se encuentran en unidades de píxel. El modelo Pin Hole describe la relación matemática entre una coordenada en un espacio tridimensional con su proyección en el plano de una imagen de una cámara estereoscópica[15]. Este modelo está definido por la ecuación 3.3[16], en donde:

- (X,Y,Z) corresponde a una coordenada 3D un espacio del mundo real
- (u,v) son las coordenadas de la proyección del objeto en píxeles
- A es la matriz de la cámara o la matriz de parámetros intrínsecos de la cámara
- (f_x, f_y) corresponden a las distancias focales en píxeles
- (c_x, c_y) es el punto central de la imagen.

$$sm' = A\,[R|t]\,M' \tag{3.2}$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_3 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.3}$$

La matriz intrínseca de la cámara es única y no varía según la escena que esta ve; para encontrarla se realiza un proceso de calibración en el que además de obtener la matriz de la cámara también se obtienen los parámetros de distorsión, los cuales ayudan a eliminar la posible distorsión que pueda presentar la imagen

debido a los sensores que la cámara pueda tener. En este caso no se realizó un proceso de calibración ya que la matriz de la cámara fue dada por el repositorio del centro tecnológico de automatización industrial (CTAI) el cual, en proyectos anteriores realizó dicha actividad. Además, la distorsión presente en la imagen entregada por el *kinect* es despreciable para el cálculo de la posición. En consecuencia, se omite el proceso de eliminación de distorsión de la imagen. En todo caso, OpenCV ofrece una gama de funciones para el cálculo de las matrices así como la calibración de la cámara, además de funciones para eliminar la distorsión de la imagen, como por ejemplo ***undistort()***.

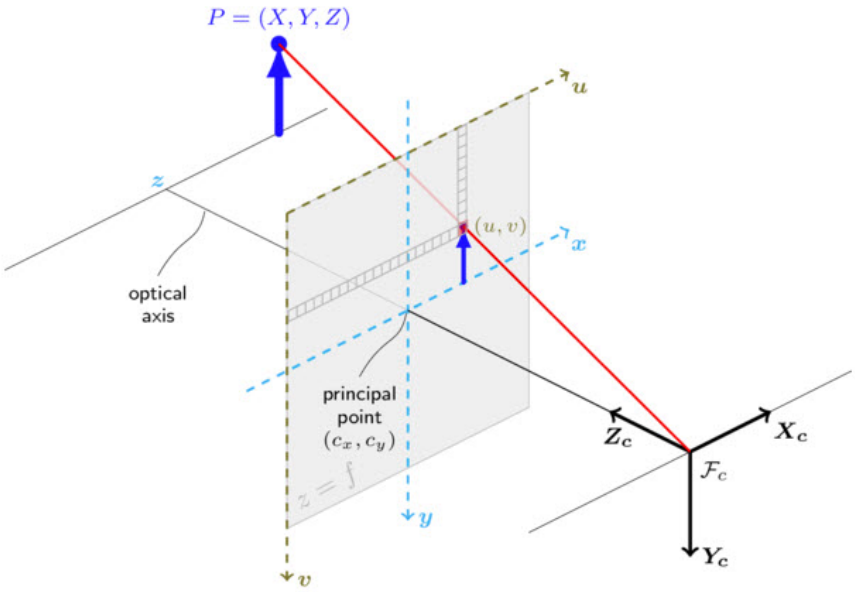


Figura 3.5: Relación entre los ejes coordenados del mundo real, la imagen y la cámara [17]

Si una imagen de la cámara es escalada por algún factor todos sus parámetros deben ser escalados, por esta razón se realizan operaciones matemáticas simples de la coordenada en píxel para llevarla a una coordenada en el mundo real en 3D. Para empezar, se parte del centroide del contorno de la botella, a esta coordenada en píxeles se restan la coordenada (c_x, c_y) para llevar el eje de referencia que se encuentra en un extremo de la imagen al centro de esta. Posteriormente, se divide el valor obtenido entre las distancias focales de la cámara (f_x, f_y) y por ultimo se multiplica este cociente por la distancia a la que se encuentra la tapa de la botella, dicha distancia se obtiene de la imagen de profundidad guardad por la función ***depthcallback()***, mencionada en la sección de **Captura de imagen**. De esta manera se ha pasado de una proyección del objeto en píxeles a una posición en el mundo real en 3D.

$$\begin{aligned} u &= \bar{x} - c_x & v &= \bar{y} - c_y \\ x' &= \frac{u}{f_x} & y' &= \frac{v}{f_y} \\ x_c &= x' * Z & y_c &= y' * Z \end{aligned}$$

Cabe recalcar que para conocer la profundidad de la tapa evaluamos la imagen de profundidad entregada por el *kinect* en la posición específica a las coordenadas del centroide de los contornos. Para evaluar la matriz se utiliza `imagen.at<float>(Point(`$\bar{x}$`, `$\bar{y}$`))`, es indispensable emplear la función ***Point()*** para indicar la posición a evaluar, dado que OpenCv recorre la matriz primero en 'y' y luego en 'x'. En consecuencia, de omitirlo se evaluaría la profundidad en un punto erróneo.

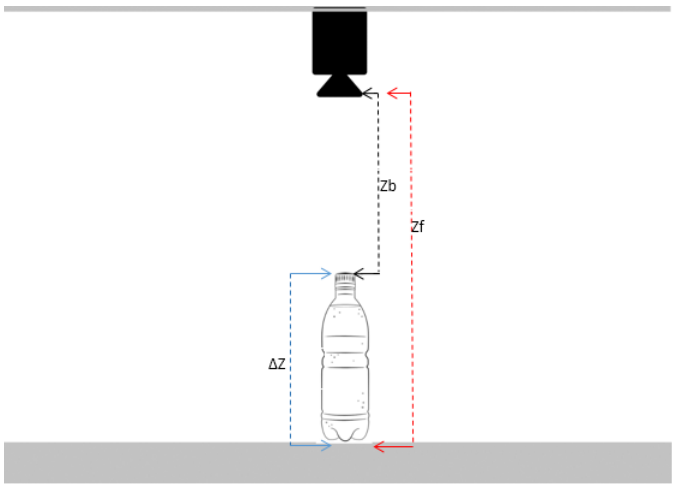
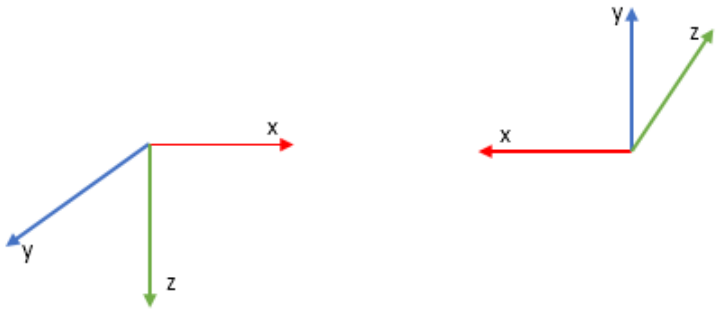


Figura 3.6: Estrategia para la posición de la botella

3.6 Estrategia para la posición de la botella

El cálculo de la posición de la botella se hace con base en la información recolectada con las 5 fases de la detección de la tapa. El cálculo de la posición se realiza relacionando las coordenadas de la tapa con la altura de la botella; esta estrategia esta dividida en dos momentos: primero, se calcula la altura de la botella obteniendo la diferencia de profundidades entre el área de trabajo y la tapa de botella; segundo, se calcula el punto medio aproximado de la botella y se realiza un ajuste en plano para que el *gripper* deseado se dirija a esa posición.

Para encontrar la profundidad del área de trabajo, se realiza el mismo proceso con el cual se obtuvo la profundidad de la tapa, pero en este caso se establece un punto fijo en la imagen, en dicho punto no habrá ninguna botella en ningún caso para que de esta forma no haya error en el valor. Posteriormente el valor obtenido se resta al ya conocido 'Z' de la tapa para obtener el δZ que equivale a la altura que se está buscando. Para asignar la coordenada en 3D de la botella por un lado, se mantienen las coordenadas (x, y) de la tapa para la botella y la coordenada en z será la mitad de la altura de la ella, de esta forma el *gripper* designado para tomarla llegaría al punto medio de esta.



(a) Eje coordenado del kinect en el mundo real

(b) Eje coordenado del kinect en el entorno de simulación

Figura 3.7: Ejes coordenados del kinect

Como se puede observar en la figura 3.7 el sistema coordenado de la cámara en el mundo real no es el

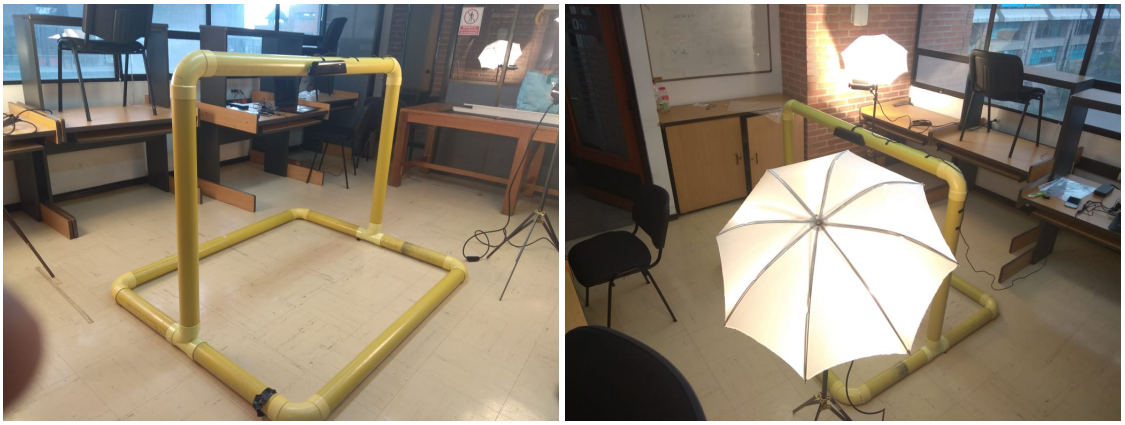
mismo en el entorno de la simulación. Por esta razón se deben asignar los valores (x, y, z) tanto de la tapa como de la botella de acuerdo al sistema coordenado del entorno de simulación. En este caso, $x_{sim} = -x_c$, $y_{sim} = -z$ y $z_{sim} = -y_c$

3.7 Protocolo de pruebas

Para comprobar el correcto funcionamiento del algoritmo para detectar la tapa de la botella y la estrategia para calcular la posición de la botella se diseñaron las siguientes pruebas:

- Calibración punto 0
- Detección y posición de un color a la vez
- Detección y posición de varios colores
- Obtención de múltiples contornos
- Detección y posición tapas de botellas
- Cálculo de la altura de diferentes botellas

Para cada prueba se realizaron varias iteraciones en el código, específicamente se realizaron 100 iteraciones por prueba con el objetivo de promediar la información y así tener mayor precisión en los resultados. El escenario de pruebas, fue el cubículo 607 de la facultad de ingeniería de la Pontificia Universidad Javeriana. En dicho lugar se construyó un área de trabajo para el kinect, con la cual se recreará la altura a la que este se encontraba respecto al área de trabajo del robot en el Centro Tecnológico de automatización industrial (CTAI), la cual era de aproximadamente 1.20m.



(a) Base para el *kinect* (b) Área de trabajo para el *kinect*

Figura 3.8: Entorno para las pruebas con el *kinect*

- **Calibración punto 0**

  Con esta prueba se busca verificar que la coordenada $(x = 0, y = 0)$ corresponda al punto de origen del plano cartesiano, es decir, que aproximadamente la posición en píxeles $(x = 320, y = 240)$ equivalga a la posición en metros $(x = 0, y = 0)$. Para el desarrollo de esta prueba se dibuja un pequeño círculo con centro en el origen sobre la imagen entregada por el *kinect*. Posteriormente en el área de trabajo se

ubica la silueta de un círculo la cual tenga su centro en el origen proyectado por la imagen del *kinect*. Luego se obtiene la posición de la silueta y se verifica que esta sea aproximadamente $(x = 0.0, y = 0,0)$, en caso de que no se cumplan estos valores, se debe restar la diferencia entre la posición esperada que es el origen, con la obtenida por el *kinect* con el objetivo de tener una medida acertada, ya que la disparidad entre las medidas se mantiene para cualquier posición de la imagen

- **Detección y posición de un color a la vez**

  La finalidad de esta prueba es corroborar primero que la segmentación de color se haga de manera correcta para los tres tonos, además de verificar que la posición (x,y,z) de los contornos se haga de manera correcta a lo largo del área de trabajo. Para el desarrollo de esta prueba se tienen tres siluetas circulares de tamaños similares (una por cada tono de color), que serán ubicadas en diferentes posiciones del área de trabajo y estarán sobre el nivel de esta, es decir, sobre el suelo. Para comprobar que la información obtenida con el algoritmo sea correcta, las posiciones de las siluetas serán asignadas de manera aleatoria, pero teniendo como referencia de medición una cuadrícula de 60cm x 60 cm, con un espacio de 1cm entre sus divisiones

- **Detección y posición de varios colores**

  En esta prueba se realiza el mismo proceso de la prueba "Detección y posición de un color a la vez"pero en este caso sobre el área de trabajo estarán las tres siluetas, de los tres colores. El objetivo de este proceso es comprobar que el sistema sea capaz de reconocer y entregar la posición correcta de los contornos de las tres circunferencias de manera simultánea.

