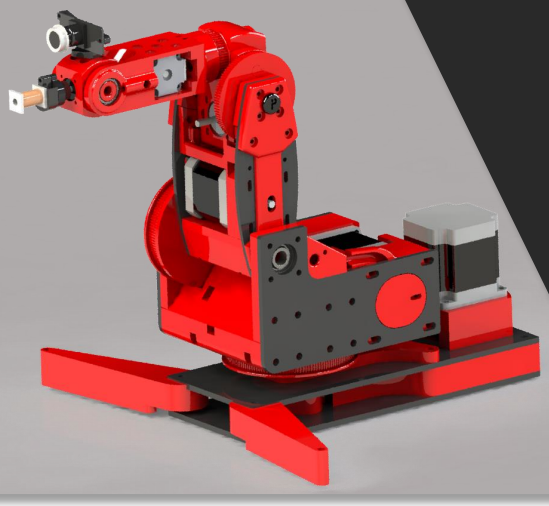


UNIVERSIDAD POLITÉCNICA DE CHIAPAS

INGENIERÍA MECATRÓNICA

M. C. Mario Wenso Morales Coutiño

ESTADÍA PROFESIONAL

CONTROL DE TRAYECTORIA DE ROBOT ARTICULADO POR VISIÓN ARTIFICIAL

PRESENTA:

TEVERA RUIZ ALEJANDRO 173049

10MO CUATRIMESTRE

COMITÁN DE DOMÍNGUEZ, CHIAPAS A 14 DE DICIEMBRE DE 2020

Índice de contenido

Introducción

La importancia que tiene los robots actualmente ha llegado a fines insospechables. Desde las industrias hasta las oficinas podemos encontrar actividad de estos seres artificiales; sin embargo, siempre ha existido una variable con la que los ingenieros han buscado mejorar. ¿Cómo hacer que estos seres piensen y actúen más como humanos y no como simples maquinas? La respuesta, mediante la programación, con un sistema operativo llamado ROS.

Dado el interés incremental en la construcción, diseño y control de robots comerciales y no comerciales, es necesaria la estandarización de su arquitectura para permitir colaboraciones efectivas con el equipo de trabajo, así como desarrollar sistemas complejos a partir de pequeños scripts con base a diversas herramientas y bibliotecas de código abierto para un avance en robótica sustancial en todo el mundo.

Considerando el enfoque tecnológico de la Universidad Politécnica de Chiapas, es importante considerar herramientas que permitan un desarrollo integral dentro de la comunidad estudiantil. De esta manera, la adopción de ROS dentro de los planes de estudio de Ingeniería Mecatrónica, otorgaría un soporte extra al desarrollo tecnológico y de innovación dentro del área de robótica; que, en conjunto con otras carreras, puede alcanzarse la ejecución de proyectos con alto impacto social, ambiental y económico. Aunado a eso, los egresados de Ingeniería Mecatrónica tendrían una importante ventaja en caso que deseen realizar estancias o estadías en empresas como Google, NASA, MIT, etc; principalmente por el conocimiento y la experiencia con ROS.

Objetivos

General

- Implementar visión artificial para el control de trayectoria de robot articulado de seis grados de libertad.

Particulares

- Desarrollar la red de nodos necesaria para el control de un brazo antropomórfico de seis grados de libertad por visión artificial mediante ROS.
- Obtener un prototipo robótico para el desarrollo de prácticas en el Laboratorio de Visión Artificial y Robótica Avanzada.

Planteamiento del problema

El desarrollo de prototipos por parte de los alumnos generalmente se ve limitada por la poca capacidad de procesamiento de las tarjetas de desarrollo comerciales. Por el contrario, pueden adquirirse brazos robots avanzados para el desarrollo de prácticas complejas con elementos como visión artificial o inteligencia artificial, pero esto supone un costo alto durante su adquisición. Además, al ser un sistema cerrado, proporciona poca adaptabilidad a periféricos ajenos a la marca, sumando costo a cualquier práctica.

De esta manera, se requiere del diseño de un prototipo robótico capaz de poder conectarse a diversos periféricos y ofertar una plataforma de desarrollo integral en computadoras económicas de placa reducida como las ofertadas por Raspberry Pi Foundation.

Antecedentes

El desarrollo de prototipos robóticos es cada vez más común, tanto por aficionados en el área, así como por instituciones educativas de prestigio. Por lo que el desarrollo y las técnicas implementadas son diversas, permitiendo enfoques, aplicaciones, diseños y controles diferentes. A continuación, se presentan algunos de los prototipos desarrollados alrededor del mundo que coinciden con ciertos aspectos con el prototipo expuesto en este documento.

