SISTEMA BASADO EN APRENDIZAJE PROFUNDO PARA EVADIR OBSTÁCULOS ESTÁTICOS CON UN ROBOT FIJO DE 6 DOF

AUTOR

JULIÁN ESTEBAN HERRERA BENAVIDES

Trabajo de grado presentado como requisito para optar al título de:

MAGISTER EN INGENIERÍA MECATRÓNICA

Director:

ROBINSON JIMÉNEZ MORENO

Codirector:

JORGE ALEXANDER APONTE RODRÍGUEZ

UNIVERSIDAD MILITAR NUEVA GRANADA

FACULTAD DE INGENIERÍA

MAESTRÍA EN INGENIERÍA MECATRÓNICA

BOGOTÁ, 16 OCTUBRE 2020

NOTA DE ADVERTENCIA

"La universidad no se hace responsable de los conceptos emitidos por sus estudiantes en sus proyectos de trabajo de grado, soló velará por la calidad académica de los mismos, en procura de garantizar su desarrollo de acuerdo a la actualidad del área disciplinar respectiva. En el caso de presentarse cualquier reclamación o acción por parte de un tercero en cuanto a los derechos de autor sobre la obra en cuestión, el estudiante – autor asumirá toda la responsabilidad y saldrá en defensa de los derechos. Para todos los derechos la universidad actúa como un tercero de buena fe". (Ley 23 de 1982, Ley 44 de 1993, Decisión Andina 351 de 1993, Decreto 460 de 1995)

AGRADECIMIENTOS

En primer lugar, agradezco a Dios y a mi madre por haberme dado la oportunidad de formarme intelectualmente. Gracias a su constante apoyo he logrado ver la importancia de apreciar la vida, el amor y el respeto en cualquier actividad que esté desarrollando.

Agradezco a mi pareja, Sara lucía Aceros Cardozo, quien con su apoyo incondicional me dio las fuerzas para culminar esta etapa. Gracias a sus consejos fue posible encontrar el método con el cual se desarrolló el algoritmo de planeación de trayectoria del robot.

Agradezco a mis tutores, los cuales con paciencia y dedicación ayudaron a desarrollar este proyecto. A partir de sus consejos guiaron mis ideas, permitiendo que lograra solucionar cada problema encontrado.

Agradezco a los ingenieros Javier Orlando Pinzón Arenas y César Giovany pachón Suescún por el tiempo que dedicaron para introducirme en el campo del Deep Learning. En especial deseo agradecer a César Pachón por haberme guiado en el estudio de gran parte de las herramientas usadas en este trabajo. Sus consejos basados en argumentos me ayudaron a escoger técnicas, programas y entornos en los cuales trabajar.

Por último, agradezco a la Universidad Militar Nueva Granada y a la maestría en ingeniería mecatrónica, quienes brindaron los docentes y los espacios requeridos para crecer intelectualmente y desarrollar este trabajo.

.

Tabla de Contenido

Pág.

LISTA DE TABLAS V

LISTA DE FIGURAS VI

LISTA DE ABREVIATURAS VIII

RESUMEN IX

1 INTRODUCCIÓN 1

1.1 FORMULACIÓN Y PLANTEAMIENTO DEL PROBLEMA 2
1.2 JUSTIFICACIÓN DEL PROYECTO 2
1.3 OBJETIVOS 3
1.4 HIPÓTESIS 4
1.5 PREGUNTAS DE INVESTIGACIÓN 4
1.6 DELIMITACIONES 4
1.7 METODOLOGÍA 5
1.8 ORGANIZACIÓN DEL DOCUMENTO 6

2 ESTADO DEL ARTE 7

3 MARCO TEÓRICO 12

4 MATERIALES Y TEMAS DE ESTUDIO 17

5 DETECCIÓN Y SEGMENTACIÓN SEMÁNTICA 19

5.1 SELECCIÓN DE ARQUITECTURA 19
5.2 MASK R-CNN 22
5.3 BASE DE DATOS Y ENTRENAMIENTO 24

6 RECONSTRUCCIÓN 3D Y NUBE DE PUNTOS 28

6.1 MODELO PINHOLE Y PROYECCIÓN DE PERSPECTIVA 28
6.2 MÉTODOS DE CALIBRACIÓN DE PARÁMETROS INTRÍNSECOS Y EXTRÍNSECOS 30
6.3 CALIBRACIÓN Y GENERACIÓN DE NUBE DE PUNTOS 34

7 MODELADO DEL ENTORNO OPERATIVO, CÁLCULOS CINEMÁTICOS Y ANÁLISIS DE COLISIONES 37

7.1 MATRICES HOMOGÉNEAS Y CINEMÁTICA DIRECTA 37
7.2 FILTRADO DE LA NUBE DE PUNTOS 39
7.3 ANÁLISIS DE COLISIONES 40
7.4 MÉTODOS PARA CALCULAR LA CINEMÁTICA INVERSA 42

8 DISCRETIZACIÓN, GENERACIÓN DE GRAFOS Y PLANEACIÓN DE MOVIMIENTO 48

8.1 DISCRETIZACIÓN DEL ESPACIO DE CONFIGURACIÓN Y CREACIÓN DE GRAFOS 48
8.2 PLANEACIÓN DE TRAYECTORIA MEDIANTE ALGORITMO DIJKSTRA 55
8.3 SISTEMA DE SEGURIDAD Y PRUEBAS DE FUNCIONAMIENTO 58

9 CONCLUSIONES 67

10 REFERENCIAS 69

Lista de Tablas

Tabla 1 Comparación de desempeño entre la red YOLO y la red Faster R-CNN(Redmon & Farhadi, 2018) 21
Tabla 2 Distribución de imágenes y objetos dentro de las bases de datos 25
Tabla 3 Puntos medidos respecto al sistema inercial 35
Tabla 4 Puntos medidos respecto al plano de la imagen 35
Tabla 5 Resultados obtenidos con el método DLT 35
Tabla 6 Errores al hallar las matrices homogéneas con el método propuesto 36
Tabla 7 Parámetros DH usados para el robot IRB4600 46
Tabla 8 Norma de la ecuación (61) al calcular la IK mediante ambos métodos para cinco matrices diferentes 47
Tabla 9 Parámetros para definir diferentes prioridades de evasión 54
Tabla 10 Características del computador sobre el cual se realizan las pruebas 58
Tabla 11 Descripción de los módulos del sistema de seguridad 59

Lista de Figuras

Figura 1 Ventas de robots entre 2006 y 2017 (Industrial robot sales increase worldwide by 31 percent, n.d.).....3
Figura 2 Método implementado para desarrollar el proyecto. 5
Figura 3 Organización del documento. 6
Figura 4 Actuadores rígidos, SCA y SEA (Lauzier & Gosselin, 2011) 8
Figura 5 Evasión de obstáculos modificando la velocidad deseada en el gripper (Mohammed et al., 2017). 9
Figura 6 Robot planar de dos grados de libertad (Reyes Cortes, 2012) 13
Figura 7 Vectores para calcular aportación de la tercera articulación en el jacobiano de un Robot planar de cinco grados de libertad. 14
Figura 8 grafo no direccionado (Pearson & Bryant, 2004) 15
Figura 9 Arquitectura para una red neuronal convolucional (Redes neuronales convolucionales. n.d.). .16
Figura 10 Software implementado. 17
Figura 11 Entorno creado en CoppeliaSim. 18
Figura 12 Temas de estudio.....18
Figura 13 Lenet-5(LeCun et al., 1998) 19
Figura 14 a) Conexión residual (He et al., 2016) b) bloque Inception (Szegedy et al., 2015). 20
Figura 15 a) Localización de personas, b) Detección de personas. 21
Figura 16 Arquitectura de la red Mask R-CNN, la cual permite generar clasificación, detección y segmentación de objetos.....22
Figura 17 Escalamiento de la imagen y Anchors generados a cada celda.....23
Figura 18 Ejemplo de polígonos usados para crear la base de datos de los operarios y los objetos de la banda.....24
Figura 19 a) Entrenamiento de la red para detección y segmentación de personas b) Entrenamiento de la red para detección y segmentación de objetos.....26
Figura 20 Resultados que se obtienen con las redes Mask R-CNN entrenadas.....27
Figura 21 Datos generados por la cámara RGB-D a) Información RGB proveniente de los tres primeros canales b) Mapa de profundidad proveniente del último canal.....28
Figura 22 Modelo PinHole para describir la proyección de las superficies sobre el plano de la imagen. .29
Figura 23 Sistemas coordenados de interés relacionados mediante matrices homogéneas.....32
Figura 24 Creación de la nube de puntos. a) Imagen capturada del entorno. b) Transformación de toda la imagen a nube de puntos. c) Transformación de los pixeles segmentados a nube de puntos.....36
Figura 25 Prismas rectangulares agregados a cada articulación del robot. 38
Figura 26 Sistema coordenado en objeto de la banda. 39
Figura 27 Filtrado de la nube de puntos.....40
Figura 28 Colisión interna y externa en el robot. 40
Figura 29 Vectores usados en el teorema de los ejes separados. 41
Figura 30 Norma de la ecuación (61) al usar los métodos explicados. Se tiene en cuenta error de posición y orientación.....46
Figura 31 Discretización del rango de operación de una articulación.....49
Figura 32 Discretización del espacio de configuración con $m_0, m_1, m_2 = 4$. Matriz de discretización. 50
Figura 33 Representación de grafo a partir de un arreglo. 51
Figura 34 Pseudocódigo para evaluar el costo entre nodos adyacentes. 52
Figura 35 Pseudocódigo de funciones para relacionar nodos, posiciones en la matriz de discretización y configuraciones del robot.....53
Figura 36 Procedimiento para hallar el costo que se asigna a una celda de la matriz de discretización. ..55
Figura 37 Pseudocódigo del método Dijkstra.....56
Figura 38 Configuración escogida en el robot para solucionar el grafo.....57
Figura 39 Prueba de planeación de movimiento a una posición arbitraria. 57
Figura 40 Tiempo de procesamiento requerido para la creación y solución de grafos. 58
Figura 41 Sistema de seguridad. 60
Figura 42 Desplazamiento del robot para sujetar objeto en la banda transportadora. 61

Figura 43 Planeación de trayectoria calculada para depositar objeto clasificado como “Tim310”.....61
Figura 44 Planeación de trayectoria calculada para depositar objeto clasificados como “Belodyne”. 61
Figura 45 Planeación de trayectoria calculada para depositar objeto clasificados como “Cámara”. 62
Figura 46 Planeación de trayectoria calculada para depositar objeto clasificados como “Rueda”. 62
Figura 47 Planeación de trayectoria, prueba 5. 63
Figura 48 Planeación de trayectoria, prueba 6. 63
Figura 49 Planeación de trayectoria, prueba 7. 63
Figura 50 Planeación de trayectoria, prueba 8. 64
Figura 51 Planeación de trayectoria, prueba 9. 64
Figura 52 Planeación de movimiento con prioridad mínima. 65
Figura 53 Planeación de movimiento con prioridad media. 65
Figura 54 Planeación de movimiento con prioridad máxima.....65
Figura 55 Distancia mínima entre el robot y el operario para la planeación de movimiento de la Figura 52. 66
Figura 56 Distancia mínima entre el robot y el operario para la planeación de movimiento de la Figura 53. 66
Figura 57 Distancia mínima entre el robot y el operario para la planeación de movimiento de la Figura 54. 66

Lista de abreviaturas

HRC= colaboración hombre-máquina (Human Robot Collaboration)
CNN= red neuronal convolucional (Convolutional Neural Network).
ML= aprendizaje de máquina (Machine Learning)
FK= cinemática directa (Forward Kinematics)
IK= cinemática inversa (Inverse Kinematics)
$^{\mathbf{W}}\mathbf{T}_{\mathbf{E}}$ = matriz de transformación homogénea (transformación del sistema E al sistema W).
$^{\mathbf{E}}\mathbf{P}$ = vector referenciado respecto al sistema E.
$\mathbf{n}, \mathbf{o}, \mathbf{a}$ = vectores unitarios de la matriz de rotación
$\mathbf{t}$ = vector de traslación de la matriz de transformación homogénea
$\mathbf{n} \cdot \mathbf{o}$ = producto punto entre vectores
$\mathbf{n} \times \mathbf{o}$ = producto cruz entre vectores
$\mathbf{d}$ = vector de velocidad lineal.
$\boldsymbol{\omega}$ = vector de velocidad angular.
$\boldsymbol{w}$ = vector de velocidades, el cual contiene al vector de velocidad lineal y angular.
$\boldsymbol{q}$ = coordenadas generalizadas o vector de estado del sistema robótico.
$^{\mathbf{W}}\mathbf{T}_{\mathbf{E}}(\mathbf{q})$ = matriz de transformación homogénea que es función del vector de estado.
$\boldsymbol{J}$ =jacobiano.
$\boldsymbol{J}_L$= componente lineal de la matriz jacobiana
$\boldsymbol{J}_A$= componente angular de la matriz jacobiana
$\boldsymbol{J}^+$ =pseudoinversa de la matriz jacobiana.

Resumen

Este documento presenta el diseño de un sistema de seguridad que evita colisiones entre un robot antropomórfico de seis grados de libertad y los operarios u objetos que se encuentren presentes en el entorno. Mediante el sistema desarrollado se permite incluir manipuladores en entornos que requieran trabajos colaborativos hombre-máquina, en los cuales está el riesgo de presentarse colisiones que pueden ser perjudiciales tanto para el robot como para los operarios. La planeación de movimiento se calcula integrando módulos que realizan segmentación semántica, nube de puntos, cálculos cinemáticos, análisis de colisiones y solución de grafos; tareas que son explicadas a lo largo del documento. Los resultados obtenidos demuestran que es posible evadir con niveles de prioridad los obstáculos, lo que permite controlar la cercanía del robot respecto a estos y así disminuir la posibilidad de colisión. Se logra obtener planeaciones de movimiento robustas con grafos de 27000 nodos, los cuales se solucionan en promedio en 396 segundos, por lo que este enfoque es útil principalmente para entornos estáticos en donde no se requiere actualizar los grafos solucionados.

Palabras clave: planeación de movimiento, modelado del entorno, cinemática directa, aprendizaje profundo, grafo no dirigido, coordenadas generalizadas, espacio de configuración.

Abstract

This document presents the design of a security system that prevents collisions between an anthropomorphic robot with six degrees of freedom and operators or objects present in the environment. Through the developed system it is allowed to include manipulators in environments that require collaborative man-machine works, in which the hazard of collisions (risky for both the robot and the operator) is latent. The motion planning is calculated by integrating modules that perform semantic segmentation, point cloud, kinematic calculations, collision analysis and graph solution; tasks that are explained throughout the document. The results show that it is possible to evade obstacles with priority levels, this allows controlling the robot's proximity to the obstacles and thus reducing the probability of collision. Robust motion planning is achieved with 27,000-node charts, these are resolved in 396 seconds on average. Thus, this approach is useful primarily for static environments where you don't need to update the resolved charts.

Key words: Path planning, environment modeling, forward kinematics, Deep Learning, undirected graph, generalized coordinates, configuration space.

1 Introducción

La robótica es una rama de la ingeniería que tiene como función analizar, diseñar, construir y programar robots para ejecutar tareas, las cuales en su mayoría son repetitivas o peligrosas. Estos manipuladores se han ido integrando poco a poco en la sociedad con el fin de mejorar la ejecución de procesos, los cuales pueden estar relacionados con una infinidad de sectores que satisfacen las necesidades de una población (Jayaraj & Divakar, 2018; Roldán et al., 2018). El creciente uso de robots ha permitido no solo mejorar la rentabilidad y la calidad en las líneas de producción; sino que, adicionalmente han dado la posibilidad de mejorar la calidad de vida de los operarios. La capacidad de los robots para realizar actividades de manera repetitiva los hace imprescindibles hoy en día para la ejecución de un gran número de actividades que pueden estar enmarcadas en el sector industrial (Dahari & Tan, 2011), médico (Beasley, 2012) y militar (Neumann, Ferrein, Kallweit, & Scholl, 2014), entre otros.

A medida que la cantidad de robots ha ido aumentando se han empezado a analizar situaciones en donde se requiere una cercana interacción hombre-máquina como, por ejemplo, en líneas de producción reconfigurables para la producción de múltiples productos (Kock et al., 2011). Actualmente existen industrias que intentan reducir la brecha entre el hombre y la máquina con el fin de mejorar la eficiencia con la cual se ejecutan algunas tareas, sin embargo, esto aumenta el riesgo laboral ya que existe la posibilidad de producirse una colisión con el operario, por lo que en ambientes industriales se trata de evitar esta situación. En respuesta a lo anterior se ha empezado a desarrollar una generación de robots denominados COBOTS (Djuric, Urbanic, & Rickli, 2016; Veloso et al., 2012) , los cuales se han diseñado pensando en trabajos colaborativos con personas. Algunos de estos sistemas se les ha limitado la fuerza y velocidad con actuadores SEA (series elastic actuators) (Calanca & Fiorini, 2014) , los cuales pueden detectar y reducir torques excesivos que pueden afectar a las personas. Entre los robots más representativos de esta generación se puede encontrar a Sawyer de Rethinkrobotics; YuMi, de ABB; UR10, de Universal Robots y LBR, de KUKA.

Los robots colaborativos se encuentran equipados con dos tipos de sistemas de seguridad denominados “activos” y pasivos”, los cuales se encargan de evitar y disminuir el impacto de cualquier colisión que pueda experimentar el robot. Los sistemas activos usan sensores y cámaras que permiten predecir colisiones, dando la posibilidad al autómata de desviar las articulaciones, activar los frenos de seguridad o disminuir la energía de los actuadores antes del impacto. Por otro lado, los sistemas pasivos deben funcionar como respaldo del sistema activo detectando la existencia de una colisión para desactivar los actuadores y así disminuir el impacto sobre el operario. La presente investigación se encuentra orientada al desarrollo de un sistema de seguridad activo que le permita a un robot de 6 DOF evadir obstáculos que se encuentren en el entorno operativo, con el fin de darle la posibilidad de continuar con sus labores sin que se desactive al detectar la presencia de estos.

En el desarrollo de este trabajo se abordan temas como la cinemática directa (mediante matrices homogéneas (Reyes Cortes, 2012)), la cinemática inversa (mediante métodos iterativos (Buss, 2004; Cisneros, Jiménez, & Navarro, 2014)), la inteligencia artificial (redes basadas en convoluciones (Liang & Hu, 2015)), la reconstrucción 3D mediante nube de puntos y teoría de

grafos para planeación de movimiento (Deng, Chen, Zhang, & Mahadevan, 2012), con los cuales se le permite al robot conocer el estado del entorno para evadir obstáculos.

1.1 Formulación y planteamiento del problema

Al momento de realizar la automatización de una línea de producción, mediante autómatas industriales, se puede observar que para la mayoría de los casos es común encontrar sistemas de seguridad que evitan situaciones de riesgo con los operarios. Estos sistemas de seguridad suelen ser acogidos por estándares que son creados por entidades de regulación como la OSHA (Occupational Safety and Health Administration), el instituto ANSI (American National Standards Institute) y la organización ISO (International Organization for Standardization). Entre los sistemas se seguridades más usados se encuentran las barreras de acceso, barreras fotoeléctricas, sensores de presencia y dispositivos de intercambio de piezas. Estos sistemas en general evitan la presencia de operarios dentro del espacio de trabajo del robot, por lo que se restringe el trabajo colaborativo. Para proteger al personal se especifica el uso de elementos que activen los frenos en cada eslabón (Faber, Bützler, & Schlick, 2015). La mayoría de los robots industriales únicamente contemplan como seguridad desactivar al autómata, y esto se debe a que los sistemas no conocen el estado del entorno, lo que impide que se calcule planeaciones de movimiento que evadan los obstáculos.

A través de los COBOTs se ha reducido enormemente los riesgos. Este tipo de robots físicamente no pueden ejercer grandes fuerzas o poseen mecanismos de detección de colisiones que son robustos como sistemas de seguridad pasivos, sin embargo, la mayoría de los robots colaborativos e industriales no conocen el estado de su entorno en detalle, por lo que no pueden evadir colisiones mientras continúan con su labor.

Poseer sistemas robóticos que no tengan la capacidad de conocer el estado del entorno los puede hacer peligrosos en cualquier trabajo en el que se requiera la presencia de un operario. Por este motivo es necesario crear sistemas de seguridad activos que eviten cualquier contacto del robot con los operarios o elementos del entorno. Aunque existen desarrollos que explican algoritmos de planeación de movimiento, no se encuentra documentación que aborde la creación de todo el sistema de seguridad, es decir, el modelamiento del entorno, la caracterización de los obstáculos, la creación del grafo y su solución.

1.2 Justificación del proyecto

El creciente uso de robots en la actualidad hace que para el año 2019 se estime la existencia de 2.6 millones de robots operando a nivel mundial (Seitz, 2017), con un crecimiento del 15% en promedio por año. Según la Federación Internacional de robots (IFR)(Industrial robot sales increase worldwide by 31 percent, n.d.), en 2017 se alcanzó un récord en ventas de robots industriales, obteniéndose 387,000 unidades vendidas a lo largo del mundo, en donde china registra el mayor crecimiento en la demanda de estos. En la Figura 1 se observa la estimación de

ventas de autómatas por cada año transcurrido desde 2006 hasta 2017. Esta muestra un patrón creciente, indicando que en los años futuros la presencia de robots será cada vez más común. Lo anterior permite inferir que los robots son herramientas que cumplen un papel fundamental en la sociedad, por lo que es indispensable evaluar la capacidad que tienen para adaptarse al entorno.

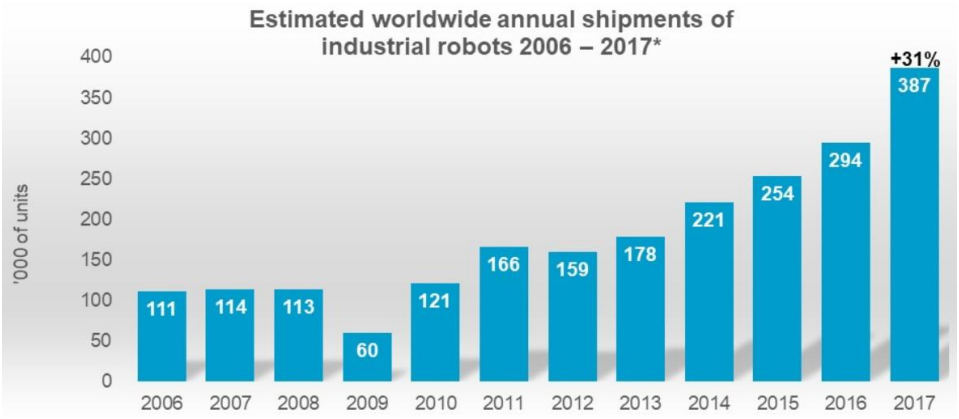


Figura 1 Ventas de robots entre 2006 y 2017 (Industrial robot sales increase worldwide by 31 percent, n.d.).

Según la OSHA, la cual hace parte del departamento de trabajo de estados unidos, desde 1984 hay registro de 41 accidentes con robots industriales que afectaron o causaron la muerte a operarios (Occupational Safety and Health Administration, n.d.). Esta cifra podría argumentar que estos sistemas son muy seguros, sin embargo, hay que prestar atención que en la mayoría de los casos los autómatas se encuentran aislados por algún sistema que los detiene una vez el operario entra al entorno operativo. Para realizar trabajos HRC es necesario mejorar la seguridad que ofrecen hoy en día estos sistemas.

Si se emplea técnicas de inteligencia artificial y algoritmos de planeación de trayectoria, se puede proponer un sistema de seguridad que sea una alternativa de solución a esta problemática. Permitirle al sistema detectar, clasificar y planear su trayectoria son funcionalidades que se requieren para permitir actividades en donde se involucra el hombre y la máquina.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar e implementar un sistema de seguridad basado en aprendizaje profundo que le permita a un robot antropomórfico detectar, priorizar y evadir obstáculos estáticos en un ambiente de trabajo controlado.

1.3.2 Objetivos Específicos

1. Desarrollar un algoritmo, basado en aprendizaje profundo, para detectar y clasificar los objetos que se encuentran dentro del área de trabajo del robot.

2. Analizar el entorno operativo del robot a través de nubes de puntos y cámaras RGB-D para detectar obstáculos con los cuales el manipulador puede colisionar.

3. Generar el modelado del entorno operativo del robot mediante matrices homogéneas para desarrollar cálculos de cinemática directa e inversa.

4. Mediante teoría de grafos desarrollar la planeación de movimiento de un robot fijo de 6 DOF para evadir obstáculos estáticos con diferentes niveles de prioridad.

5. Implementar un entorno virtual de simulación para realizar pruebas de validación del sistema desarrollado.

1.4 Hipótesis

Integrar herramientas de la inteligencia artificial y la robótica con algoritmos de planeación de trayectoria permite a un robot antropomórfico de 6 DOF ejecutar labores de manera segura en entornos con obstáculos estáticos.

1.5 Preguntas de investigación

Para poder solucionar la problemática expuesta es necesario responder las siguientes preguntas:

- ¿cómo es posible clasificar los objetos, entre obstáculos y objetivos?
- ¿cómo se puede determinar la ubicación de estos respecto al robot?
- ¿cómo el robot puede evadir obstáculos y dirigirse al objetivo?
- ¿cómo es posible evadir con niveles de prioridad los obstáculos?

1.6 Delimitaciones

La investigación está delimitada por los siguientes ítems:

- El algoritmo basado en inteligencia artificial reconoce al operario y hasta 4 tipos de objetos, siempre y cuando estén dentro del área de trabajo del robot.

- El algoritmo encargado de realizar la planeación de trayectoria le permite al autómata evadir obstáculos mientras ejecuta una labor específica.

- Los algoritmos se implementan sobre un robot antropomórfico de 6 grados de libertad.

- La validación del algoritmo se realiza en un entorno de simulación.

1.7 Metodología

Para desarrollar la investigación y cumplir con los objetivos propuestos, se usó el método presentado en la Figura 2, el cual se compone de 9 fases. Las fases 5 a 8 indican el objetivo específico que están solucionando.

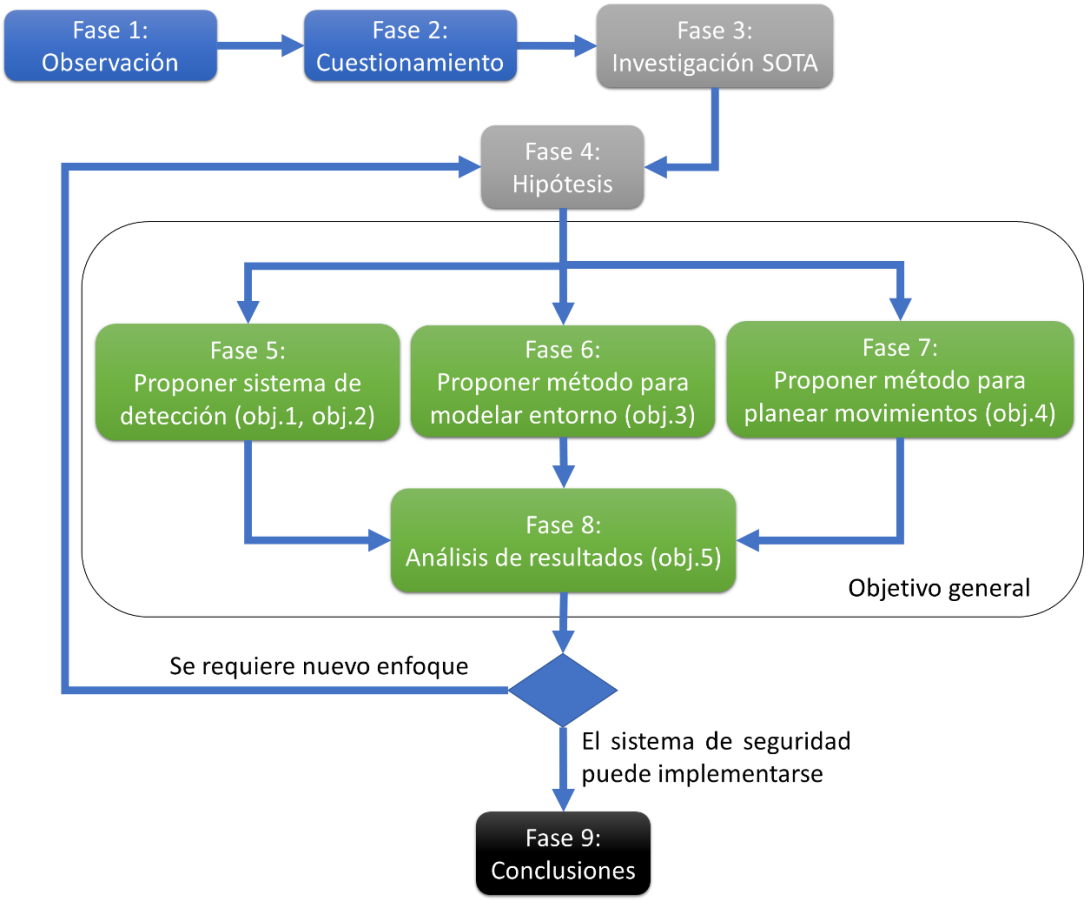


Figura 2 Método implementado para desarrollar el proyecto.

En la primera fase del método se realiza una observación en la cual se identifica la forma de operar de los sistemas robóticos actuales. A grandes rasgos se identifican los sistemas de seguridad, lo que posteriormente lleva a realizar cuestionamientos acerca del funcionamiento de estos y la forma en la que permiten adaptar los manipuladores a entornos con personas. La fase 3 implica una

revisión más detallada del estado del arte, en la cual se investiga los tipos de sistemas de seguridad, la forma en la operan, los resultados que pueden generar y los principios teóricos sobre los que funcionan. Esta etapa permite identificar las diferencias entre el sistema desarrollado y los actuales, además, permite identificar técnicas que pueden ser de utilidad. En la fase 4 se plantea la hipótesis, la cual debe demostrarse a lo largo de la investigación. Una vez se comprende lo que desea desarrollar y se tiene conocimiento sobre los sistemas actuales, se procede a proponer y a desarrollar los módulos que componen al sistema propuesto, los cuales deben trabajar en conjunto para solucionar el objetivo general.

En la fase 5 se propone un sistema de detección de obstáculos que funcione a partir de una red basada en convoluciones y un modelo para reconstrucción 3D. En esta fase se investiga sobre las diferentes redes, se escoge una con base a un métrica y finalmente se ajusta para solucionar el problema actual. Esta red se implementa en conjunto con el modelo que permite realizar la reconstrucción 3D para detectar todos los posibles obstáculos, de alta y baja prioridad, con los cuales el robot puede colisionar. La fase 6 desarrolla el modelado del entorno con el cual se comprende cómo se encuentra el robot y los objetos a su alrededor. La fase 7 propone el método para realizar la planeación de movimiento del robot, este estudia cómo crear grafos y solucionarlos para desplazar el robot en el entorno sin que colisione. La fase 8 usa todos los productos obtenidos por las fases 5 a 7 y desarrolla pruebas de funcionamiento en el entorno de simulación para determinar si se obtienen resultados adecuados. En esta fase se determina si algún planteamiento es incorrecto y si se deben realizar ajustes tanto en la hipótesis como en los objetivos que deben solucionar las fases 5 a 7. Finalmente, la fase 9 presenta las conclusiones obtenidas.

1.8 Organización del documento

La organización del documento se puede observar en la Figura 3. En el primer capítulo se desarrolla la introducción al proyecto, en donde se contextualiza al lector y se indica la problemática a solucionar, además, se mencionan los objetivos, la hipótesis, las preguntas de investigación, las delimitaciones y la metodología. El segundo capítulo describe la investigación documental que se relaciona con la problemática expuesta, con la cual se identifica los métodos usados por otros investigadores. El tercer y cuarto capítulo exponen las herramientas teóricas, los materiales usados y los temas que se abordan en las siguientes secciones. Los siguientes capítulos se enfocan en solucionar los objetivos del proyecto, mostrar las pruebas de funcionamiento e indicar las conclusiones del trabajo.

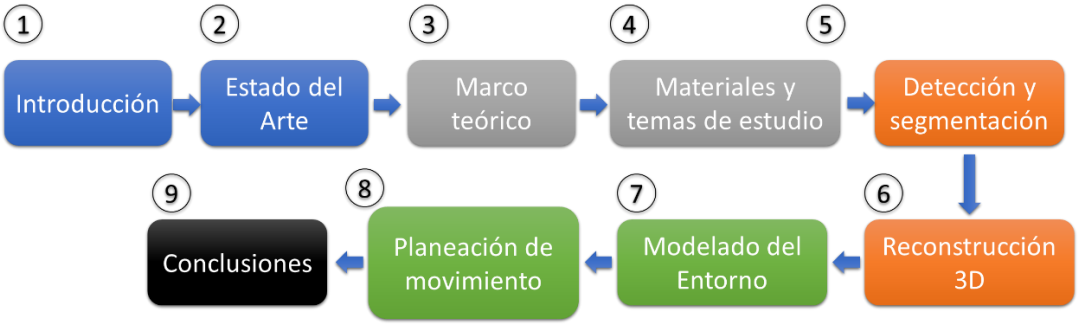


Figura 3 Organización del documento.