- **Obtención de múltiples contornos**

  Esta prueba se centra en la detección de la mayor cantidad de figuras posibles a la vez y verificar que los contornos generados sean los correctos. Para el desarrollo de esta prueba se tendrán cinco siluetas circulares de diferentes tamaños para cada color, dichas siluetas serán ubicadas a lo largo del área de trabajo. Primero se hará la prueba por cada color y posteriormente se mezclaran entre sí las siluetas.

- **Detección y posición de tapas de botellas**

  En esta prueba se calcula la posición de las tapas de las botellas y se verifica la capacidad del sistema para reconocer siluetas más pequeñas y la obtención correcta de la profundidad de un punto en específico en el plano. Para esta prueba se cuenta con tres botellas de diferentes tamaños, cada una con un color de tapa diferente. Para verificar el cálculo correcto de la posición de tapa se tiene una cuadricula de 60cm x 60cm con divisiones de 1cm, la cual es colocada justo sobre la botella, dado que el cálculo de la posición se hace con la información de profundidad de la tapa. En consecuencia, la posición (x,y,z) debe se verificar a la misma altura de la tapa de la botella.

- **Cálculo de la altura de diferentes botellas**

  En esta prueba se verifica que se obtenga de manera correcta la profundidad sobre diferentes botellas con el fin de comprobar que se puede implementar de manera efectiva la estrategia para la posición de estas. Para el desarrollo se colocarán diferentes botellas, de diferentes alturas; a lo largo del área de trabajo y posteriormente se obtendrá la profundidad de sus tapas y a la vez la altura de la botella en cuestión.

3.8 Resultados obtenidos

A continuación se presentan los resultados de cada uno de los puntos del protocolo de pruebas así como información sobre su desarrollo.

- **Calibración punto 0**: Para empezar, se dibujó un pequeño círculo sobre la imagen del kinect con el objetivo de conocer qué punto en el área de trabajo seria determinado como el origen. Al colocar el centro de la silueta sobre ese punto se obtuvieron los resultados de la tabla 3.2, como se puede observar, la coordenada en 'x' presenta una desviación de aproximadamente 5cm, mientras que la coordenada en 'y' de 2cm. Después de realizar el ajuste en el origen, se obtuvieron los resultados de la tabla 3.3

	Promedio
Coordenada X	0,050259 m
Coordenada Y	-0,024187 m

Tabla 3.2: Coordenada (x=0,y=0) sin ajuste

	Promedio
Coordenada X	0,0004641295 m
Coordenada Y	0,0003851932 m

Tabla 3.3: Coordenada (x=0,y=0) con ajuste

Aunque no es posible calcular el error en esta prueba, ya que se está midiendo respecto a la coordenada $(x = 0, y = 0)$ y el resultado sería indefinido, se observa que la diferencia entre lo esperado y lo obtenido con el kinect se encuentra en el rango de las décimas de milímetros.

- **Detección y posición de un color a la vez**

Como se mencionó en el protocolo de pruebas, para esta parte se tienen a disposición tres siluetas circulares una por cada tono de color a detectar (rojo, amarillo o azul). Se realizaron diez pruebas diferentes por cada tono, además de eso se calculó el error entre la medición del kinect y la posición establecida en el plano real. A continuación, se presentan los resultados obtenidos de las pruebas para cada color, así como las siluetas utilizadas.

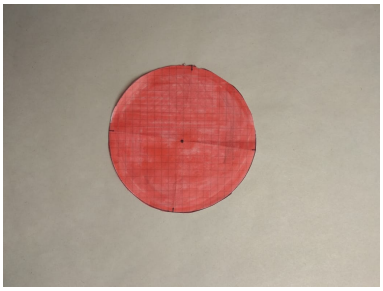


Figura 3.9: Círculo rojo utilizado en la prueba

Prueba	Cord. X [m]	Cord. Y [m]	Cord. Z [m]	Ref X [m]	Ref Y [m]	Ref Z [m]	Error X [%]	Error Y [%]	Error Z [%]
1	-0,001	-0,002	1,168	0,000	0,000	1,170			
2	-0,266	-0,214	1,165	-0,261	-0,216	1,170	1,933	0,948	0,427
3	0,287	-0,289	1,153	0,288	-0,272	1,170	0,347	6,355	1,435
4	0,400	0,226	1,183	0,381	0,215	1,170	5,016	4,974	1,130
5	-0,269	0,270	1,181	-0,258	0,269	1,170	4,163	0,380	0,957
6	-0,002	0,422	1,189	-0,002	0,424	1,170	0,000	0,460	1,583
7	-0,545	-0,321	1,165	-0,524	-0,311	1,170	4,006	3,345	0,464
8	-0,285	0,001	1,166	-0,270	0,001	1,170	5,465	0,000	0,312
9	-0,577	0,184	1,168	-0,551	0,183	1,170	4,808	0,683	0,180
10	0,501	-0,214	1,157	0,472	-0,209	1,170	6,210	2,232	1,087

Tabla 3.4: Coordenadas en (x,y,z) para un círculo de color rojo

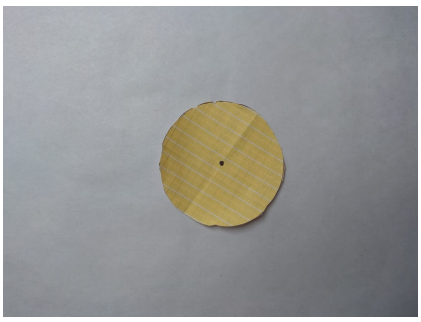


Figura 3.10: Círculo amarillo utilizado en la prueba

Prueba	Cord. X [m]	Cord. Y [m]	Cord. Z [m]	Ref X [m]	Ref Y [m]	Ref Z [m]	Error X [%]	Error Y [%]	Error Z [%]
1	-0,410	0,248	1,179	-0,394	0,242	1,170	4,014	2,311	0,781
2	-0,257	-0,286	1,163	-0,246	-0,276	1,170	4,510	3,724	0,624
3	0,379	0,295	1,186	0,363	0,281	1,170	4,439	4,974	1,394
4	-0,386	0,276	1,181	-0,371	0,269	1,170	4,003	2,583	0,940
5	0,045	0,458	1,193	0,043	0,436	1,170	4,626	5,102	1,949
6	0,181	0,117	1,179	0,173	0,113	1,170	4,824	3,831	0,781
7	-0,456	-0,370	1,160	-0,427	-0,341	1,170	6,680	8,641	0,834
8	-0,445	0,449	1,190	-0,425	0,429	1,170	4,609	4,613	1,692
9	0,534	0,277	1,195	0,503	0,271	1,170	6,089	2,089	2,127
10	0,021	0,021	1,165	0,023	0,022	1,170	9,318	6,793	0,427

Tabla 3.5: Coordenadas en (x,y,z) para un círculo de color amarillo

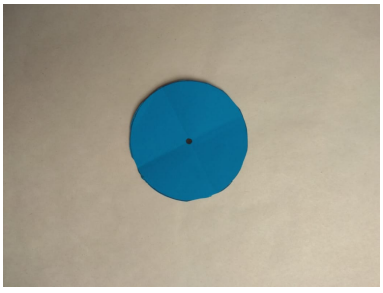


Figura 3.11: círculo azul utilizado en la prueba

Prueba	Cord. X [m]	Cord. Y [m]	Cord. Z [m]	Ref X [m]	Ref Y [m]	Ref Z [m]	Error X [%]	Error Y [%]	Error Z [%]
1	-0,410	0,248	1,179	-0,394	0,242	1,170	4,014	2,311	0,781
2	-0,257	-0,286	1,163	-0,246	-0,276	1,170	4,510	3,724	0,624
3	0,379	0,295	1,186	0,363	0,281	1,170	4,439	4,974	1,394
4	-0,386	0,276	1,181	-0,371	0,269	1,170	4,003	2,583	0,940
5	0,045	0,458	1,193	0,043	0,436	1,170	4,626	5,102	1,949
6	0,181	0,117	1,179	0,173	0,113	1,170	4,824	3,831	0,781
7	-0,456	-0,370	1,160	-0,427	-0,341	1,170	6,680	8,641	0,834
8	-0,445	0,449	1,190	-0,425	0,429	1,170	4,609	4,613	1,692
9	0,534	0,277	1,195	0,503	0,271	1,170	6,089	2,089	2,127
10	0,021	0,021	1,165	0,023	0,022	1,170	9,318	6,793	0,427

Tabla 3.6: Coordenadas en (x,y,z) para un círculo azul

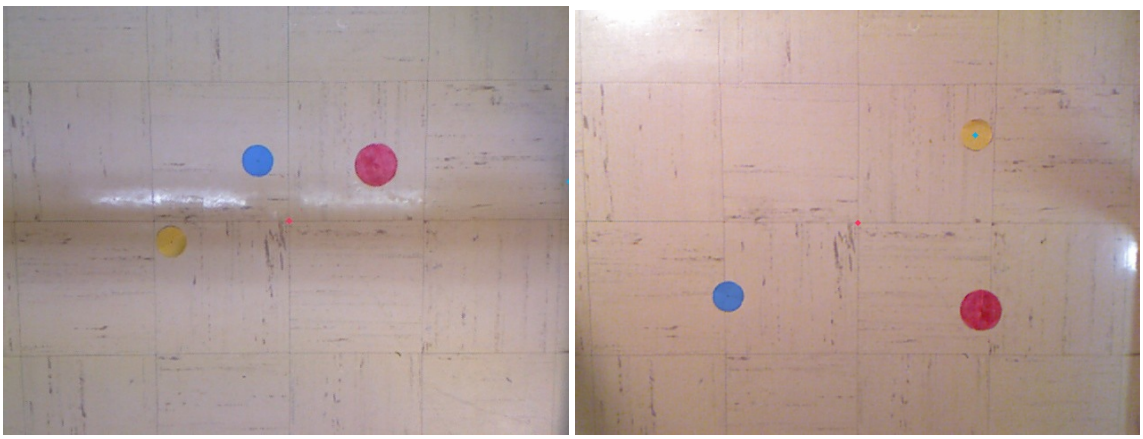
Al observar los errores porcentuales de las tablas 3.4, 3.5 y 3.6 se puede observar que se cumple el objetivo de tener una precisión de mínimo el 90%, lo que es equivalente a tener un error máximo del 10%.

- **Detección y posición de varios colores**

  Para esta prueba se ubicó cada círculo en dos posiciones diferentes del área de trabajo, los resultados de sus posiciones en (x, y, z) se presentan en la tabla 3.7.

Prueba	Color	Cord. X [m]	Cord. Y [m]	Cord. Z [m]	Ref X [m]	Ref Y [m]	Ref Z [m]	Error X [%]	Error Y [%]	Error Z [%]
1	Amarillo	-0,304	0,053	1,166	-0,295	0,052	1,170	3,167	1,801	0,360
	Rojo	0,228	-0,145	1,173	0,215	-0,135	1,170	6,059	7,612	0,256
	Azul	-0,080	-0,155	1,137	-0,078	-0,152	1,170	2,655	2,256	2,821
2	Amarillo	0,308	-0,226	1,161	0,293	-0,216	1,17	5,275	4,683	0,786
	Rojo	0,328	0,230	1,186	0,312	0,223	1,17	5,050	2,963	1,360
	Azul	-0,343	0,192	1,183	-0,326	0,193	1,17	5,265	0,503	1,136

Tabla 3.7: Coordenadas en (x,y,z) para las tres siluetas a la vez

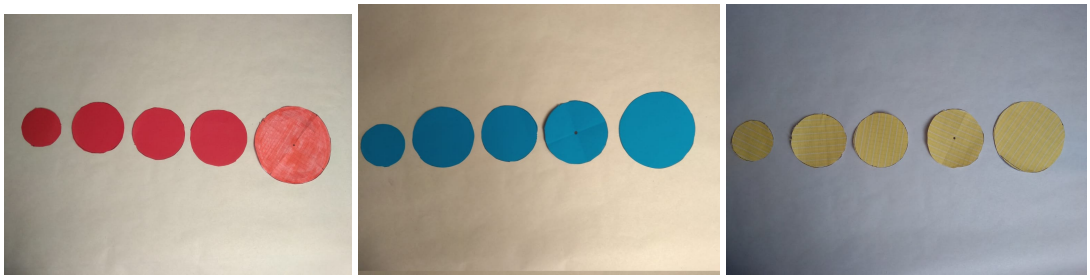


(a) Posiciones para la prueba 1 (b) Posiciones para la prueba 2

Figura 3.12: Posiciones de las siluetas en el espacio para realizar las pruebas de ”Detección y posición de varios colores”