Raspberry Pi Robot Arm with simple Computer Vision por Tobias Kuhn

Con base a (Kuhn, 2017) puede observarse en la Figura 1 el desarrollo del prototipo está enfocado al uso de servomotores Dynamixel AX-12A para el movimiento de las cuatro articulaciones. Como efector final utiliza un embobinado casero para la interacción con pequeños tornillos al ser detectados por una cámara Raspberry Pi instalada en el penúltimo eslabón del robot.

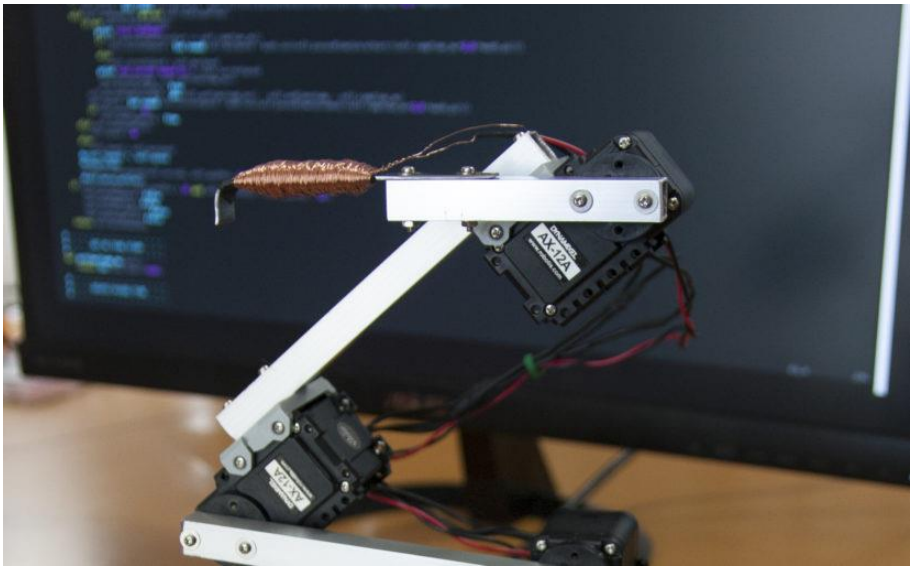


Figura 1. Raspberry Pi Robot Arm with simple Computer Vision (Kuhn, 2017)

Como característica principal, el autor menciona que no se ha utilizado OpenCV para el desarrollo de visión artificial, por lo que la programación puede resultar más personal y liviana para procesamiento de datos por parte de la Raspberry Pi 2 Modelo B. El software del robot puede encontrarse en (Kuhn, ScrewPicker, 2017).

BCN3D Moveo ROS por Jesse Weisberg

Utilizando la unidad mecánica desarrollada por la compañía BCN3D (BCN3D, 2016), Jesse Weisberg desarrolló un sistema de detección de objetos utilizando Machine Learning para el posterior control del robot de cinco grados de libertad mostrado en la Figura 2.

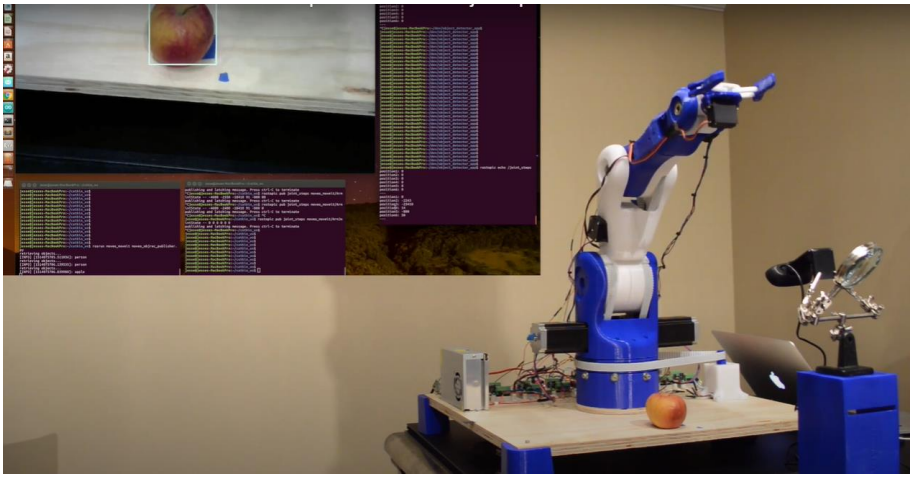


Figura 2. BCN3D Moveo ROS (Weisberg, 2018)

Cada grado de libertad es movido mediante motores a paso, mientas que el gripper, como efector final, es accionado por un servomotor. Todo el sistema es controlado mediante ROS (Weisberg, 2018), por lo que permite el desarrollo flexible de nuevas habilidades.

Arduino Robot Arm with Computer Vision por Gaultier Lecaillon

Utilizando servomotores MG996R, Lecaillon construyó un robot antropomórfico de tres grados de libertad con un gripper como efector final, presente en la Figura 3. El desarrollo del proyecto está basado puramente en OpenCV y comunicación serial con una placa Arduino UNO para la interacción con piezas de corcho para posicionarlas dentro de un contenedor (Lecaillon, 2019).