2 Estado del Arte

Desde la aparición de los primeros Cobots en 1995 (Peshkin, Colgate, Akella, & Wannasuphoprasit, 2000) se han propuesto diferentes sistemas de seguridad que están presentes tanto en el software como en el hardware de los sistemas robóticos, los cuales se pueden clasificar como activos o pasivos. Los denominados activos tienen como objetivo prevenir las colisiones haciendo uso de sensores, mientras que los pasivos disminuyen el impacto de la colisión reduciendo la energía cinética de los eslabones. El primer Cobot, el cual fue creado por los ingenieros Edward Colgate y Michael Peshki (Morris, 20), contemplaba la seguridad como la no existencia o carencia de fuentes de alimentación. Actualmente se han integrado a los robots colaborativos herramientas relacionadas con el control, la visión de máquina y la inteligencia artificial que permiten una interacción más segura y eficiente con las personas. A continuación, se presentan las investigaciones que se han desarrollado al respecto.

En general los sistemas relacionados con la seguridad pasiva presentan una metodología similar para la detección de las colisiones. Estos disminuyen el impacto detectando cambios en la respuesta dinámica del robot (Erden & Tomiyama, 2010), la cual puede ser estimada gracias a que se posee un modelo matemático del manipulador. Como se observa en (Cho, Kim, Lee, & Song, 2012; De Luca, Albu-Schaffer, Haddadin, & Hirzinger, 2006) , se implementa el modelo del robot y con base en la energía del sistema y al momento generalizado se puede calcular una variable residual que es sensible únicamente a las colisiones, dando la posibilidad de tomar decisiones como, por ejemplo, cortar el suministro energético a los eslabones. En (Cho, Kim, Kim, Song, & Kyung, 2012; Haddadin, Albu-Schaffer, De Luca, & Hirzinger, 2008) se implementan observadores de estado para estimar la fuerza externa que ingresa al sistema, la cual se relaciona directamente con colisiones del manipulador. Estos sistemas de seguridad son eficientes, pero deben ser implementados en conjunto con sistemas activos para tener robots mucho más seguros. En (De Luca & Flacco, 2012) se muestra la relevancia de tener estos dos sistemas funcionando simultáneamente, en donde el sistema de evasión propuesto hace uso de vectores repulsivos que modifican la velocidad del efector final para impedir colisiones.

Dentro de la categoría de los sistemas pasivos también se encuentra el uso de actuadores SEA (Series Elastic Actuator) (Schütz, Nejadfard, Kötting, & Berns, 2016), los cuales poseen elementos elásticos ubicados en serie a la fuente de energía (Zhang, C., Yan, Wen, Yang, & Yu, 2018) que permiten limitar el impacto de la colisión. Estos elementos además ayudan a disminuir los picos de energía que se producen al iniciar el movimiento (Grimmer, Eslamy, & Seyfarth, 2014). El robot Baxter de Rethink Robotics es un ejemplo de un robot con actuadores SEA, por lo que es usual implementarlo en entornos colaborativos y educativos. Por otra parte, también existen una serie de actuadores de embrague SCA (Series Clutch Actuators) (Lauzier & Gosselin, 2011) , los cuales limitan la magnitud de la fuerza con el objetivo de evitar niveles excesivos que sean peligrosos en interacciones con personas. En la Figura 4 se indica cómo actúa la fuerza en diferentes tipos de actuadores. Como se puede observar, los rígidos son los menos seguros ya que la fuerza no está restringida. Combinaciones de estos actuadores son presentados en (Alò, Bottiglione, & Mantriota, 2016; Rouse, Mooney, & Herr, 2014).

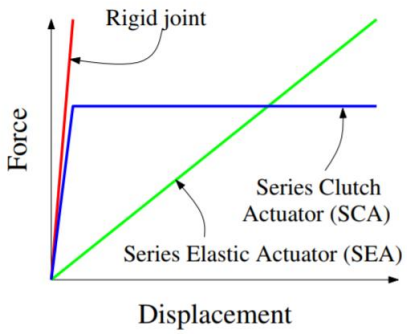


Figura 4 Actuadores rígidos, SCA y SEA (Lauzier & Gosselin, 2011) .

Respecto a los sistemas de seguridad activos se identificó 3 tipos de categorías. La primera se enfoca en usar todo tipo de sensores que permitan medir la distancia de los objetos respecto al robot para disminuir la velocidad de las articulaciones. La segunda implementa algoritmos que crean barreras artificiales con el fin de restringir espacios en los que no puede operar el autómata. La tercera aplica algoritmos de planeación de trayectoria para calcular el desplazamiento del robot en función a los obstáculos presentes. Algunas de estas investigaciones son presentadas a continuación.

Sistemas basados en sensores se pueden encontrar en (Gageik, Müller, & Montenegro, 2012; Ruan & Li, 2014; Tanaka, Kon, & Tanaka, 2015) . Estos se enfocan en medir la posición relativa de los obstáculos respecto al robot, para luego enviar esta información a un sistema de control que toma decisiones respecto al movimiento de los actuadores. Enfoques interesantes han sido propuestos en (Lam, Yip, Qian, & Xu, 2012; Navarro et al., 2013), en donde se intenta crear una armadura de sensores capacitivos que permiten medir la presencia de obtáculos. Estos en algunas investigaciones son presentados como elemetos bio-inspirados (Prescott, Pearson, Mitchinson, Sullivan, & Pipe, 2009; Schlegl, Kröger, Gaschler, Khatib, & Zangl, 2013; Sullivan et al., 2011) debido a que imitan el funcionamiento de las vibrisas de los mamíferos, las cuales obtienen información del ambiente midiendo la tensión superficial que es producto de la deformación de las mismas. En (Vogel, Fritzsche, & Elkmann, 2016) se plantea un sistema innovador para permitir la interacción hombre-máquina con robots industriales convencionales sin necesidad de realizar modificaciones fisicas a estos; se propone incorporar sensores táctiles en la superficie del área de trabajo en la cual pueden estar en movimiento los operarios. Mediante esta metodología se logra medir con precisión la posicion de las personas respecto al robot, por lo que se puede evaluar si este debe disminuir la velocidad para conservar la seguridad de los operarios.

Un sistema más robusto se puede encontrar en (Lasota, Rossano, & Shah, 2014) , en donde se implementa un sensor PhaseSpace para capturar el movimiento del operario. Este sistema usa cámaras que pueden identificar hasta 256 marcadores con los cuales se puede calcular la posición de las extremidades de una persona. Mediante esta metodología se puede tener cálculos más robustos que los obtenidos en (Vogel et al., 2016), ya que la cantidad de información abstraida del ambiente es mayor y más exacta, lo que permite calcular la distancia del robot respecto al operario con mayor confiabilidad. Las investigaciones mencionadas anteirormente tienen un sistema de control que reduce la velocidad del autómata en función a su cercanía con el operario (obstáculo) (Fenucci, Indri, & Romanelli, 2014; Ferraro, Indri, & Lazzero, 2012), por lo que estas

metodologías en realidad sólo impiden que se genere la colisión desactivando el autómata en lugar de realizar planeaciones de movimiento.

Por otro lado se pueden encontrar sistemas de seguridad que tratan de crear barreras artificiales entre el robot y las personas para evitar posibles colisiones. Como se observa en (Bogue, 2017), la empresa KUKA ha desarrollado un software llamado "KUKA.safeoperation" el cual permite restringir espacios de trabajo del robot de tal manera que se le limita el desplazamiento de este para que no pueda alcanzar trayectorias con las cuales se pueda afectar al operario. De igual manera, la empresa ABB ha creado el software SafeMove2 (SafeMove 2 - IRC5 options (IRC5 controller), n.d), el cual permite la integración hombre máquina con la funcionalidad "Saves floor space", el cual en esencia es similar a KUKA.safeoperation.

Por último, se identificó desarrollos que tienen como objetivo no solo evitar una colisión, sino que, adicionalmente calculan la planeación de trayectoria del robot. Principalmente se encuentra desarrollos que usan la cinemática diferencial del robot a través de la matriz jacobiana para modificar la velocidad de este y a si mismo su trayectoria (Fratu, Vermeiren, & Dequidt, 2010; Ping, Wei, Li, & Luo, 2009; Puiu & Moldoveanu, 2011). Como se observa en (Mohammed, Schmidt, & Wang, 2017), se propone modificar la velocidad deseada en el gripper en función a la distancia entre el obstáculo y el efector final. Esto genera una trayecotria que lleva al robot de un punto inicial a uno final evitando el obstáculo (ver Figura 5). En (Flacco, Kröger, De Luca, & Khatib, 2012) se presenta una metodología similar que es más robusta, esta toma en cuenta la cercanía de todos los obstáculos respecto a los eslabones del robot para modificar la velocidad de cada articulación. Los vectores que modifican la velocidad de las articuaciones se les llama vectores de repulsión. Mediante estos el robot puede evadir obstáculos teniendo en cuenta colisiones en toda la cadena cinemática. Esta metodología puede presentar buenos resultados pero exige al robot ejecutar velocidades que en muchas posiciones pueden ser inalcanzables, además, se sabe que los métodos que usan vectores de repulsión pueden presentar indeterminaciones debido a la existencia de mínimos locales, los cuales impiden al robot llegar a la posición objetivo (Conkur, 2005; Yagnik, Ren, & Liscano, 2010). Una de las primeras investigaciones que implementa el concepto de campo potencial para la evasión de obstáculos fue publicada en el año 1986 (Khatib, 1986). Esta ha sido punto de partida para las investigaciones actuales.

Figura 5 Evasión de obstáculos modificando la velocidad deseada en el gripper (Mohammed et al., 2017).

Un algoritmo de gran importancia para la planeación de trayectoria de robots móviles se puede encontrar en (Benavides, Julián Esteban Herrera, Corredor, Moreno, & Hernández, 2018; Jabbar, 2016). A este algoritmo se le llama Flood Fill y se basa en la técnica de inundación de una matriz para calcular la planeación de trayectoria. Aunque este es un algoritmo aplicado a robots móviles

que deben evadir obstáculos, se puede encontrar una investigación que lo usa para determinar la planeación de trayectoria de un robot Scara (Benavides, J.E.H., Suescún, C.G.P. and Moreno, R.J, 2018), dando una alternativa que es aplicable a robots fijos no redundantes de 3 DOF. Lo que destaca de esta investigación es que se calcula una trayectoria, libre de obstáculos, usando un algoritmo exacto, es decir, uno que llega a la solución óptima según como se plantea la discretización del entorno. Aunque en estas investigaciones no se mencione un grafo, esta técnica en esencia se basa en construir uno para solucionarlo y así obtener la planeación de trayectoria.

Cuando se analizan robots con varios grados de libertad se empieza a trabajar con espacios de configuración de alta dimensión. La planeación de trayectoria en este tipo de robots se vuelve compleja ya que existe una combinación enorme de posibilidades en el desplazamiento del robot. En respuesta a esta problemática se han planteado algoritmos probabilísticos que se basan en muestreos aleatorios del espacio de configuración. Uno de los más famosos es denominado RRT (Rapidly-Exploring Random Trees) (LaValle, 1998), el cual fue publicado en 1998. Desde entonces se ha desarrollado algunas variantes obteniéndose algoritmos como RRT*(Lee, Baek, & Kim, 2015) RRT-Conect (Kuffner & LaValle, 2000) e informed-RRT*(Gammell, Srinivasa, & Barfoot, 2014), entre otros que pueden encontrarse en OMPL (Open Motion Planning Library) (Sucan, Moll, & Kavraki, 2012). Lo interesante de estas metodologías es que se puede analizar planeaciones de trayectoria para robots móviles no holonómicos y robots fijos con espacios de configuración de alta dimensionalidad. Estos algoritmos pueden abordar problemas denominados "kinodynamic", los cuales requieren analizar obstáculos, fuerzas, velocidades, aceleraciones y límites en articulaciones, motivo por el cuál son de gran importancia en el estudio de robots móviles. Si el robot que se implementa es holonómico, es posible construir grafos no probabilísticos que pueden estar uniformemente distribuidos a lo largo del espacio de configuración. Este es el enfoque que se da en esta investigación.

Existe una rama de la inteligencia artificial llamada Deep Learning (DL), la cual ha mostrado grandes avances en el reconocimiento de patrones, por lo que múltiples campos del conocimiento están mostrándose atraídas hacia esta, y la robótica no es la excepción. El DL ha mostrado lo robustas que pueden ser varias arquitecturas para solucionar problemas de clasificación, detección y de toma de decisiones. Como se muestra en (Miyajima, 2017), el DL se ha usado para automatizar problemas de sujeción de objetos. Este es un problema de gran dificultad que incluso ha generado desafíos como el "Amazon Picking Challenge" (Correll et al., 2016a), el cual exige almacenar objetos con formas variadas en estantes. Lo interesante de este desafío es que dos tercios de los participantes se apoyaron de técnicas de DL, incluyendo a los que obtuvieron las mejores puntuaciones. Hay varias aplicaciones que pueden referenciarse, pero quizás una de las impresionantes es la estimación de la posición y la orientación de los objetos (Tekin, Sinha, & Fua, 2018). En general las redes que se encuentran en este campo son usadas para detectar objetos, es decir, obtener su ubicación dentro de la imagen y la clase a la que pertenece. Entre las redes más famosas de detección se encuentra la Faster-RCNN (Ren, He, Girshick, & Sun, 2015) y la YOLO (Shafiee, Chywl, Li, & Wong, 2017). En este trabajo se usa una arquitectura para detectar operarios y objectos que el robot debe evadir o manipular.

El DL además de usarse en tareas de detección en imágenes, se ha implementado para abordar el problema de evasión de obstáculos. Como se muestra en (Gaya et al., 2016), se puede aplicar una red CNN para calcular un mapa de transmisión para indicarle a un vehículo submarino autónomo

(AUV) la dirección de escape que debe hacer para evitar colisiones. La red convolucional en esta situación no calcula el desplazamiento del robot, sólo apoya en el cálculo de una matriz que es usada para detectar las direcciones de escape. Existen enfoques en donde la red CNN es entrenada de forma supervisada para que genere una salida multiclase que indique la dirección de escape para evitar obstáculos (Gandhi, Pinto, & Gupta, 2017; Tai, Li, & Liu, 2016). Incluso se pueden manejar principios similares, pero con un entrenamiento no supervisado (Xie, Wang, Markham, & Trigoni, 2017), en donde no se requiere crear bases de datos ya que el sistema por si solo aprende a realizar la tarea maximizando una función de recompensa futura acumulativa. La mayoría de los trabajos que trabajan con obstáculos y DL se enfocan en desarrollar planeadores de trayectoria locales, es decir, sistemas que le indican al robot cómo desplazarse sólo con la información actual que puede ver el robot.

3 Marco Teórico

La investigación que se realiza en este proyecto está soportada por teorías y técnicas que se relacionan con el aprendizaje profundo, la planeación de trayectorias, la cinemática directa y la cinemática inversa. Con el fin de otorgar una base sobre la cual se empiecen a analizar los algoritmos que se presentan en secciones posteriores, se expone la teoría fundamental que permite comprender el proyecto en su totalidad.

Para poder analizar el robot y el entorno en el que se encuentra es fundamental realizar el modelado del espacio de trabajo (Cisneros et al., 2014) . Este modelamiento permite conocer la ubicación y orientación de cada entidad respecto a un marco de referencia inercial, lo que da la posibilidad de comprender como se encuentra el robot y su entorno. Para esto es necesario implementar herramientas matemáticas como lo son las matrices de transformación homogénea, las cuales son ampliamente tratadas en (Craig, 2009; Spong & Vidyasagar, 2008) , entre otros.

La matriz de transformación homogénea permite comprender la relación que existe entre dos sistemas coordenados. Esta indica cómo se encuentra un sistema coordenado, en términos de posición y orientación, respecto a otro sistema coordenado, lo que permite realizar transformaciones de uno a otro. La matriz de transformación homogénea se presenta en la ecuación (1) (Cisneros et al., 2014).

$$ {}^{\mathbf{w}}\mathbf{T}_{\mathbf{E}} = \begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & t_x \\ n_y & o_y & a_y & t_y \\ n_z & o_z & a_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

Dicha matriz permite realizar transformaciones de un sistema coordenado E a uno W. Esta se compone por un vector de traslación $\boldsymbol{t}$ y una matriz de rotación que se identifica por los vectores $\boldsymbol{n}, \boldsymbol{o}$ y $\boldsymbol{a}$. La última fila se compone de tres valores de escalamiento y uno de perspectiva. A partir de esta matriz es posible relacionar cualquier elemento respecto a un marco de referencia o sistema coordenado base.

Un ejemplo en el que se transforma un punto del sistema E al W y viceversa se muestra en (2) y (3).

$$ {}^{\mathbf{w}}\mathbf{P} = {}^{\mathbf{w}}\mathbf{T}_{\mathbf{E}} \cdot {}^{\mathbf{E}}\mathbf{P} \tag{2} $$

$$ {}^{\mathbf{E}}\mathbf{P} = {}^{\mathbf{E}}\mathbf{T}_{\mathbf{w}} \cdot {}^{\mathbf{w}}\mathbf{P} = ({}^{\mathbf{w}}\mathbf{T}_{\mathbf{E}})^{-1} \cdot {}^{\mathbf{w}}\mathbf{P} \tag{3} $$

Es importante tener en cuenta que se puede realizar lo que se conoce como la composición entre matrices. Esto permite relacionar sistemas que se encuentran en cadena como, por ejemplo, los eslabones de un robot fijo. La posición del efector final se calcula mediante la composición de varias matrices que relacionan, de manera ordenada, cada sistema coordenado desde el sistema base hasta el gripper. Un ejemplo de esto se puede ver en (4), la cual describe al robot planar de

la Figura 6. La variable β_1 representa las distancias entre el sistema coordenado cero y uno; β_2 la distancia entre el sistema coordenado uno y dos.

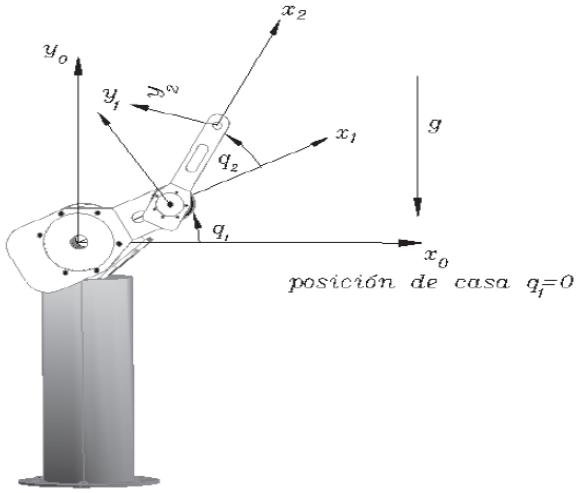


Figura 6 Robot planar de dos grados de libertad (Reyes Cortes, 2012) .

$$ {}^{0}\mathbf{H}_{2} = {}^{0}\mathbf{H}_{1}\,{}^{1}\mathbf{H}_{2} = \begin{bmatrix} \cos(q_1+q_2) & -sen(q_1+q_2) & 0 & l_1\cos(q_1)+l_2\cos(q_1+q_2) \\ sen(q_1+q_2) & \cos(q_1+q_2) & 0 & l_1 sen(q_1)+l_2 sen(q_1+q_2) \\ 0 & 0 & 1 & \beta_1+\beta_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4) $$

Una vez se conoce la posición de cada elemento el siguiente problema es llevar al robot a posiciones de interés, problema que se soluciona con la cinemática inversa. Esta calcula el valor que deben tomar las coordenadas generalizadas para llevar al actuador final a una orientación y ubicación deseada. En la literatura se encuentran diferentes métodos entre los cuales destacan el método geométrico (Craig, 2009), el desacoplamiento cinemático (Reyes Cortes, 2012), el método algebraico (Husty, Pfurner, Schröcker, & Brunnthaler, 2007; Spong & Vidyasagar, 2008), el método iterativo con la inversa de la matriz jacobiana y la pseudoinversa de la matriz jacobiana (Prempraneerach & Kulvanit, 2010) , el método Damped Least Squares (DLS) (Buss & Kim, 2005) y el método Cyclic Coordinate Descent (CCD) (Kenwright, 2012); de los cuales los métodos iterativos se implementar en robot redundantes. Los métodos iterativos se pueden entender estudiando la cinemática diferencial de un robot (ver ecuación (5)), la cual relaciona las velocidades articulares $\dot{\boldsymbol{q}}$ (derivada del vector de estado $\boldsymbol{q}$) con las velocidades del gripper $\boldsymbol{W}$. La matriz jacobiana $\boldsymbol{J}$ (ver ecuacion (6)) está compuesta por varios vectores, uno por cada grado de libertad, y cada uno de estos vectores es definido según la naturaleza de la articulación a la que hacen referencia. Si la articulación es rotacional debe usarse la ecuación (7), por el contrario, si es prismática debe usarse la ecuación (8). El eje **z** de cada articulación debe estar ubicado sobre el eje de rotación o de traslación de la articulación, por lo que debe usarse la convención Denavit-Hartenberg (DH). En las ecuaciones el superíndice indica que el vector ha sido referenciado respecto al sistema base, mientras que el subíndice indica el vector analizado, el cual proviene del eslabón *i*.

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{d} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \\ d_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = [\boldsymbol{J}]\dot{\boldsymbol{q}} = \begin{bmatrix} \boldsymbol{J}_L \\ \boldsymbol{J}_A \end{bmatrix} \dot{\boldsymbol{q}} \tag{5}$$

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_0 & \mathbf{J}_1 & \cdots & \mathbf{J}_{n-1} \end{bmatrix} \tag{6}$$

$$\mathbf{J}_i = \begin{bmatrix} {}^b\mathbf{z}_{i-1} \times \left({}^b\mathbf{t}_n - {}^b\mathbf{t}_{i-1} \right) \\ {}^b\mathbf{z}_{i-1} \end{bmatrix} \tag{7}$$

$$\mathbf{J}_i = \begin{bmatrix} {}^b\mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix} \tag{8}$$

Si a un robot de 5 articulaciones rotacionales se le analiza la aportación de la articulación 3, las variables que deben usarse en (7) se deben entender como se muestra en la Figura 7. Una explicación más detallada de esto se puede ver en (Cisneros et al., 2014).

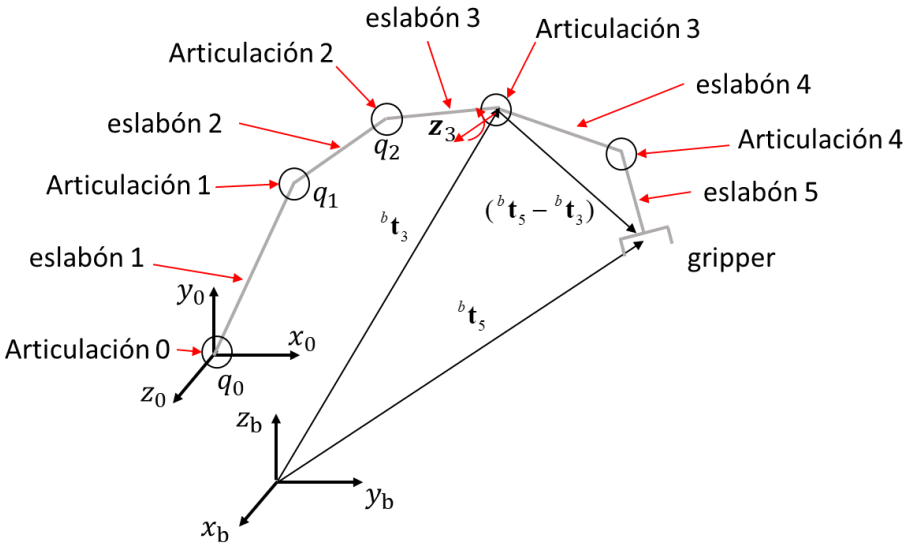


Figura 7 Vectores para calcular aportación de la tercera articulación en el jacobiano de un Robot planar de cinco grados de libertad.

A través de estas relaciones se pueden hallar las velocidades que describen el cambio de posición y orientación de un sistema coordenado del robot respecto al sistema base. El sistema coordenado puede estar sujeto a una cámara o un gripper y mediante métodos iterativos es posible posicionar y dirigir el efector final a posiciones y orientaciones deseadas, es decir, realizar la cinemática inversa del manipulador. Esto es de gran utilidad en secciones futuras para la planeación de movimiento del robot, ya que primero es necesario conocer la configuración a la que se desea llegar antes de calcular la planeación de movimiento.

Un concepto importante tratado en este trabajo son los grafos no direccionados (ver Figura 8), los cuales son conformados mediante nodos y vértices. Los nodos representan el estado de un sistema y los vértices el costo entre estos. Cada nodo puede representar una configuración específica del

robot, y mediante la información de las conexiones es posible implementar algoritmos de búsqueda del camino más corto para determinar el recorrido de un nodo a otro. Existen varios métodos entre los cuales destaca “Dijkstra”, el cual puede ser estudiado en (Pearson & Bryant, 2004). Este indica cuál es el recorrido más corto de un nodo a todos los demás nodos en el grafo. Específicamente en este problema la solución obtenida representa la planeación de movimiento que debe realizar el robot para evadir obstáculos, y los costos entre los nodos determinan cómo será este movimiento.

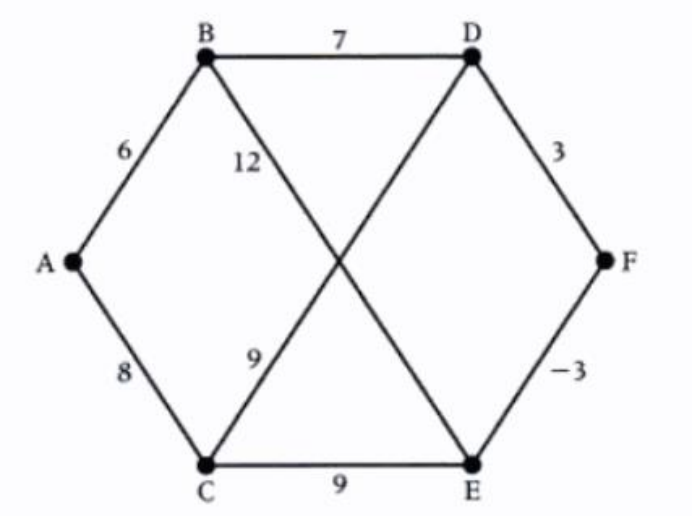


Figura 8 grafo no direccionado (Pearson & Bryant, 2004) .

Los valores que están asignados a los vértices afectan directamente el recorrido hallado por Dijkstra. Este evade los nodos que sean demasiado costosos, los cuales en esta situación son las configuraciones que están muy cerca a los obstáculos de mayor importancia. Esto quiere decir que se puede manejar la importancia con la que se debe evadir un obstáculo si se alteran los costos de los vértices.

El aprendizaje de máquina se puede describir como una práctica enfocada a la creación de algoritmos que procesan información para abstraer patrones de los cuales se puede predecir algún resultado. Esto es de especial interés ya que permite crear un sistema que tenga la capacidad de detectar operarios o elementos con los cuales el robot no debe colisionar. El aprendizaje de máquina se divide en varias ramas en las cuales se puede encontrar el aprendizaje profundo (Deep learning), el cual implementa algoritmos que son inspirados en la estructura y funcionamiento de las redes neuronales cerebrales. El aprendizaje de las redes neuronales puede clasificarse en entrenamiento supervisado o no supervisado, en donde el supervisado posee información que ha sido etiquetada previamente al entrenamiento.

Sin importar el tipo de arquitectura que se vea en este campo, se puede apreciar que la mayoría poseen elementos en común, una etapa de extracción de características y otra de inferencia. Como ejemplo puede tomarse la red Alexnet (Redes neuronales convolucionales, n.d.; Krizhevsky, Sutskever, & Hinton, 2012), la cual en primer lugar analiza las imágenes con convoluciones para luego clasificar la imagen con base a los patrones obtenidos. Esta red se hizo famosa en el concurso ImageNet ILSVRC-2012 al lograr una tasa de error de 15.3% en la clasificación de 1.2 millones

de imágenes dentro de 1000 categorías, lo cual para el estado del arte de su momento era un gran avance. En la Figura 9 se muestra una configuración general para una red de clasificación que posee las dos etapas mencionadas.

La primera etapa se encarga de realizar la extracción de características o patrones que facilitan el proceso de clasificación. Estos patrones detectados son ingresados a la etapa de clasificación que mediante una capa "fully connected" y una función "softmax" indican la probabilidad de pertenencia de la imagen a una de las categorías. Toda la red se compone de capas de convolución, funciones de activación, capas de discretización, una capa totalmente conectada y una capa softmax; las cuales fueron ajustadas en el entrenamiento de la red para que sus parámetros internos produzcan el menor error posible en la clasificación. Un estudio de estas capas puede encontrarse en (Goodfellow, Bengio, & Courville, 2016), en donde se podrá encontrar los parámetros que las definen y los métodos de entrenamiento que suelen usarse, como por ejemplo el descenso del gradiente estocástico, el cual implementa el gradiente de la función objetivo.

Las redes convolucionales más famosas son: Resnet(He, Zhang, Ren, & Sun, 2016), YOLOv3(Redmon & Farhadi, 2018a), Faster-RCNN(Ren et al., 2015) , Inception(Szegedy, Ioffe, Vanhoucke, & Alemi, 2017) y Mask-RCNN(He, Gkioxari, Dollár, & Girshick, 2017). En este trabajo se usa una red Mask-RCNN para detectar y segmentar objetos a los que el robot debe dirigirse o evadir.

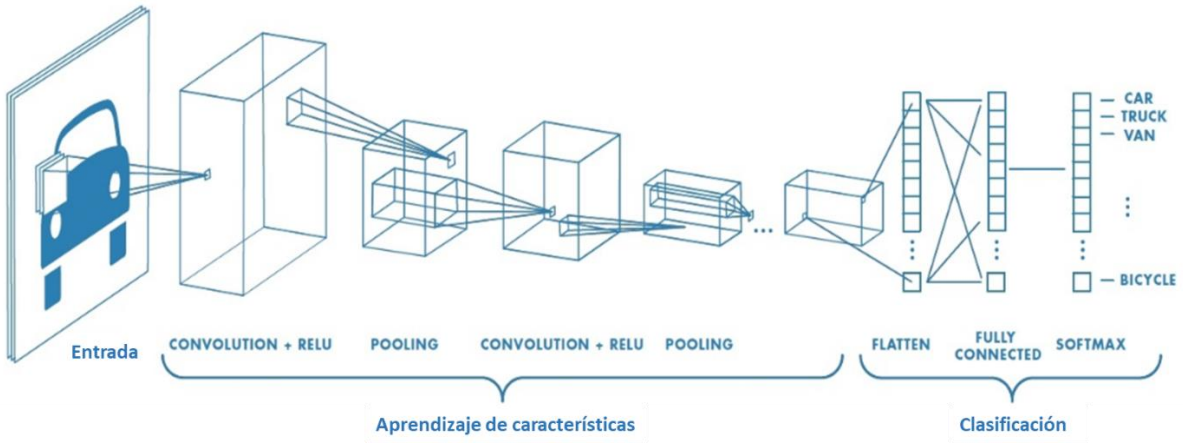


Figura 9 Arquitectura para una red neuronal convolucional (Redes neuronales convolucionales. n.d.).

4 Materiales y temas de estudio

El desarrollo de este proyecto contempla únicamente como materiales los programas con los cuales se simula el entorno operativo del robot y se crea los algoritmos que permiten calcular la planeación de movimiento. El lenguaje de programación escogido ha sido Python ya que sobre este se desarrolló el marco de trabajo Pytroch, el cual es líder en el manejo de redes basadas en DL. Grandes empresas como Facebook han creado librerías como Detectron2 que facilitan el desarrollo y entrenamiento de redes de detección, además, Python es un lenguaje que se puede integrar fácilmente a la mayoría de las plataformas de simulación de robots. Para simular el entorno de trabajo del robot se implementa el software CoppeliaSim ya que, además de ser libre, viene con modelos de robots comerciales que pueden ser fácilmente controlados desde Python mediante APIs, por lo que los programas desarrollados únicamente se encargan de enviar instrucciones al simulador. CoppeliaSim además de permitir un control simplificado de las entidades, da la posibilidad de construir entornos de trabajo sin gran esfuerzo ya que posee entidades prefabricadas con las cuales se puede construir variedad de entornos. Los materiales implementados en este trabajo se pueden ver en la Figura 10.

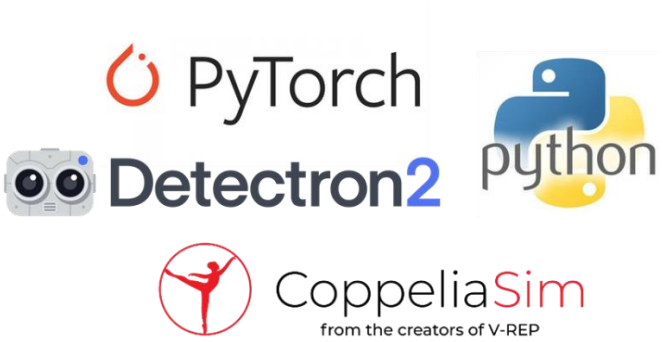


Figura 10 Software implementado.