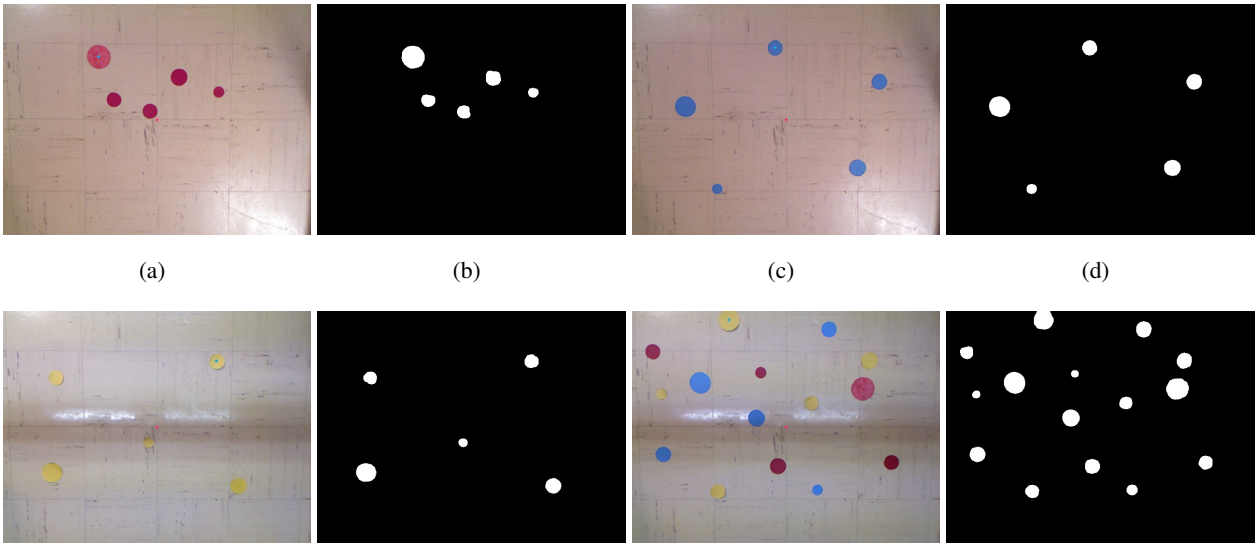
Como se puede observar en la información recolectada para esta prueba, el algoritmo logra tanto detectar de manera correcta los tres colores a la vez como entregar su posición en el espacio. Los errores obtenidos para esta prueba para cada una de las coordenadas de igual forma no exceden el umbral establecido de 10 %.

- **Detección de múltiples contornos**
  Como se mencionó en el protocolo de pruebas, para esta sección se colocaron en el área de trabajo 5 siluetas de diferentes tamaños por cada tono de color, las siluetas para cada color se presentan en la figura 3.13. Los resultados de la detección del color y los contornos se visualizan en la figura 3.14.



(a) Grupo rojo (b) Grupo azul (c) Grupo amarillo

Figura 3.13: Grupos de figuras para la prueba



(a) (b) (c) (d)

(e) (f) (g) (h)

Figura 3.14: Máscaras generadas e imágenes RGB

Como se puede observar en la figura 3.14 el sistema es capaz de reconocer hasta 15 siluetas a lo largo del espacio. Cabe mencionar que el sensor Kinect es bastante sensible a cambios de iluminación y a pesar de utilizar el espacio de color HSV, el desarrollo de la prueba se realizó bajo condiciones específicas de iluminación.

- **Detección y posición de botellas** Para esta prueba se cuenta con tres botellas de diferentes altura, como se mencionó en las especificaciones del sistema estas botellas plasticas de bebidas. Los resultados obtenidos se presentan en la tabla 3.8.

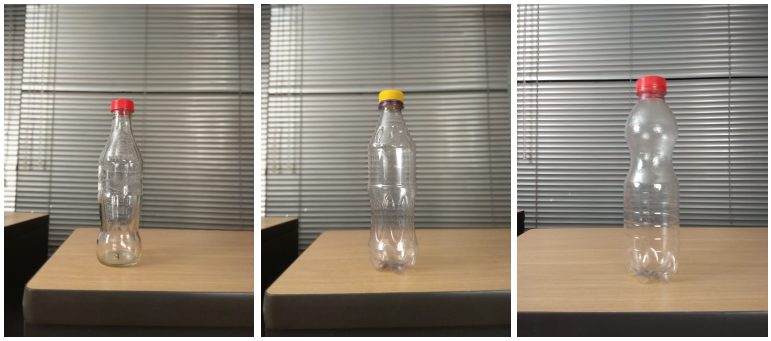
Prueba	Cord. X [m]	Cord. Y [m]	Cord. Z [m]	Altura [m]	Ref X [m]	Ref Y [m]	Ref Z [m]	R Altura [m]
Botella 1	-0,227	-0,208	0,949	0,208	-0,218	-0,206	0,955	0,206
Botella 2	-0,204	-0,221	0,938	0,227	-0,192	-0,218	0,943	0,221
Botella 3	0,063	-0,165	0,935	0,230	0,062	-0,175	0,931	0,226

Tabla 3.8: Posición (x,y,z) y altura de las botellas de prueba

Prueba	Error X [%]	Error Y [%]	Error Z [%]	Er Altura [%]
Botella 1	4,126	0,889	0,618	0,893
Botella 2	6,186	1,559	0,544	2,792
Botella 3	0,959	5,581	0,379	1,978

Tabla 3.9: Error porcentual para la posición de las botellas y su altura

Como se observa en la tabla 3.9 el mayor error es de aproximadamente 5.5 %, el cual se encuentra en el rango deseado. De esta forma se comprueba que el sistema es capaz de entregar la posición de una botella con una precisión de al menos 90 %.



(a) Botella prueba 1 (b) Botella prueba 2 (c) Botella prueba 3

Figura 3.15: Botellas de prueba

- **Cálculo de la altura de diferentes botellas** En esta prueba se contó con once botellas de alturas variadas con el objetivo de comprobar la capacidad del sistema de obtener diferentes alturas y paralelamente comprobar la eficacia de la estrategia para el cálculo de la altura de la botella. Los resultados obtenidos se muestran en la tabla 3.10.

Prueba	Altura [m]	Altura Ref [m]	Error [%]
Botella 1	0,368	0,362	1,668
Botella 2	0,351	0,345	1,831
Botella 3	0,352	0,346	1,660
Botella 4	0,265	0,256	3,458
Botella 5	0,250	0,250	0,111
Botella 6	0,325	0,322	0,806
Botella 7	0,256	0,245	4,647
Botella 8	0,353	0,353	0,115
Botella 9	0,191	0,180	6,381
Botella 10	0,241	0,239	0,713
Botella 11	0,216	0,211	2,487

Tabla 3.10: Prueba de altura de las botellas

Como se puede evidenciar en los errores porcentuales de la tabla 3.10, se cumple la especificación de precisión en el cálculo de la altura de la botella. En consecuencia, la posición en z de esta tiene un error menor al 10%. A continuación se presentan 3 de las botellas empleadas en las pruebas.

(a) Botella prueba 1 (b) Botella prueba 5 (c) Botella prueba 9

Figura 3.16: Algunas de las botellas de prueba para el cálculo de la altura

- **Matriz de confusión**: Para realizar la matriz de confusión del sistema de visión, en especifico de la detección de la tapa de la botella, se utilizaron 3000 datos en los cuales se observa sí el sistema es capaz de detectar o no la tapa de la botella y a la vez obtener la posición en píxeles de esta. En el caso de no lograr reconocer la tapa, la posición entregada por el sistema correspondería a $''-nan''$.

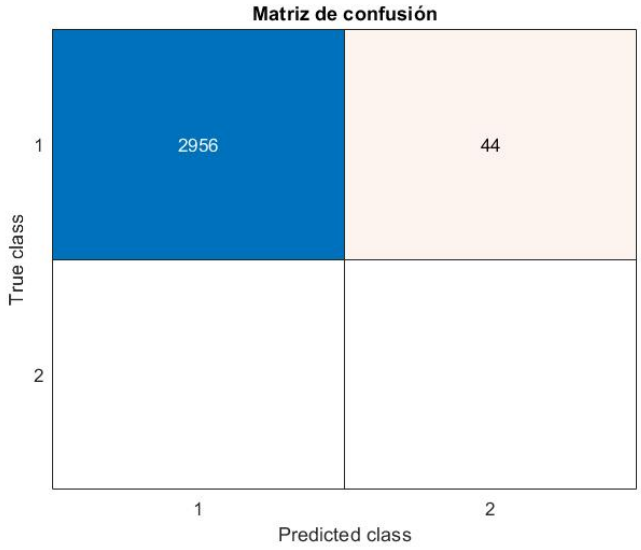


Figura 3.17: Matriz de confusión para la detección de la tapa de la botella

Como se puede observar en la figura 3.17 se tiene un total de 3000 datos, en los cuales 2956 son verdaderos positivos y 44 falsos negativos. En nuestra aplicación se categorizaron como ”Bien” o ”Mal” para una detección correcta y la no detección de la tapa respectivamente. Los datos de referencia para realizar la matriz de confusión son 3000 casos en los que el sistema es capaz de detectar las tapas, por esta razón en la matriz no se tienen verdaderos negativos, ya que el sistema solo tiene como referencia casos de detección correcta y, en consecuencia, los casos de una detección fallida los cataloga como falsos negativos. Con base en la información entregada por la matriz de confusión se obtuvo las siguientes medidas.

1. **Acurracy**

$$Acurracy = \frac{VP+VN}{Total} \tag{3.4}$$

$$Acurracy = \frac{2956}{3000} = 0.9853 \tag{3.5}$$

2. **Error**

$$Error = 1 - Acurracy \tag{3.6}$$

$$Error = 0.0147 \tag{3.7}$$

3. **True Positive Rate**

$$TPR = \frac{VP}{VP+FP} \tag{3.8}$$

$$TPR = 1 \tag{3.9}$$

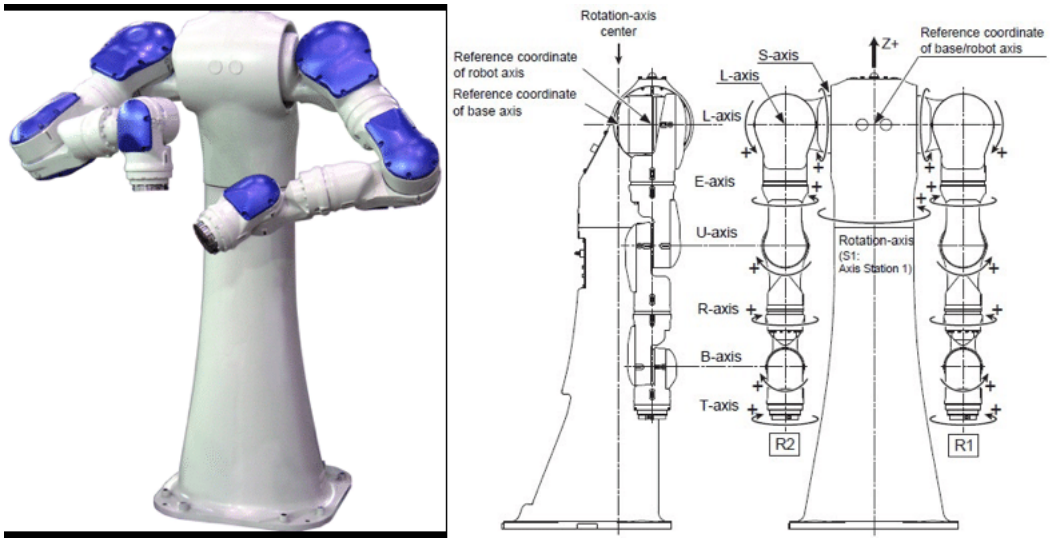
Como se observa en los resultados de las ecuaciones 3.4, 3.6 y 3.8, se obtuvo que un 98.5 % de los datos se clasificaron de manera correcta, obteniendo de esta forma un error menor al 2 %, el cual se ajusta a lo establecido en los objetivos del proyecto.

4. Robot Manipulador

En este capítulo se presentan los items más importantes para la simulación del robot, teniendo en cuenta los *grippers* con los que cuenta el autómata, para, a partir de esto, construir la secuencia de movimientos necesaria para la extracción de la tapa de la botella.

4.1 Estructura del robot

Es importante tener en cuenta que la solución está planteada con un robot Motoman Yaskawa SDA10f, el cual es un robot de dos brazos que cuenta con la capacidad de trabajar con los dos brazos en simultáneo. Además, el robot cuenta con 15 ejes de rotación (7 en cada brazo y la rotación del torso como se puede ver en la figura 4.1(a)), lo que permite que pueda desempeñar diferentes labores que antes solo podían ser ejecutadas por humanos. En la figura 4.1 se ilustra el robot y los grippers (marca Robotiq) adecuados al mismo para la solución.