La principal ventaja es el entorno de desarrollo utilizado; ya que al utilizar OpenCV, puede implementarse dicho control en computadoras con Sistema Operativo Microsoft Windows, evitando tener que migrar de S.O. como en el caso de los prototipos anteriores.

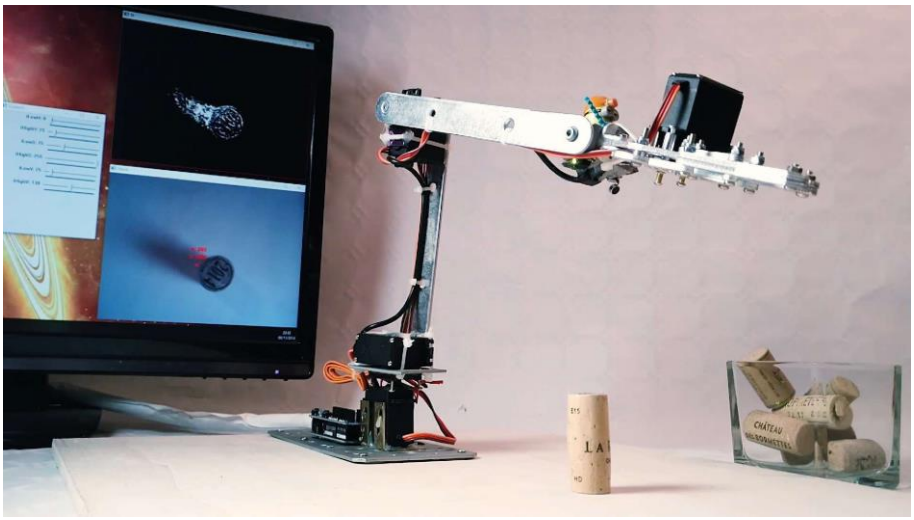


Figura 3. Arduino Robot arm with Computer Vision (Lecaillon, 2019)

Considerando los prototipos anteriores, es constante el desarrollo de prototipos enfocados al control de la unidad mecánica mediante visión artificial; por lo que pueden tenerse diversos caminos para alcanzar dicha meta.

Personalmente, el desarrollo hecho por Jesse Weisberg es impresionante; puesto que combina diversas técnicas de reconocimiento de objetos dentro del entorno de ROS. Razón por la cual, dentro de los tres prototipos presentados es mejor desarrollado.

Estado del arte

Fundamentos de robótica

La robótica es una ciencia o rama de la tecnología, que estudia el análisis, diseño, construcción y aplicación de robots principalmente en tareas laboriosas, repetitivas o peligrosas. Causa por la que existen diversas áreas y tipos de robots dependiendo del ambiente en que nos encontremos.

En específico, un robot industrial de acuerdo a la RIA[1] ≪*es un manipulador multifuncional reprogramable, capaz de mover piezas, herramientas o dispositivos especiales; según trayectorias variables, programadas para realizar tareas diversas*≫.

Clasificación

De acuerdo a su estructura mecánica, los robots industriales o brazos robóticos pueden clasificarse en:

- **Cartesiano** - Se caracterizan por posicionarse mediante 3 articulaciones lineales. Además, hacen movimientos perpendiculares a partir de los 3 ejes cartesianos *X, Y* y *Z*.

  Este robot proporciona un nivel adecuado de precisión y repetibilidad, programación fácil y se adquiere por un costo económico. Por lo tanto, es una solución económica para la industria, ya que ejecuta operaciones sencillas. Estas pueden ser: soldar, colocar o escoger de manera eficiente y barata.

- **SCARA** - Se mueve en planos cartesianos que el anterior. Su diferencia radica en la incorporación de un eje final del plano Z. Con este es posible hacer girar la herramienta en la parte final del brazo robótico.

[1] Asociación de Industrias Robóticas

Por ello, los robots SCARA son ideales para procesos de ensamblaje. Sin embargo, no son tan universales, ya que la terminación del brazo acorta su alcance.

- **Antropomórfico** - Son llamados así por sus articulaciones al asemejarse al brazo humano, ya que pueden colocar su efector final en una posición y orientación específica; para esto se requiere al menos seis grados de libertad. Lo que permite tener más flexibilidad para las diferentes tareas o aplicaciones industriales. En varias ocasiones, tienden a utilizarse como robots colaborativos.

Actuadores

Las cadenas cinemáticas requieren de dispositivos específicos para cambiar su posición de acuerdo a la programación establecida en la tarea a realizar. En los brazos robóticos industriales generalmente se utilizan servomotores de corriente alterna, dada la alta precisión en sus movimientos, control relativamente sencillo y alto torque para procesos donde sea requerido.