En la Figura 11, se presenta el entorno creado. Este se adecuó para que un robot ejecute una tarea de manipulación mientras evade obstáculos. Se seleccionó el manipulador ABB IRB-4600 debido a que es un robot de 6DOF que se encuentra prefabricado dentro de CoppeliaSim. La razón por la cual se estudia la evasión de obstáculos con una cadena cinemática de 6DOF es debido a que esta es la cantidad mínima con la cual se puede orientar y posicionar un objeto en el espacio, y se desea que el robot manipule objetos sin restricciones. Los objetos que el robot debe manipular provienen de una banda transportadora. Estos deben ser depositados en un sitio específico teniendo en cuenta que al trasladarlos hay varios obstáculos (banda transportadora, cámaras, suelo, paredes) entre los cuales puede haber unos de alta prioridad (personas). Para solucionar este problema se requiere detectar los obstáculos, crear la reconstrucción 3d del entorno, relacionar toda la información respecto al robot, discretizar el espacio de configuración del robot, crear un grafo que represente el espacio de configuración, solucionar el grafo para obtener la planeación de movimiento y finalmente desarrollar el sistema de seguridad que controle el movimiento del robot. Todas estas tareas que son objeto de estudio se abordan en el orden mostrado en la Figura 12.

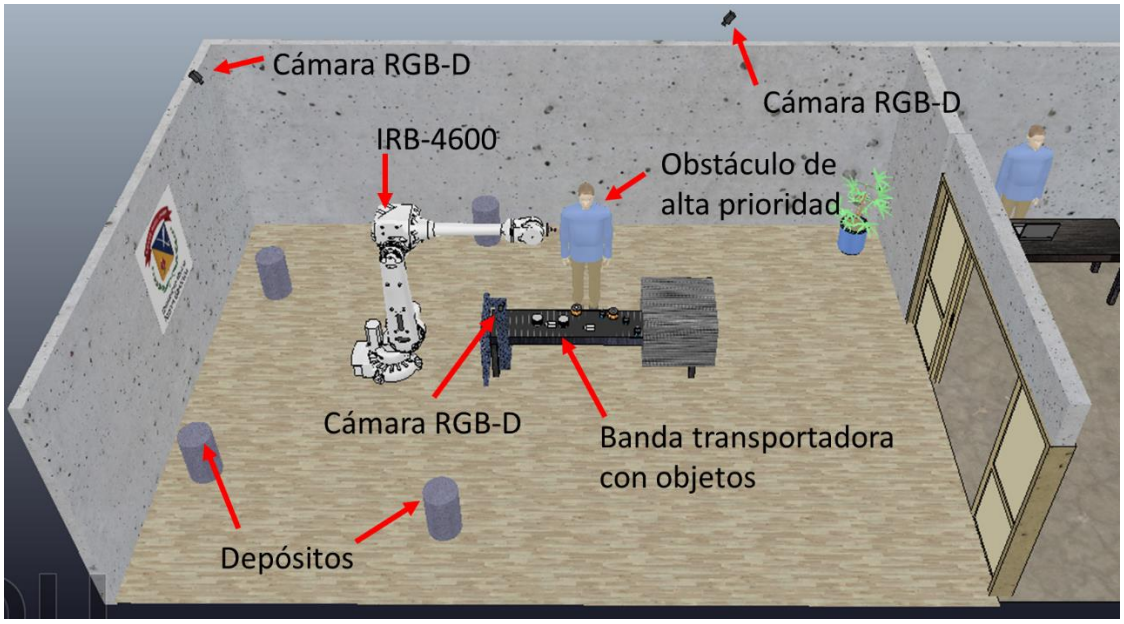


Figura 11 Entorno creado en CoppeliaSim.

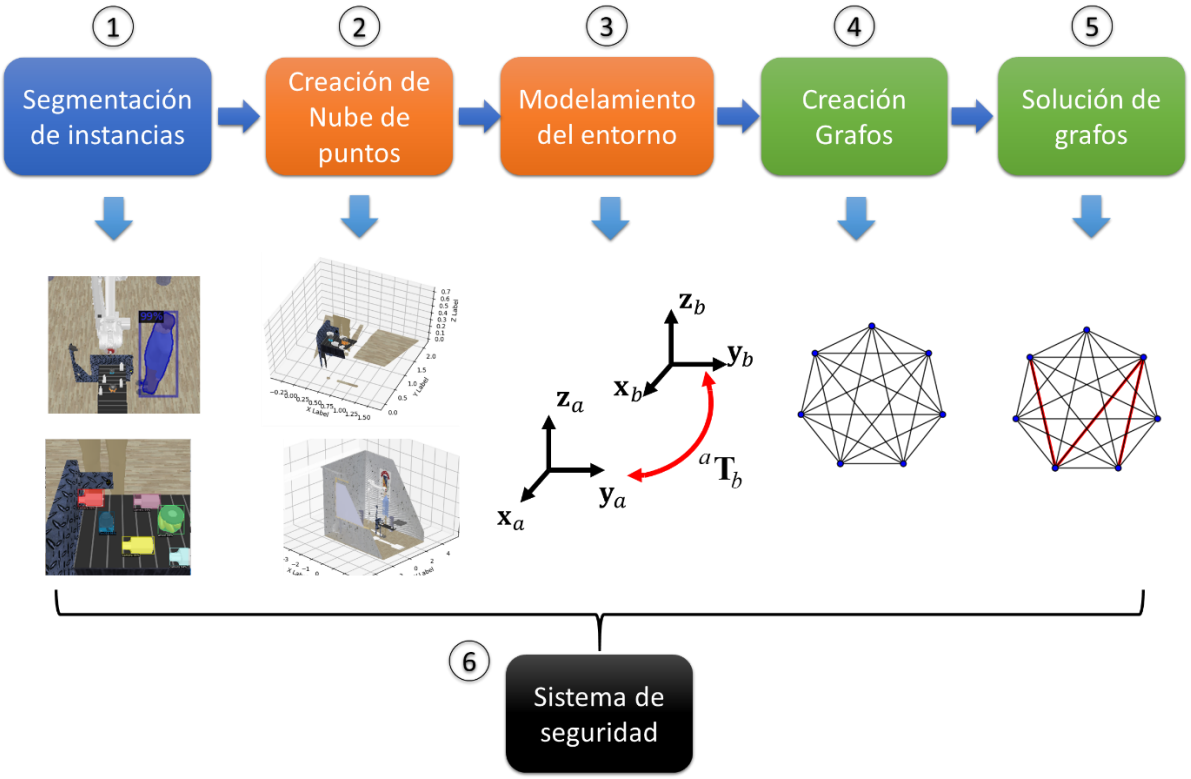


Figura 12 Temas de estudio.

5 Detección y segmentación semántica

La primera tarea que se debe solucionar para crear el sistema de seguridad es detectar y clasificar las entidades que se encuentran dentro del entorno. El robot debe identificar la diferencia entre un obstáculo de baja y alta prioridad, además, debe ubicar los objetos que debe manipular. Para satisfacer esta necesidad se propone usar una arquitectura proveniente del DL ya que el estado del arte muestra que para realizar tareas con imágenes lo más conveniente es implementar redes profundas basadas en convoluciones. Este capítulo explica cómo se soluciona el objetivo específico 1.

5.1 Selección de arquitectura

El primer paso para escoger el tipo de red que se requiere en este problema es revisar el estado del arte. En este proceso se identifica que una práctica altamente usada es implementar una red pre entrenada para hacer Transfer Learning. La red escogida se le hace un ajuste fino para adaptarla al problema actual. Esto permite que el entrenamiento sea mucho más rápido y que no se requieran bases de datos enormes, las cuales generalmente son demoradas de construir (Tan et al., 2018). También se identifica que las redes basadas en convoluciones son de gran utilidad para hacer inferencias en imágenes. Las convoluciones se implementan para abstraer características de las imágenes, las cuales sirven para realizar mejores clasificaciones. Ejemplo de esto se puede ver en la red Lenet-5(LeCun, Bottou, Bengio, & Haffner, 1998), la cual fue usada para detectar dígitos dentro de imágenes de 32x32 pixeles (ver Figura 13).

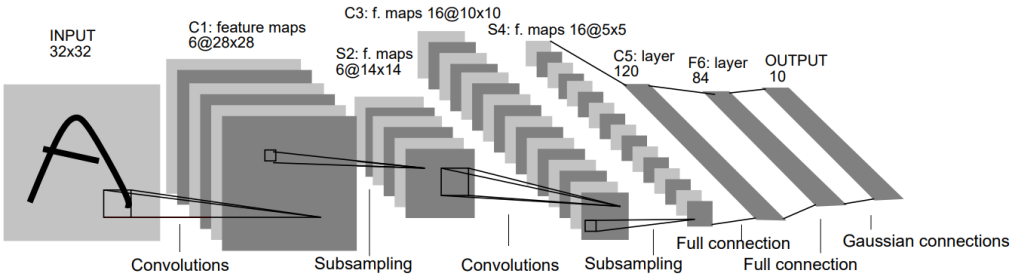


Figura 13 Lenet-5(LeCun et al., 1998) .

Las convoluciones permiten reducir la cantidad de parámetros requeridos para analizar las imágenes gracias a que poseen lo que se conoce como "Parameter Sharing", lo cual significa que el kernel de la convolución recorre toda la imagen permitiendo que los parámetros se compartan. La salida que produce la convolución tiene además la ventaja de reducir la cantidad de conexiones que existe entre las neuronas de entrada y las de salidas (sparsity of conection), lo que reduce el sobre entrenamiento y aumenta la generalización.

Gracias a estas ventajas que ofrece la convolución se han creado redes más profundas que solucionan problemas más complejos. Entre las redes más famosas se encuentran Alexnet (Krizhevsky et al., 2012) y VGG (Simonyan & Zisserman, 2014), la cuales en resumen aumentan la cantidad de capas de convolución para lograr clasificar imágenes en una gran cantidad de categorías. Estas redes son secuenciales, es decir, el flujo a lo largo de la red tiene sólo un recorrido. Podría pensarse que a medida que se aumenta la profundidad de la red se pueden obtener resultados más robustos, sin embargo, el desvanecimiento del gradiente es uno de los principales problemas, por lo que surge la red residual RESNET (He et al., 2016). Esta básicamente conecta el resultado de dos capas como se muestra en la Figura 14.a. A partir de esta configuración, los autores lograron obtener el primer puesto en el ILSVRC 2015 al obtener un error del 3.57%, mostrando que redes DAG (Directed Acyclic Graph) son de gran utilidad en el campo del DL. Siguiendo esta metodología se encuentra la red Inception (Szegedy et al., 2015), la cual permite ir mucho más profundo que la RESNET. El principio fundamental de la red está en poner en paralelo varias capas como se muestra en la Figura 14.b.

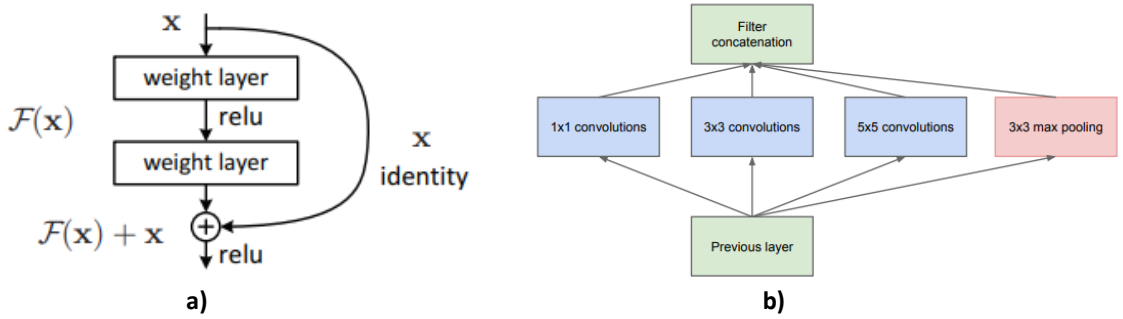


Figura 14 a) Conexión residual (He et al., 2016) b) bloque Inception (Szegedy et al., 2015).

Las arquitecturas mencionadas anteriormente se pueden usar tanto para problemas de regresión como de clasificación, por lo que se pueden entrenar para identificar si en una imagen se encuentra un obstáculo o para localizar dicho obstáculo (localización de objeto). El inconveniente de esto radica en que este problema no puede ser tratado con redes de localización ya que existe un número indefinido de objetos que deben ser identificados al mismo tiempo, es decir, se requiere un sistema de detección de objetos, motivo por el cual las redes anteriores quedan descartadas. La diferencia entre localización y detección se presenta la Figura 15. Un sistema de localización genera sólo un cuadro delimitador, por lo que adentro de este pueden estar muchos objetos, mientras que un sistema de detección a cada objeto le asigna un cuadro delimitador.

Hay varias redes de detección que se pueden citar, sin embargo, las más famosas son la Faster R-CNN (Ren et al., 2015) y la YOLOv3 (Redmon & Farhadi, 2018b). Ambas generan resultados robustos, por lo que para conocer cuál se debe seleccionar es necesario usar alguna métrica que evalúe el desempeño en la detección. Una ampliamente usada es la precisión promedio (Average Precision) con un IoU (intersecion over unión) del 50%, AP_{50}, la cual es conocida gracias al reto PASCAL (Everingham et al., 2015). A partir de esta métrica se puede determinar que la Faster R-CNN es más robusta detectando objetos (ver Tabla 1). Aunque la ventaja de la YOLO es su rapidez de ejecución, en este trabajo se prestó más importantica al desempeño en la detección. Un enfoque importante que manejan en redes de detección es usar redes CNN, como las mencionadas

anteriormente, para realizar la extracción de características de la imagen. Tanto la red YOLO como la red Faster R-CNN usan una red convolucional que generalmente referencian como "columna vertebral" (backbone).

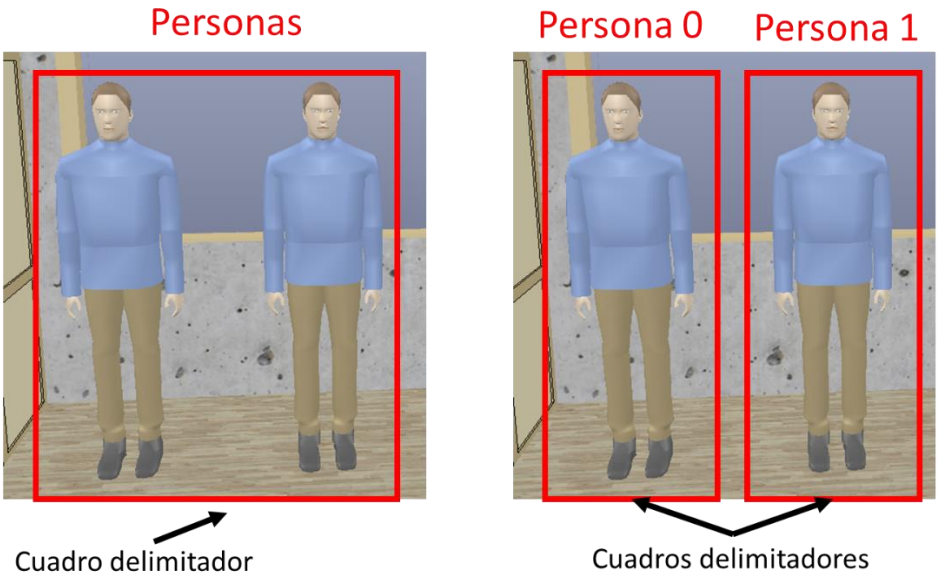


Figura 15 a) Localización de personas, b) Detección de personas.

Tabla 1 Comparación de desempeño entre la red YOLO y la red Faster R-CNN(Redmon & Farhadi, 2018)

Red de detección	Red de extracción de características	AP_{50}
YOLOv3	Darknet-53	57.9
Faster R-CNN	ResNet-101-C4	55.7
Faster R-CNN	ResNet-101-FPN	59.1

Mediante esta comparativa la red candidata es la Faster-RCNN. Al estudiarla se puede encontrar una arquitectura sucesora llamada Mask-RCNN, que implementa el mismo principio de funcionamiento, pero permite realizar segmentación de instancia de objeto, lo cual significa que permite realizar detección y segmentación semántica al mismo tiempo. Esta característica es de gran importancia ya que al realizar segmentación semántica se puede conocer todos los pixeles que pertenecen a un objeto en específico, por lo que se puede realizar la nube de puntos de cada objeto independientemente. Esto es de gran utilidad al momento de generar los grafos, por lo que la red escogida finalmente es la Mask-RCNN. Esta arquitectura además logra generar un AP_{50} igual a 62.3 en tareas de detección con la base de datos de COCO (Lin et al., 2014), demostrando que es más robusta que su antecesora y que la YOLO.

5.2 Mask R-CNN

La arquitectura llamada Mask R-CNN se puede ver en la Figura 16. Esta se basa en una faster R-CNN a la cual se le agrega una rama con capas convolucionales para realizar la segmentación de las imágenes. Esta arquitectura tiene la capacidad de detectar, clasificar y segmentar objetos con gran desempeño. Su funcionamiento se puede explicar en 6 etapas las cuales se enlistan a continuación.

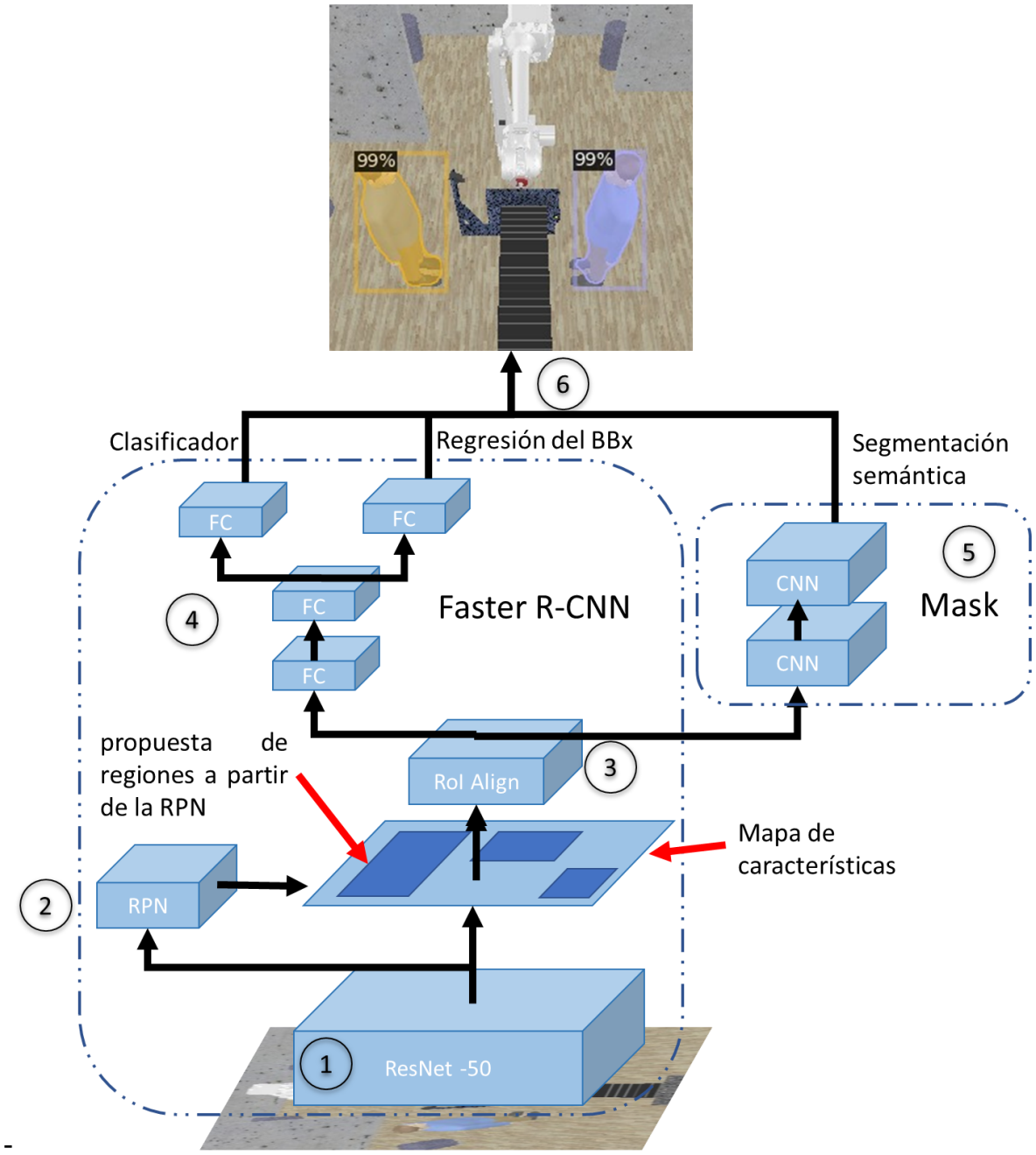


Figura 16 Arquitectura de la red Mask R-CNN, la cual permite generar clasificación, detección y segmentación de objetos.

Funcionamiento de la Mask RCNN:

1. La imagen ingresa a la una red convolucional encargada de extraer el mapa de características. Generalmente se usa redes famosas como Resnet-50, Resnet-101 o VGG-16. El objetivo de esta etapa es hallar todas las características requeridas por las demás ramas de la red.
2. La RPN escala la imagen de entrada y asigna a cada una de las nuevas posiciones 9 Anchors (ver Figura 17), los cuales representan regiones en donde podría haber un objeto. Para cada una de estas regiones la red calcula la probabilidad de que haya o no un objeto, además, modifica el tamaño por defecto de estos Anchors para extraer mejor al objeto de interés. Una vez se obtienen los resultados para cada uno de los Anchors, se aplica la técnica "supresión no máxima" (NMS) para eliminar todos aquellos RoIs (regiones de interés) que no son relevantes.

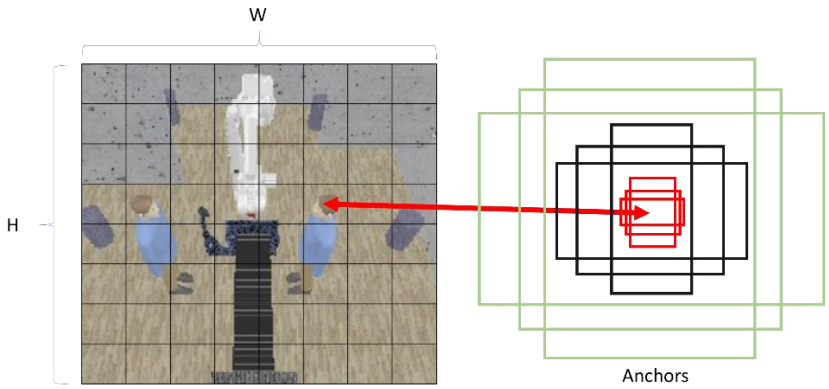


Figura 17 Escalamiento de la imagen y Anchors generados a cada celda.

3. Con los resultados de la RPN se extrae del mapa de características los datos asociados a los RoIs. RoI Aling obtiene todos estos recortes del mapa de características y les modifica el tamaño para entregar un tensor de dimensiones fijas. Por ejemplo, el tamaño del tensor resultante de RoI Aling puede ser de (N,7,7,512), indicando que se tienen N regiones de interés, 512 canales y un tamaño fijo de 7x7 al cual se redimensionan todos los recortes.
4. El tensor resultante de RoI Aling pasa a través de redes totalmente conectadas que calculan la clasificación del objeto y la posición del BBx (se realiza la localización del objeto). Si existen P clases, la red de clasificación entrega (N,P) datos y la red que calcula el BBx entrega (N,P,4) datos. De esta manera cada BBx entregado se especializa para una clase en específico. Cuando se determina a qué clase se asocia una región de interés, se procede a escoger el BBx asociado a dicha clase. Los datos que se obtienen de un BBx son las coordenadas de la esquina superior izquierda, el ancho y el alto.

5. Se aplican capas de convolución para producir la máscara que genera la segmentación del objeto. El tensor resultante genera una máscara para cada clase existente, por lo que se debe escoger la máscara que se asocia a la clase que fue determinada por la rama de clasificación.
6. Se une la información de las clasificaciones, los BBx y la máscara para realizar la segmentación de instancia de objeto.

Un análisis más detallado de la arquitectura puede hacerse revisando la Fast R-CNN (Girshick, 2015), la Faster R-CNN(Ren et al., 2015) y la Mask R-CNN(He et al., 2017).

5.3 Base de datos y Entrenamiento

Para este sistema se plantea usar dos Mask R-CNN. Una para detectar los objetos de la banda transportadora y una para detectar a los operarios del entorno (obstáculos de alta prioridad). Esto se hace con la finalidad de especializar las redes y hacer que la detección de personas sea más confiable. En este sentido se debe generar la base de datos de entrenamiento y de prueba tanto para los operarios como para los objetos que estarán en la banda. Las bases de datos deben indicar para cada objeto el BBx, la clase a la que pertenece y los pixeles que ocupa en la imagen. Ejemplos de cómo se encuentra etiquetada la base de datos se puede ver en la Figura 18. Cada polígono permite indicar los pixeles del objeto al mismo tiempo que su BBx.

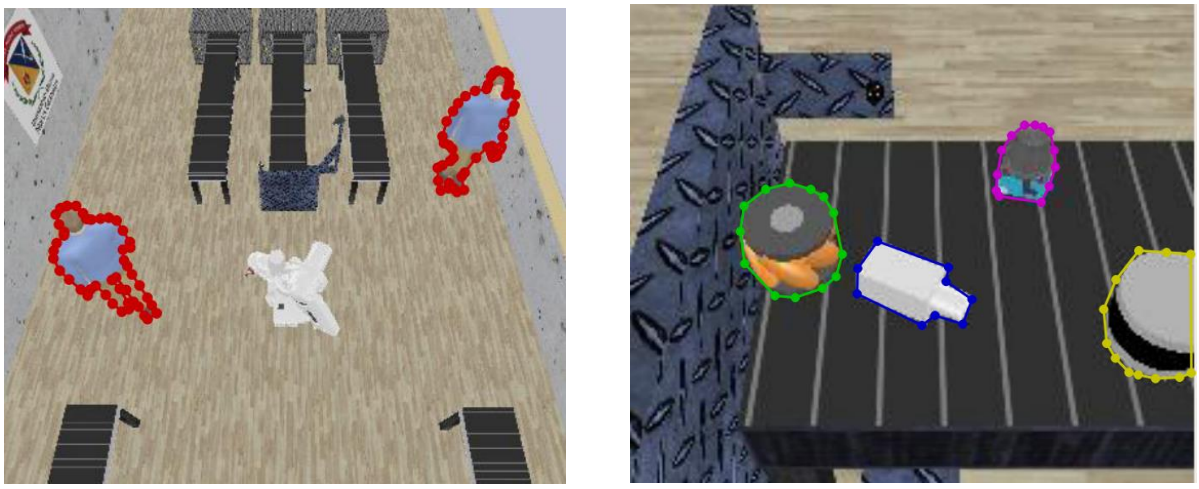


Figura 18 Ejemplo de polígonos usados para crear la base de datos de los operarios y los objetos de la banda.

Debido a que se está trabajando con una red pre entrenada se puede usar bases de datos pequeñas. Se plantea iniciar con 100 imágenes de entrenamiento y 40 de prueba para luego ir aumentando este valor hasta obtener un AP_{50} mayor o igual a 90 con las imágenes de prueba. Sin embargo, lo que se logra evidenciar es que con esta cantidad inicial las redes obtienen un AP_{50} mayor 90, lo que demuestra que estas sólo requerían un pequeño ajuste para abordar el problema actual. Las redes antes de su entrenamiento ya detectaban personas y una gran cantidad de objetos, por lo que sus pesos ya estaban muy cerca a los valores requeridos.

La distribución de la cantidad de objetos a lo largo de la base de datos se puede ver en la Tabla 2. Los objetos que provienen de la banda transportadora se han escogido arbitrariamente. El único requisito analizado fue que pudieran ser manipulados por el sistema robótico para realizar una tarea de ordenamiento. La Tabla 2 indica la cantidad de imágenes usadas y la cantidad de objetos presentes en todas las imágenes. La cantidad de objetos no es la misma en la base de datos de los objetos debido a que estos fueron creados aleatoriamente mientras la banda estaba en movimiento. Aunque no se tienen las mismas cantidades, esto no es muy relevante debido a que se está haciendo Transfer Learning.

Tabla 2 Distribución de imágenes y objetos dentro de las bases de datos

Base de datos	Imágenes	Personas	Rueda	Velodyne	Cámara	Tim310
Entrenamiento Personas	100	200	0	0	0	0
Prueba Personas	40	80	0	0	0	0
Entrenamiento Objetos	100	0	99	77	134	83
Prueba Objetos	40	0	24	12	20	23

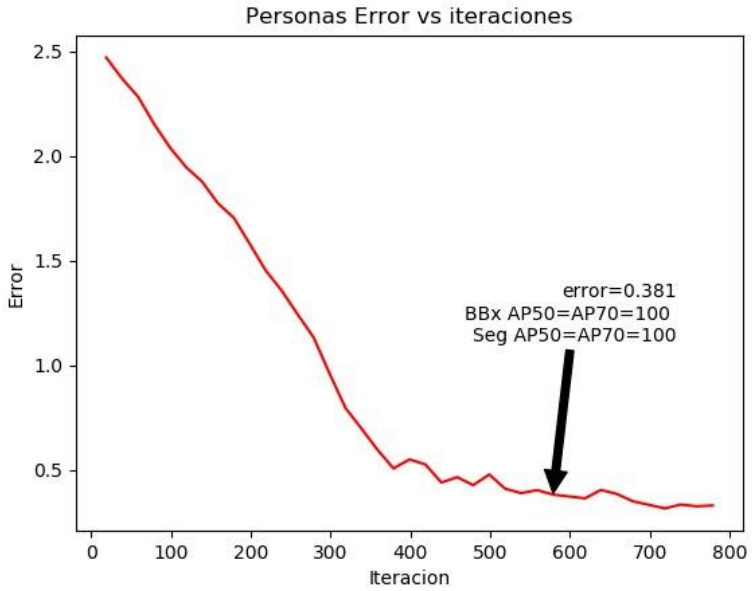
El objetivo del entrenamiento es disminuir el valor de una función objetivo o error. Para el caso de la Mask R-CNN se tienen dos funciones objetivo que se indican en (9) y (10). La función objetivo de la rama de detección y de segmentación está dada por (9), esta se compone por el error de la clase L_{cls}, el error del BBx L_{box} y el error de la máscara L_{mask}. Por otro lado, la función objetivo de la RPN se compone por el error de clasificación L_{cls} y el error del L_{RoI} (ver ecuación (10)). En el entrenamiento lo que se hace es hallar el gradiente de estas funciones a partir de la propagación hacia atrás del error, para luego aplicar el método de optimización del gradiente descendente estocástico con momento (SGD) y así actualizar el valor de los parámetros de la red. Los detalles de la implementación de la arquitectura y de las funciones de costo se pueden encontrar en la librería Detectron2 (Wu, Kirillov, Massa, Lo, & Girshick, 2019). En este desarrollo se usan los parámetros por defecto que dejan los autores en su implementación.

$$L_{\text{det}\_mask} = L_{cls} + L_{bbx} + L_{mask} \tag{9}$$

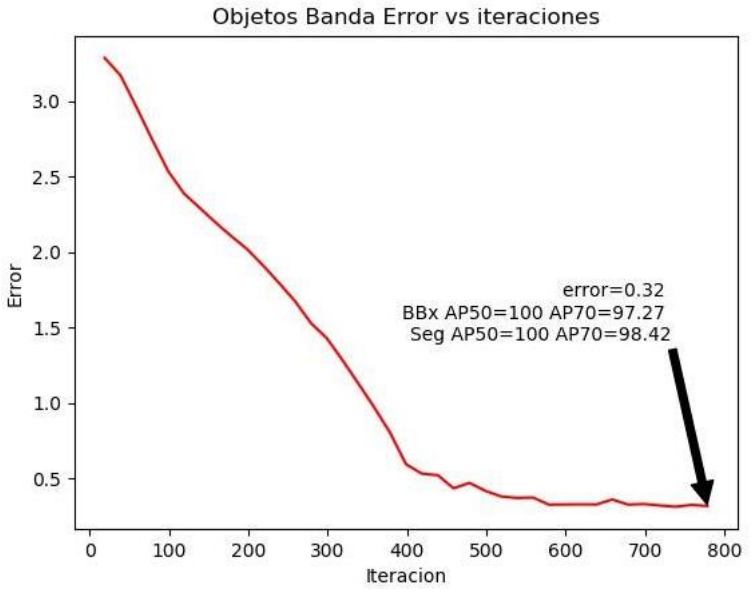
$$L_{rpn} = L_{cls} + L_{RoI} \tag{10}$$

El error total de las redes a lo largo del entrenamiento se puede ver en la Figura 19. Este es calculado sobre la base de datos de entrenamiento, mientras que el AP_{50} y AP_{70} mostrado en las gráficas es calculado sobre la base de datos de prueba. Como se puede observar en las imágenes, el error en ambas redes disminuye obteniéndose un desempeño adecuado. La red enfocada en detección de personas obtiene un desempeño perfecto en la detección del BBx y en la segmentación. Aunque la red enfocada en detección de objetos no alcanzó el 100, un valor por encima de 95 ya es robusto para esta aplicación. En la Figura 20 se muestra los resultados que se pueden obtener con las redes. Es importante señalar que la red de detección de personas mantuvo un desempeño perfecto a pesar de haber cambiado el entorno (presencia de paredes y ausencia de bandas transportadoras), por lo que se puede concluir que la red puede generalizar la detección de

personas. Estas redes lograron correr en promedio a 4fps (fotogramas por segundo), por lo que analizar una imagen les toma alrededor de 0.25s.



a)



b)

Figura 19 a) Entrenamiento de la red para detección y segmentación de personas b) Entrenamiento de la red para detección y segmentación de objetos.