(a) grados de libertad del Robot Motoman SDA10f [18]

(b) Gripper de dos dedos

(c) Gripper de tres dedos

Figura 4.1: Robot Motoman SDA10F y grippers

Como se puede observar en la figura 4.1(b), se cuenta con un *gripper* de dos dedos, el cual está dispuesto en el brazo izquierdo del robot y será usado, principalmente, para la extracción de la tapa de la botella; mientras que el *gripper* de tres dedos, ilustrado en la figura 4.1(c) está ubicado en el brazo derecho y será usado para la manipulación del cuerpo de la botella.

4.2 Programación del robot

Para el desarrollo de la programación de alto nivel para el funcionamiento del robot, es indispensable tener en cuenta que se realiza mediante *ROS (Robotic Operative System)*, el cual es un pseudo sistema operativo de libre acceso, creado específicamente para robots, el cual permite simular e implementar los entornos en los cuales se desempeña un robot, además de contar con diferentes herramientas de abstracción del mundo real, controladores de dispositivos de bajo nivel, mensajes para la comunicación entre diferentes procesos y cuenta además con soporte continuo **[19]**. Cabe resaltar que existen varias versiones, pero para este caso se utiliza **ROS-kinetic** y **ROS-Industrial**, el cual está soportado en el sistema operativo **Ubuntu 16.04**.

Ahora bien, para el desempeño del robot, existe una plataforma llamada **MoveIt**, la cual se ejecuta basada en ROS y provee las funcionalidades para la cinemática del robot, la planificación de trayectorias, revisión de colisión, percepción 3D (mediante sensores) entre muchas otras cosas más.

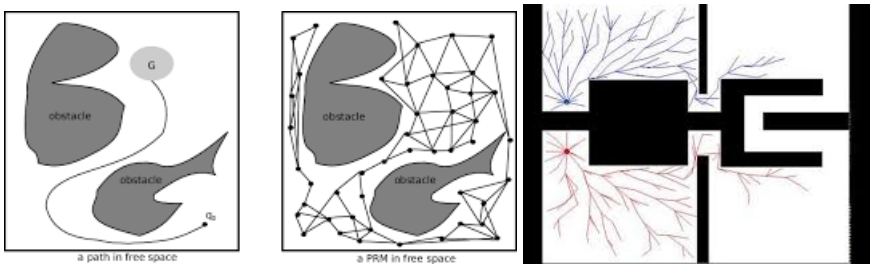
4.2.1 Planificador de trayectoria

- **Chequeo de colisión**

  Como se mencionó anteriormente, **MoveIt** permite realizar la planificación de trayectorias, lo cual es posible mediante diferentes tipos de planificadores ofrecidos por esta plataforma, mediante la libería de acceso libre llamada *OMPL (Open Motion Planning Library)* **[20]**. Dentro de esta librería, se encuentran dos grandes grupos de planificadores: "*Geometric Planners (GP)*" y "*Control Based Planners (CBP)*", en donde, para GP, se tienen en cuenta específicamente las restricciones geométricas y cinemáticas del sistema [21], mientras que los CBP evalúan restricciones diferenciales, siendo por ende muy usado en robots móviles [22]. Como en este caso no se usa un robot móvil, se escoge el grupo de los GP, entrando entonces a evaluar los algoritmos de planificación que se encuentran dentro de este grupo.

  Después de observar el funcionamiento de cada algoritmo y reconocer los mas usados en este ámbito, se compararon dos planificadores, los cuales se muestran a continuación:

  - ***Probabilistic Roadmap Method (PRM)***: es un planificador que genera una ruta de metas que se dirigen a un objetivo final, trazando la trayectoria que le indica este espacio de estados. (Ver figura 4.2(a)
  - ***Rapidly-Exploring Random Trees (RRT)***: es un planificador cuyo objetivo es encontrar la ruta óptima mediante un muestreo de trayectorias con respecto a la posición final. (Ver figura 4.2(b)



(a) PRM (b) RRT

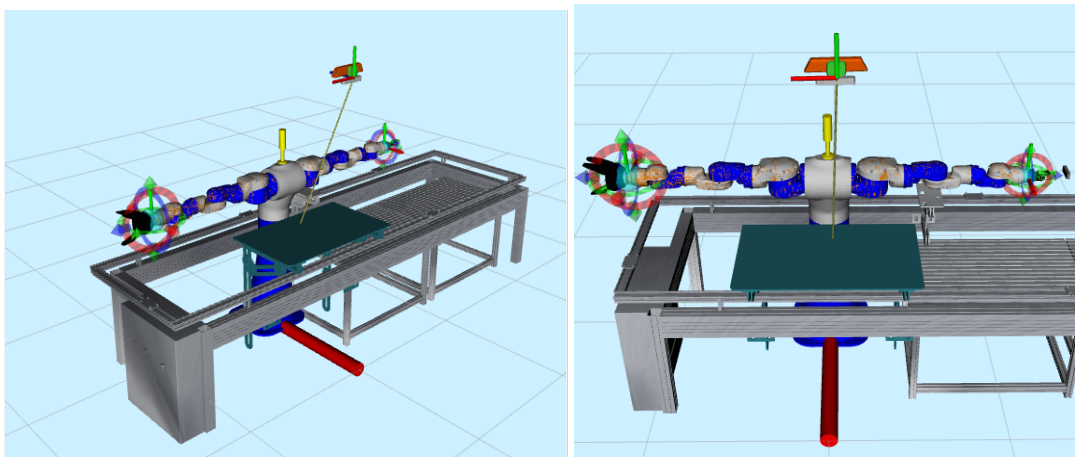
Figura 4.2: Planificadores: PRM vs RRT

De acuerdo con lo visto en la figura 4.2, se escoge el planificador RRT debido a que tiene una revisión más exahustiva de las posibles trayectorias, mediante la generación de todo un árbol completo que permite reconocer casi completamente el entorno en comparación con el PRM que si bien, no tiene un mal desempeño, no reconstruye de manera completa el entorno.

- ***Inverse Kinematics Solver***
  Por otra parte, se debe tener en cuenta el calculador para la cinemática inversa (IK) del robot, debido a que se quiere, mediante código, enviar la posición y orientación al efector final. Para esto, debido a que en **MoveIt!** se encuentra una herramienta llamada *Moveit_setup_assistant*, cuya función es generar los archivos *URDF/SRDF* del robot, también genera el archivo del calculador de la *IK* por defecto llamado *KDLKinematicsPlugin* [24]. Según la documentación leída, se puede usar este *IK solver*, sin tener que usar otro más potente como el *IKFast* o el *TRAC-IK*, debido a que el robot tiene más de 6 grados de libertad (DOF). En el **Capítulo 7 Sección 7.3** se puede ver la configuración del *IK solver* con un pequeño cambio con respecto al generado por defecto desde el *Moveit_setup_assistant*.

4.2.2 Secuencia para destapar la botella

Para la secuencia que tiene como objetivo destapar la botella es importante tener en cuenta el entorno de trabajo del robot, el cuál se muestra en la figura 4.3(a). Dentro de dicho entorno se observa, al frente del robot, una especie de plataforma, que actúa como mesa, la cual es el espacio donde el robot va a realizar el proceso de agarre y destape de la botella. Dicha mesa, que se puede observar mejor en la figura 4.3(b), se encuentra a una altura de 0.9 metros del suelo, aproximadamente. La mesa tiene las siguientes medidas: ancho = 50 cm, largo = 90 cm. Por esta razón, y debido a que el robot logra ir a todas las posiciones con el movimiento único de los brazos, se decide no planificar con respecto al torso del robot, es decir, el torso del robot no se va a mover para implementar esta solución.



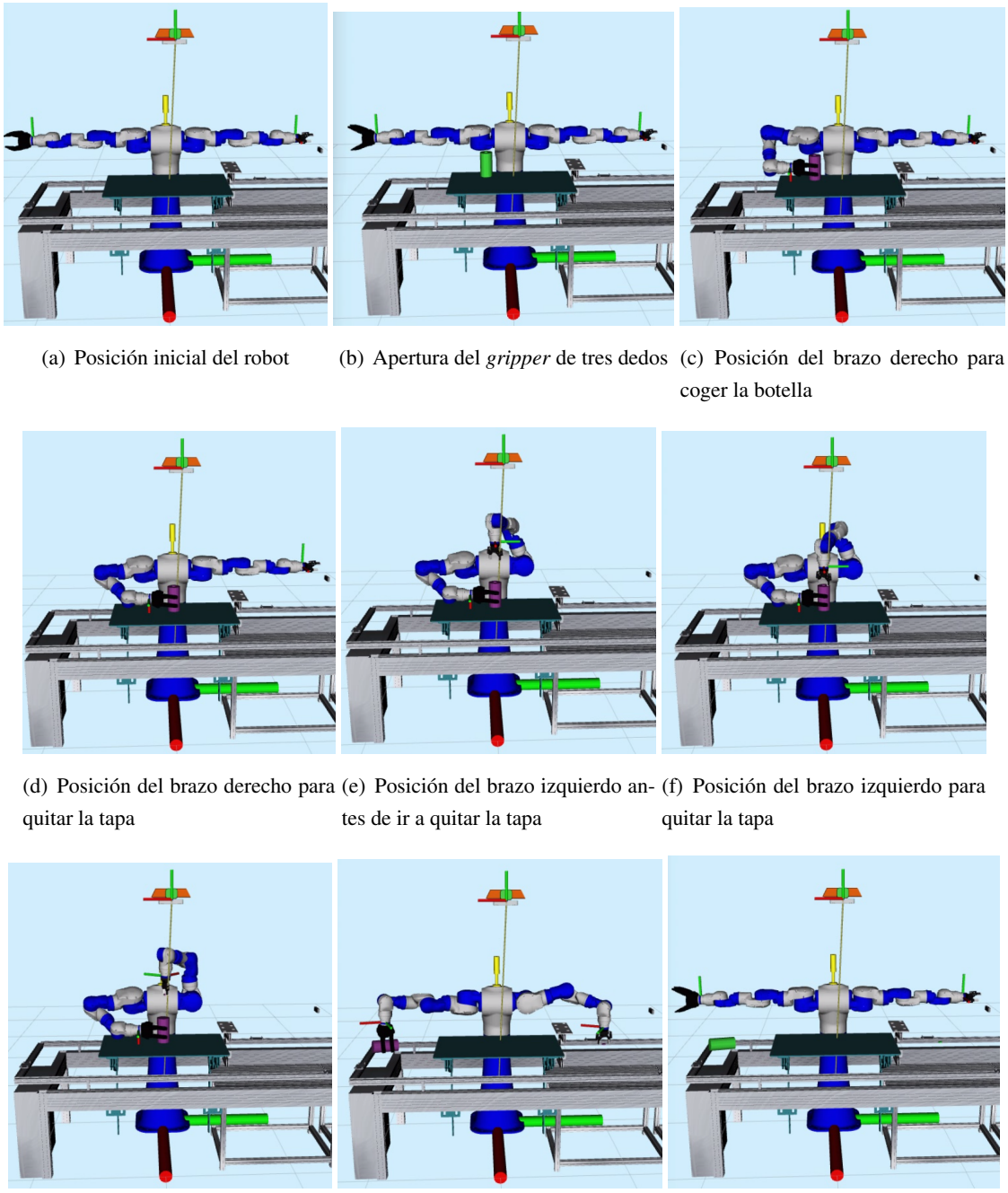
(a) Entorno de trabajo general (b) Enfoque sobre la mesa

Figura 4.3: Entorno de trabajo del robot

Recordando los *gripper* con los cuales cuenta el robot, se tiene presente que para el agarre de la botella es mejor hacer uso del *gripper* de tres dedos (adaptado al brazo derecho del robot), y para extraer la tapa, se hace uso del *gripper* de dos dedos (adaptado al brazo izquierdo del robot).

Para el agarre inicial se debe tener en cuenta la posición de la botella, que es enviada desde el *Kinect* para poder agarrar de manera óptima la botella y posteriormente llevarla al punto en el que se le quita la tapa. Este proceso es detallado posteriormente en el **Capítulo 5**.

La secuencia completa se puede apreciar en la figura 4.4, donde se muestran paso a paso los movimientos del robot para realizar el destapado de la botella.