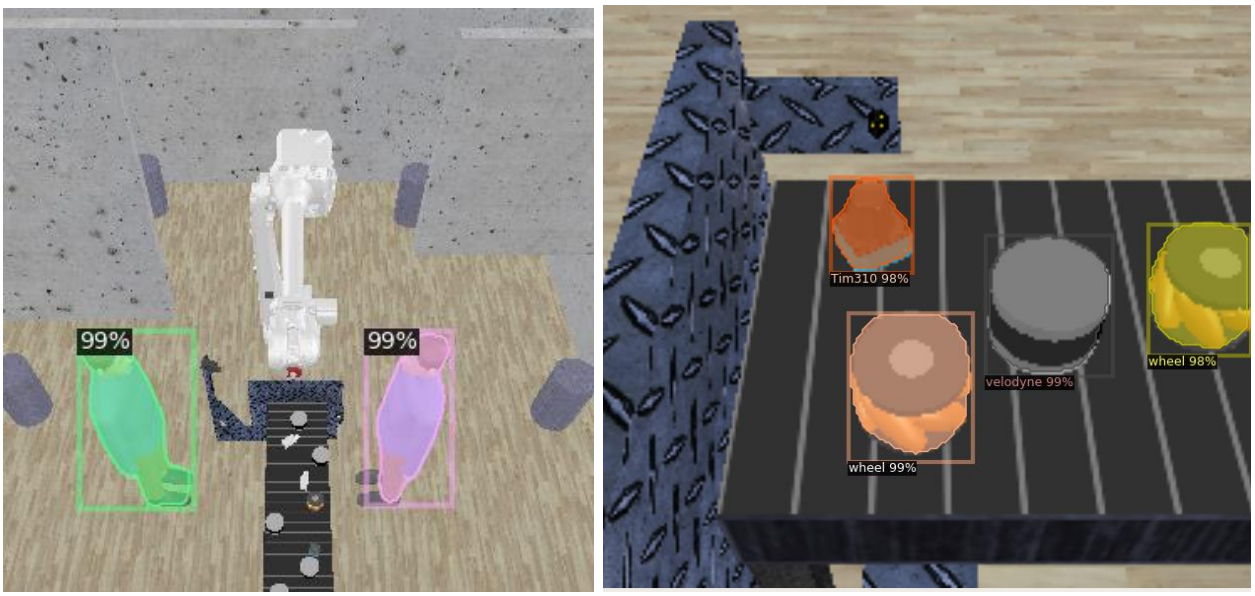


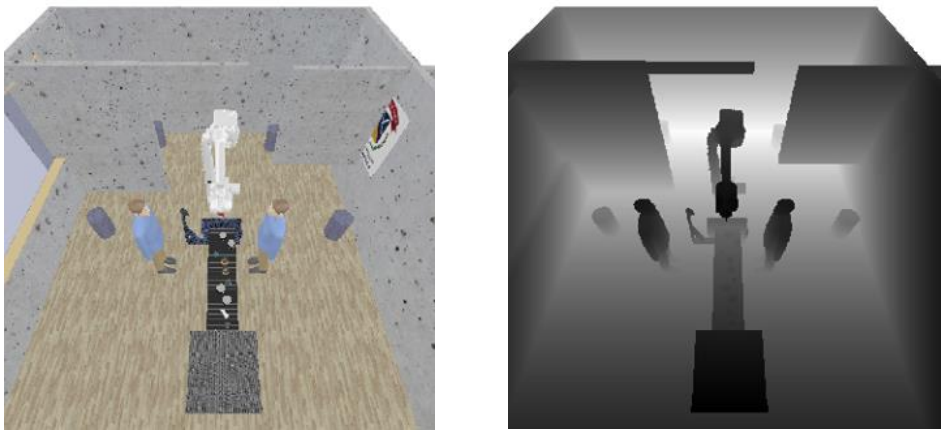
Figura 20 Resultados que se obtienen con las redes Mask R-CNN entrenadas.

Es importante aclarar que la red creada para detectar objetos en la banda transportadora tiene como función únicamente permitir una tarea de ordenamiento con el manipulador, esta puede omitirse al momento de crear el sistema de seguridad activo, el cual debe detectar obstáculos. La red que detecta personas en realidad lo que hace es detectar obstáculos que deben ser evitados con mayor importancia. Todo el entorno contiene obstáculos que debe detectar y evadir el robot, pero los que indica la Mask-RCNN son los que deben ser tratados con prioridad. En este caso sólo se toma en cuenta un tipo de obstáculo con prioridad, sin embargo, se puede incluir más categorías e incluir una diferente prioridad en la evasión. El manejo de prioridades para evasión de obstáculos es presentado en el capítulo 8.

6 Reconstrucción 3D y nube de puntos

Una tarea importante que debe hacer el sistema de seguridad es analizar si el robot colisiona con el entorno. Antes de realizar la planeación de movimiento es necesario conocer qué configuraciones no pueden ser alcanzadas. Por este motivo se hace indispensable analizar cómo se encuentra el entorno, lo cual se puede abordar mediante la nube de puntos. Esta se puede describir como un muestreo de la superficie del entorno que resulta en un conjunto de puntos que para esta situación representan obstáculos. Conociendo la nube de puntos, se pueden detectar todos los obstáculos presentes en el entorno. Este capítulo desarrolla el objetivo específico 2.

La nube de puntos puede ser creada a partir cámaras RGB-D. Estos sensores entregan cuatro datos por cada píxel en la imagen. Los primeros tres se relacionan con la intensidad de los colores rojo, verde y azul, mientras que el último indica la profundidad del píxel (ver Figura 21). El objetivo de esta sección es transformar esta información para crear la nube de puntos, lo cual da paso al análisis de colisiones. Para entender esta transformación se debe analizar el modelo que describe a la cámara.



a) **b)**

Figura 21 Datos generados por la cámara RGB-D a) Información RGB proveniente de los tres primeros canales b) Mapa de profundidad proveniente del último canal.

6.1 Modelo PinHole y proyección de perspectiva

El modelo más conocido se llama “PinHole” (Hartley & Zisserman, 2003) , el cual en esencia es una proyección de perspectiva (Woo, Neider, Davis, & Shreiner, 1999). También se puede encontrar con el nombre de “Frustum projection”. Este modelo asigna un sistema coordenado a la cámara y lo relacionado con el sistema inercial (sistema coordenado del robot) mediante una matriz

de transformación homogénea. Todo punto $\mathbf{P}$ que se encuentra definido respecto al sistema coordenado inercial es proyectado al plano de la imagen en las coordenadas (p_x, p_y) (ver Figura 22). La descripción matemática de esta proyección puede ser vista en (11) a (13).

El vector $\mathbf{P}$, el cual se encuentra en coordenadas homogéneas, es referenciado respecto al sistema coordenado de la cámara mediante la matriz homogénea $\mathbf{H}$ (ver ecuación (11)). Luego se realiza una proyección al plano de la imagen usando la matriz de parámetros intrínsecos $\mathbf{K}$ (ver ecuación (12)), la cual genera un vector que nuevamente está en coordenadas homogéneas. El objetivo de calibrar la cámara es calcular las matrices $\mathbf{H}$ y $\mathbf{K}$. Generalmente se suele representar el problema mediante sólo una matriz $\mathbf{T}$ (ver ecuación (13)), la cual luego de ser estimada se descompone en las matrices de interés.

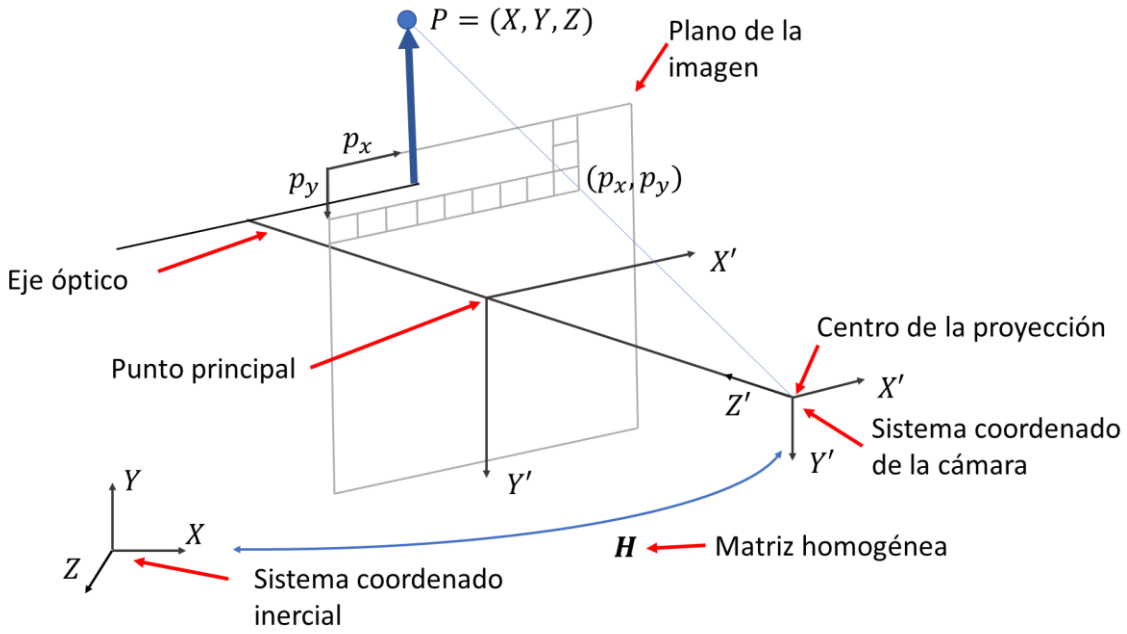


Figura 22 Modelo PinHole para describir la proyección de las superficies sobre el plano de la imagen.

$$\mathbf{P}' = \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \mathbf{HP} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{P} \tag{11}$$

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} p_x Z' \\ p_y Z' \\ Z' \end{bmatrix} = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_x & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \mathbf{KP}' \tag{12}$$

$$\mathbf{p} = \mathbf{TP} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]\mathbf{P} = \mathbf{KR}[\mathbf{I} \mid -\mathbf{X_o}]\mathbf{P} \tag{13}$$

De (12) se puede observar que, si se conocen los parámetros intrínsecos de la cámara y la profundidad asociada a cada píxel, se puede transformar pixeles a coordenadas cartesianas, es decir, se puede crear la nube de puntos del entorno respecto al sistema coordenado de la cámara. Esta nube de puntos puede ser referenciada respecto al sistema coordenado del robot usando la

matriz homogénea para así conocer la posición de los obstáculos respecto al robot. Por este motivo surge la necesidad de hallar la matriz de parámetros intrínsecos $\mathbf{K}$ y la matriz de parámetros extrínsecos $\mathbf{H}$. A continuación, se presentan dos formas de abordar este problema.

6.2 Métodos de Calibración de parámetros intrínsecos y extrínsecos

Cuando se desea calibrar una cámara lo más usual es implementar el método de Zhang (Zhang, Z., 2000). Este usa las intersecciones de los cuadros de un tablero de ajedrez para calibrar la matriz $\mathbf{T}$, volviéndolo un método muy flexible ya que los usuarios no requieren medir manualmente ningún punto. El inconveniente radica en que la matriz homogénea obtenida relaciona el sistema coordenado de la cámara con el del cuadro de ajedrez, y en este problema se debe relacionar con el de un robot. Por este motivo este método sólo debería usarse para calcular los parámetros intrínsecos ya que estos no dependen de la ubicación de la cámara. Existe otro método llamado DLT (Dubrofsky, 2009) que es muy similar al método de Zhang. La diferencia radica en que el método DLT usa mínimo 6 puntos los cuales no deben estar sobre un plano, por lo que es más aplicable a robots. A continuación, se presenta el método DLT y posteriormente se expone un enfoque desarrollado en esta investigación para calibrar los parámetros extrínsecos.

6.2.1 Método DLT (Direct Lineal Transform)

El método DLT inicia su planteamiento separando la matriz $\mathbf{T}$ en tres vectores $\mathbf{A}, \mathbf{B}$ y $\mathbf{C}$ (ver ecuación (14)) para luego manipular algebraicamente a (13) y así expresar el problema como se indica en (15), la cual usa los vectores mostrados en (16) y (17). La ecuación (15) contempla sólo un punto en el espacio, por lo que para tener en cuenta mínimo 6 se debe agregar más vectores como se indica en (18) y (19). Lo anterior provoca que (19) ya no sea igual a cero, por lo que se genera un vector de contradicciones $\mathbf{l}$. Para reducir estas contradicciones lo que se hace es plantear un problema de minimización que halle una matriz $\mathbf{T}$ diferente de cero que reduzca en lo posible el valor Ω (ver ecuación (20)). Este problema de minimización puede ser solucionado usando una descomposición por valores singulares (SVD) (Förstner & Wrobel, 2016) de la matriz $\mathbf{M}$, la cual genera tres matrices como las mostradas en (21). La solución estará dada por la columna de $\mathbf{V}$ que pertenece al menor valor singular de la matriz $\mathbf{S}$.

Una vez se tiene estimada a la matriz $\mathbf{T}$, esta se puede separar como se muestra en (22) y (23). El vector $\mathbf{X_0}$ se puede obtener como se indica en (24), mientras que la matriz de rotación y la matriz de parámetros intrínsecos se debe calcular de una descomposición QR como se muestra en (25).

$$\mathbf{T} = \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} & t_{1,4} \\ t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} \\ t_{3,1} & t_{3,2} & t_{3,3} & t_{3,4} \end{bmatrix} = \begin{bmatrix} \mathbf{A}^T \\ \mathbf{B}^T \\ \mathbf{C}^T \end{bmatrix} \tag{14}$$

$$\begin{bmatrix} \mathbf{a}_x^T \\ \mathbf{a}_y^T \end{bmatrix} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \end{bmatrix} = \mathbf{0} \tag{15}$$

$$\mathbf{a}_x^T = \begin{bmatrix} -X & -Y & -Z & -1 & 0 & 0 & 0 & 0 & p_x X & p_x Y & p_x Z & p_x \end{bmatrix} \tag{16}$$

$$\mathbf{a}_y^T = \begin{bmatrix} 0 & 0 & 0 & 0 & -X & -Y & -Z & -1 & p_y X & p_y Y & p_y Z & p_y \end{bmatrix} \tag{17}$$

$$\mathbf{M} = \begin{bmatrix} \mathbf{a}_{x1}^T \\ \mathbf{a}_{y1}^T \\ \mathbf{a}_{x2}^T \\ \mathbf{a}_{y2}^T \\ \vdots \\ \mathbf{a}_{xn}^T \\ \mathbf{a}_{yn}^T \end{bmatrix} \tag{18}$$

$$\mathbf{M} \begin{bmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{C} \end{bmatrix} = \mathbf{M}\mathbf{b} = \mathbf{l} \tag{19}$$

$$\hat{\mathbf{b}} = \arg\min_{\mathbf{b}} \Omega = \arg\min_{\mathbf{b}} \mathbf{l}^T \mathbf{l} \tag{20}$$

$$\mathbf{M} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{21}$$

$$\hat{\mathbf{H}}_\infty = \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,1} & t_{3,2} & t_{3,3} \end{bmatrix} \tag{22}$$

$$\hat{\mathbf{h}} = \begin{bmatrix} t_{1,4} \\ t_{2,4} \\ t_{3,4} \end{bmatrix} \tag{23}$$

$$\mathbf{X_o} = -\hat{\mathbf{H}}_\infty^{-1} \hat{\mathbf{h}} \tag{24}$$

$$-\hat{\mathbf{H}}_\infty^{-1} = \mathbf{R}^{-1}\mathbf{K}^{-1} \tag{25}$$

Es importante mencionar que para algunas situaciones es necesario hacer un ajuste en las matrices de rotación y de parámetros intrínsecos. Si se agrega una matriz identidad a (13) (ver ecuación (26)) y se descompone en dos matrices de rotación de 180 grados sobre un eje, como por ejemplo el eje "*y*", se puede obtener nuevas matrices de parámetros intrínsecos y extrínsecos como se indica en (27) y (28). En muchas situaciones estas nuevas matrices son las que se están buscando, por lo que cuando los signos de las matrices no parezcan estar bien, es necesario evaluar esto.

$$\mathbf{KIR}\left[\mathbf{I} \mid -\mathbf{X_o}\right]\mathbf{P} = \mathbf{KR}(y,\pi)\mathbf{R}(y,\pi)\mathbf{R}\left[\mathbf{I} \mid -\mathbf{X_o}\right]\mathbf{P} \tag{26}$$

$$\tilde{\mathbf{K}} = \mathbf{KR}(y,\pi) \tag{27}$$

$$\tilde{\mathbf{R}} = \mathbf{R}(y,\pi)\mathbf{R} \tag{28}$$

6.2.2 Calibración de parámetros extrínsecos (método propuesto)

El método que se explica a continuación es aplicable a situaciones en las que sólo se requiere calcular la matriz homogénea que relaciona dos sistemas coordenados. Se puede aplicar a problemas que no requieren hacer una calibración de parámetros intrínsecos, como es el caso de cámaras previamente calibradas o en situaciones en las que se desea hallar la matriz homogénea que relaciona los sistemas coordenados de varios manipuladores. Este es una alternativa al método DLT que aplica sólo cuando se desea hallar la relación espacial entre dos objetos.

Si existen dos sistemas coordenados a los cuales no se les conoce su relación espacial, pero se puede medir ciertos puntos respecto a estos, entonces el problema se puede representar como se indica en la Figura 23. Existen dos sistemas coordenados ***a*** y ***b*** los cuales conocen la posición de los puntos p_oa p_3. Si se asigna un sistema coordenado auxiliar **r** al punto p_o con la misma orientación que el sistema ***a***, se pueden escribir las relaciones mostradas en (29) y (30). En (29) la matriz homogénea está compuesta por una matriz identidad y el vector de traslación p_o el cual se mide respecto al sistema coordenado ***a***. Para poder hallar la matriz de interés ${}^{a}\mathbf{T}_{b}$ debe aplicarse la ecuación (30), la cual requiere conocer a la matriz ${}^{b}\mathbf{T}_{r}$. Si se calcula a ${}^{b}\mathbf{T}_{r}$, el problema queda solucionado.

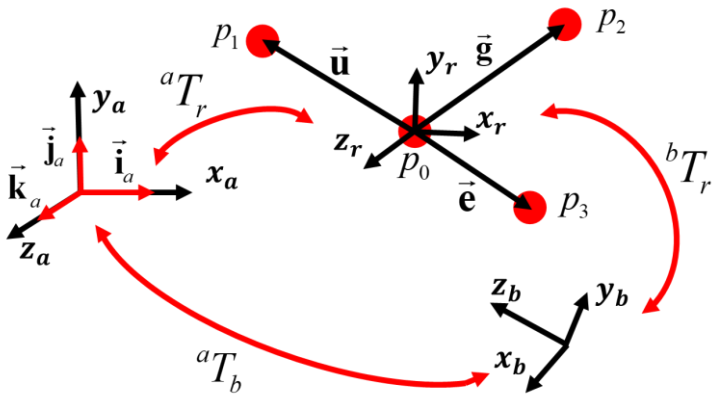


Figura 23 Sistemas coordenados de interés relacionados mediante matrices homogéneas.

$$ {}^{a}\mathbf{T}_{r} = \begin{bmatrix} \mathbf{I} & {}^{a}\mathbf{p}_{0} \\ \mathbf{0} & 1 \end{bmatrix} \tag{29} $$

$$ {}^{a}\mathbf{T}_{b} = {}^{a}\mathbf{T}_{r}\left({}^{b}\mathbf{T}_{r}\right)^{-1} \tag{30} $$

Cada sistema coordenado tiene 3 vectores unitarios proyectados sobre cada eje coordenado. La matriz ${}^{b}\mathbf{T}_{r}$ puede ser calculada si se poseen los vectores mostrados en (31), los cuales están medidos respecto al sistema ***b***. Como la matriz de rotación es ortogonal, no es necesario medir el vector unitario que está proyectado sobre el eje **z**, por el contrario, se deduce a partir de un producto cruz. El vector ${}^{b}\mathbf{p}_{0}$ se conoce ya que como se mencionó anteriormente, todos los puntos ya han sido medidos respecto a cada sistema coordenado. Como se verá a continuación, las únicas incógnitas ${}^{b}\vec{\mathbf{i}}_{r}$ y ${}^{b}\vec{\mathbf{j}}_{r}$ se pueden estimar usando una regresión lineal. Para hacer esto se debe usar

la ecuación del producto punto en conjunto con tres vectores auxiliares ***u*, g** y **e** (ver Figura 23). Estos vectores pueden ser medidos respecto al sistema coordenado **r** y volverse unitarios como se indica en (32) a (34). Como la orientación del sistema **r** y **a** son iguales, es válido usar los puntos medidos respecto al sistema **a**. Las componentes de estos vectores unitarios representan los cosenos directores de los vectores ***u*, g** y **e**, los cuales son usados en el producto punto con los vectores ${}^b\vec{\mathbf{i}}_r$, ${}^b\vec{\mathbf{j}}_r$ y ${}^b\vec{\mathbf{k}}_r$.

$$ {}^b\mathbf{T}_r = \begin{bmatrix} {}^b\mathbf{i}_r & {}^b\mathbf{j}_r & {}^b\mathbf{i}_r \times {}^b\mathbf{j}_r & {}^b\mathbf{p}_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{31} $$

$$ \mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \frac{{}^a\mathbf{u}}{\left|{}^a\mathbf{u}\right|} = \frac{{}^a\mathbf{p}_1 - {}^a\mathbf{p}_0}{\left|{}^a\mathbf{p}_1 - {}^a\mathbf{p}_0\right|} \tag{32} $$

$$ \mathbf{c} = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} = \frac{{}^a\mathbf{g}}{\left|{}^a\mathbf{g}\right|} = \frac{{}^a\mathbf{p}_2 - {}^a\mathbf{p}_0}{\left|{}^a\mathbf{p}_2 - {}^a\mathbf{p}_0\right|} \tag{33} $$

$$ \mathbf{d} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \frac{{}^a\mathbf{e}}{\left|{}^a\mathbf{e}\right|} = \frac{{}^a\mathbf{p}_3 - {}^a\mathbf{p}_0}{\left|{}^a\mathbf{p}_3 - {}^a\mathbf{p}_0\right|} \tag{34} $$

Si se conoce el coseno del ángulo de cada vector ***u*, g** y **e** respecto a cada vector ${}^b\vec{\mathbf{i}}_r$, ${}^b\vec{\mathbf{j}}_r$ y ${}^b\vec{\mathbf{k}}_r$, se pueden plantear los productos punto indicados en (35) y (36), los cuales representan un conjunto de ecuaciones en donde las únicas incógnitas son los vectores que deben usarse en la matriz de rotación de (31). Las ecuaciones (35) y (36) son un modelo lineal que puede escribirse como se indica en (37), en donde hay una variable explicativa $\mathbf{X}$ que puede entenderse como la entrada del sistema, unas variable de respuesta $\mathbf{Y}$ que son los valores esperados, unos parámetros o coeficientes $\boldsymbol{\beta}$ que deben ser estimados y un error aleatorio $\mathbf{e}$ que se supone con una distribución normal con media cero y varianza constante. El modelo aproximado de (37) se presenta en (38), el cual se implementa en un problema de minimización en el que se debe hallar los estimadores $\hat{\boldsymbol{\beta}}$ que reducen en lo posible la suma de los errores cuadráticos entre los valores esperados y estimados (ver ecuación (39)). La solución de este sistema se presenta en (40), en donde se implementa una pseudoinversa izquierda.

$$ \begin{bmatrix} \left|{}^b\mathbf{u}\right| v_x \\ \left|{}^b\mathbf{g}\right| c_x \\ \left|{}^b\mathbf{e}\right| d_x \end{bmatrix} = \begin{bmatrix} {}^b\vec{\mathbf{i}}_r \cdot {}^b\mathbf{u} \\ {}^b\vec{\mathbf{i}}_r \cdot {}^b\mathbf{g} \\ {}^b\vec{\mathbf{i}}_r \cdot {}^b\mathbf{e} \end{bmatrix} = \begin{bmatrix} {}^b\mathbf{u}^T \\ {}^b\mathbf{g}^T \\ {}^b\mathbf{e}^T \end{bmatrix} {}^b\vec{\mathbf{i}}_r \tag{35} $$

$$ \begin{bmatrix} \left|{}^b\mathbf{u}\right| v_y \\ \left|{}^b\mathbf{g}\right| c_y \\ \left|{}^b\mathbf{e}\right| d_y \end{bmatrix} = \begin{bmatrix} {}^b\vec{\mathbf{j}}_r \cdot {}^b\mathbf{u} \\ {}^b\vec{\mathbf{j}}_r \cdot {}^b\mathbf{g} \\ {}^b\vec{\mathbf{j}}_r \cdot {}^b\mathbf{e} \end{bmatrix} = \begin{bmatrix} {}^b\mathbf{u}^T \\ {}^b\mathbf{g}^T \\ {}^b\mathbf{e}^T \end{bmatrix} {}^b\vec{\mathbf{j}}_r \tag{36} $$

$$ \mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{e} \tag{37} $$

$$\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\beta}} \tag{38}$$

$$\hat{\boldsymbol{\beta}} = \arg\min_{\hat{\boldsymbol{\beta}}\in\mathbb{R}^3} \Omega = \arg\min_{\hat{\boldsymbol{\beta}}\in\mathbb{R}^3} \left(\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}\right)^T \left(\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}\right) \tag{39}$$

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y} \tag{40}$$

Como se puede observar, en la metodología planteada es necesario conocer mínimo 4 puntos en el espacio que puedan ser medidos respecto a los dos sistemas coordenados de interés. La limitación de este método radica en que todos los cálculos se apoyan de un vector ${}^b\mathbf{p}_0$, por lo que este debe tener el menor error posible. El método DLT toma en cuenta todos los puntos, pero no se apoya de uno solo para hacer todos los cálculos, por lo que el resultado final depende en gran medida de la exactitud de ${}^b\mathbf{p}_0$. También es importante resaltar que el método DLT calcula al tiempo los parámetros intrínsecos, mientras que el método explicado solo puede ser usado para calcular la matriz homogénea que relaciona dos sistemas, no una matriz de proyección. Ambos métodos estiman los parámetros mediante un problema de minimización, con la diferencia de que DLT debe tratar con un sistema homogéneo que posee la solución trivial. Una característica importante del método propuesto es que no es necesario analizar si la orientación de la matriz de rotación es la adecuada. Debido a que no se involucra a los parámetros intrínsecos, el resultado obtenido es inmediatamente la solución esperada.

Poseer la matriz de parámetros intrínsecos y extrínsecos permite usar las ecuaciones (11) y (12) para realizar la reconstrucción 3D del entorno a partir de la información de la coordenada de cada píxel y la profundidad de este. Si un sistema de segmentación semántica indica que ciertos pixeles son de interés, se puede hallar las coordenadas cartesianas de estos y por ende generar nubes de puntos en donde sólo se encuentran los objetos detectados.

6.3 Calibración y generación de nube de puntos

Para poder calibrar la cámara en primer lugar se necesita la posición de mínimo seis puntos. Estos se deben medir respecto al sistema coordenado inercial y respecto al sistema coordenado de la imagen. Si se agrega una esfera roja al gripper del robot para generar los puntos, se puede calcular la posición de esta respecto al sistema inercial a partir de la cinemática directa, la posición respecto a la imagen se puede calcular a partir de filtros morfológicos o redes convolucionales de detección. Las posiciones que fueron capturadas dentro del simulador se pueden ver en la Tabla 3 y Tabla 4. La Tabla 3 indica las coordenadas de los puntos respecto al sistema inercial, mientras que la Tabla 4 indica las coordenadas de los puntos respecto a la imagen. Todos los datos de una fila de la tabla son los puntos usados para calibrar a una cámara

Se está trabajando con tres cámaras. Dos de ellas rodean al robot mientras que una está cerca a los objetos que llegan de la banda transportadora (ver Figura 11). De esta manera se logra abarcar todo el espacio de configuración reduciendo en gran medida los puntos ciegos.

Los resultados obtenidos con el método DLT se muestran en la Tabla 5. A cada cámara se le ha indicado las matrices obtenidas y el error de la minimización que proviene de (20). Los resultados

poseen un error muy bajo debido a que estas mediciones fueron hechas dentro de un simulador. En un entorno real efectos no lineales debido al lente de la cámara pueden afectar los resultados.

Tabla 3 Puntos medidos respecto al sistema inercial

Cámara\Puntos	P_0[m]	P_1[m]	P_2[m]	P_3[m]	P_4[m]	P_5[m]
Cámara1, Cámara 2	$\begin{bmatrix}-0.9\\-0.65\\1.625\end{bmatrix}$	$\begin{bmatrix}0.125\\-1.25\\2.05\end{bmatrix}$	$\begin{bmatrix}0.65\\0.525\\1.475\end{bmatrix}$	$\begin{bmatrix}-0.675\\1.125\\1.975\end{bmatrix}$	$\begin{bmatrix}0.675\\1.05\\2.275\end{bmatrix}$	$\begin{bmatrix}-0.025\\-1.675\\2.375\end{bmatrix}$
Cámara 3	$\begin{bmatrix}0.7\\0.15\\0.8\end{bmatrix}$	$\begin{bmatrix}0.875\\0.125\\0.975\end{bmatrix}$	$\begin{bmatrix}0.825\\-0.1\\1.125\end{bmatrix}$	$\begin{bmatrix}0.975\\-0.0675\\1.025\end{bmatrix}$	$\begin{bmatrix}0.95\\-0.002\\0.525\end{bmatrix}$	$\begin{bmatrix}1.05\\0.275\\0.45\end{bmatrix}$

Tabla 4 Puntos medidos respecto al plano de la imagen

Cámara\Puntos	P_0[px]	P_1[px]	P_2[px]	P_3[px]	P_4[px]	P_5[px]
Cámara1	$\begin{bmatrix}372.02\\356.27\end{bmatrix}$	$\begin{bmatrix}428.93\\241.27\end{bmatrix}$	$\begin{bmatrix}256.85\\243.89\end{bmatrix}$	$\begin{bmatrix}169.94\\314.37\end{bmatrix}$	$\begin{bmatrix}196.72\\186.07\end{bmatrix}$	$\begin{bmatrix}491.49\\227.16\end{bmatrix}$
Cámara 2	$\begin{bmatrix}222.59\\160.72\end{bmatrix}$	$\begin{bmatrix}121.497\\189.069\end{bmatrix}$	$\begin{bmatrix}341.22\\285.12\end{bmatrix}$	$\begin{bmatrix}400.83\\141.61\end{bmatrix}$	$\begin{bmatrix}431.45\\213.83\end{bmatrix}$	$\begin{bmatrix}52.706\\140.23\end{bmatrix}$
Cámara 3	$\begin{bmatrix}139.47\\100.89\end{bmatrix}$	$\begin{bmatrix}279.22\\27.61\end{bmatrix}$	$\begin{bmatrix}563.08\\12.39\end{bmatrix}$	$\begin{bmatrix}544.73\\112.48\end{bmatrix}$	$\begin{bmatrix}241.3\\228.1\end{bmatrix}$	$\begin{bmatrix}237.23\\145.61\end{bmatrix}$

Tabla 5 Resultados obtenidos con el método DLT

Cámara\ Puntos	K	H	error Ω
Cámara 1	$\begin{bmatrix}360.07 & 1.68x10^{-10} & 303.141\\0 & 360.329 & 302.595\\0 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}5.35x10^{-7} & -1 & 3.74x10^{-7} & -0.025\\-0.819 & -6.53x10^{-7} & -0.573 & 0.681\\0.573 & 5.9x10^{-13} & -0.819 & 5.113\\0 & 0 & 0 & 1\end{bmatrix}$	1.37×10^{-26}
Cámara 2	$\begin{bmatrix}360.45 & 1.28x10^{-10} & 302.83\\0 & 360.364 & 302.83\\0 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}-5.35x10^{-7} & 1 & 3.74x10^{-7} & -0.199\\0.819 & 6.53x10^{-7} & -0.573 & 0.163\\-0.573 & -4.35x10^{-13} & -0.819 & 4.633\\0 & 0 & 0 & 1\end{bmatrix}$	1.66×10^{-27}
Cámara 3	$\begin{bmatrix}268.12 & 8.39x10^{-12} & 112.39\\0 & 268.18 & 112.25\\0 & 0 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 5x10^{-7} & 4.2x10^{-7} & -0.648\\6.53x10^{-7} & -0.766 & -0.642 & 0.607\\-2.45x10^{-14} & 0.642 & -0.766 & 1.031\\0 & 0 & 0 & 1\end{bmatrix}$	1.71×10^{-29}

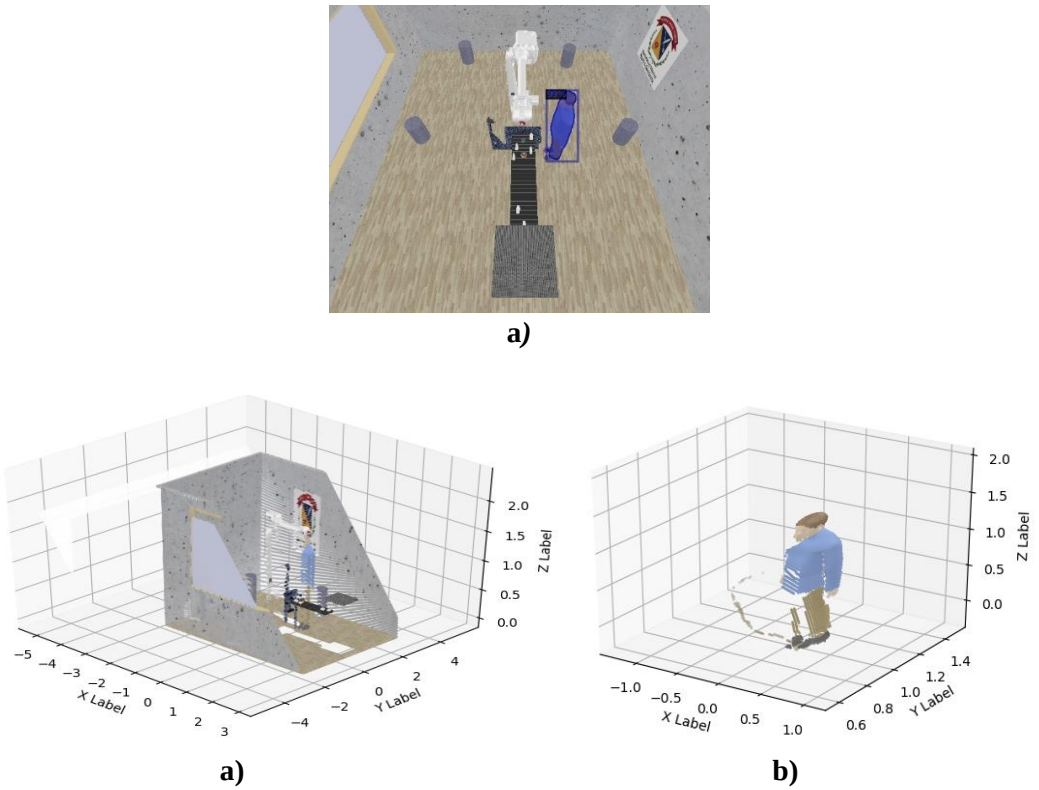
Para poner a prueba el método propuesto se deben medir al menos tres puntos de la Tabla 3 respecto al sistema coordenado de cada cámara. Usando las matrices homogéneas calculadas en la Tabla 5 se transforman todos estos puntos al sistema de la cámara y luego se aplica el método para

calcular las matrices homogéneas. El resultado debe ser la misma matriz homogénea usada para transformar los puntos. Lo que se pudo observar es que el método funciona satisfactoriamente con errores mínimos ya que los datos no poseen algún error asociado a su medición. Debido a que las matrices halladas son esencialmente las mismas que se muestran en la Tabla 5, sólo se indica en la Tabla 6 los errores obtenidos. Estos errores se calculan según como se plantea la función objetivo en (39). Cada cámara obtuvo errores que pueden ser considerados iguales a cero, lo que significa que la calibración fue satisfactoria.

Tabla 6 Errores al hallar las matrices homogéneas con el método propuesto

Error/cámara	Cámara 1	Cámara 2	Cámara 3
Error Ω	7.42×10^{-31}	4.61×10^{-31}	1.61×10^{-31}

Debido a que (11) y (12) ya están completamente definidas, se puede transformar pixeles a puntos medidos respecto al sistema coordenado inercial. Esto puede hacerse ya sea para toda una imagen o para los pixeles segmentados por la Mask R-CNN (ver Figura 24.b). La Figura 24.a contiene la información de todos los obstáculos presentes en el entorno. Es claro que parte de la información obtenida representa al robot, por lo que es necesario filtrar estos puntos para que sólo queden los que representan obstáculos, esto se trata en el siguiente capítulo. Lo importante a tener en cuenta de los resultados es que se detectan todos los obstáculos en la Figura 24.a y con las Mask R-CNN se obtienen los obstáculos de importancia.



a)

a) **b)**

Figura 24 Creación de la nube de puntos. a) Imagen capturada del entorno. b) Transformación de toda la imagen a nube de puntos. c) Transformación de los pixeles segmentados a nube de puntos.

7 Modelado del entorno operativo, cálculos cinemáticos y análisis de colisiones

Modelar el entorno puede describirse como el uso de herramientas matemáticas para describir el estado de los elementos. En robótica el modelado del entorno operativo se enfoca en describir la posición y orientación de los elementos, y la forma más aceptada para desarrollar esto es a partir de matrices de transformación homogénea, las mismas usadas en la calibración de las cámaras RGB-D. En este desarrollo es necesario caracterizar la posición de varios objetos del entorno como lo son los eslabones que conforman al robot, los objetos que se deben manipular, los obstáculos, los contenedores donde se depositan los objetos y las cámaras RGB-D. Conocer esta información permite analizar qué configuraciones debe tomar el robot para manipular un objeto, o qué configuraciones no debe alcanzar para evitar colisiones. El análisis de colisiones permite restringir el espacio de configuración excluyendo todos los vectores de estado que generan colisiones, lo que permite crear un subconjunto con el cual el algoritmo de planeación de trayectoria produce resultados factibles.

Este capítulo analiza la cinemática directa, el filtrado de la nube de puntos, el análisis de colisiones y la cinemática inversa del robot. Los análisis realizados son fundamentales para permitir la creación de grafos con los cuales se hace la planeación de movimiento. En resumen, en este capítulo da las herramientas requeridas para trabajar en el capítulo 8 la generación de grafos. Además, se presenta un enfoque diferente al convencional para calcular la cinemática inversa, el cual produce resultados más exactos, pero a un mayor costo computacional. Este capítulo desarrolla el objetivo específico 3.

7.1 Matrices homogéneas y cinemática directa

La cinemática directa del robot se genera a partir de la composición de matrices homogéneas como se muestra en (41). Cada matriz homogénea relaciona dos sistemas coordenados. Por ejemplo, la matriz ${}^{b}\mathbf{T}_{6}(\mathbf{q})$, que depende del vector de estado del robot $\mathbf{q}$, relaciona el sistema coordenado 6 con el inercial b. Este vector de estado está formado por 6 componentes (ver ecuación (42)), uno asociado a cada grado de libertad. La cinemática directa se basa en calcular la ecuación (41) para una configuración específica del robot y así conocer cuál es la ubicación y orientación del gripper en el sistema inercial. La definición de estas matrices generalmente se hace a partir de la convención Denavit-Harnenberg (Cisneros et al., 2014), convención que vale la pena implementar para facilitar la creación de las matrices y el cálculo de la cinemática inversa por el método convencional. Para poder realizar el filtrado de la nube de puntos y analizar el cálculo de las colisiones se plantea agregar un prisma rectangular a cada articulación como se ve en la Figura 25. Se han asignado siete prismas, y cada sistema coordenado se ha ubicado en el centro de estos. La matriz homogénea asociada al prisma *i* debe calcularse como se indica en (43). En esta ecuación

se debe utilizar la cinemática directa hasta el sistema coordenado *i* y la matriz homogénea que relaciona al prisma con el sistema coordenado *i*.

$$ {}^{b}\mathbf{T}_{6}(\mathbf{q}) = {}^{b}\mathbf{T}_{0}\,{}^{0}\mathbf{T}_{1}(q_0)\,{}^{1}\mathbf{T}_{2}(q_1)\,{}^{2}\mathbf{T}_{3}(q_2)\,{}^{3}\mathbf{T}_{4}(q_3)\,{}^{4}\mathbf{T}_{5}(q_4)\,{}^{5}\mathbf{T}_{6}(q_5) \tag{41} $$

$$ \mathbf{q} = \left[q_0, q_1, q_2, q_3, q_4, q_5 \right]^{T} \tag{42} $$

$$ {}^{b}\mathbf{T}_{pi}(\mathbf{q}) = {}^{b}\mathbf{T}_{i}(\mathbf{q})\,{}^{i}\mathbf{T}_{pi} \tag{43} $$

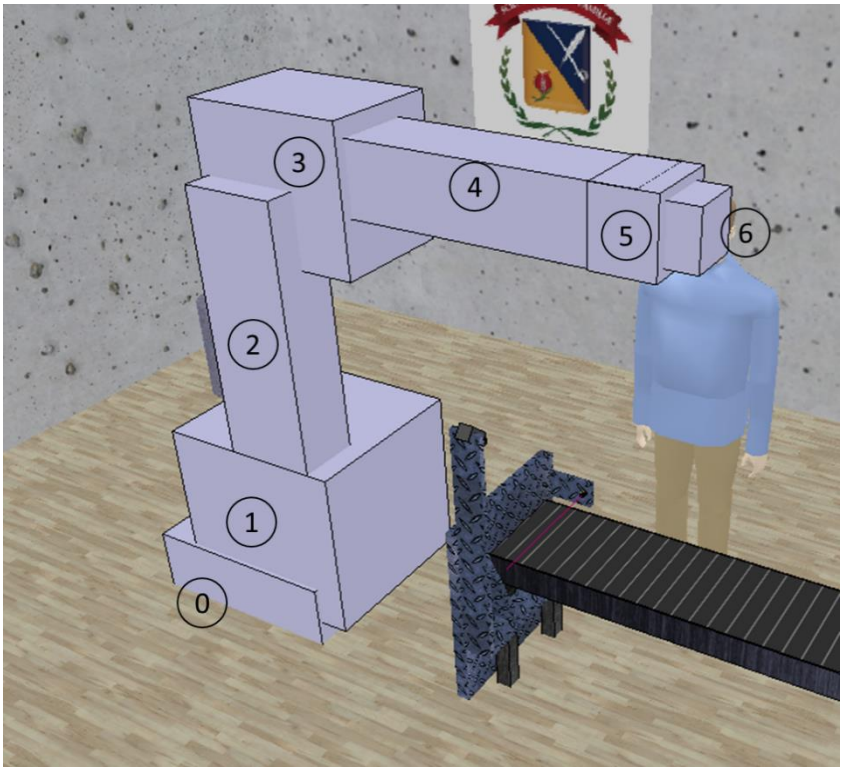


Figura 25 Prismas rectangulares agregados a cada articulación del robot.

A partir de las ecuaciones anteriores se describe por completo al sistema robótico y a cada uno de sus eslabones. Los obstáculos del entorno se describen a partir de puntos como se indicó el capítulo anterior, por lo que no es necesario definir ninguna matriz para estos. A los contenedores y a los objetos de la banda se les debe asignar una matriz homogénea para que el robot tenga una referencia a la cual dirigir el gripper, a esta matriz se le representa como se indica en (44). A los objetos de la banda transportadora se les conoce únicamente la nube de puntos, por lo que se plantea que la matriz (44) en dicho caso contenga una matriz de rotación igual a la identidad y un vector de posición equivalente al centroide de los puntos superiores del objeto (ver Figura 26). En esta investigación no se pretende analizar a profundidad las formas de definir a (44) según la nube de puntos de los objetos. Este es un problema de alto nivel que incluso ha creado retos como el desafío de agarre de Amazon (Correll et al., 2016b).

$$ {}^{b}\mathbf{T}_{d} = \begin{bmatrix} {}^{b}\mathbf{n}_{d} & {}^{b}\mathbf{o}_{d} & {}^{b}\mathbf{a}_{d} & {}^{b}\mathbf{t}_{d} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{44} $$

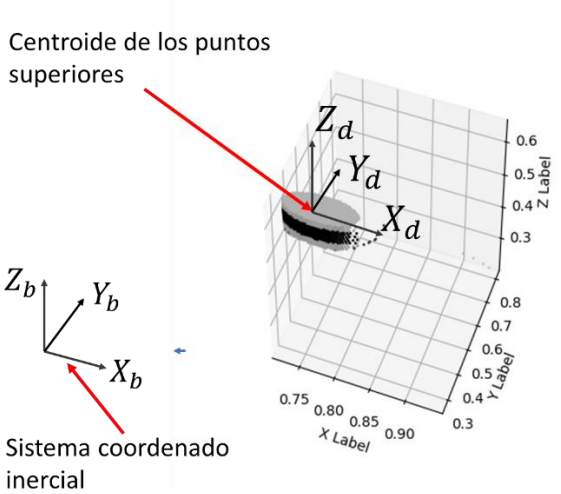


Figura 26 Sistema coordenado en objeto de la banda.

7.2 Filtrado de la nube de puntos

El primer filtro que se aplica elimina todos los puntos que se encuentren fuera del área de trabajo del robot. A cada punto se le debe calcular su hipotenusa, si esta es mayor al radio que abarca el espacio de trabajo, entonces este punto debe ser eliminado. Esto permite reducir la cantidad de información que se debe analizar.

El segundo filtro se enfoca en eliminar todos los datos que representan a la superficie del robot. Para hacer esto se debe referenciar toda la información respecto al sistema coordenado de cada prisma (ver ecuación (45)). Todo punto ${}^{pi}P$ que esté contenido dentro del prisma debe ser eliminado. Esto se puede evaluar rápidamente a partir de las ecuaciones (46) a (48). Si para un punto con coordenadas (*x*,*y*,*z*) todas las ecuaciones (46) a (48) dan falsas, significa que el punto debe ser eliminado. Las constantes *w*, *h* y *d* indican el ancho, alto y largo del prisma. La nube de puntos resultante se muestra en la figura 26.

$$ {}^{pi}P = \left({}^{b}\mathbf{T}_{pi}(\mathbf{q}) \right)^{-1} {}^{b}P \tag{45} $$

$$ |x| \geq w/2 \tag{46} $$

$$ |y| \geq h/2 \tag{47} $$

$$ |z| \geq d/2 \tag{48} $$

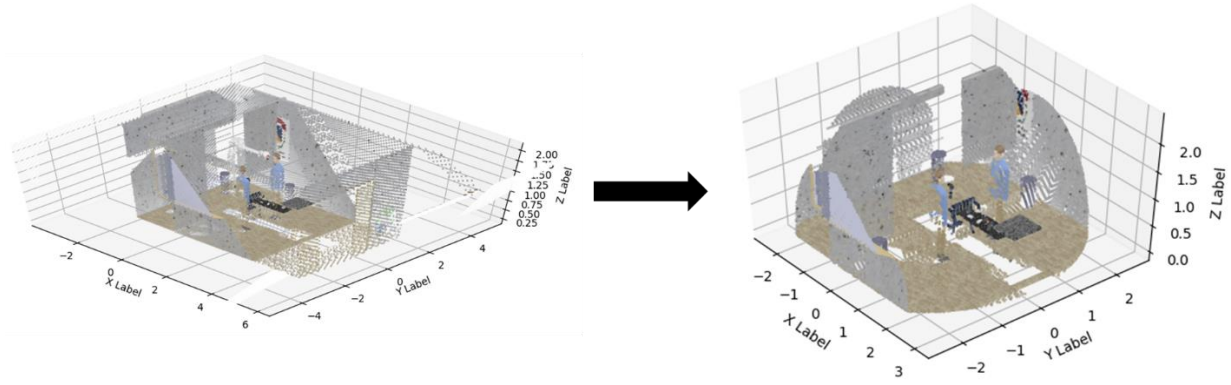


Figura 27 Filtrado de la nube de puntos.

Este proceso debe hacerse para cada nube de puntos. Si un punto que representa al robot es tomado como un obstáculo, el analizador de colisiones generará resultados erróneos que provocarán que la planeación de movimiento se calcule mal o que incluso no se encuentren soluciones.

7.3 Análisis de colisiones

El objetivo de analizar las colisiones es poder conocer las configuraciones que no debe alcanzar el robot. La planeación de movimiento debe contener únicamente vectores de estado que no produzcan colisiones, por lo que el algoritmo que genera este resultado debe conocer en primer lugar qué configuraciones no puede usar. Existen dos tipos de colisiones que deben evaluarse, estas se han llamado colisiones internas y externas. Una colisión interna se genera cuando el robot colisiona consigo mismo, mientras una colisión externa se genera cuando el robot colisiona con el entorno (ver Figura 28). Los prismas asignados a cada eslabón permiten calcular si para una configuración el robot presenta alguna de las dos colisiones mencionadas.

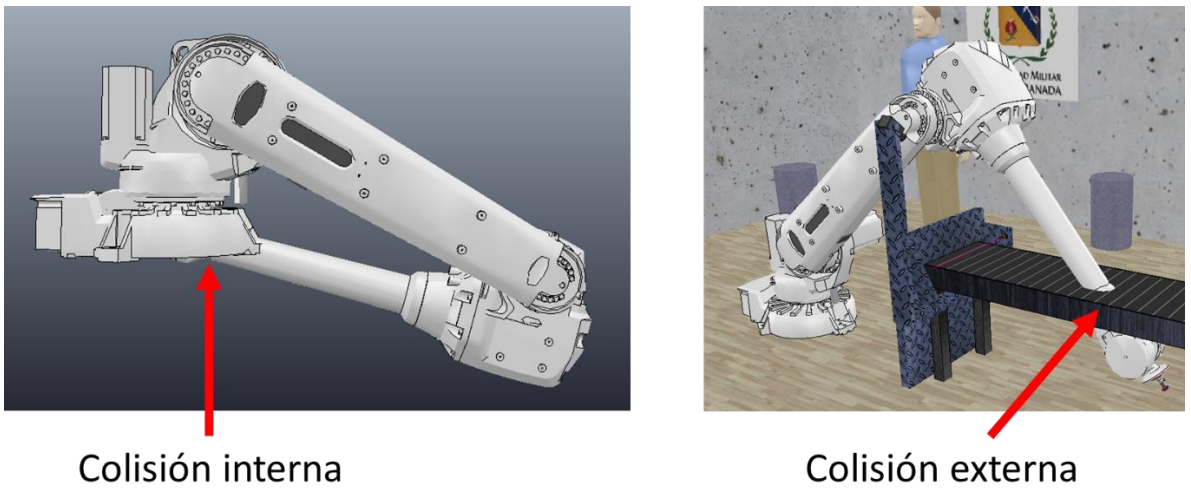


Figura 28 Colisión interna y externa en el robot.

La razón por la que se ha escogido prismas rectangulares para representar a los eslabones se debe a que el análisis de colisiones para estas figuras es más rápido que para otra forma geométrica convexa. El analizador de colisiones externas usa la ecuación (45) para referenciar la nube de puntos respecto a cada prisma, luego revisa si algún punto no cumple todas las ecuaciones (46) a (48). Si un punto no cumple las tres ecuaciones, entonces el analizador indicará que el robot colisiona con el entorno.

Para evaluar las colisiones internas es necesario usar el teorema de los ejes separados. Este no debe ser confundido con el teorema de los ejes paralelos, el cual es usado en mecánica para transformar segundos momentos de área o momentos de inercia. El teorema de los ejes separados es descrito en (Huynh, 2009). Este usa los vectores que se indican en la Figura 29. Cada prisma rectangular tiene un sistema coordenado en el centro, el cual contiene un vector que se dirige hasta una esquina del prisma. La distancia entre los dos sistemas coordenados se caracteriza a través del vector ***T***. Este teorema plantea que existe un plano separador entre los dos prismas si y solo si existe un eje ***L*** para el cual se cumple la ecuación (49). El vector ***L*** representa un eje al cual se le está evaluando la existencia de un plano separador. ***L*** se debe reemplazar por cada eje de los dos sistemas coordenados, además, se debe reemplazar por el vector resultante del producto cruz entre cada uno de los vectores unitarios del sistema coordenado "A" con cada uno de los vectores unitarios del sistema coordenado "B". En total ***L*** debe ser reemplazado por 15 ejes. Si para un eje la ecuación (49) se cumple, se dice que dicho eje contiene un plano separador y por lo tanto los dos prismas no colisionan.

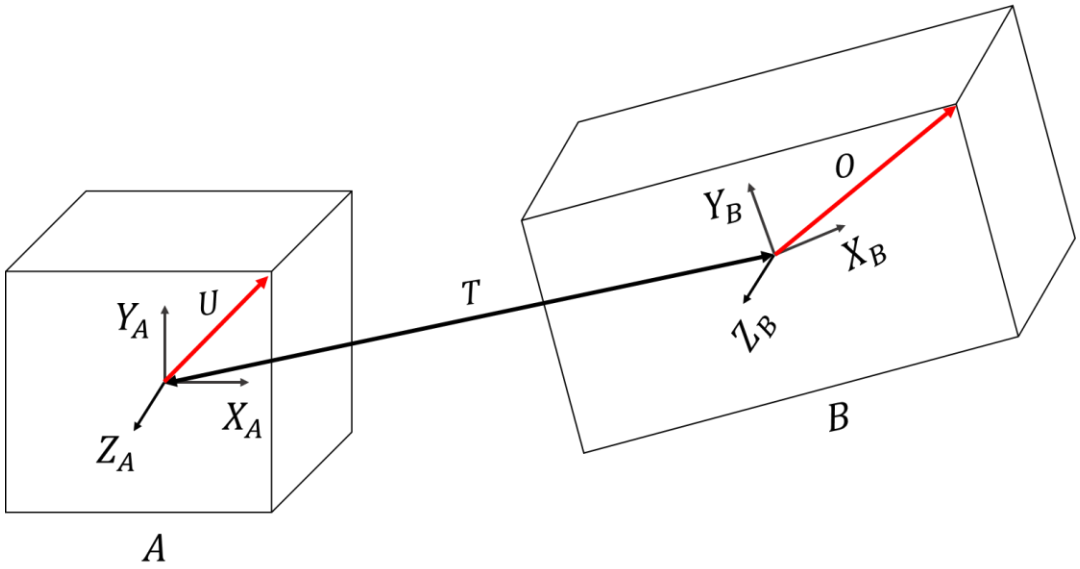


Figura 29 Vectores usados en el teorema de los ejes separados.

$$|\mathbf{T}\cdot\mathbf{L}| > |\mathbf{U}\cdot\mathbf{L}| + |\mathbf{O}\cdot\mathbf{L}| \tag{49}$$

Cada vector mostrado en la Figura 29 se puede hallar a partir de las matrices homogéneas asociadas a los prismas. La matriz de rotación contiene los vectores unitarios mientras que el vector de traslación contiene la posición del sistema coordenado. Por lo que para una configuración específica la ecuación (43) contiene toda la información necesaria para proceder a calcular este teorema. El análisis de colisiones no se debe efectuar entre todos los prismas, ya que prismas consecutivos generarían colisiones para cualquier posición, como el prisma cero con el prisma uno. Se supone que prismas consecutivos no colisionan. En este desarrollo las colisiones se evalúan entre los prismas cero y uno con los prismas cuatro, cinco y seis.

7.4 Métodos para calcular la cinemática inversa

Cuando se desea desplazar el gripper del robot a una posición y orientación específica, es necesario aplicar la cinemática inversa. Esta debe generar un vector de estado que reduzca en lo posible el error que existe entre la matriz de la ecuación (41) con la matriz de la ecuación (44). Existen varios métodos para hacer esto; sin embargo, en esta sección sólo se habla de dos. El primero de estos es el método clásico, el cual implementa la cinemática diferencial para calcular iterativamente la solución. Este relaciona a través de un jacobiano las velocidades articulares con las velocidades que experimenta el gripper, y a partir de esta relación puede calcular un vector que reduce el error entre los sistemas coordenados. El segundo método que se presenta es desarrollado en esta investigación y es una alternativa que genera resultados más exactos. Este al igual que el método clásico es iterativo y usa un jacobiano, sin embargo, este no se relaciona con la cinemática diferencial.

7.4.1 Método clásico

El método clásico calcula la cinemática inversa de un robot de manera iterativa. Este actualizar el vector de estado $\mathbf{q}$ en función a un $\Delta\mathbf{q}$ (ver ecuación (50)). En esta expresión se observa que el valor actualizado del vector de estado $\mathbf{q}_{k+1}$ es igual a su valor actual $\mathbf{q}_k$ más un cambio definido por el vector de velocidad $\mathbf{W}_k$, la pseudo inversa de la matriz jacobiana $\mathbf{J}^+$ y un vector de pesos $\boldsymbol{\alpha}$ que se encarga de controlar la velocidad de cambio de las articulaciones. Esta ecuación proviene de la cinemática diferencial del robot, en donde se relaciona las velocidades angulares y articulares. Para aplicar el método es necesario calcular un vector de velocidades tal que se reduzca el error entre el sistema coordenado del gripper (ver ecuación (51)) y el deseado (ver ecuación (44)).

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta\mathbf{q} = \mathbf{q}_k + \boldsymbol{\alpha}^T \left(\mathbf{J}^+ \mathbf{W}_k \right) \tag{50}$$

$${}^b\mathbf{T}_n = \begin{bmatrix} {}^b\mathbf{n}_n & {}^b\mathbf{o}_n & {}^b\mathbf{a}_n & {}^b\mathbf{t}_n \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^b\mathbf{T}_6 \tag{51}$$

Para lograr reducir el error entre los sistemas coordenados, es necesario reducir la distancia entre estos al mismo tiempo que se orientan los tres vectores de la matriz de rotación. Esto se puede

realizar si el vector de velocidades se define según como se expresa en la ecuación (52), la cual indica que el vector de velocidad lineal $\mathbf{d}$ es el error de posición, mientras que el vector de velocidad angular $\boldsymbol{\omega}$ es definido por el producto cruz que se ha indicado (deducción que proviene de la fórmula de Rodríguez (Cisneros et al., 2014)). De esta manera aplicando la ecuación (50) iterativamente, es decir volviendo el vector de estado futuro el actual, se logra obtener el valor de las coordenadas generalizadas y por lo tanto la configuración que debe tener el robot.

$$\mathbf{W}_k = \begin{bmatrix} \mathbf{d} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} {}^b\mathbf{t}_d - {}^b\mathbf{t}_n \\ {}^b\mathbf{n}_n \times {}^b\mathbf{n}_d + {}^b\mathbf{o}_n \times {}^b\mathbf{o}_d + {}^b\mathbf{a}_n \times {}^b\mathbf{a}_d \end{bmatrix} \tag{52}$$

La forma convencional para el cálculo del jacobiano es tomada de (Cisneros et al., 2014) , el cual es descrito como se indica en (53). Este tiene varios vectores, uno por cada grado de libertad, y cada uno de estos vectores es definido según la naturaleza de la articulación a la que hacen referencia. Si la articulación es rotacional debe usarse la ecuación (54), por el contrario, si es prismática debe usarse la ecuación (55). El vector ${}^b\mathbf{z}_{i-1}$ se encuentra ubicado sobre el eje de rotación o el eje de traslación de la articulación, por lo que debe usarse la convención Denavit-Hartenberg (DH). El superíndice indica que el vector ha sido referenciado respecto al sistema base, mientras que el subíndice indica el sistema coordenado al que pertenece, el cual proviene del eslabón i .

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_0 & \mathbf{J}_1 & \cdots & \mathbf{J}_{n-1} \end{bmatrix} \tag{53}$$

$$\mathbf{J}_i = \begin{bmatrix} {}^b\mathbf{z}_{i-1} \times \left({}^b\mathbf{t}_n - {}^b\mathbf{t}_{i-1} \right) \\ {}^b\mathbf{z}_{i-1} \end{bmatrix} \tag{54}$$

$$\mathbf{J}_i = \begin{bmatrix} {}^b\mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix} \tag{55}$$

7.4.2 Método propuesto

El método que se propone en este documento se basa en aplicar el método numérico Newton-Raphson(Chapra, 2012) a una serie de ecuaciones que se relacionan con la cinemática directa del robot y las derivadas de las matrices que la componen. Este método no usa la cinemática diferencial del robot, el jacobiano que se implementa proviene de una función a la cual se le está hallando las raíces. Lo que se logra apreciar en el método propuesto es que se obtienen resultados más exactos en comparación con el método clásico.

Para explicar el método propuesto en primer lugar se define el método numérico Newton-Raphson para una función vectorial multivariable. Sea $f(\mathbf{X})$ (ver ecuación (56)) una función vectorial multivariable no lineal a la cual se le desean hallar las raíces. Esta depende de un vector $\mathbf{X}$ (ver ecuación (57)) que representan las variables del sistema, que para este caso específico son las coordenadas generalizadas de un robot de 6 DOF. La función puede ser aproximada por series de

Taylor a una función lineal que actúa sobre un punto de operación $\mathbf{a}$ como se indica en la (58). Si se despeja el vector $\mathbf{X}$ con $f(\mathbf{X})$ igual a un vector nulo, se obtiene la ecuación (59), la cual puede usarse iterativamente para hallar las raíces del sistema (ver ecuación (60)). Note que se ha cambiado la inversa del jacobiano por su pseudo inversa para evitar problemas de invertibilidad.

$$f(\mathbf{X}) = \begin{bmatrix} f_1(\mathbf{X}) \\ f_2(\mathbf{X}) \\ \vdots \\ f_N(\mathbf{X}) \end{bmatrix} \tag{56}$$

$$\mathbf{X} = \begin{bmatrix} x_1, x_2, \cdots, x_n \end{bmatrix}^T = \mathbf{q} = \begin{bmatrix} \mathrm{q}_0, \mathrm{q}_1, \mathrm{q}_2, \mathrm{q}_3, \mathrm{q}_4, \mathrm{q}_5 \end{bmatrix}^T \tag{57}$$

$$f(\mathbf{X}) \simeq \begin{bmatrix} f_1(\mathbf{a}) \\ f_2(\mathbf{a}) \\ \vdots \\ f_N(\mathbf{a}) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1(\mathbf{a})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{a})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N(\mathbf{a})}{\partial x_1} & \cdots & \frac{\partial f_N(\mathbf{a})}{\partial x_n} \end{bmatrix} \begin{bmatrix} \mathbf{X} - \mathbf{a} \end{bmatrix} = f(\mathbf{a}) + \mathbf{J} \begin{bmatrix} \mathbf{X} - \mathbf{a} \end{bmatrix} \tag{58}$$

$$\mathbf{X} = \mathbf{a} - (\mathbf{J}(\mathbf{a}))^{-1} f(\mathbf{a}) \tag{59}$$

$$\mathbf{X}_{i+1} = \mathbf{X}_i - (\mathbf{J}(\mathbf{X}_i))^{+} f(\mathbf{X}_i) \tag{60}$$

Ahora se debe definir la función vectorial a la cual se le aplica el método. Lo que se debe hacer es plantear una función cuyas raíces representen un vector de estado que iguale las matrices homogéneas de las ecuaciones (44) y (51). Una forma de hacer esto es planteando una función vectorial que analice el error entre los vectores de posición y el error de proyección entre los vectores de las matrices de rotación (ver ecuación (61)). La función $F_{pos}(\mathbf{X})$ representa el error de posición, por lo que se deben restar los vectores de traslación de las matrices homogéneas, mientras que el error de orientación $F_{orient}(\mathbf{X})$ contiene el error del producto punto entre los vectores de las matrices de rotación. Si se desea orientar únicamente un eje del gripper, sólo es necesario usar el producto punto del eje que se desea orientar, mientras que, si se desea orientar todos los ejes, sólo es necesario usar el error del producto punto de dos ejes ya que las matrices de rotación son ortogonales.

$$F(\mathbf{X}) = \begin{bmatrix} F_{pos}(\mathbf{X}) \\ F_{orient}(\mathbf{X}) \end{bmatrix} = \begin{bmatrix} {}^b\mathbf{t}_d - {}^b\mathbf{t}_n \\ -1 + {}^b\mathbf{n}_n \cdot {}^b\mathbf{n}_d \\ -1 + {}^b\mathbf{o}_n \cdot {}^b\mathbf{o}_d \\ -1 + {}^b\mathbf{a}_n \cdot {}^b\mathbf{a}_d \end{bmatrix} \tag{61}$$

Como se puede observar en la ecuación (58), es necesario calcular el gradiente de cada función para generar el jacobiano. Un primer enfoque para realizar esto es describir la ecuación (61) de manera simbólica (ver ecuación (62)), y de esta manera tener los vectores de interés expresados en función del vector de estado para posteriormente calcular el jacobiano simbólico. El inconveniente de esto radica en que es computacionalmente costoso calcular un jacobiano

simbólico. Para evitar esto, se analizan las derivadas parciales como se muestra en las ecuaciones (63) y (64).

$$ {}^{b}\mathbf{T}_{n}(\mathbf{q}) = \begin{bmatrix} {}^{b}\mathbf{n}_{n}(\mathbf{q}) & {}^{b}\mathbf{o}_{n}(\mathbf{q}) & {}^{b}\mathbf{a}_{n}(\mathbf{q}) & {}^{b}\mathbf{t}_{n}(\mathbf{q}) \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^{b}\mathbf{T}_{0}\,{}^{0}\mathbf{T}(q_0)_{1}\,{}^{1}\mathbf{T}_{2}(q_1)\,{}^{2}\mathbf{T}_{3}(q_2)\,{}^{3}\mathbf{T}_{4}(q_3)\,{}^{4}\mathbf{T}_{5}(q_4)\,{}^{5}\mathbf{T}_{6}(q_5) \tag{62} $$

$$ \frac{\partial F_{pos}(\mathbf{q})}{\partial \mathbf{q}} = \frac{\partial({}^{b}\mathbf{t}_{d} - {}^{b}\mathbf{t}_{n})}{\partial \mathbf{q}} = \frac{-\partial {}^{b}\mathbf{t}_{n}}{\partial \mathbf{q}} = -\begin{bmatrix} \frac{\partial {}^{b}\mathbf{t}_{n}}{\partial q_0} & \frac{\partial {}^{b}\mathbf{t}_{n}}{\partial q_1} & \cdots & \frac{\partial {}^{b}\mathbf{t}_{n}}{\partial q_{n-1}} \end{bmatrix} \tag{63} $$

$$ \frac{\partial F_{orient}(\mathbf{X})}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial\left(-1+{}^{b}\mathbf{n}_{n}\cdot{}^{b}\mathbf{n}_{d}\right)}{\partial \mathbf{q}} \\ \frac{\partial\left(-1+{}^{b}\mathbf{o}_{n}\cdot{}^{b}\mathbf{o}_{d}\right)}{\partial \mathbf{q}} \\ \frac{\partial\left(-1+{}^{b}\mathbf{a}_{n}\cdot{}^{b}\mathbf{a}_{d}\right)}{\partial \mathbf{q}} \end{bmatrix} = \begin{bmatrix} \frac{\partial {}^{b}\mathbf{n}_{n}}{\partial q_0}\cdot{}^{b}\mathbf{n}_{d} & \frac{\partial {}^{b}\mathbf{n}_{n}}{\partial q_1}\cdot{}^{b}\mathbf{n}_{d} & \cdots & \frac{\partial {}^{b}\mathbf{n}_{n}}{\partial q_{n-1}}\cdot{}^{b}\mathbf{n}_{d} \\ \frac{\partial {}^{b}\mathbf{o}_{n}}{\partial q_0}\cdot{}^{b}\mathbf{o}_{d} & \frac{\partial {}^{b}\mathbf{o}_{n}}{\partial q_1}\cdot{}^{b}\mathbf{o}_{d} & \cdots & \frac{\partial {}^{b}\mathbf{o}_{n}}{\partial q_{n-1}}\cdot{}^{b}\mathbf{o}_{d} \\ \frac{\partial {}^{b}\mathbf{a}_{n}}{\partial q_0}\cdot{}^{b}\mathbf{a}_{d} & \frac{\partial {}^{b}\mathbf{a}_{n}}{\partial q_1}\cdot{}^{b}\mathbf{a}_{d} & \cdots & \frac{\partial {}^{b}\mathbf{a}_{n}}{\partial q_{n-1}}\cdot{}^{b}\mathbf{a}_{d} \end{bmatrix} \tag{64} $$

Las derivadas parciales resultantes pueden ser halladas como se indica en la ecuación (65), la cual puede simplificarse en la ecuación (66). Este resultado indica que para calcular las derivadas parciales de los vectores de la matriz homogénea ${}^{b}\mathbf{T}_{n}$ respecto a una componente del vector de estado $\mathbf{q}$, es necesario usar la cinemática directa del robot cambiando únicamente la matriz que contiene la componente analizada del vector de estado por su derivada parcial respecto a esta misma componente. De esta manera se puede ir completando cada columna de las ecuaciones (63) y (64), las cuales conforman al jacobiano.

$$ \frac{\partial {}^{b}\mathbf{T}_{n}}{\partial q_{i-1}} = \begin{bmatrix} \frac{\partial {}^{b}\mathbf{n}_{n}}{\partial q_{i-1}} & \frac{\partial {}^{b}\mathbf{o}_{n}}{\partial q_{i-1}} & \frac{\partial {}^{b}\mathbf{a}_{n}}{\partial q_{i-1}} & \frac{\partial {}^{b}\mathbf{t}_{n}}{\partial q_{i-1}} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \frac{\partial\left({}^{b}\mathbf{T}_{i-1}\,{}^{i-1}\mathbf{T}(q_{i-1})_{i}\,{}^{i}\mathbf{T}_{n}\right)}{\partial q_{i-1}} \tag{65} $$

$$ \frac{\partial {}^{b}\mathbf{T}_{n}}{\partial q_{i-1}} = {}^{b}\mathbf{T}_{i-1}\frac{\partial {}^{i-1}\mathbf{T}(q_{i-1})_{i}}{\partial q_{i-1}}\,{}^{i}\mathbf{T}_{n} \tag{66} $$

7.4.3 Resultados aplicando cinemática inversa.

Para poner a prueba los métodos anteriores se debe definir las matrices homogéneas de la cinemática directa, tener un vector de estado inicial y una matriz homogénea de referencia. Dentro del simulador se obtiene los parámetros DH del robot IRB 4600, los cuales se muestran en la Tabla 7.