(a) Posición inicial del robot (b) Apertura del *gripper* de tres dedos (c) Posición del brazo derecho para coger la botella

(d) Posición del brazo derecho para quitar la tapa (e) Posición del brazo izquierdo antes de ir a quitar la tapa (f) Posición del brazo izquierdo para quitar la tapa

(g) Posición del brazo izquierdo después de quitar la tapa (h) Posición de los brazos para dejar la botella y la tapa (i) Los brazos vuelven a la posición inicial

Figura 4.4: Secuencia para destapar la botella

En la **Fig.4.4(a)** se tiene la posición inicial de los brazos, posteriormente, como se observa en la figura 4.4(b) se hace la apertura del *gripper* de tres dedos, con el fin de acercarse a la botella para agarrarla, como se muestra en la figura 4.4(c); a continuación, se lleva la botella al centro de la mesa, como se puede observar en la figura 4.4(d) para que posteriormente llegue el brazo izquierdo por la parte superior de la botella como se ve en la figura 4.4(e), con el fin de acercarse a la tapa, lo que se evidencia en la figura 4.4(f). En esta posición, el *gripper* de dos dedos debe apretar, girar 360° en sentido anti-horario y como esta articulación

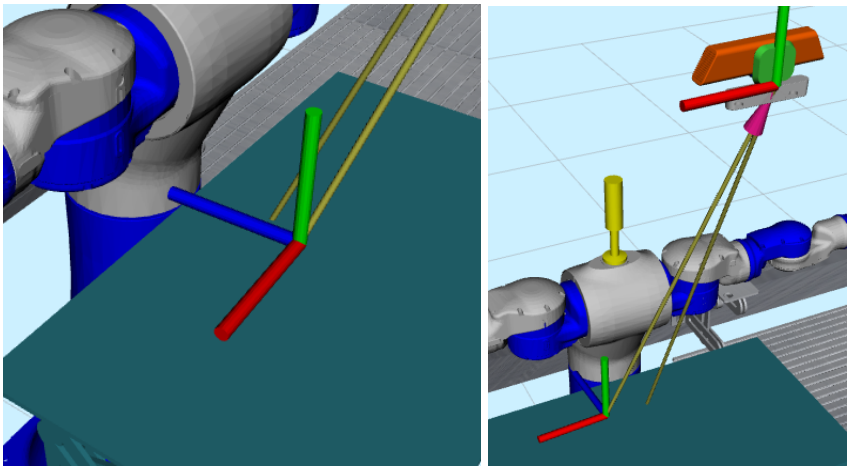
no puede girar indefinidamente, debe soltar y regresar 360° para poder dar otro giro en sentido anti-horario y culminar la extracción de la tapa. Posteriormente el brazo izquierdo sube con la tapa como se ve en la figura 4.4(g). Como penúltimo paso los brazos llevan la botella y la tapa a botarlos en un recipiente (ver figura 4.4(h); y por último los brazos vuelven a la posición inicial.

4.3 *Frames* y *TF*

De acuerdo con la secuencia mostrada anteriormente, es importante tener en cuenta la percepción del entorno basado en el punto de referencia del robot, el cual es llamado *"Torso_Base_Link"*. La importancia de lo anterior radica en el momento en que el *Kinect* envía la posición de la botella, para la cual se debe hacer una transformación y construir el *frame* de dicha posición. A continuación, se explican dichos conceptos.

Los *frames* son sistemas de ejes "X, Y, Z" que indican la posición y orientación de un objeto con respecto a la posición de otro objeto, por ejemplo, en nuestro caso, la posición de la botella con respecto al *kinect*.

Los *frames* son importantes a la hora de verificar qué objetos se encuentran en el entorno. Es por esto que a la hora de la detección de la botella por parte del *Kinect*, se debe generar en simulación un "punto" en la misma posición que detecta dicho dispositivo, esto, con el fin de añadirlo al árbol de TF's, y posteriormente realizar la conversión de este punto que está con respecto al *Kinect* para que tenga como referencia la base del robot, el cual es el punto de referencia de todos los *links* y *joints* del robot. En la figura 4.5 se observan: el *frame* que se genera para una botella ubicada en algún punto del espacio (ver figura 4.5(a) y el tf, el cual es la flecha que indica a que objeto del entorno está asociado, en este caso al *Kinect* (ver figura 4.5(b).



(a) *Frame* (b) TF

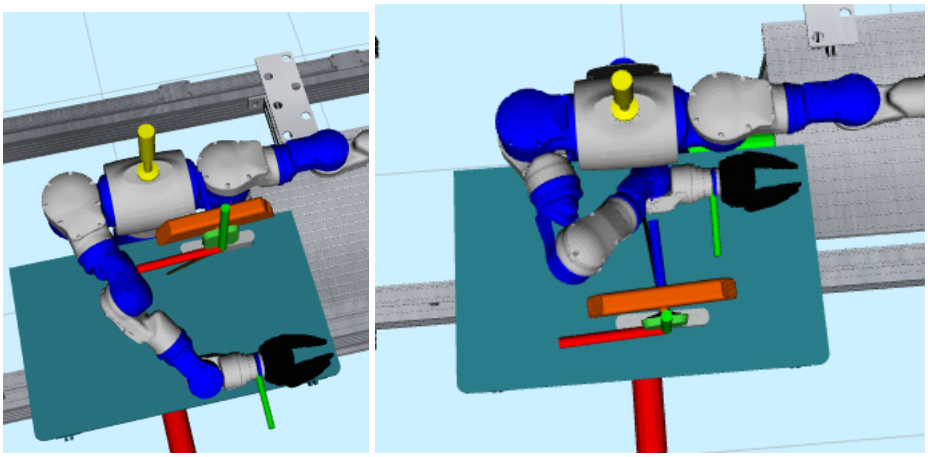
Figura 4.5: *Frame* y TF de un punto en el espacio con respecto al Kinect

Para la generación de dicho *frame* en simulación, se programa un *broadcaster* [23], cuyo código se muestra en el **Capítulo 7** en la **sección 7.4.1**.

4.4 Protocolo de pruebas

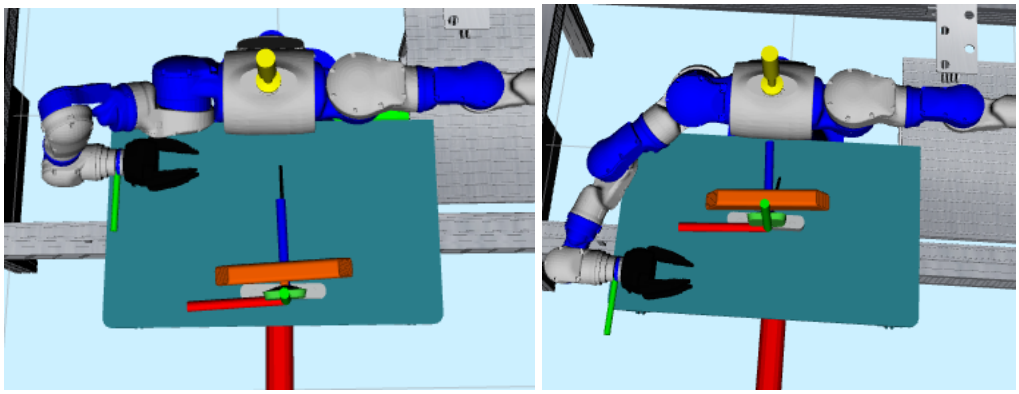
- Verificación de cubrimiento de todos los puntos de la mesa por parte del brazo derecho del robot: En este paso se decidió probar que el brazo derecho pudiera llegar a los límites de la mesa, encontrando que en la parte derecha de la mesa tiene un margen de aproximadamente 10cm donde no puede acceder de manera óptima para agarrar un objeto con la orientación que se tiene planeada. En la figura 4.6

se aprecian los límites que alcanza el robot sobre la mesa para lograr acceder a una botella que se encuentre dentro de ese rango.



(a) Brazo derecho en la esquina superior izquierda de la mesa

(b) Brazo derecho en la esquina inferior izquierda de la mesa



(c) Brazo derecho en la esquina inferior derecha de la mesa

(d) Brazo derecho en la esquina superior derecha de la mesa

Figura 4.6: Cobertura del espacio de la mesa por parte del brazo derecho del robot

4.5 Resultados obtenidos

De acuerdo con lo mostrado en la figura 4.6, se limita el espacio de trabajo por el costado derecho de la mesa, además, justo en frente del robot (parte inferior de la mesa) se pueden generar problemas porque las botellas posiblemente no quepan, debido a la forma del torso, por ende, se estima que el espacio de trabajo queda establecido de la siguiente forma: ancho = 72cm, largo = 40cm, aproximadamente.

5. Visión y manipulador

En este capítulo se explica el proceso realizado para entablar una comunicación exitosa entre el entorno de simulación del robot y el *Kinect*, además de la creación del servicio con el cual se realiza el envío de las coordenadas en el espacio tanto de la botella como la tapa que posee. Por último, se presenta el diseño del protocolo de pruebas así como los resultados obtenidos.

5.1 Conexión *Machine to Machine*

Para la conexión entre los dos computadores se debe establecer primero cual dispositivo va a ser el maestro y cual el esclavo, en donde, para este caso, se escogió que el maestro sea el que maneja el entorno de simulación del robot, mientras que el esclavo es el que realiza la detección de la tapa de la botella mediante el *Kinect*. Sabiendo esto, es importante tener en cuenta que los dos computadores deben encontrarse en la misma red de conexión para poder tener comunicación. Ahora bien, se hace necesario ingresar una serie de comandos desde consola, aclarando cuál PC es maestro y cuál no, mediante las direcciones IP asignadas a cada computador. Abajo se muestran los comandos ingresados en la terminal de cada dispositivo para cumplir con este procedimiento. Después de realizar esto, se comprueba la conexión haciendo un *ping* hacia la dirección IP del otro computador, corroborando que lleguen dichos paquetes de información.

- **En el pc maestro:**

  export ROS_HOSTNAME=A.B.C.D #Dirección IP de este PC

  export ROS_MASTER_URI=http://A.B.C.D:11311 #Se usa el puerto 11311 para este propósito.

  export ROS_IP=A.B.C.D #Dirección IP de este PC

- **En el pc esclavo:**

  export ROS_HOSTNAME=E.F.G.H #Dirección IP de este PC

  export ROS_MASTER_URI=http://A.B.C.D:11311 #Se usa el puerto 11311 para este propósito y se pone la dirección ip del PC maestro.

  export ROS_IP=E.F.G.H #Dirección IP de este PC

5.2 Servicio para la posición de las botellas

Dentro de una de las funcionalidades de ROS, está la comunicación mediante diferentes tipos de mensajes entre nodos: Tópicos, servicios, acciones y parámetros **[25]**. Evaluando las funcionalidades de cada uno de estos mensajes, se escogió el servicio, el cual maneja un relación cliente-servidor, donde el cliente (PC maestro) solicita información cuando lo requiera al servidor (PC esclavo), cuyo contenido para este caso es la posición del centro de la tapa y el punto central de la botella para poder realizar de manera correcta la rutina de agarre y destapado de la botella. En la figura 5.1 se ilustra mas claro este proceso de comunicación.

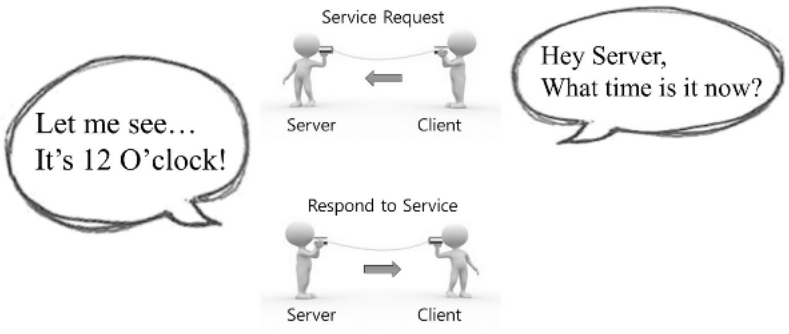
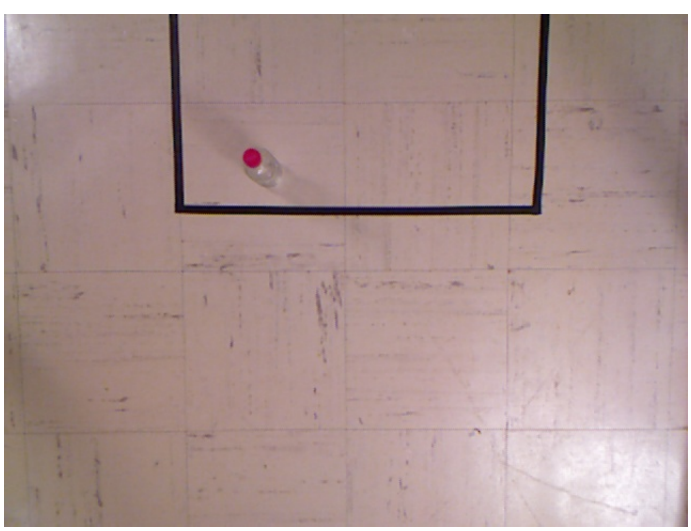


Figura 5.1: Imagen ilustrativa del proceso de comunicación mediante servicios [25].