Tabla 7 Parámetros DH usados para el robot IRB4600

Matriz	θ[deg]	$d[m]$	$a[m]$	α[deg]
${}^{b}\mathbf{T}_{0}$	−175.4	0	0.5146	0
${}^{0}\mathbf{T}_{1}(q_0)$	$-150.1+q_0$	0.495	0.2122	–90
${}^{1}\mathbf{T}_{2}(q_1)$	$-78.1+q_1$	0.3366	1.1191	–34.5
${}^{2}\mathbf{T}_{3}(q_2)$	$36.3+q_2$	–0.5463	0.1616	–90.6
${}^{3}\mathbf{T}_{4}(q_3)$	q_3	0.7962	0.8272	95
${}^{4}\mathbf{T}_{5}(q_4)$	$-139.6+q_4$	–0.0034	0.0037	93.6
${}^{5}\mathbf{T}_{6}(q_6)$	$-176.6+q_6$	0.0006	0.0098	–93.7

Al momento de implementar el método convencional se omite el vector de pesos y al igual que el método propuesto, se ha dejado iterar 19 veces ya que cerca a esta cantidad de iteraciones se aprecia una estabilidad en la variación del error. Para analizar el error total entre la matriz homogénea deseada y la del gripper se ha tomado como referencia la norma del vector que resulta de la ecuación (61), a la cual se le ha omitido la sexta componente. En la primera prueba de cinemática inversa se ha escogido la matriz homogénea mostrada en (67), además, se ha escogido un vector de estado inicial igual a un vector nulo. En la Figura 30 se muestra los resultados obtenidos al realizar la cinemática inversa tomando en cuenta tanto la orientación como la posición del gripper.

$$ {}^{b}\mathbf{T}_{d} = \begin{bmatrix} 0.707 & -0.707 & 0 & -1.2 \\ 0.707 & 0.707 & 0 & -0.041 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{67} $$

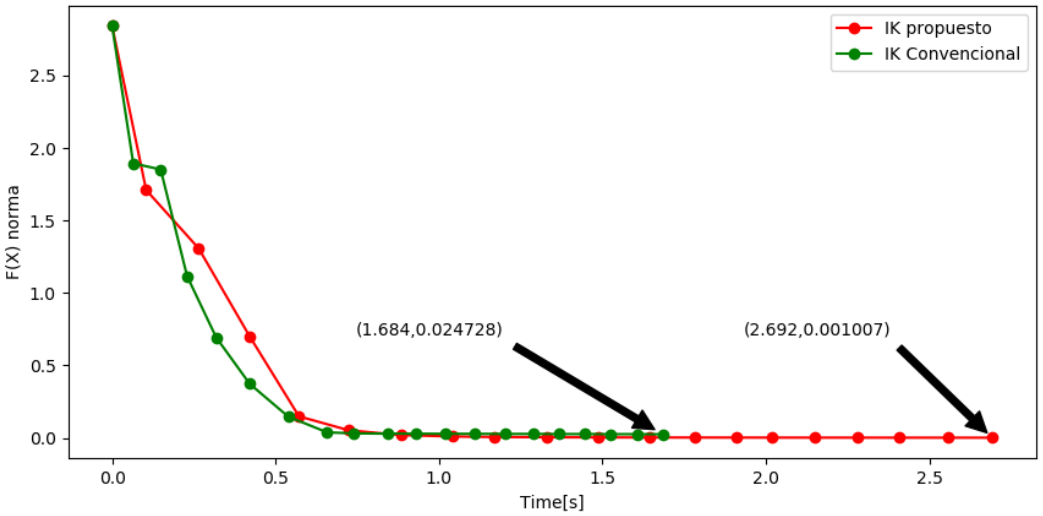


Figura 30 Norma de la ecuación (61) al usar los métodos explicados. Se tiene en cuenta error de posición y orientación.

Según la gráfica de la Figura 30, se puede concluir que los métodos permiten realizar la cinemática inversa, sin embargo, hay una diferencia importante en la velocidad de ejecución de estos. Se ha medido que el cálculo del jacobiano es la sección de los algoritmos que más consume tiempo. El método convencional es el que se ejecuta más rápido debido a que el jacobiano requiere conocer unos vectores que se pueden hallar a medida que se realiza la cinemática directa del robot, esto significa que calculando una vez la cinemática directa se puede construir todo el jacobiano. En el método propuesto cada columna del jacobiano se halla a partir de una multiplicación matricial que tiene la misma cantidad de matrices que las halladas en la cinemática directa, lo que causa que sea más demorado. A pesar de esto el método propuesto logra una exactitud mayor. En cinco pruebas realizadas se identificó que siempre el método propuesto era más exacto (ver Tabla 8). Los tiempos obtenidos son extensos en los dos métodos debido a la librería Sympy que se implementó para realizar las multiplicaciones matriciales. Librerías más rápidas como Numpy pueden reducir más los tiempos obtenidos.

Tabla 8 Norma de la ecuación (61) al calcular la IK mediante ambos métodos para cinco matrices diferentes

Método	1	2	3	4	5
Convencional	$78.9x10^{-3}$	$93.03x10^{-3}$	$3.02x10^{-3}$	$1.04x10^{-3}$	$75.3x10^{-3}$
Propuesto	$2.26x10^{-5}$	$2.568x10^{-5}$	$7.81x10^{-5}$	$4.18x10^{-5}$	$2.1x10^{-4}$

8 Discretización, generación de grafos y planeación de movimiento

A través de las herramientas presentadas en los anteriores capítulos es posible analizar el entorno operativo para crear grafos con los cuales planear el movimiento del robot. Los grafos pueden ser construidos independientemente del problema que se esté trabajando. Estos pueden usarse con robots móviles, aéreos y fijos, entre otros. Si se posee la información suficiente para realizar el grafo, la búsqueda del camino más corto puede ser abordada con algoritmos clásicos como Dijkstra o Floyd-warshall. La solución de los grafos ya ha sido estudiada fuertemente por la ciencia de la computación, por lo que el problema real se encuentra en cómo construirlos. Para esto se debe conocer las restricciones del robot, el espacio de configuración libre y la forma en la que se conectan las configuraciones del robot.

En este capítulo se aborda la creación de grafos para un robot antropomórfico de seis grados de libertad y la solución de estos a partir del algoritmo Dijkstra. Lo anterior significa que este capítulo se encarga de explicar cómo realizar la planeación de movimiento del robot para que este pueda desplazarse sin colisionar con el entorno, además, presenta los resultados obtenidos en la planeación de movimiento. Por este motivo en esta sección se abordan los objetivos específicos 4 y 5.

8.1 Discretización del espacio de configuración y creación de grafos

El espacio de configuración del robot son todos los posibles estados en los que se pueden encontrar sus articulaciones. Este es un espacio vectorial en el cual a los vectores se les suele llamar vectores de estado o vectores de coordenadas generalizadas, en donde cada componente de estos contiene el valor instantáneo de una articulación (ver ecuación (68)). El espacio de configuración se determina según como se muestra en (69), el cuál limita el valor de cada componente a partir de unas constantes l_i, que se definen según el rango de operación de las articulaciones. Debido a que este espacio vectorial es infinito, se puede intuir que es necesario discretizarlo, ya que un algoritmo no puede analizar infinitos estados. En este punto la capacidad de cómputo empieza a jugar un papel importante ya que en función a esta se debe definir cómo hacer el muestreo del espacio de configuración. En esta investigación el algoritmo que se propone se inspira en el método RRT (LaValle, 1998), el cual es muy conocido en robótica para hacer planeaciones de trayectoria. RRT analizar configuraciones aleatorias para crear un grafo cuyas aristas pertenecen al subespacio de configuración libre $\mathbf{C}_{libre}$, el cual está contenido en $\mathbf{C}$.

$$\mathbf{q} = \left[q_0, q_1, q_2, q_3, q_4, q_5\right]^T \tag{68}$$

$$\mathbf{C} = \begin{Bmatrix} (q_0, q_1, q_2, q_3, q_4, q_5) \in \mathbb{R}^6 \mid \\ l_0 \le q_0 \le l_1, l_2 \le q_1 \le l_3, l_4 \le q_2 \le l_5, l_6 \le q_3 \le l_7, l_8 \le q_4 \le l_9, l_{10} \le q_5 \le l_{11} \end{Bmatrix} \tag{69}$$

El método RRT menciona que existen dos subespacios de configuración denominados $\mathbf{C}_{libre}$ y $\mathbf{C}_{obs}$. Ambos están contenidos por $\mathbf{C}$, sin embargo, no se conoce una representación explícita de estos, por lo que inevitablemente se tiene que realizar un muestreo de $\mathbf{C}$ para analizar cada configuración y determinar si esta se encuentra en $\mathbf{C}_{libre}$ o en $\mathbf{C}_{obs}$. Para determinar a qué subespacio pertenece una configuración $\mathbf{q}$, se debe implementar un analizador de colisiones como el propuesto en la sección 7.3.

RRT antes de crear el grafo define una cantidad por defecto de aristas. Luego empieza a analizar aleatoriamente configuraciones para conectarlas a la arista más próxima dentro del grafo siempre y cuando esta y su trayectoria (vértice entre aristas) se encuentre en $\mathbf{C}_{libre}$. Esto lo hace repetidamente hasta completar el tamaño de aristas requerido. Finalmente, el grafo resultante es usado por un planeador de trayectoria para indicar el camino entre dos pares de nodos.

El método que se propone desarrolla un enfoque similar, pero con un grafo cuyas aristas no se escogen aleatoriamente, por el contrario, se escogen de tal forma que estén uniformemente distribuidas a lo largo del espacio de configuración. Esto se puede hacer discretizando el rango de operación de cada articulación en *m* posiciones. Como ejemplo se presenta la Figura 31, en donde una articulación con un rango de operación definido por l_4 y l_5 se ha discretizado en 5 posiciones.

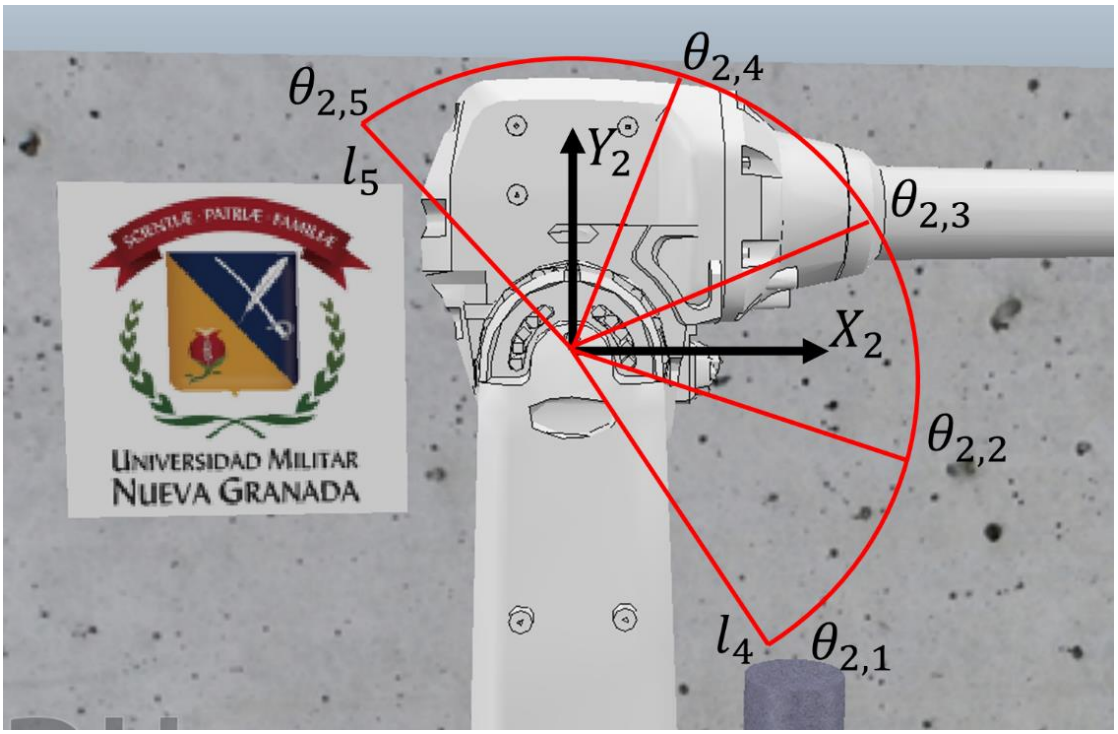


Figura 31 Discretización del rango de operación de una articulación.

Si se divide todas las articulaciones como se describe en la Figura 31, entonces la cantidad de configuraciones a evaluar está dada por (70). El inconveniente de dividir el rango de operación de todas las articulaciones es que la cantidad de configuraciones que se deben analizar es enorme, por lo que deja de ser viable. Si se fija los tres últimos grados de libertad y discretizar las tres primeras articulaciones, la cantidad de nodos estará dada por (71), lo cual reduce en gran medida los

cálculos. Esto es viable debido a que las tres primeras articulaciones se encargan de desplazar al gripper mientras que las tres últimas dan orientación, y al momento de evadir obstáculos es más importante posicionar.

$$v = m_0 \cdot m_1 \cdot m_2 \cdot m_3 \cdot m_4 \cdot m_5 \tag{70}$$

$$v = m_0 \cdot m_1 \cdot m_2 \tag{71}$$

Sabiendo que el espacio de configuración se discretiza con tres coordenadas fijas, entonces todas las posibles configuraciones se pueden representar en una matriz como se muestra en la Figura 32, a la cual se le da el nombre de "Matriz de discretización". En este ejemplo cada articulación se discretiza en cuatro posiciones. Para cualquier posición dentro de la matriz el vector de estado tiene constantes en las tres últimas componentes, mientras que para los extremos de la matriz las tres primeras componentes usan las constantes l_0 a l_5, las cuales definen el rango de operación de las articulaciones.

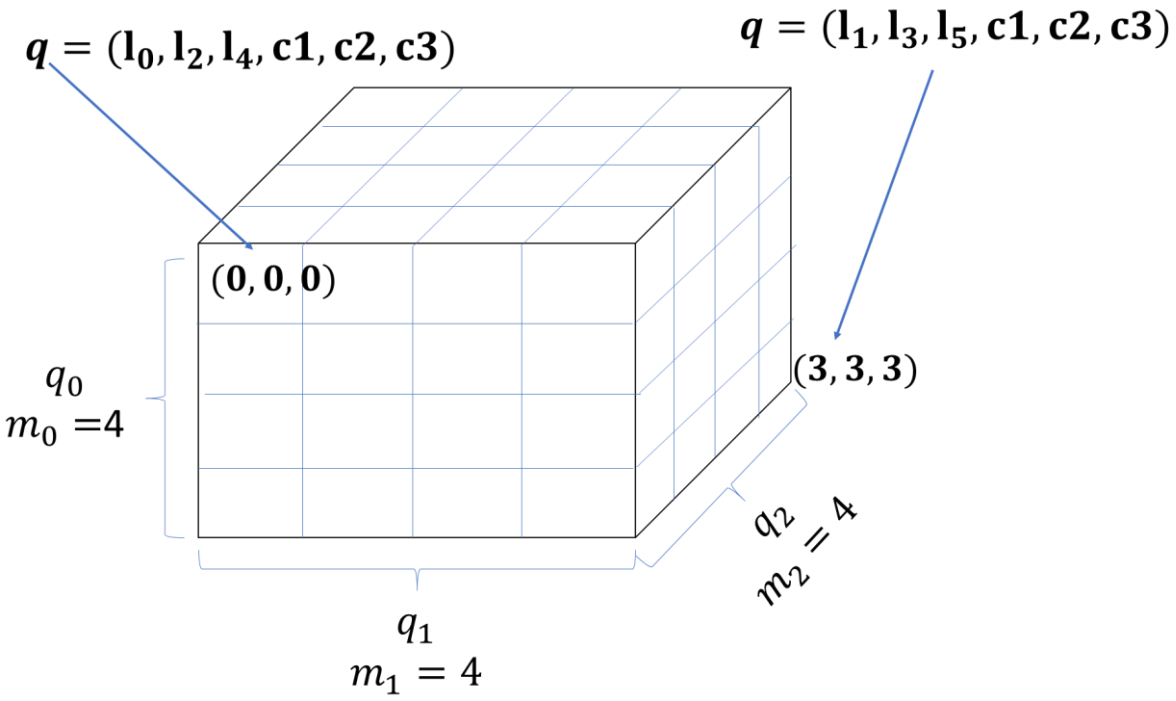


Figura 32 Discretización del espacio de configuración con $m_0, m_1, m_2 = 4$. Matriz de discretización.

La Figura 32 muestra una matriz de discretización en donde las celdas representan una configuración específica del robot. Todas las configuraciones, aunque pertenecen a $\mathbf{C}$, se les desconoce a que subconjunto están asociadas. Es en este punto en donde se debe aplicar el analizador de colisiones y clasificar cada configuración en alguno de los dos subespacios $\mathbf{C}_{libre}$ o $\mathbf{C}_{obs}$. Cada celda de la matriz debe almacenar esta clasificación para que, al momento de crear el grafo, se conozca qué configuraciones no usar. Por ejemplo, para la celda en la posición (0,0,0), que representa la configuración $\mathbf{q} = (l_0, l_2, l_4, c_1, c_2, c_3)$, se ha determinado que pertenece a $\mathbf{C}_{obs}$, entonces la posición (0,0,0) debe almacenar esta información. Otro dato que deben almacenar las

celdas de esta matriz es un costo asociado a la distancia que existe entre los obstáculos de alta prioridad y el robot, esto se analiza más adelante.

El grafo dentro del algoritmo se construye a partir de un arreglo de dos dimensiones, en donde las filas y las columnas representan nodos. Un nodo en el grafo es una configuración del robot, por lo que todas las celdas de la matriz de la Figura 32 son todos los nodos que tendrá el grafo. Para una cantidad de nodos definida por (71), el grafo tendrá la forma mostrada en la Figura 33. La información que contiene este arreglo es el costo que existe para desplazarse localmente entre nodos, por este motivo se le llama matriz de costos. La fila escogida es el nodo de origen, mientras que la columna es el nodo de destino, por lo que el valor $a_{1,v-1}$ representa el costo que hay para desplazar el robot del nodo 1 al nodo *v-1*. No todos los nodos están conectados entre sí localmente, por lo que, para estas posiciones de la matriz contiene un valor igual a infinito. A pesar de que dos nodos pueden estar conectados a través de otros nodos, esta matriz sólo indica costos de conexiones locales, es decir, nodos con una conexión directa. Si para una fila de la matriz todas las posiciones contienen valores infinitos, significa que esta configuración no puede ser alcanzada por el robot.

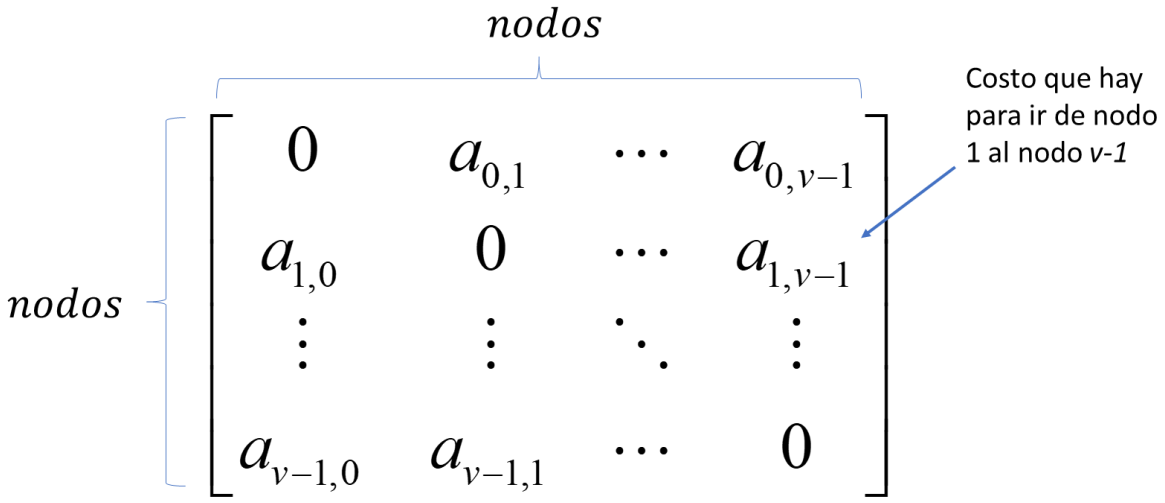


Figura 33 Representación de grafo a partir de un arreglo.

El algoritmo de planeación de trayectoria usa la matriz de costos para calcular cuál es el desplazamiento entre nodos para ir desde un nodo "A" a uno "B". Aunque exista un gran número de trayectorias factibles, el algoritmo busca la que genera el menor costo. Esto significa que modificando la matriz de costos se puede hacer que el robot evite en lo posible acercar los eslabones a los objectos de alta prioridad (personas).

Los costos locales se definen a partir de tres condiciones. El pseudocódigo de esto se puede ver en la Figura 34. En primer lugar, el nodo de origen y destino se les obtiene la posición dentro de la matriz de discretización. Luego en las líneas 7 y 8 se analiza si para estas posiciones existe colisión. En caso de que alguno de los dos nodos presente colisión, se retorna un costo igual a infinito. De igual manera si los dos nodos dentro de la matriz de discretización no son adyacentes, se retorna infinito. En caso de que los nodos estén conectados localmente y no presenten colisión, el costo total se genera a partir de la suma de dos costos. El primero se asocia con el costo que indica la

matriz de discretización y el segundo con el tipo de desplazamiento que se debe realizar para ir del nodo “A” al nodo “B”. Este pseudocódigo se debe evaluar para todos los nodos de la matriz de costos y así definir al grafo.

```
CostosLoclales(nodoA,nodoB)
{

  posicionA = nodos2Matriz(nodoA)
  posicionB = nodos2Matriz(nodoB)

  colisionA = MatrizDiscretizacion[posicionA]["colision"]
  colisionB = MatrizDiscretizacion[posicionB]["colision"]

  if colisionA == "no hay colision" && colisionB == "no hay colision"
    if posicionA es adyacente a posicionB == True

      costo1 = MatrizDiscretizacion[posicionB]["costo"]

      if posicionA y posicionB tienen todas las componentes iguales menos una
        costo2 = 0.5

      else
        costo2 = 0.8

      return costo1+costo2

    else
      return "infinito"

  else
    return "infinito"
}
```

Figura 34 Pseudocódigo para evaluar el costo entre nodos adyacentes.

Es necesario definir funciones que permitan relacionar las configuraciones del robot, las posiciones de la matriz de discretización y los nodos de la matriz de costos. En la Figura 35 se muestran el pseudocódigo para estas transformaciones. De esta manera cualquier nodo puede ser transformado en un vector de estado, o un vector de estado puede ser relacionado con la matriz de discretización o un nodo. En este momento lo último que falta por definir es cómo se obtiene la variable “costo1” del pseudocódigo de la Figura 34. Este dato debe estar almacenado en la matriz de discretización y se relaciona con la cercanía de los eslabones con los obstáculos. Esto significa que para una configuración en la que los eslabones se encuentren muy cerca al operario esta variable tendrá un valor alto, haciendo que una trayectoria por este nodo sea más costosa, lo que ocasionará que el algoritmo de planeación de trayectoria los evite.

```
cMat2Nod=[m0*m1,m0]

nodos2Matriz(nodo){
#transforma nodos a posiciones de la matriz de discretizacion

  posMat_q0 = int((nodo%cMat2Nod[0])/cMat2Nod[1])
  posMat_q1 = (nodo%cMat2Nod[0])%cMat2Nod[1]
  posMat_q3 = int(nodo/cMat2Nod[0])
  posicionMat = [posMat_q0,posMat_q1,posMat_q3]
  return posicionMat
}

matriz2nodo(posicionMat){
#transforma posiciones de la matriz de discretizacion a nodos

  return int(posicionMat[2]*cMat2Nod[0]+posicionMat[0]*cMat2Nod[1]+posicionMat[1])
}


mMat2deg[0] = (l1-l0)/(m0-1)
mMat2deg[1] = (l3-l2)/(m1-1)
mMat2deg[2] = (l5-l4)/(m2-1)
cMat2deg = [l0,l2,l4]

matriz2deg(posicionMat){
#transforma posiciones de la matriz de discretizacion a un estado de configuracion
  q0 = mMat2deg[0]*posicionMat[0]+cMat2deg[0]
  q1 = mMat2deg[1]*posicionMat[1]+cMat2deg[1]
  q2 = mMat2deg[2]*posicionMat[2]+cMat2deg[2]
  return [q0,q1,q2]
}

deg2Matriz(cordenadas_q){
#transforma un estado de configuracion a una posicion de la matriz de discretizacion
  posMat_q0 = int((cordenadas_q[0]-cMat2deg[0])/mMat2deg[0])
  posMat_q1 = int((cordenadas_q[1]-cMat2deg[1])/mMat2deg[1])
  posMat_q2 = int((cordenadas_q[2]-cMat2deg[2])/mMat2deg[2])
  posicionMat = [posMat_q0,posMat_q1,posMat_q3]
  return posicionMat
}
```

Figura 35 Pseudocódigo de funciones para relacionar nodos, posiciones en la matriz de discretización y configuraciones del robot.

Para obtener el costo que se almacena en una celda de la matriz de discretización, se debe analizar todas las nubes de puntos y a cada una calcularle un costo, luego el mayor costo detectado es el que se almacena. El costo asociado a la nube de puntos *i* se calcula con las ecuaciones (72) a (76). La ecuación (72) referencia la nube de puntos respecto al sistema coordenado de un prisma. A cada punto resultante con coordenadas (*x*,*y*,*z*) se le hallan los tres radios mostrados en las ecuaciones (73) a (75). Si se analizan todos los prismas y se escoge el menor radio *r* calculado, entonces el costo para la nube de puntos se obtiene con la ecuación (76), la cual es una función a trozos. Esto se hace para todas las nubes de puntos para posteriormente escoger el costo más alto, el cual es almacenado en la celda analizada de la matriz de discretización.

$$ {}^{pi}P = \left({}^{b}\mathbf{T}_{pi}(\mathbf{q}) \right)^{-1} {}^{b}P \tag{72}$$

$$r_1 = \sqrt{\left(|x| - \frac{w}{2}\right)^2 + y^2 + z^2} \tag{73}$$

$$r_2 = \sqrt{x^2 + \left(|y| - \frac{h}{2}\right)^2 + z^2} \tag{74}$$

$$r_3 = \sqrt{x^2 + y^2 + \left(|z| - \frac{d}{2}\right)^2} \tag{75}$$

$$\text{costoNube}_i = \begin{cases} \left(\frac{-\text{costoMaximo}}{\text{limiteRadio}}\right) * r + \text{costoMaximo} & \text{si } r \le \text{limiteRadio} \\ 0 & \text{si } r > \text{limiteRadio} \end{cases} \tag{76}$$

Las constantes de la ecuación controlan el incremento del costo, por lo que se puede definir grupos de constantes para definir diferentes prioridades en la evasión de los obstáculos. En la Tabla 9 se muestran las constantes usadas en este trabajo, las cuales se hallaron experimentalmente. Se definió tres prioridades: máxima, media y mínima. Prioridad mínima en este trabajo significa no usar la ecuación 76 a la nube de puntos analizada, esto es arbitrario y si el desarrollador desea definir nuevas constantes o incluso más niveles de prioridad, puede hacerlo.

Tabla 9 Parámetros para definir diferentes prioridades de evasión

Prioridad	Costo Máximo	Limite Radio
Prioridad máxima	10	85cm
Prioridad media	10	25cm
Prioridad mínima	-	-

Suponga que se está obteniendo el costo para una posición de la matriz de discretización y se tienen las nubes de puntos del entorno, las personas, las paredes y las bandas transportadoras. El procedimiento para hallar el costo que se debe asignar a la celda puede desarrollarse como se indica en la Figura 36.

Para calcular el costo primero se deben tener definidas las diferentes nubes de puntos respecto al robot. Se puede tener la nube de puntos del entorno, en la cual están todos los obstáculos detectados; la nube de puntos de las personas, las cuales se tratan con prioridad máxima; las nubes de puntos de las paredes, las cuales se tratan con prioridad media y las nubes de puntos de las bandas transportadoras, las cuales se tratan con prioridad media. A cada nube de puntos se le calcula el costo con la ecuación (76), luego se escoge el costo más alto y este es el valor retornado para almacenar en la celda analizada de la matriz de discretización. En este trabajo sólo se están obteniendo dos tipos de nubes de puntos, la que representa a todo el entorno y la que representa a las personas. Para ver el efecto de las prioridades, en la sección de resultados se muestran diferentes planeaciones de trayectoria para las cuales las nubes de puntos de las personas se les asigna las tres prioridades presentadas en la tabla Tabla 9.