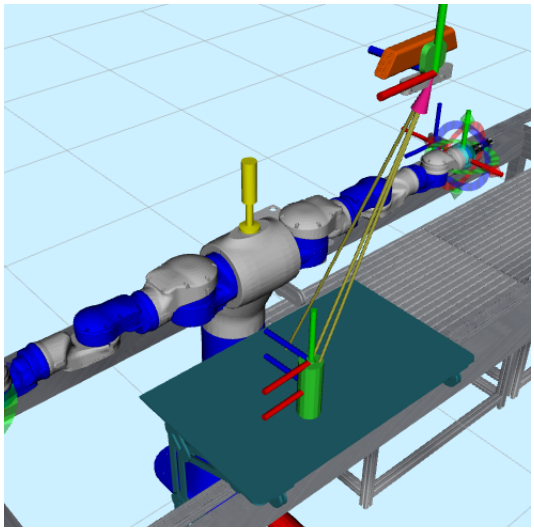
El servicio diseñado para la comunicación del kinect y la información de las coordenadas del mundo real con el robot manipulador y el entorno de simulación consta de 4 variables. Primero se tiene una variable de solicitud enviada por el cliente, en este caso el robot, con la cual se le informa al módulo de visión que realice el proceso de detección de la tapa de la botella y posteriormente el envío de dicha información. Como respuesta del servicio se tienen tres variables: Confirmación, PositionCap y Position Bottle. Confirmación es un mensaje de tipo entero con el que se le indica al robot que las coordenadas de la botella y la tapa se calcularon de manera correcta. PositionCap y PostitionBottle son mensajes de tipo *geometry_msgs* que envían la posición en una variable tipo Point (x,y,z), de esta forma el robot recibe estas posiciones para asignarlas a sus variables internas y poder realizar la planificación de la trayectoria.

5.3 Protocolo de pruebas

- Generación del *frame* (en simulación), de la botella detectada: Se procuró probar que la simulación detectara de forma correcta lo que ocurre en el mundo real; es decir, a la hora de detectar una botella, se generen los *frames* necesarios para poder disponer del árbol de TF's y así planificar satisfactoriamente toda la secuencia de destapado. En las figuras 5.2, 5.3, 5.4 y 5.5 se observan 4 pruebas particulares en las que se muestra la ubicación en el espacio real de una botella (diferente para cada prueba y ubicada en una posición diferente a las demás) y su respectivo *frame* generado en simulación, con la posición adecuada, además de generar un cilindro en simulación, emulando a la botella con la misma altura que arroja la medición del *Kinect*.

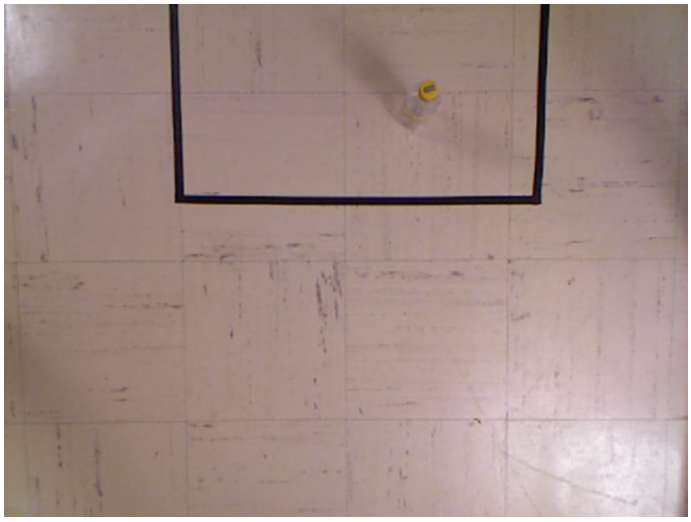


(a) Prueba 1: Botella real

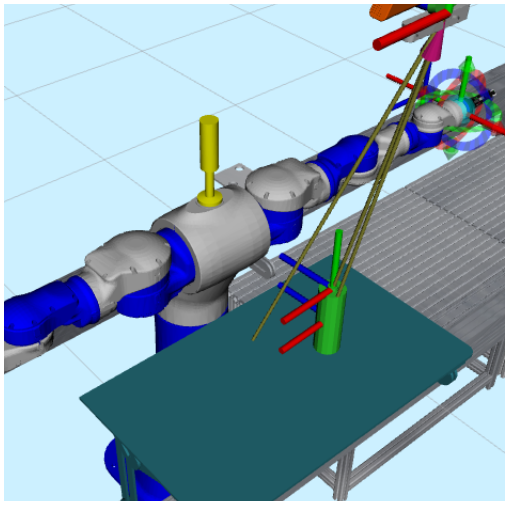


(b) Prueba 1: Simulación

Figura 5.2: Prueba 1: Botella y *frame*

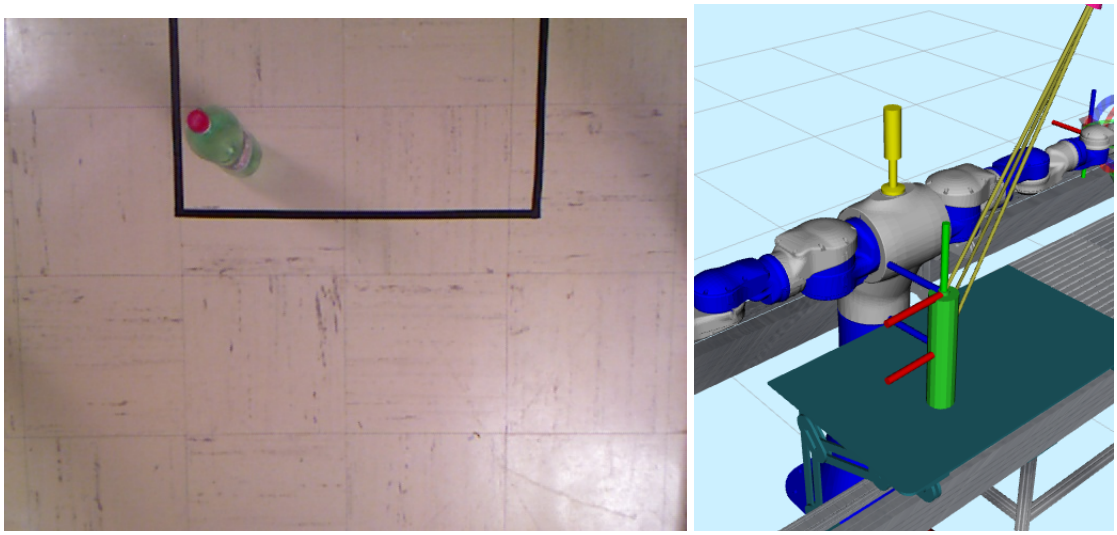


(a) Prueba 2: Botella real



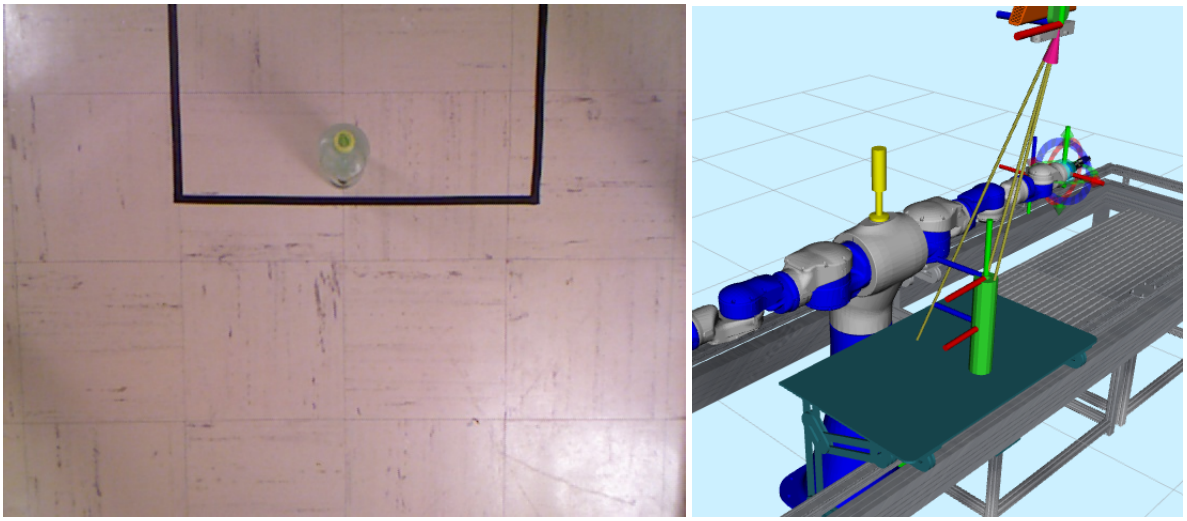
(b) Prueba 2: Simulación

Figura 5.3: Prueba 2: Botella y *frame*



(a) Prueba 3: Botella real (b) Prueba 3: Simulación

Figura 5.4: Prueba 3: Botella y *frame*



(a) Prueba 4: Botella real (b) Prueba 4: Simulación

Figura 5.5: Prueba 4: Botella y *frame*

En las anteriores figuras, se observan las imágenes RGB tomadas por el *Kinect* a la izquierda, allí se evidencia el campo de trabajo limitado por una cinta negra, indicando los máximos puntos a los que llegará el brazo robótico en la orientación requerida para el agarre de la botella (Esta delimitación se dió gracias a las pruebas realizadas en el **Capítulo 4**). En el costado derecho se ve la generación de los *frames* de las botellas en la ubicación que se tomó del entorno físico, logrando apreciar que se generan con una ubicación proporcional.

En el **Capítulo 7** en la **sección 7.2** se encuentran 9 imágenes adicionales relacionadas con estas pruebas, realizadas a otras 9 botellas diferentes en posiciones distintas.

- Duración de la secuencia de destapado: Es importante, de acuerdo con los objetivos planteados, saber el tiempo que se demora el robot en realizar una secuencia completa para destapar la botella. De acuerdo con diferentes posiciones de la botella, el robot en ocasiones puede demorarse más o menos, dependiendo de la trayectoria, para lo cual se realizó una prueba con varias posiciones y diferentes tamaños de la botella (16 posiciones), comprobando que en ninguna prueba se excediera de 2 minutos. En la tabla 5.1 se reporta dicha información, observando que para ninguna de las posiciones establecidas se excedió del tiempo límite. Por otra parte, la tabla da un estimado de los límites de la mesa para los cuales el robot alcanza a realizar correctamente la labor. Se debe tener en cuenta que las coordenadas "X" e "Y" en la tabla, están dadas con respecto a la base del robot y no con respecto al *Kinect*.

  En la tabla 5.2 se verifica la duración promedio (1 minuto con 45 segundos) de las secuencias, la máxima duración (1 minuto con 59 segundos) y la mínima duración (1 minuto con 38 segundos)

# secuencia	Posición botella (m)		Altura Botella (m)	Tiempo(s)
	x	y	z	
1	0.4	-0.3	0.2	109
2	0.4	-0.3	0.3	104
3	0.4	-0.3	0.15	108
4	0.35	-0.33	0.15	106
5	0.35	-0.33	0.2	105
6	0.4	0.2	0.3	98
7	0.4	0.4	0.3	99
8	0.7	0.4	0.3	104
9	0.7	0.4	0.2	119
10	0.4	-0.2	0.2	105
11	0.6	-0.2	0.2	105
12	0.6	-0.2	0.3	107
13	0.65	-0.2	0.3	106
14	0.65	-0.2	0.2	105
15	0.65	-0.3	0.2	108
16	0.65	-0.3	0.3	107

Tabla 5.1: Duración de la secuencia

# de secuencias	Duración promedio (s)	Máxima duración (s)	Mínima duración (s)
16	105.93	119	98

Tabla 5.2: Resumen duración de secuencias

6. Conclusiones y recomendaciones

6.1 Conclusiones

- De acuerdo con los resultados obtenidos para el protocolo de pruebas del capitulo de Visión, se cumple con el objetivo de identificar la posición de una o varias botellas con una precisión de al meno 90 %. Como consecuencia a lo anterior, la detección de la tapa hecha por segmentación de color es exitosa, ademas la estrategia para calcular la posición de la botella con base a la posición de la tapa funciona de manera correcta

- El algoritmo de alto nivel diseñado para el control de los brazos del robot funciona correctamente, basados en los resultados obtenidos para el protocolo de pruebas del capitulo *Robot manipulador* se logra realizar actividades de manera cooperativa con las dos extremidades del robot.

- La secuencia para destapar una botella se ajusta a los objetivos, realizándose en un tiempo menor a 2 minutos esto comprobado gracias al protocolo de pruebas diseñado para dicha labor

- Se realizó una comunicación exitosa entre los diferentes nodos del sistema, logrando una integración efectiva y por ende un funcionamiento adecuado para la extracción de la tapa.

- El protocolo de pruebas permite comprobar el funcionamiento del sistema en simulación, evidenciando cada una de sus fases y la integración completa del mismo.

6.2 Recomendaciones

- Para futuros proyectos con el robot es indispensable empezar a revisar todo el tema de ROS con bastante antelación debido a que es un mundo muy amplio de aprendizaje, razón por la cual puede retrasar un poco las tareas planeadas para un proyecto de este tipo.