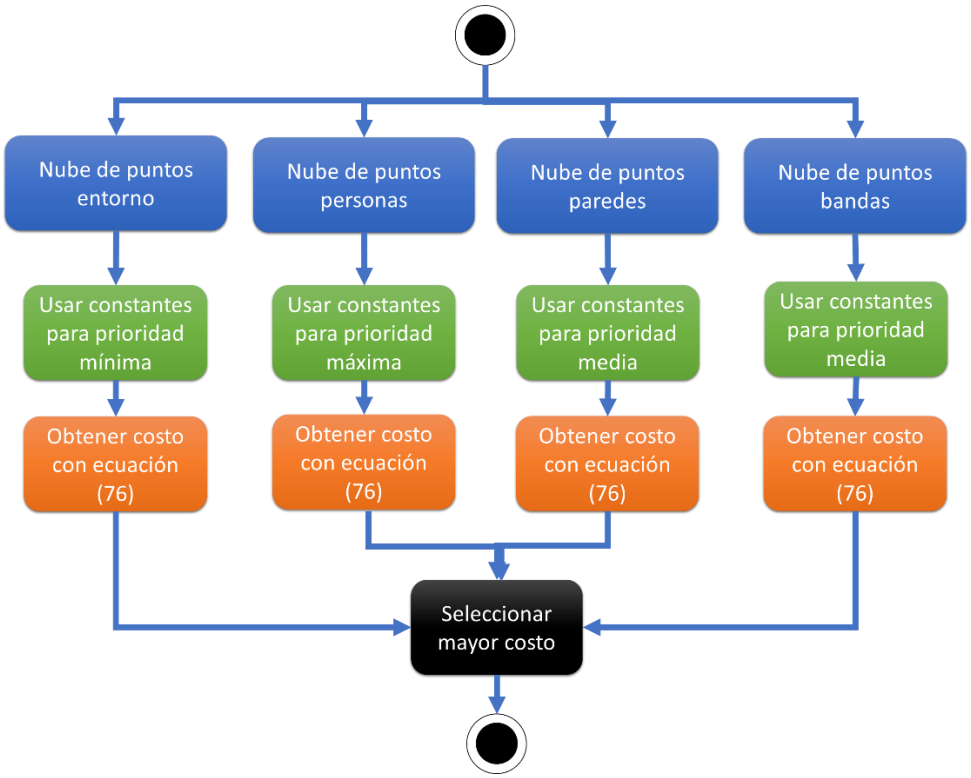


Figura 36 Procedimiento para hallar el costo que se asigna a una celda de la matriz de discretización.

Con todo lo planteado anteriormente ya es posible crear la matriz de costos y transformar un nodo a una coordenada generalizada y viceversa. Esto significa que es momento de aplicar un algoritmo que encuentre el camino más corto entre dos nodos, para luego transformar la solución obtenida, la cual es una secuencia de nodos, a una secuencia de coordenadas generalizadas con las cuales se puede realizar la planeación de movimiento. Es importante mencionar que el analizador de colisiones sólo revisa si una configuración hace colisionar al robot, no analiza si el trayecto entre dos configuraciones adyacentes contiene un obstáculo. Esto puede generar problemas para matrices de discretización muy pequeñas. Entre más grande sea esta matriz, este problema tendrá menos efecto, sin embargo, para futuros desarrollos puede plantearse analizar la trayectoria entre nodos en el analizador de colisiones para reducir el tamaño de la matriz de discretización, y así reducir el tiempo de ejecución del algoritmo.

8.2 Planeación de trayectoria mediante algoritmo Dijkstra

Los dos algoritmos más usados para solucionar grafos son Dijkstra y Floyd-Warshall. Estos sólo requieren conocer las conexiones y costos locales de cada nodo para generar la solución. La diferencia entre estos dos está en que Dijkstra obtiene la trayectoria para ir de un nodo de origen a cualquier nodo dentro del grafo, mientras que Floyd-Warshall obtiene la solución para ir desde cualquier nodo a cualquier otro nodo. Esto en tiempo de procesamiento tiene un gran efecto. Para

comparar los algoritmos se debe revisar el tiempo de complejidad. Si V son la cantidad de vértices y E el número máximo de aristas, entonces la complejidad computacional de Dijkstra es como se indica en (77) y la de Floyd-Warshall como se muestra en (78). Lo que se puede concluir es que Floyd-Warshall es muy costoso, por lo que se prefiere implementar Dijkstra.

$$O\left(VE\log(V)\right) \tag{77}$$

$$O\left(V^3\right) \tag{78}$$

El Pseudocódigo de Dijkstra se puede ver en la Figura 37. Este genera la solución en un arreglo de dos dimensiones en donde la cantidad de filas es igual a la cantidad de nodos del grafo y la cantidad de columnas es igual a 2. La primera columna indica los costos para ir de un nodo cualquiera al nodo origen, mientras que la segunda columna indica el nodo predecesor que debe ser tomado para dirigirse al nodo origen.

```
74  Dijkstra(){
75
76    crear matriz solucion con vx2 celdas # primera columna representa costos y segunda conexiones
77    para el vertice inicial indicar que se conecta al vertice inicial
78    para el vertice inicial indicar un costo igual a cero
79    para los demas vertices indicar un costo igual a infinito y dejar vacia la columna de conexiones
80
81    crear una lista para almacenar los vertices no visitados
82
83    while siempre que hayan vertices no visitados
84
85      visitar el nodo no visitado con el menor costo anotado en la matriz solucion
86      for para cada vecino no visitado del nodo actual
87        calcular la distancia desde el nodo inicial al nodo vecino
88        if la distancia calculada  para el vertice vecino es menor que la distancia conocida
89          acutalizar la distancia conocida del nodo vecino
90          actualizar al nodo anterior del nodo vecino por el actual
91      indicar que el nodo actual ya fue visitado
92
93    return matriz solucion
94
95  }
```

Figura 37 Pseudocódigo del método Dijkstra.

La matriz solución generada por Dijkstra permite escoger un nodo arbitrario y conocer el recorrido entre nodos que lo dirige al nodo inicial. Debido a que todo el grafo es solucionado para ir desde cualquier nodo a un nodo específico, se debe escoger con cuidado sobre qué nodo se generará la solución. Se ha escogido el nodo para el cual la configuración del robot tiene un vector de estado nulo (ver Figura 38). Esta configuración es de interés ya que desde esta se realiza el agarre de los objetos de la banda transportadora.

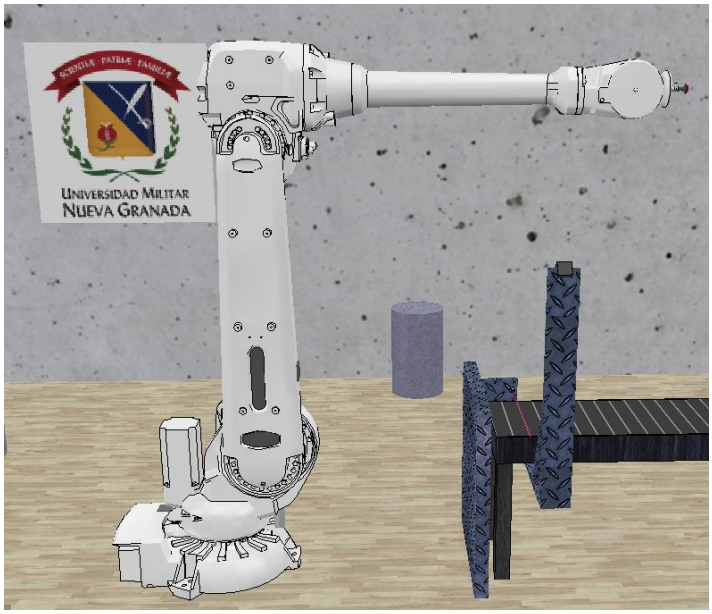


Figura 38 Configuración escogida en el robot para solucionar el grafo.

La solución obtenida por Dijkstra es una secuencia de nodos que deben ser transformados a coordenadas generalizadas para realizar el desplazamiento del robot. En muchas ocasiones sólo se conoce la posición y orientación que debe tener el gripper, por lo que en estas situaciones es necesario aplicar alguno de los métodos de IK explicados en la sección 7.4 y así obtener una vector de estado que luego debe ser aproximada a un nodo del grafo.

Un resultado obtenido para una matriz de discretización de 30x30x30 celdas se puede ver en la Figura 39. El desplazamiento del gripper se ha indicado a partir de una línea blanca. Lo que se observa es que el robot logra evadir tanto personas como objetos mientras se dirige al nodo objetivo.

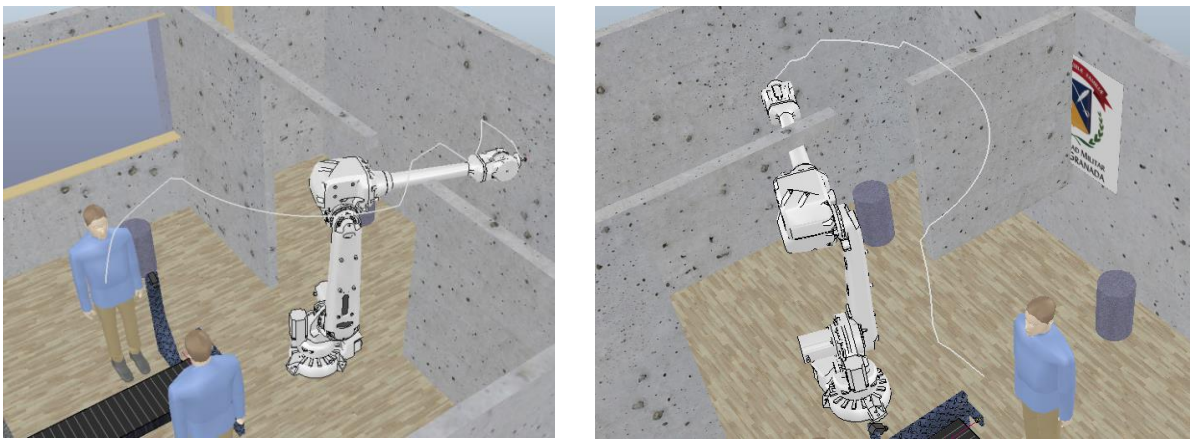


Figura 39 Prueba de planeación de movimiento a una posición arbitraria.

Los tiempos de ejecución obtenidos para crear y solucionar diferentes tamaños de matrices de discretización se indican en la Figura 40. Crear la matriz de discretización es mucho más costoso que solucionar el grafo. Los tiempos de procesamiento no son lo suficientemente rápidos como

para actualizar y solucionar el grafo en tiempo real, lo que causa que no se puedan evadir obstáculos en movimiento. Lo anterior significa que este enfoque sólo puede aplicarse para entornos estáticos. Una forma de incluir personas en movimiento dentro del entorno es restringir espacios a partir de obstáculos ficticios. Si el obstáculo abarca todo el espacio de trabajo que implementa el operario, entonces el robot no colisionará con este. Las características del computador sobre el cual se realizan las pruebas se pueden ver en la Tabla 10.

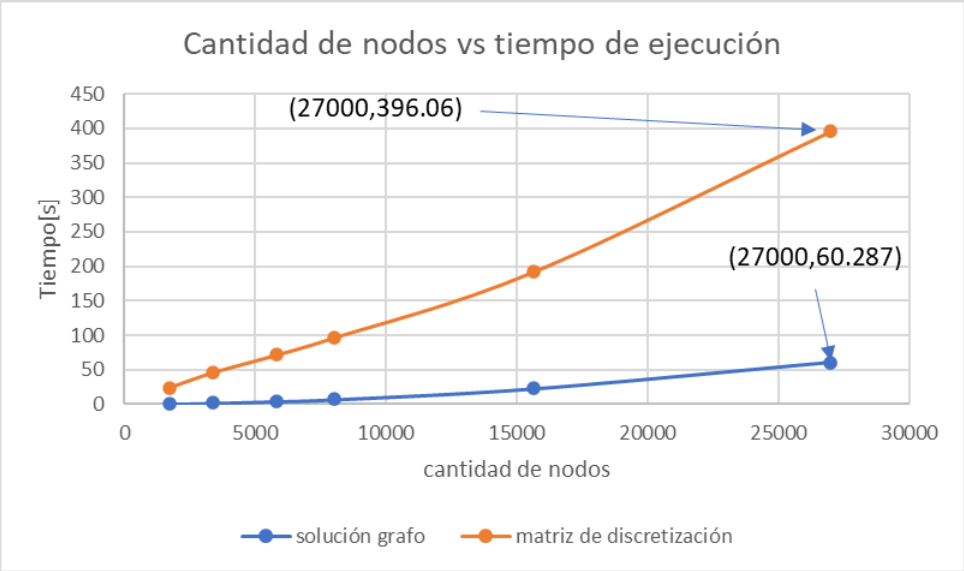


Figura 40 Tiempo de procesamiento requerido para la creación y solución de grafos.

Tabla 10 Características del computador sobre el cual se realizan las pruebas

CPU	RAM	GPU	OS
Core i7-7700HQ	32 DDR4	GTX 1050	Ubuntu 18.04

8.3 Sistema de seguridad y pruebas de funcionamiento

A continuación, se resume cómo unir todos los planteamientos anteriores para realizar planeaciones de movimiento en una tarea de manipulación de objetos. El sistema que une todos los conceptos presentados se muestra en la Figura 41, en donde se aprecia la existencia de 9 módulos que se encargan de procesar la información y controlar el movimiento del robot. A continuación, cada módulo se explica brevemente. Este sistema se estructuró específicamente para que controlara el robot dentro del simulador CoppeliaSim, por este motivo se ve el uso de una API que comunica el simulador y el programa desarrollado en Python. En la Tabla 11, se enlistan los módulos y se describe su funcionamiento.

Tabla 11 Descripción de los módulos del sistema de seguridad

módulo	descripción
1	API encargada de intercambiar información entre el simulador y el sistema desarrollado en Python
2	Controla el movimiento del robot para seguir planeaciones de movimiento. Envía información a otros módulos de lo que capturan las cámaras y se encarga de controlar el accionamiento del gripper y la banda transportadora.
3	Genera la segmentación de instancia de objetos de la imagen.
4	Construye la nube de puntos de todo el entorno y de los objetos segmentados. La nube de puntos se forma a partir de la información entregada por tres cámaras y las segmentaciones realizadas por la Mask R-CNN. Cada nube de puntos es filtrada para eliminar cualquier punto que pertenezca a la superficie del robot.
5	A partir de la nube de puntos de los objetos de la banda crea dos matrices homogéneas a las cuales se debe dirigir el gripper para sujetar y depositar un objeto.
6	Calcula los vectores de estado que ubican al gripper en las posiciones y orientaciones especificadas por las matrices homogéneas generadas en el módulo 5.
7	Crea la matriz de discretización. Este módulo implementa el analizador de colisiones para clasificar todas las celdas de la matriz, además, asigna el costo de cada posición. Este módulo se ejecuta sólo cuando el entorno ha cambiado, de lo contrario no actualiza la matriz de discretización.
8	Aplica el algoritmo Dijkstra para hallar la solución del grafo. Esta solución indica cómo ir de un nodo inicial a cualquier nodo del grafo. Este módulo sólo se ejecuta cuando la matriz de discretización ha cambiado.
9	Genera la planeación de movimiento del robot a partir de dos arreglos. El primer arreglo indica cómo desplazar el robot para sujetar el objeto y el segundo indica cómo desplazar el objeto al depósito. El último arreglo se obtiene de la solución dada por Dijkstra.

Todo el sistema de seguridad mostrado en la Figura 41 permite efectuar de manera segura tareas de manipulación de objetos. A partir de este se puede evadir objetos con diferentes niveles de prioridad, por lo que se ha cumplido con el objetivo de esta investigación. Antes de pasar a mostrar los resultados obtenidos, se aclara que el segundo arreglo del módulo 9 indica una planeación de trayectoria en la cual las últimas tres componentes de los vectores de estado son unas constantes. Esto significa que el nodo alcanzado con Dijkstra no tendrá la orientación final requerida para depositar el objeto. Cuando se alcanza el último nodo de la solución, se debe mover las tres últimas articulaciones del robot para luego depositar el objeto. Se recuerda que Dijkstra sólo se encarga de desplazar al robot sin generar colisiones, la posición final no tendrá la orientación deseada en el gripper. El primer arreglo generado por el módulo 9 no implementa la solución dada por Dijkstra para mover el gripper debido a que en este desplazamiento no hay obstáculos que eviten ese movimiento.

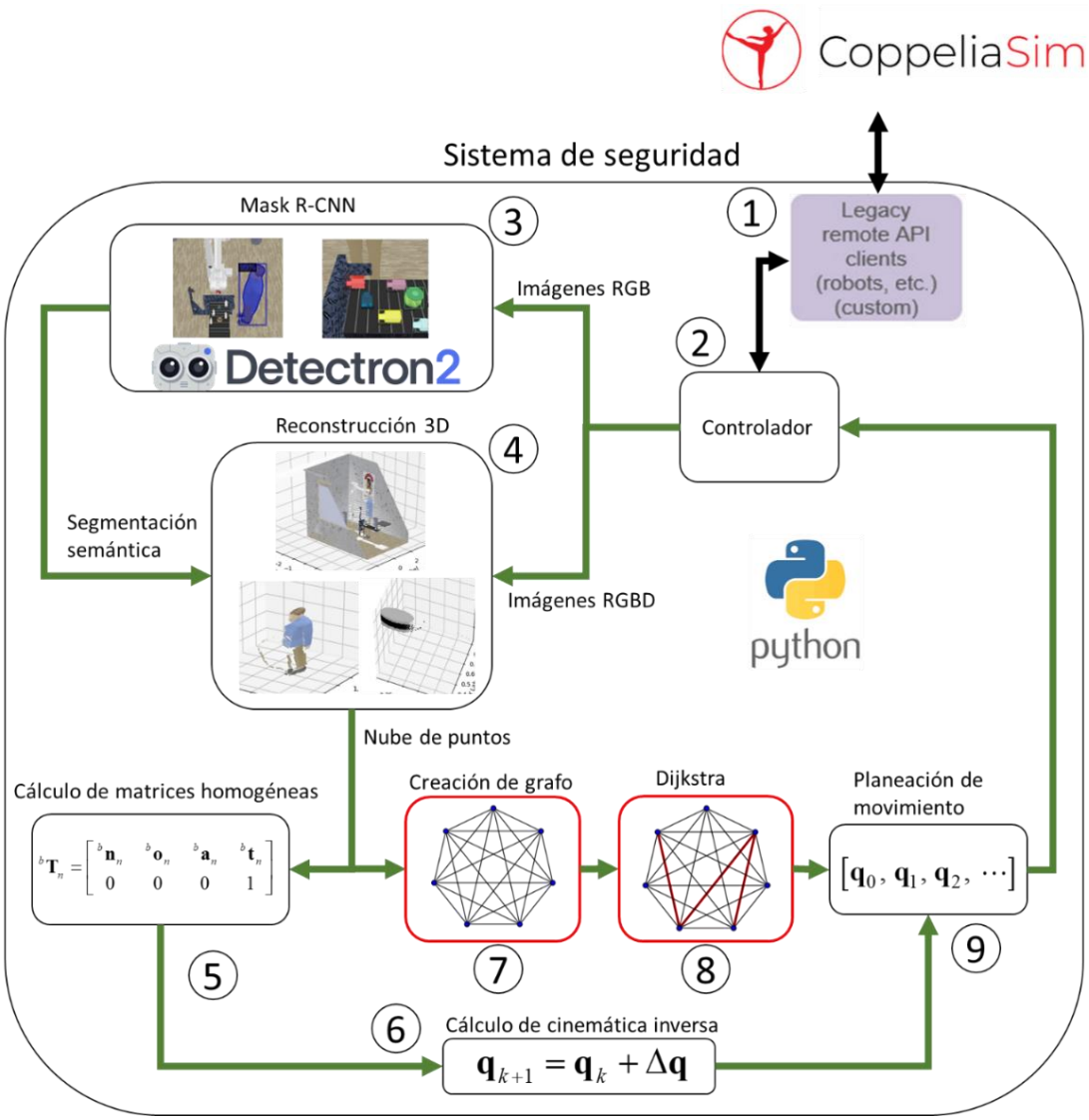


Figura 41 Sistema de seguridad.

Las imágenes que se presentan a continuación muestran las pruebas de funcionamiento hechas en el entorno virtual. El entorno creado contiene varios obstáculos que complican el desplazamiento del robot (paredes, banda transportadora, suelo, cámara, depósitos y operarios). Las personas son analizadas como obstáculos de alta prioridad, por lo que el sistema implementa la ecuación (76) para ajustar los costos del grafo. Obstáculos como paredes, banda transportadora, piso y demás, son tratados con prioridad mínima. En las siguientes figuras se puede observar el funcionamiento del sistema. Las soluciones mostradas fueron calculadas con una matriz de discretización de 30x30x30 celdas.

El primer desplazamiento que realiza el robot se muestra en la Figura 42. Este dirige el gripper hasta el objeto en la banda para sujetarlo. Debido a que en ese desplazamiento no hay obstáculos, el sistema no debe usar la solución dada por Dijkstra. Luego de sujetar el objeto y volver a la configuración inicial, obtiene de Dijkstra la planeación de movimiento para depositar el objeto (ver Figura 43). El desplazamiento realizado por el gripper evitó tanto al operario como a las

paredes. Se puede apreciar que la cercanía del gripper con las paredes es mucho más corta que con los operarios, esto se debe a la priorización dada. Los desplazamientos obtenidos para los demás objetos de la banda se pueden desde la Figura 43 a la Figura 46**.** Como se puede apreciar, todas las planeaciones de trayectoria fueron adecuadas.

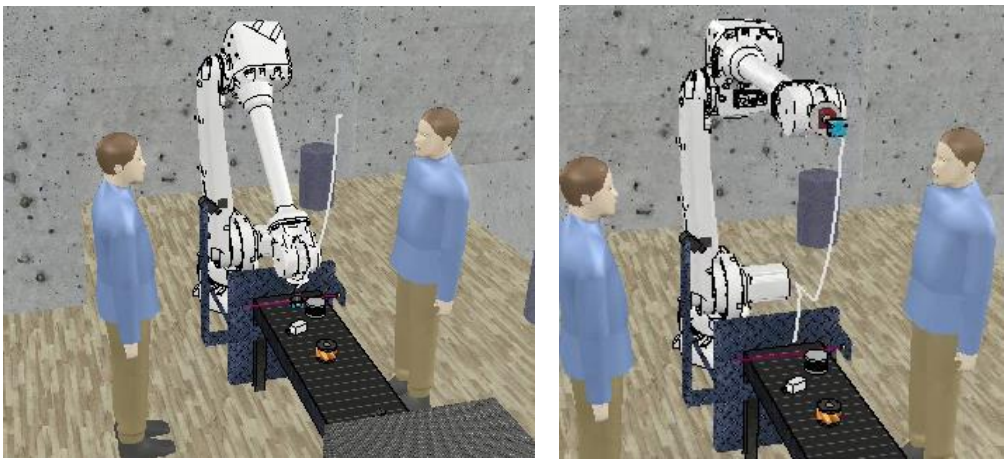


Figura 42 Desplazamiento del robot para sujetar objeto en la banda transportadora.

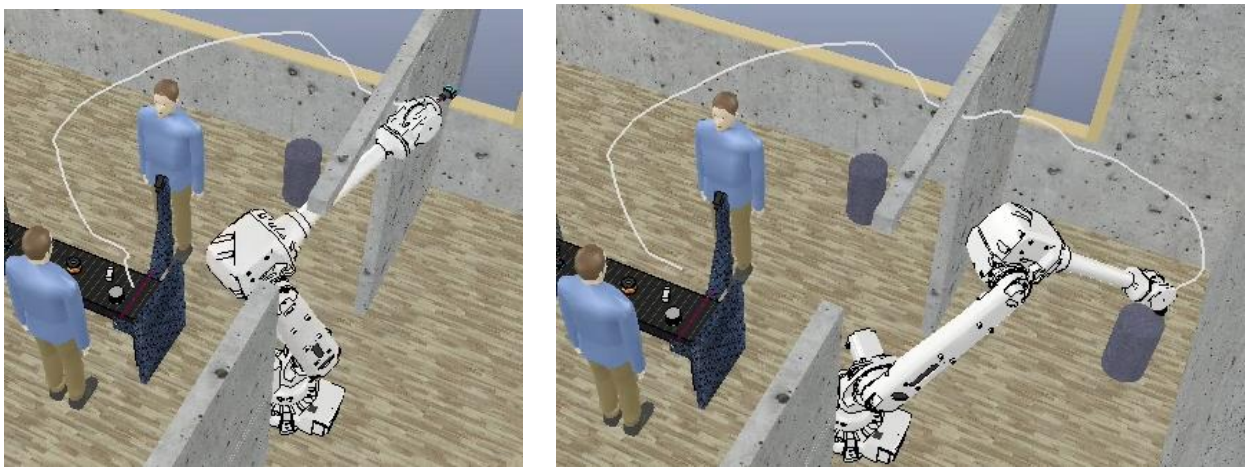


Figura 43 Planeación de trayectoria calculada para depositar objeto clasificado como "Tim310".

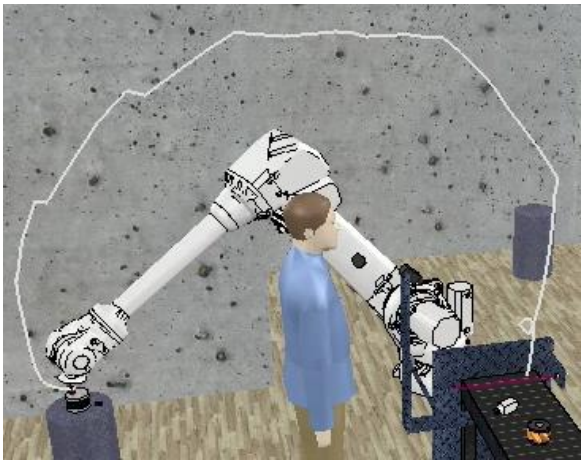


Figura 44 Planeación de trayectoria calculada para depositar objeto clasificados como "Belodyne".

Figura 45 Planeación de trayectoria calculada para depositar objeto clasificados como "Cámara".

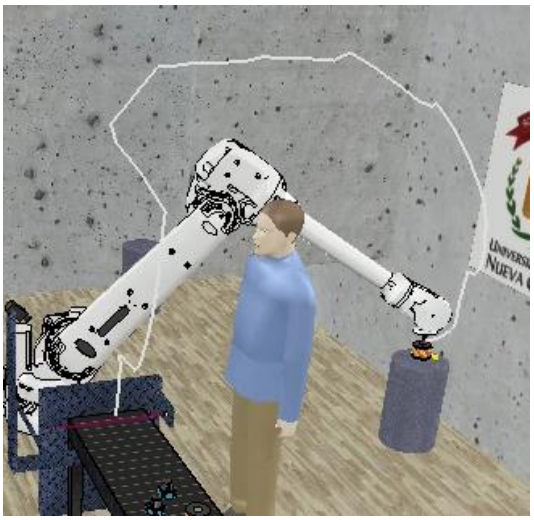


Figura 46 Planeación de trayectoria calculada para depositar objeto clasificados como "Rueda".

Planeaciones de trayectoria para entornos diferentes se muestran desde la Figura 47 a la Figura 51. En la Figura 49 se implementa un obstáculo ficticio para restringir la zona de trabajo del robot. Esto es similar a lo que hacen las empresas ABB y KUKA con los programas "Saves floor space" y "KUKA.safeoperation".

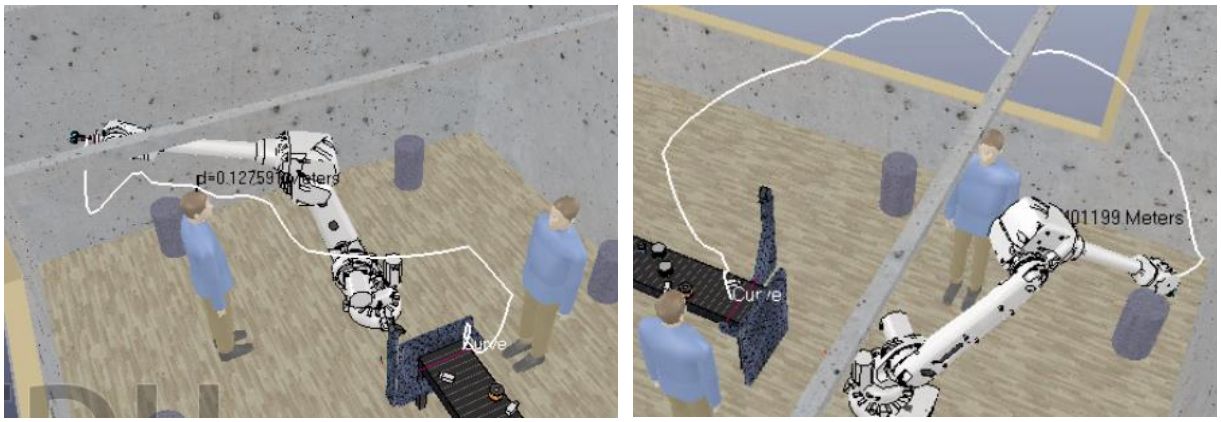


Figura 47 Planeación de trayectoria, prueba 5.

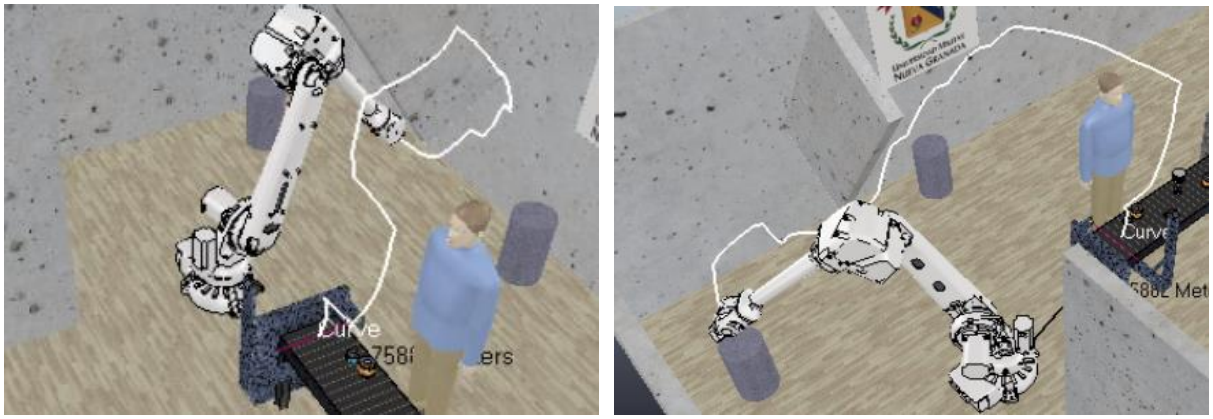


Figura 48 Planeación de trayectoria, prueba 6.

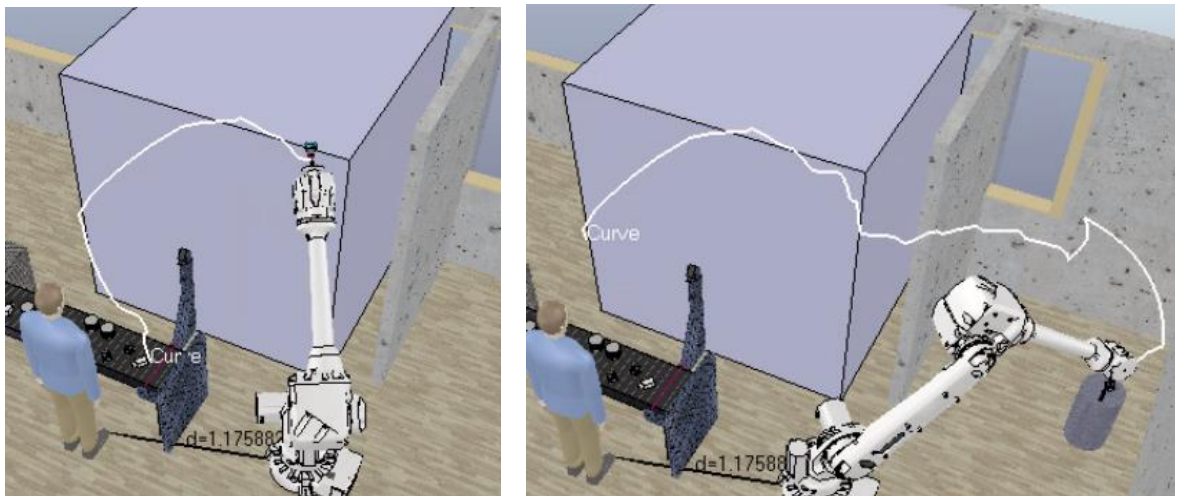


Figura 49 Planeación de trayectoria, prueba 7.

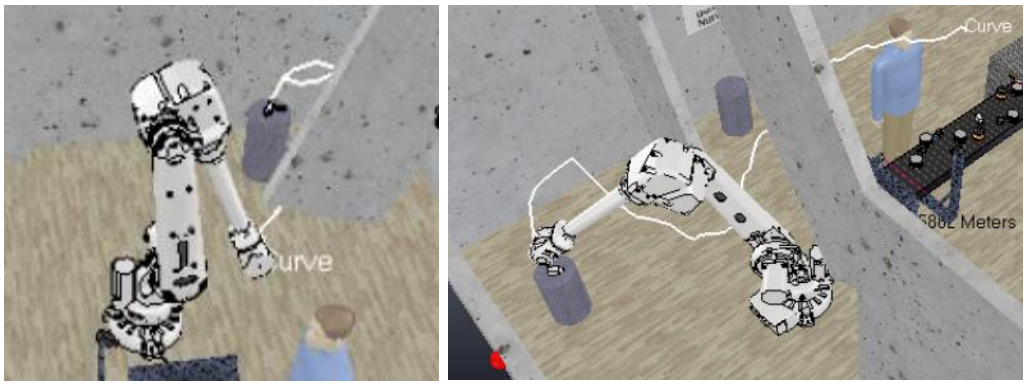


Figura 50 Planeación de trayectoria, prueba 8.

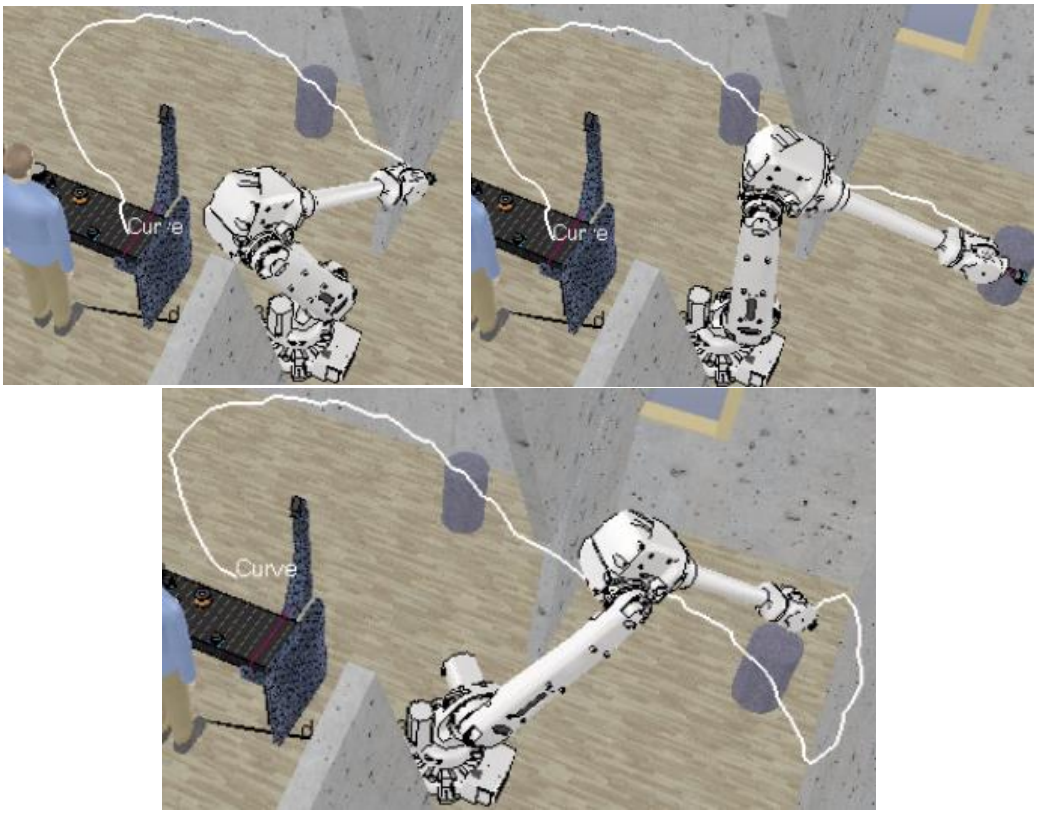


Figura 51 Planeación de trayectoria, prueba 9.

El efecto de usar diferentes prioridades en la evasión de obstáculos se puede ver desde la Figura 52 a la Figura 54. En estas se observa que entre más prioridad se asigne al operario, el robot más alejado intentará estar de este. Esto permite hacer que los trabajos colaborativos con personas puedan ser más seguros. Para revisar más a detalle la distancia mínima entre el robot y el obstáculo a lo largo del desplazamiento, se puede ver desde la Figura 55 a la Figura 57. Como se puede observar, la distancia mínima entre el robot y el operario para una prioridad mínima, media y máxima es de 0.15m, 0.205m. y 0.61m respectivamente.

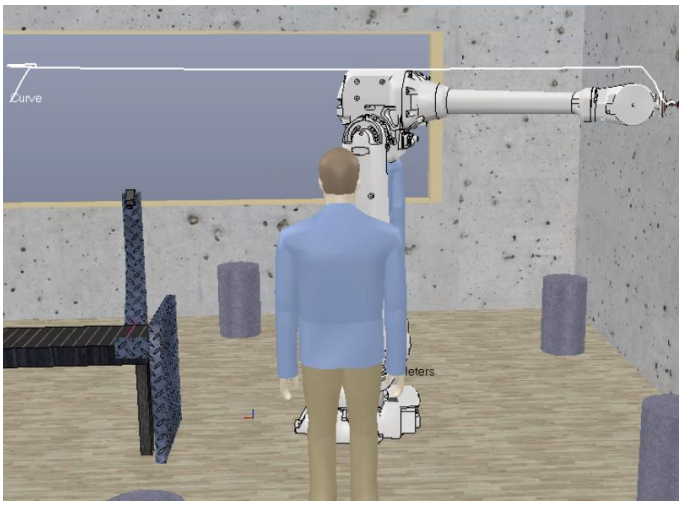


Figura 52 Planeación de movimiento con prioridad mínima.

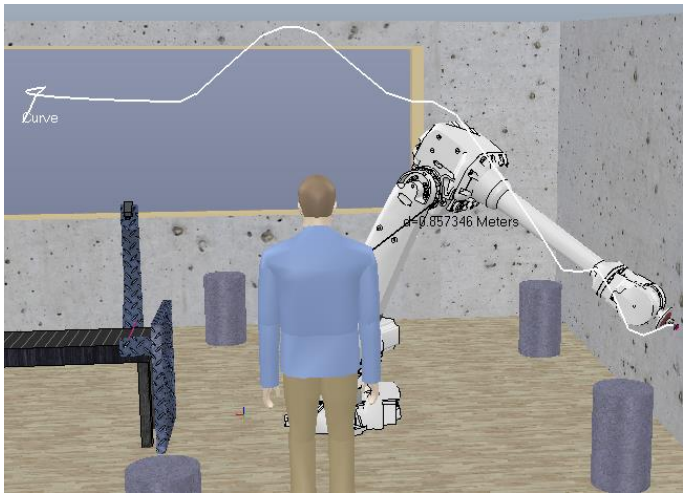


Figura 53 Planeación de movimiento con prioridad media.

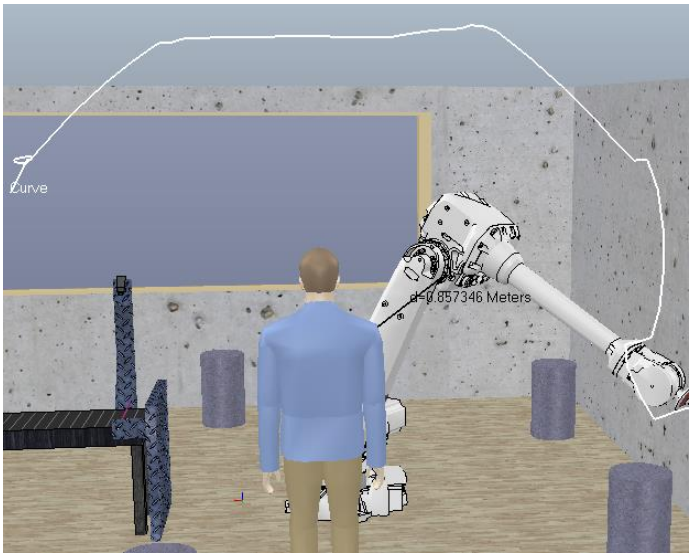


Figura 54 Planeación de movimiento con prioridad máxima.

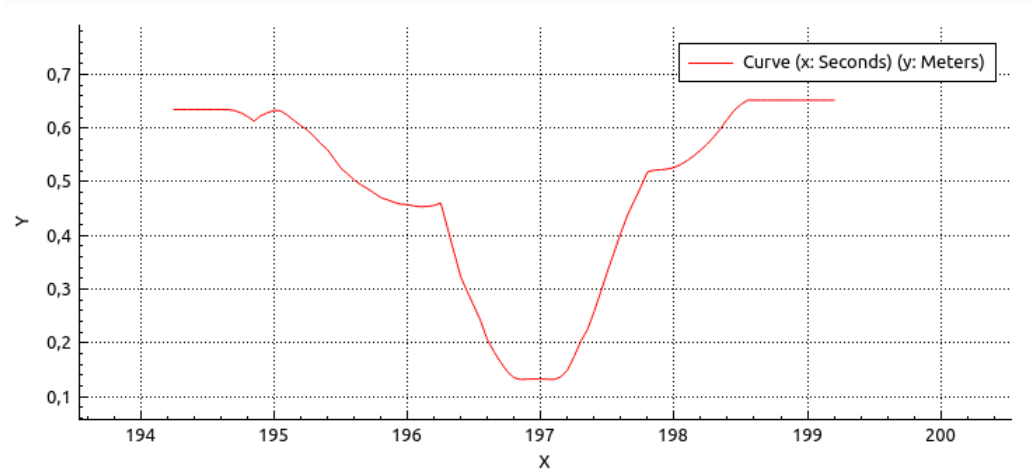


Figura 55 Distancia mínima entre el robot y el operario para la planeación de movimiento de la Figura 52.

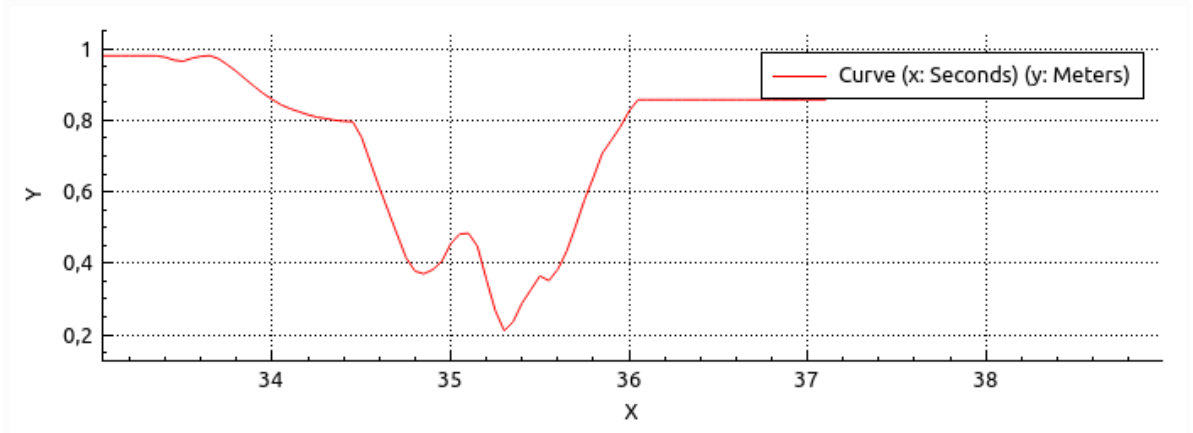


Figura 56 Distancia mínima entre el robot y el operario para la planeación de movimiento de la Figura 53.

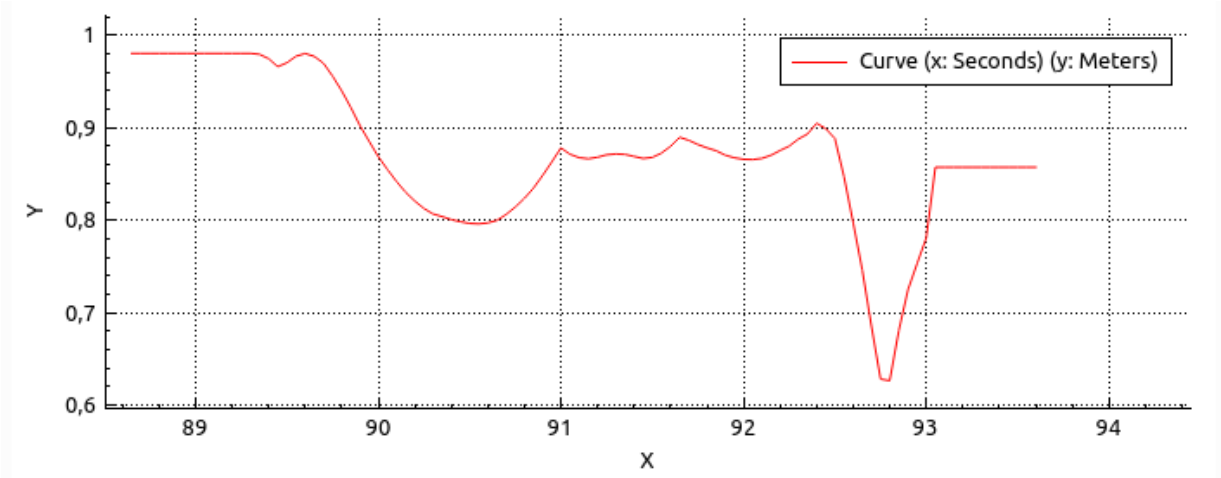


Figura 57 Distancia mínima entre el robot y el operario para la planeación de movimiento de la Figura 54.

9 Conclusiones

A partir de la investigación realizada se logra identificar que es viable desarrollar sistemas de seguridad activos para promover la integración de robots a entornos de alto riesgo. Incorporar herramientas de campos como el Deep Learning, la robótica y teoría de grafos, permiten crear un sistema de alta seguridad para evadir obstáculos estáticos con diferentes niveles de prioridad, lo que permite fomentar la integración de los robots en varios entornos. Aunque los resultados indican que el sistema sólo es viable en entornos estáticos, este puede implementarse para restringir zonas en las que los operarios pueden estar en movimiento, siendo de gran utilidad en la robótica colaborativa y estudiantil. Los planteamientos analizados dentro de esta investigación sirven de referencia para iniciar a analizar obstáculos dinámicos. Si se estudia cómo optimizar los algoritmos y reducir la cantidad de análisis dentro del entorno, es posible reducir el tiempo de ejecución del sistema y así poder evadir obstáculos en movimiento. Este sistema demuestra que el análisis del entorno y la planeación de movimiento son labores que pueden ser automatizadas, lo que permite aumentar la autonomía de los sistemas robóticos.

Implementar redes basadas en aprendizaje profundo da la posibilidad de mejorar el desempeño de los sistemas de seguridad. A partir de la segmentación de instancia de objetos es posible identificar y clasificar obstáculos de alta prioridad para hacer que la planeación de movimiento del robot los evite. Esto hace que los sistemas sean más seguros ya que se aumenta la distancia entre el robot y los operarios, por lo que se disminuye la probabilidad de presentarse colisiones de alto riesgo.

Implementar cámaras RGB-D es un enfoque altamente viable para analiza el estado del entorno y detectar obstáculos. Mediante la nube de puntos se puede representar la superficie de los obstáculos con los cuales puede colisionar el robot. De esta manera se caracteriza todo el entorno para calcular las configuraciones que el robot no debe alcanzar para evitar colisiones. Esto permite identificar todo el grupo de vectores de estado que el algoritmo de planeación de trayectoria debe usar para generar su solución.

El modelado del entorno operativo puede desarrollarse mediante matrices de transformación homogénea. Estas dan la posibilidad de caracterizar la posición de cada entidad dentro del entorno, lo que permite realizar análisis cinemáticos y de colisiones. Se ha demostrado que el vector de estado del robot puede calcularse mediante el método numérico Newton Raphson, el cual logra resultados más exactos en comparación con el método convencional, sin embargo, a un costo computacional mayor.

Discretizar y representar el entorno a partir de grafos es una metodología adecuada para calcular planeaciones de movimiento que deben evadir obstáculos. Los costos entre nodos que contiene el grafo pueden ser alterados para modificar la prioridad de evasión de los obstáculos, lo que permite hacer que el robot realice movimientos que alejen sus eslabones de los operarios.

La discretización que se plantee para crear el grafo debe desarrollarse con cuidado, ya que la cantidad de nodos usados se relaciona directamente con el costo computacional. Este es el principal motivo por el cual no se puede abordar entornos con obstáculos dinámicos. La creación de los grafos es la sección más costosa dentro del sistema, por lo que para incorporar objetos en

movimiento es necesario estudiar más a profundidad cómo construirlos, mientras que la solución de estos se puede generar mediante métodos clásicos que busquen el camino más corto entre nodos, como por ejemplo el algoritmo Dijkstra.

El entorno de simulación implementado demuestra que el sistema desarrollado tiene la capacidad de controlar al robot de manera segura en un entorno con obstáculos. El robot logró desarrollar tareas de ordenamiento de objetos evadiendo obstáculos con diferentes prioridades, lo que permitió concluir que los planteamientos descritos a lo largo del documento son viables para controlar sistemas de robóticos de manera segura en entornos estáticos.

10 Referencias

Alò, R., Bottiglione, F., & Mantriota, G. (2016). Artificial knee joints actuators with energy recovery capabilities. Journal of Robotics, 2016, 5.

Beasley, R. A. (2012). Medical robots: Current systems and research directions. *Journal of Robotics, 2012*

Benavides, J.E.H., Suescún, C.G.P. and Moreno, R.J. (2018). Scara robot path planning through flood fill algorithm. *International Journal of Applied Engineering Research, 13*(19), 14273-14281.

Benavides, J. E. H., Corredor, D. E. E., Moreno, R. J., & Hernández, R. D. (2018). Flood fill algorithm dividing matrices for robotic path planning. *International Journal of Applied Engineering Research, 13*(11), 8862-8870.

Bogue, R. (2017). Robots that interact with humans: A review of safety technologies and standards. *Industrial Robot: An International Journal, 44*(4), 395-400.

Buss, S. R. (2004). Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation, 17*(1-19), 16.

Buss, S. R., & Kim, J. (2005). Selectively damped least squares for inverse kinematics. *Journal of Graphics Tools, 10*(3), 37-49.

Calanca, A., & Fiorini, P. (2014). Human-adaptive control of series elastic actuators. *Robotica, 32*(8), 1301-1316.

Chapra, S. C. (2012). *Applied numerical methods with MATLAB for engineers and scientists* New York: McGraw-Hill,.

Cho, C., Kim, J., Kim, Y., Song, J., & Kyung, J. (2012). Collision detection algorithm to distinguish between intended contact and unexpected collision. *Advanced Robotics, 26*(16), 1825-1840.

Cho, C., Kim, J., Lee, S., & Song, J. (2012). Collision detection and reaction on 7 DOF service robot arm using residual observer. *Journal of Mechanical Science and Technology, 26*(4), 1197-1203.

Cisneros, M. A. P., Jiménez, E. V. C., & Navarro, D. Z. (2014). *Fundamentos de robótica y mecatrónica con MATLAB© y simulink©* RA-MA Editorial.

Conkur, E. S. (2005). Path planning using potential fields for highly redundant manipulators. *Robotics and Autonomous Systems, 52*(2-3), 209-228.

Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., . . . Wurman, P. R. (2016a). Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering, 15*(1), 172-188.

Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., . . . Wurman, P. R. (2016b). Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering, 15*(1), 172-188.

Craig, J. J. (2009). *Introduction to robotics: Mechanics and control, 3/E* Pearson Education India.

Dahari, M., & Tan, J. (2011). Forward and inverse kinematics model for robotic welding process using KR-16KS KUKA robot. Paper presented at the *2011 Fourth International Conference on Modeling, Simulation and Applied Optimization*, 1-6.

De Luca, A., Albu-Schaffer, A., Haddadin, S., & Hirzinger, G. (2006). Collision detection and safe reaction with the DLR-III lightweight manipulator arm. Paper presented at the *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1623-1630.

De Luca, A., & Flacco, F. (2012). Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration. Paper presented at the *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 288-295.

Deng, Y., Chen, Y., Zhang, Y., & Mahadevan, S. (2012). Fuzzy dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing, 12*(3), 1231-1237.

Djuric, A. M., Urbanic, R. J., & Rickli, J. L. (2016). A framework for collaborative robot (CoBot) integration in advanced manufacturing systems. *SAE International Journal of Materials and Manufacturing, 9*(2), 457-464.

Dubrofsky, E. (2009). Homography estimation. diplomová práce. *Vancouver: Univerzita Britské Kolumbie*,

Erden, M. S., & Tomiyama, T. (2010). Human-intent detection and physically interactive control of a robot without force sensors. *IEEE Transactions on Robotics, 26*(2), 370-382.

Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision, 111*(1), 98-136.

Faber, M., Bützler, J., & Schlick, C. M. (2015). Human-robot cooperation in future production systems: Analysis of requirements for designing an ergonomic work system. *Procedia Manufacturing, 3*, 510-517.

Fenucci, A., Indri, M., & Romanelli, F. (2014). A real time distributed approach to collision avoidance for industrial manipulators. Paper presented at the *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, 1-8.

Ferraro, A., Indri, M., & Lazzero, I. (2012). Dynamic update of a virtual cell for programming and safe monitoring of an industrial robot. *IFAC Proceedings Volumes, 45*(22), 822-827.

Flacco, F., Kröger, T., De Luca, A., & Khatib, O. (2012). A depth space approach to human-robot collision avoidance. Paper presented at the *2012 IEEE International Conference on Robotics and Automation,* 338-345.

Förstner, W., & Wrobel, B. P. (2016). *Photogrammetric computer vision* Springer.

Fratu, A., Vermeiren, L., & Dequidt, A. (2010). Using the redundant inverse kinematics system for collision avoidance. Paper presented at the *2010 3rd International Symposium on Electrical and Electronics Engineering (ISEEE),* 88-93.

Gageik, N., Müller, T., & Montenegro, S. (2012). Obstacle detection and collision avoidance using ultrasonic distance sensors for an autonomous quadrocopter. *University of Wurzburg, Aerospace Information Technologhy (Germany) Wurzburg, ,* 3-23.

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2014). Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. Paper presented at the *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2997-3004.

Gandhi, D., Pinto, L., & Gupta, A. (2017). Learning to fly by crashing. Paper presented at the *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* 3948-3955.

Gaya, J. O., Gonçalves, L. T., Duarte, A. C., Zanchetta, B., Drews, P., & Botelho, S. S. (2016). Vision-based obstacle avoidance using deep learning. Paper presented at the *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR),* 7-12.

Girshick, R. (2015). Fast r-cnn. Paper presented at the *Proceedings of the IEEE International Conference on Computer Vision,* 1440-1448.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* MIT press.

Grimmer, M., Eslamy, M., & Seyfarth, A. (2014). Energetic and peak power advantages of series elastic actuators in an actuated prosthetic leg for walking and running. Paper presented at the *Actuators, , 3*(1) 1-19.

Haddadin, S., Albu-Schaffer, A., De Luca, A., & Hirzinger, G. (2008). Collision detection and reaction: A contribution to safe physical human-robot interaction. Paper presented at the *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 3356-3363.

Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision* Cambridge university press.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. Paper presented at the *Proceedings of the IEEE International Conference on Computer Vision,* 2961-2969.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 770-778.

Husty, M. L., Pfurner, M., Schröcker, H., & Brunnthaler, K. (2007). Algebraic methods in mechanism analysis and synthesis. *Robotica, 25*(6), 661-675.

Huynh, J. (2009). Separating axis theorem for oriented bounding boxes. *URL: Jkh.Me/Files/Tutorials/Separating% 20Axis% 20Theorem% 20for% 20Oriented% 20Bounding% 20Boxes.Pdf,*

Ifr. (n.d.). Industrial robot sales increase worldwide by 31 percent. Retrieved from https://ifr.org/ifr-press-releases/news/industrial-robot-sales-increase-worldwide-by-29-percent

Jabbar, A. M. (2016). Autonomous navigation of mobile robot based on flood fill algorithm. *Iraqi Journal for Electrical and Electronic Engineering, 12*(1), 79-84.

Jayaraj, A., & Divakar, H. N. (2018). Robotics in construction industry. Paper presented at the *IOP Conference Series: Materials Science and Engineering, , 376*(1) 012114.

Kenwright, B. (2012). Inverse kinematics–cyclic coordinate descent (CCD). *Journal of Graphics Tools, 16*(4), 177-217.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Autonomous robot vehicles* (pp. 396-404) Springer.

Kock, S., Vittor, T., Matthias, B., Jerregard, H., Källman, M., Lundberg, I., . . . Hedelind, M. (2011). Robot concept for scalable, flexible assembly automation: A technology study on a harmless dual-armed robot. Paper presented at the *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM),* 1-5.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Paper presented at the *Advances in Neural Information Processing Systems,* 1097-1105.

Kuffner, J. J., & LaValle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning. Paper presented at the *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. no. 00CH37065), , 2* 995-1001.

Lam, T. L., Yip, H. W., Qian, H., & Xu, Y. (2012). Collision avoidance of industrial robot arms using an invisible sensitive skin. Paper presented at the *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 4542-4543.

Lasota, P. A., Rossano, G. F., & Shah, J. A. (2014). Toward safe close-proximity human-robot interaction with standard industrial robots. Paper presented at the *2014 IEEE International Conference on Automation Science and Engineering (CASE),* 339-344.

Lauzier, N., & Gosselin, C. (2011). Series clutch actuators for safe physical human-robot interaction. Paper presented at the *2011 IEEE International Conference on Robotics and Automation,* 5401-5406.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.

Lee, S., Baek, S., & Kim, J. (2015). Arm trajectory generation based on RRT* for humanoid robot. *Robot intelligence technology and applications 3* (pp. 373-383) Springer.

Liang, M., & Hu, X. (2015). Recurrent convolutional neural network for object recognition. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3367-3375.

Lin, T., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., . . . Zitnick, C. L. (2014). Microsoft coco: Common objects in context. Paper presented at the *European Conference on Computer Vision*, 740-755.

Miyajima, R. (2017). Deep learning triggers a new era in industrial robotics. *IEEE Multimedia, 24*(4), 91-96.

Mohammed, A., Schmidt, B., & Wang, L. (2017). Active collision avoidance for human–robot collaboration driven by vision sensors. *International Journal of Computer Integrated Manufacturing, 30*(9), 970-980.

Morris, A. (20). Years later: Cobots co-opt assembly lines. *Northwestern University: McCormick School of Engineering*,

Navarro, S. E., Marufo, M., Ding, Y., Puls, S., Göger, D., Hein, B., & Wörn, H. (2013). Methods for safe human-robot-interaction using capacitive tactile proximity sensors. Paper presented at the *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1149-1154.

Neumann, T., Ferrein, A., Kallweit, S., & Scholl, I. (2014). Towards a mobile mapping robot for underground mines. Paper presented at the *Proceedings of the 2014 PRASA, RobMech and AfLaT International Joint Symposium, Cape Town, South Africa*, 27-28.

Occupational Safety and Health Administration.Accident search results. (n.d.). Retrieved from https://www.osha.gov/pls/imis/accidentsearch.search?sic=&sicgroup=&naics=&acc_description=&acc_abstract=&acc_keyword=%22Robot%22&inspnr=&fatal=&officetype=&office=&startmonth=&startday=&startyear=&endmonth=&endday=&endyear=&keyword_list=on&p_start=&p_finish=0&p_sort=&p_desc=DESC&p_direction=Next&p_show=20

Pearson, D., & Bryant, V. (2004). *Decision maths 1* Heinemann.

Peshkin, M., Colgate, J. E., Akella, P., & Wannasuphoprasit, W. (2000). Cobots in materials handling. *Northwestern University, Evanston*,

Ping, G., Wei, B., Li, X., & Luo, X. (2009). Real time obstacle avoidance for redundant robot. Paper presented at the *2009 International Conference on Mechatronics and Automation*, 223-228.

Prempraneerach, P., & Kulvanit, P. (2010). Implementation of resolved motion rate controller with 5-axis robot manipulator arm. Paper presented at the *The First TSME International Conference on Mechanical Engineering*, 1-8.

Prescott, T. J., Pearson, M. J., Mitchinson, B., Sullivan, J. C. W., & Pipe, A. G. (2009). Whisking with robots. *IEEE Robotics & Automation Magazine, 16*(3), 42-50.

Puiu, D., & Moldoveanu, F. (2011). Real-time collision avoidance for redundant manipulators. Paper presented at the *2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 403-408.

Redes neuronales convolucionales. (n.d.). Retrieved from https://la.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Redmon, J., & Farhadi, A. (2018a). Yolov3: An incremental improvement. *arXiv Preprint arXiv:1804.02767*,

Redmon, J., & Farhadi, A. (2018b). Yolov3: An incremental improvement. *arXiv Preprint arXiv:1804.02767*,

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Paper presented at the *Advances in Neural Information Processing Systems*, 91-99.

Reyes Cortes, F. (2012). *Matlab aplicado a robótica y mecatrónica*

Roldán, J. J., del Cerro, J., Garzón-Ramos, D., Garcia-Aunon, P., Garzón, M., de León, J., & Barrientos, A. (2018). Robots in agriculture: State of art and practical experiences. *Service Robots*,

Rouse, E. J., Mooney, L. M., & Herr, H. M. (2014). Clutchable series-elastic actuator: Implications for prosthetic knee design. *The International Journal of Robotics Research, 33*(13), 1611-1625.

Ruan, X., & Li, W. (2014). Ultrasonic sensor based two-wheeled self-balancing robot obstacle avoidance control system. Paper presented at the *2014 IEEE International Conference on Mechatronics and Automation*, 896-900.

SafeMove 2 - IRC5 options (IRC5 controller). (n.d.). Retrieved from https://new.abb.com/products/robotics/controllers/irc5/irc5-options/safemove-2

Schlegl, T., Kröger, T., Gaschler, A., Khatib, O., & Zangl, H. (2013). Virtual whiskers—Highly responsive robot collision avoidance. Paper presented at the *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5373-5379.

Schütz, S., Nejadfard, A., Kötting, C., & Berns, K. (2016). An intuitive and comprehensive two-load model for series elastic actuators. Paper presented at the *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, 573-580.

Seitz, M. (2017). Qué países tienen más robots en sus fábricas y cuán cierto es que nos están robando los puestos de trabajo. Retrieved from https://www.bbc.com/mundo/noticias-39267567

Shafiee, M. J., Chywl, B., Li, F., & Wong, A. (2017). Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *arXiv Preprint arXiv:1709.05943,*

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv Preprint arXiv:1409.1556,*

Spong, M. W., & Vidyasagar, M. (2008). *Robot dynamics and control* John Wiley & Sons.

Sucan, I. A., Moll, M., & Kavraki, L. E. (2012). The open motion planning library. *IEEE Robotics & Automation Magazine, 19*(4), 72-82.

Sullivan, J. C., Mitchinson, B., Pearson, M. J., Evans, M., Lepora, N. F., Fox, C. W., . . . Prescott, T. J. (2011). Tactile discrimination using active whisker sensors. *IEEE Sensors Journal, 12*(2), 350-362.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. Paper presented at the *Thirty-First AAAI Conference on Artificial Intelligence,*

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 1-9.

Tai, L., Li, S., & Liu, M. (2016). A deep-network solution towards model-less obstacle avoidance. Paper presented at the *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* 2759-2764.

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A survey on deep transfer learning. Paper presented at the *International Conference on Artificial Neural Networks,* 270-279.

Tanaka, M., Kon, K., & Tanaka, K. (2015). Range-sensor-based semiautonomous whole-body collision avoidance of a snake robot. *IEEE Transactions on Control Systems Technology, 23*(5), 1927-1934.

Tekin, B., Sinha, S. N., & Fua, P. (2018). Real-time seamless single shot 6d object pose prediction. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 292-301.

Veloso, M., Biswas, J., Coltin, B., Rosenthal, S., Kollar, T., Mericli, C., . . . Ventura, R. (2012). Cobots: Collaborative robots servicing multi-floor buildings. Paper presented at the *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 5446-5447.

Vogel, C., Fritzsche, M., & Elkmann, N. (2016). Safe human-robot cooperation with high-payload robots in industrial applications. Paper presented at the *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI),* 529-530.

Woo, M., Neider, J., Davis, T., & Shreiner, D. (1999). *OpenGL programming guide: The official guide to learning OpenGL, version 1.2* Addison-Wesley Longman Publishing Co., Inc.

Wu, Y., Kirillov, A., Massa, F., Lo, W., & Girshick, R. (2019). No title. *Detectron2,*

Xie, L., Wang, S., Markham, A., & Trigoni, N. (2017). Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv Preprint arXiv:1706.09829,*

Yagnik, D., Ren, J., & Liscano, R. (2010). Motion planning for multi-link robots using artificial potential fields and modified simulated annealing. Paper presented at the *Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications,* 421-427.

Zhang, C., Yan, Y., Wen, C., Yang, J., & Yu, H. (2018). A nonsmooth composite control design framework for nonlinear systems with mismatched disturbances: Algorithms and experimental tests. *IEEE Transactions on Industrial Electronics, 65*(11), 8828-8839.

Zhang, Z. (2000). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(11), 1330-1334.