- Para proyectos donde se plantee necesariamente la implementación sobre el robot real, sería muy útil ofrecer algún tipo de inducción a los estudiantes que vayan a realizar el proyecto, sobre el manejo físico del mismo, con esto se puede disponer de un mejor tiempo para trabajar con el autómata.

- Sería conveniente que se pudiera implementar en el robot real esta solución, aunque se pueden abordar otras problemáticas con un desempeño similar al mostrado en este documento por parte del robot.

Bibliografía

[1] S. LAVILLE AND M. TAYLOR, *El mundo compra un millón de botellas de plástico por minuto que acaban en vertederos o en el mar*, 2017, [ONLINE] Available: https://www.eldiario.es/theguardian/compra-botellas-plastico-mayoria-vertederos_0_659684375.html [Accessed: 05-Mar-2019].

[2] BBC MUNDO, *"Hay tantos residuos de plástico en el mundo que podrían cubrir un país como Argentina": la advertencia de un grupo científicos sobre la contaminación que acecha a nuestro planeta*, 2017, [ONLINE] Available: https://www.bbc.com/mundo/noticias-40664725 [Accessed: 05-Mar-2019].

[3] J. STASS, *How are plastic bottles recycled – How It Works, How are plastic bottles recycled?*, [ONLINE] Available: https://www.howitworksdaily.com/how-are-plastic-bottles-recycled/. [Accessed: 05-Mar-2019].

[4] I. JUSTE, *"El proceso de reciclaje de una botella de plástico - te lo explicamos," El proceso de reciclaje de una botella de plástico*, 2018. [ONLINE]. Available: https://www.ecologiaverde.com/el-proceso-de-reciclaje-de-una-botella-de-plastico-152.html. [Accessed: 05-Mar-2019].

[5] NISMRC, *OpenNI, el driver Open Source oficial de Kinect*, [ONLINE] Disponible: https://www.muylinux.com/2010/12/17/openni-el-driver-open-source-oficial-de-kinect/. [Accessed: 18-Nov-2019].

[6] ROS.ORG, *Converting between ROS images and OpenCV images (C++)*, [ONLINE] Disponible: http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages. [Accessed: 18-Nov-2019].

[7] *Modelo de color HSV*, [ONLINE] Disponible: http://recursos.normalpopayan.edu.co:8983/wikipedia_es_all_2017-08/A/HSV.html. [Accessed: 18-Nov-2019].

[8] OPENCV, *Miscellaneous Image Transformations*, [ONLINE] Disponible: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html. [Accessed: 18-Nov-2019].

[9] OPENCV,*Operations on Arrays*, [ONLINE] Dispoible: https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html. [Accessed: 18-Nov-2019].

[10] *Detección de contronos en OpenCv Python*, [ONLINE] Disponible: http://acodigo.blogspot.com/2017/08/deteccion-de-contornos-con-opencv-python.html. [Accessed: 18-Nov-2019].

[11] OPENCV, *Morphological Transformations* , [ONLINE] Disponible: https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html. [Accessed: 18-Nov-2019]

[12] F. CALDERÓN AND J.FLÓREZ, *Operaciones morfológicas*, Bogotá, pp. 2-6

[13] OPENCV,*Structural Analysis and Shape Descriptors*, [ONLINE] Disponible: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours. [Accessed: 18-Nov-2019].

[14] GL4R3,*Buscar el centro de un contorno con OpenCV python*, [ONLINE] Disponible: https://robologs.net/2019/06/24/buscar-el-centro-de-un-contorno-con-opencv-python/. [Accessed: 18-Nov-2019].

[15] WIKIPEDIA, *Pinhole camera model*, [ONLINE] Disponible: https://en.wikipedia.org/wiki/Pinhole_camera_model. [Accessed: 18-Nov-2019].

[16] OPENCV, *Camera Calibration and 3D Reconstruction*, [ONLINE] Disponible: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. [Accessed: 18-Nov-2019].

[17] FDXLABS,*Calculate X, Y, Z Real World Coordinates from Image Coordinates using OpenCV* , [ONLINE] Disponible: https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/. [Accessed: 18-Nov-2019].

[18] RESEARCHGATE, *Robot SDA10F*, [ONLINE] Disponible: https://www.researchgate.net/figure/Robot-SDA10F-Image-taken-from-7_fig1_315063645.[Accessed: 25-Jul-2019].

[19] ROS, *What is ROS?*, [ONLINE] Disponible: http://wiki.ros.org/ROS/Introduction. [Accessed: 5-Ago-2019].

[20] OMPL, *The Open Motion Planning Library*, [ONLINE] Disponible: https://ompl.kavrakilab.org/. [Accessed: 28-Ago-2019].

[21] OMPL, *Geometric Planners*, [ONLINE] Disponible: https://ompl.kavrakilab.org/planners.html#geometric_planners. [Accessed: 28-Ago-2019].

[22] OMPL, *Control-Based Planners*, [ONLINE] Disponible: https://ompl.kavrakilab.org/planners.html#control_planners. [Accessed: 28-Ago-2019].

[23] ROS.ORG, *Writing a tf broadcaster (C++)*, [ONLINE] Disponible: http://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20broadcaster%20%28C%2B%2B%29. [Accessed: 05-OCT-2019].

[24] MOVEIT, *MoveIt Setup Assistant*, [ONLINE] Disponible: http://docs.ros.org/melodic/api/moveit_tutorials/html/doc/kinematics_configuration/kinematics_configuration_tutorial.html. [Accessed: 10-Ago-2019].

[25] YOONSEOK PYO, HANCHEOL CHO, RYUWOON JUNG AND TAEHOON LIM, *ROS Robot Programming: From the basic concept to practical programming and robot application*, Republic of Korea, 2017.

7. Anexos

A continuación encontrará un código QR y un link para acceder de cualquier forma a la carpeta general que contiene los anexos de este libro.

Figura 7.1: Codigo QR para acceder a la carpeta de anexos

Link carpeta de anexos: `https://www.dropbox.com/sh/c84zdjkbmigm0h0/AADacZVntU9Pui_PhfgVjfI4a?dl=0`

Posteriormente, en las secciones de este capítulo, encontrará el link específico de cada anexo para evitar la búsqueda de cada uno en la carpeta general.

7.1 Anexos Capitulo 3 *Visión*

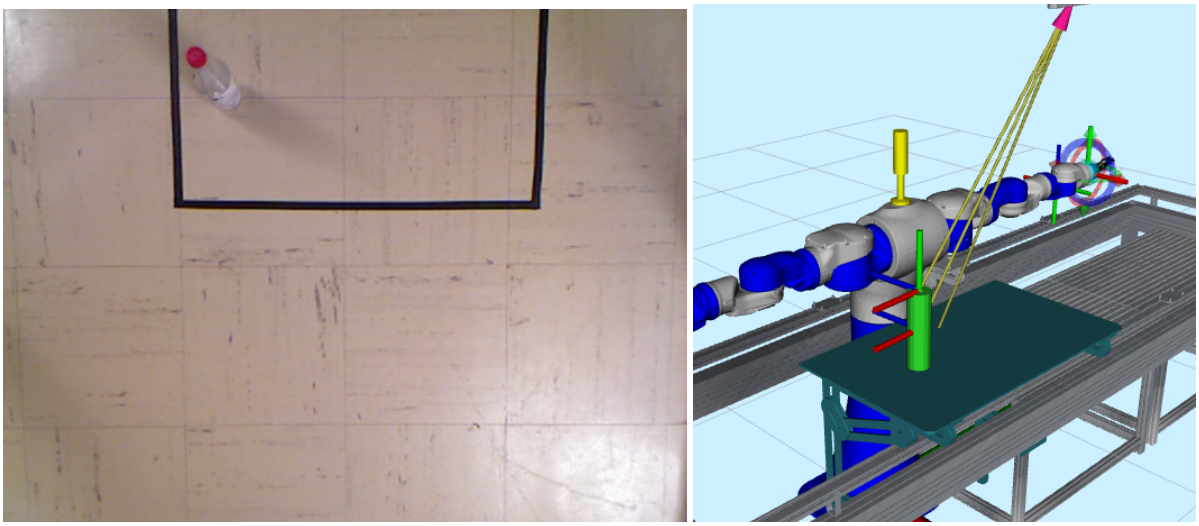
Los anexos para el capitulo de *Visión* se pueden encontrar en la nube leyendo el código QR de la **Fig.7.1** o ingresando al link correspondiente que se tiene a continuación.

Link: `https://www.dropbox.com/sh/6ys0x28o6w5jxg2/AACFcq6bXNrLEDBGoBM7TmZGa?dl=0`.

En este link se encuentran los registros fotográficos de las botellas utilizadas en las pruebas, las diferentes pruebas realizadas y las tablas de datos con la información recolectada en el protocolo de pruebas.

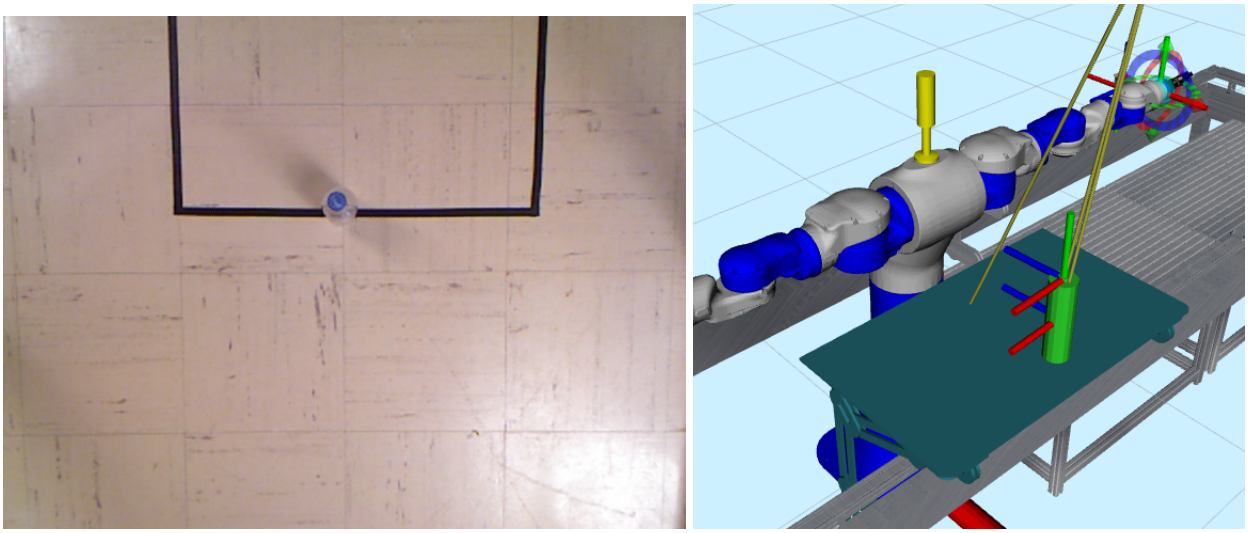
7.2 Protocolo de pruebas: *frames* y tf's

A continuación se presentan 9 imágenes adicionales como soporte realizado a las pruebas descritas en la **sección 5.3.**



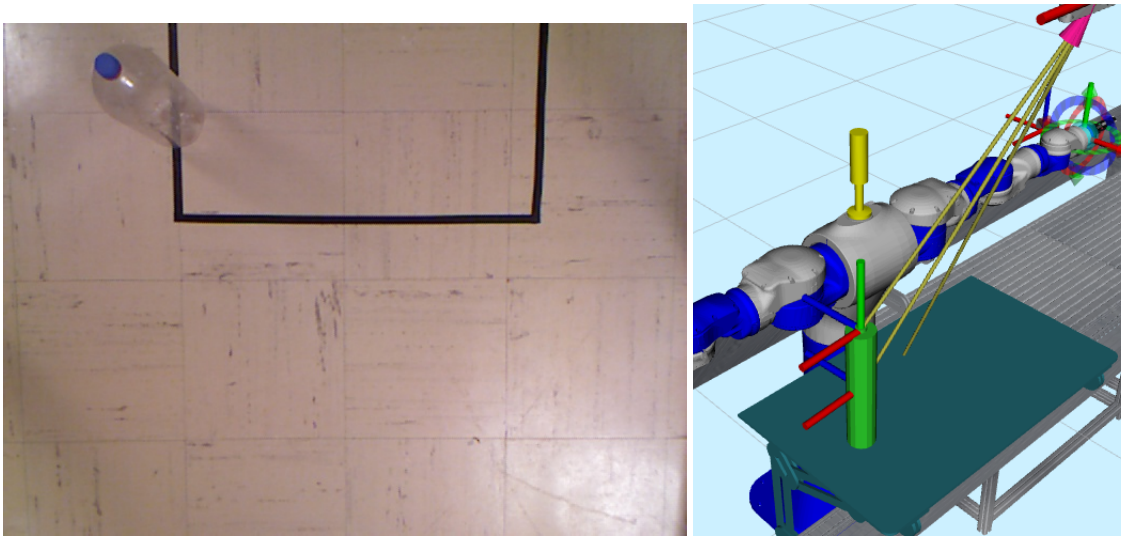
(a) Prueba 5: Botella real (b) Prueba 5: Simulación

Figura 7.2: Prueba 5: Botella y *frame*



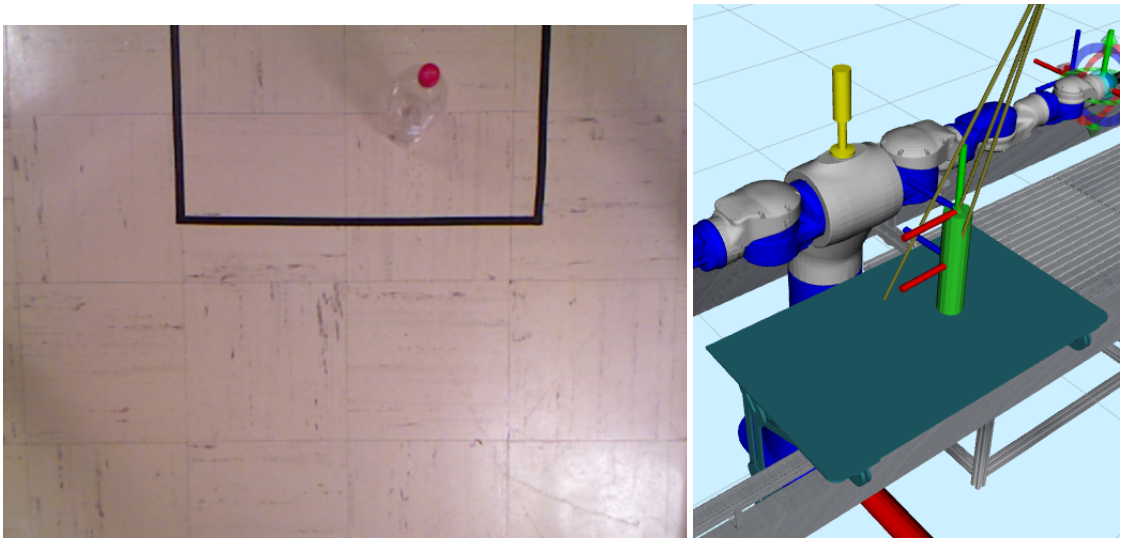
(a) Prueba 6: Botella real (b) Prueba 6: Simulación

Figura 7.3: Prueba 6: Botella y *frame*



(a) Prueba 7: Botella real (b) Prueba 7: Simulación

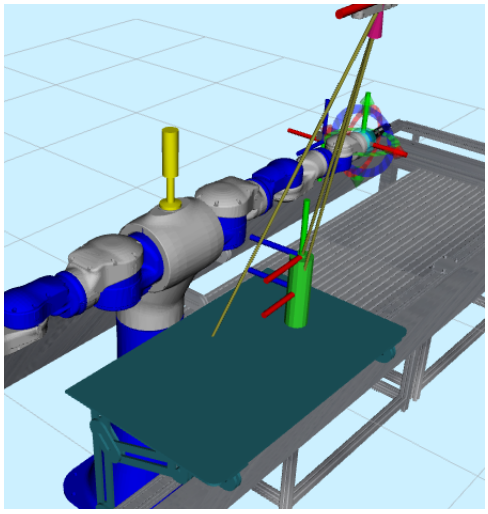
Figura 7.4: Prueba 7: Botella y *frame*



(a) Prueba 8: Botella real (b) Prueba 8: Simulación

Figura 7.5: Prueba 8: Botella y *frame*

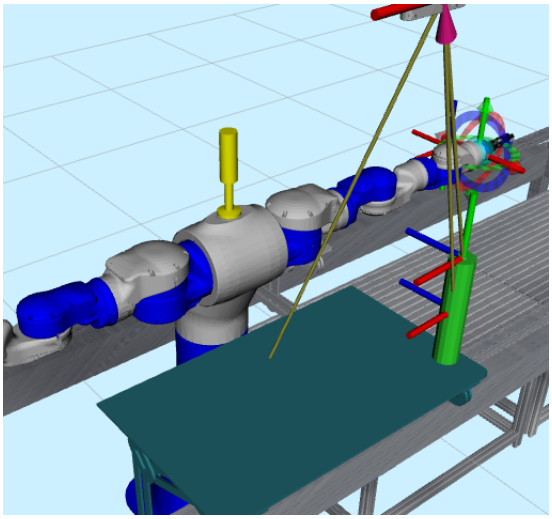
(a) Prueba 9: Botella real



(b) Prueba 9: Simulación

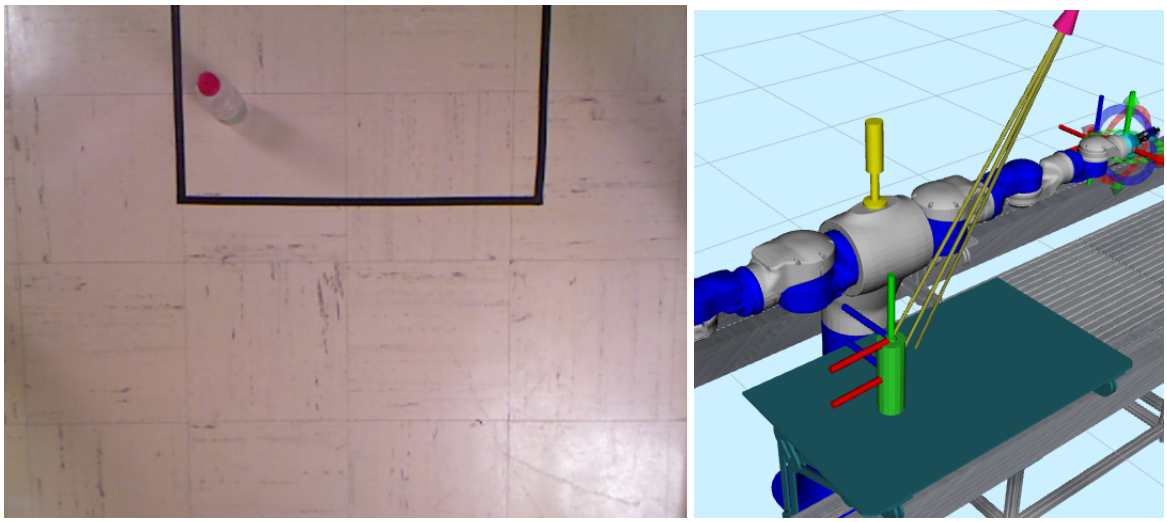
Figura 7.6: Prueba 9: Botella y *frame*

(a) Prueba 10: Botella real



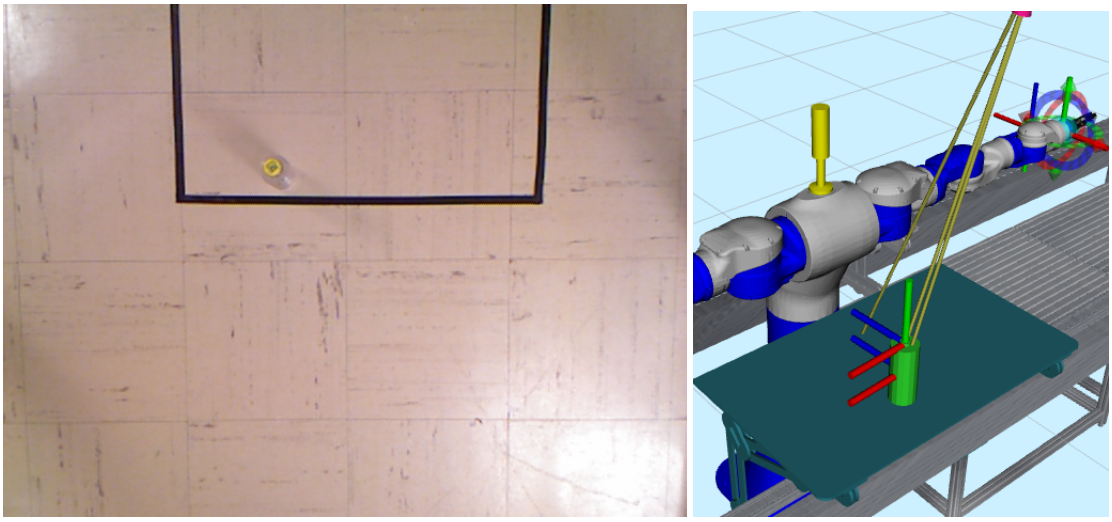
(b) Prueba 10: Simulación

Figura 7.7: Prueba 10: Botella y *frame*



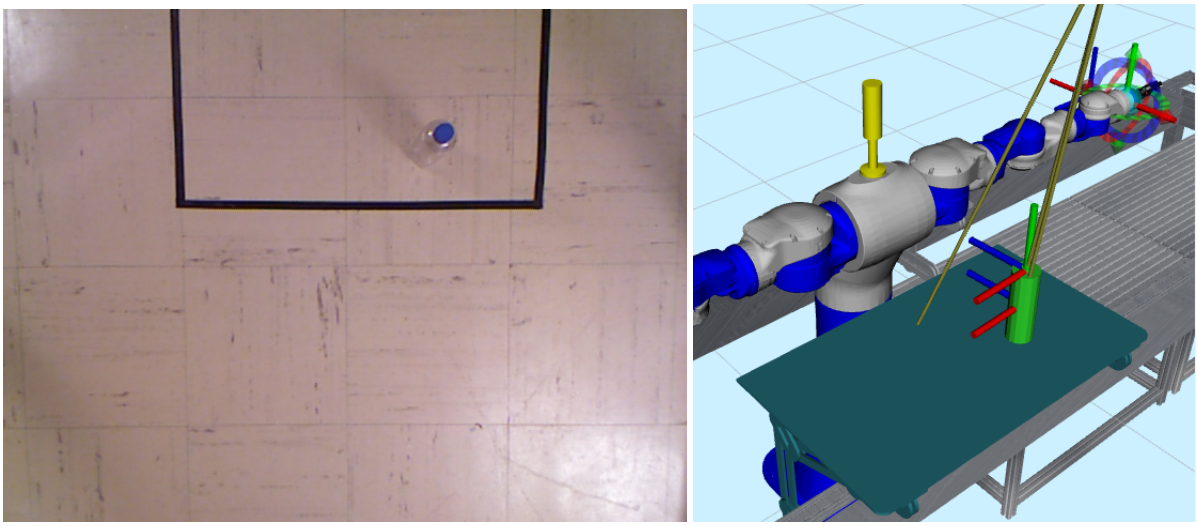
(a) Prueba 11: Botella real (b) Prueba 11: Simulación

Figura 7.8: Prueba 11: Botella y *frame*



(a) Prueba 12: Botella real (b) Prueba 12: Simulación

Figura 7.9: Prueba 12: Botella y *frame*



(a) Prueba 13: Botella real (b) Prueba 13: Simulación

Figura 7.10: Prueba 13: Botella y *frame*

7.3 IK solver

En el link que se tiene a continuación, se tiene la configuración del archivo **kinematics.yaml** generado para establecer el *IK solver* del robot.

Link: `https://www.dropbox.com/sh/ps9ddhgd9unolge/AADkGet08q86cKTos1X-GXffa?dl=0`

7.4 Códigos realizados

7.4.1 Cliente y *Broadcaster*

En el siguiente link se encuentra el código realizado para realizar el cliente necesario para la comunicación con el otro PC y el broadcaster necesario para la generación del *frame* de la botella en simulación.

Link: `https://www.dropbox.com/sh/0e9dtefg1z77yhq/AABzUthHed1bCuCS1KtmuORxa?dl=0`

7.4.2 *Listener, collision objects* y secuencia

En el siguiente link se encuentra el código realizado para realizar el listener necesario para obtener la posición de la botella con respecto a la base del robot, además de se agrega el cilindro como un *Collision Object* emulando la botella de la vida real y se genera toda la secuencia realizada por el robot para destapar la botella.

Link: `https://www.dropbox.com/sh/0b9w5jwfgg91quc/AACRPaGmwVJ73Z7xLFGzT6cQa?dl=0`

7.4.3 Estructura del servicio para la detección de las tapas "PoseCam.srv"

En el siguiente link se tiene el archivo *PoseCam.srv* necesario para la comunicación entre el cliente y el servidor.

Link: `https://www.dropbox.com/sh/tl8fx4stithftva/AAD_Ofc-f4wvj8oRNJZ-hx4la?dl=0`

7.4.4 Servicio para la detección de las tapas de las botellas

El código del servicio para realizar la detección de la tapa, calcular su posición y enviar la información al entorno de simulación se encuentra en el archivo *pose_cam_server.cpp*, el link de este archivo es el siguiente: `https://www.dropbox.com/s/n1769ql9ozedwku/pose_cam_server.cpp?dl=0`