Para el caso de prototipos educativos o de primera etapa, se opta por actuadores más económicos, además que su etapa de potencia es mucho más sencilla que un servomotor de corriente alterna. Dentro de ellos puede encontrarse:

- Motor a pasos – Requieren de un encoder extra para determinar su posición real y calcular el error para tener un control preciso. Dependiendo del modelo, puede alcanzar un torque considerable.
- Servomotor de corriente directa – Dado su encoder integrado, su uso es más sencillo, pero carece de torque en tareas más demandantes.

Efectores finales

De acuerdo a lo visto anteriormente, los robots están diseñados para tener una interacción con el ambiente en que se encuentran de acuerdo a la tarea asignada. De esta manera, el humano ha requerido desarrollar diversas herramientas, dentro de ellas pueden encontrarse:

- **Gripper** – Sujeta un objeto, normalmente la pieza de trabajo, de acuerdo al ciclo de trabajo del robot. Generalmente se emplean casquetes de sujeción, imanes, ganchos y cucharas.
- **Pinza soldadura por puntos** – Realiza el proceso de soldado al cerrarse los dos electrodos sobre la pieza a soldar.
- **Soplete de soldadura de arco** – Aporta el flujo de electrodo que se funde para realizar el proceso de soldado a una pieza o ensamble.
- **Cucharón para colada** – Manejo de piezas calientes en procesos de fundición.
- **Atornillador (Desarmador)** – Ajusta tornillos, generalmente para el ensamble de productos.
- **Fresa/lija** – Perfila, elimina rebabas o pule superficies.
- **Pistola de pintura** – Pulveriza la pintura necesaria para darle un acabado a un producto o pieza.
- **Cañón láser** – Realiza cortes en materiales, proceso de soldado o inspección de piezas de acuerdo a la tarea asignada.
- **Cañón de agua a presión** – Ejecuta cortes en materiales.
- **Electroimán** – Sujeta piezas metálicas mediante la fuerza magnética generada por el paso de corriente dentro de la bobina.

Fundamentos de electrónica

Unidades de procesamiento

Para el control de sistemas, comúnmente se requiere de microcontroladores; circuitos integrados digitales programables. Están compuestos por una unidad central de proceso (CPU), memorias (ROM y RAM) y líneas de entrada y salida (periféricos).

De manera comercial, Arduino ha tomado la delantera en el diseño y comercio de placas de desarrollo basadas en microcontroladores. Su ventaja es la facilidad en su programación, permitiendo ser una plataforma de inicio en la programación de sistemas automáticos. Desafortunadamente, las placas de Arduino son poco potentes en procesamiento y no permiten multiprocesos en paralelo, lo que limita el uso de sistemas de control más sofisticados.

En caso se requiera la aplicación de sistemas más robustos, se recomienda el uso de dispositivos más potentes. Dentro de lo comercial, el procesador ESP32 desarrollado por el fabricante *Espressif* tiene mucha más potencia de procesamiento, así como soporte de procesamiento en paralelo gracias a sus dos núcleos de 32 bits, ofertando además comunicación WiFi y Bluetooth de manera integrada. Razones por las cuales, este dispositivo ha tenido gran aceptación dentro del desarrollo de proyectos de IoT[2]. En la Figura 4 puede observarse su distribución de pines.

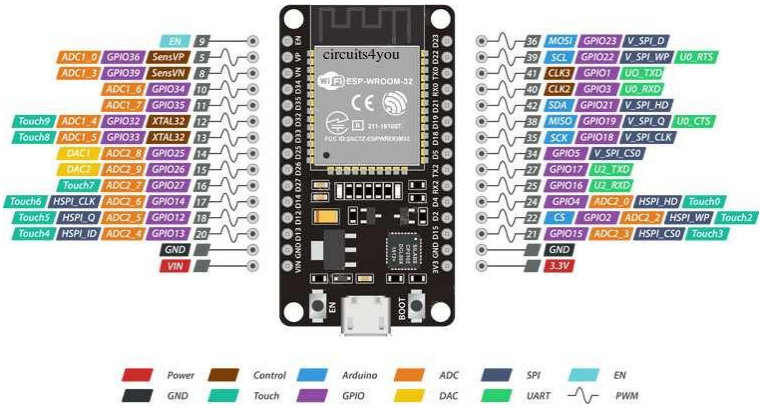


Figura 4. Pinout de ESP32 (Circuits4you, 2018)

[2] Internet de las cosas

Computadoras de placa única

Desarrolladas por la Fundación Raspberry Pi, las computadoras de placa única Raspberry Pi tienen el objetivo de poner en manos de las personas de todo el mundo el poder de la informática y la creación digital. Si bien el modelo original buscaba la promoción de la enseñanza de informática en las escuelas, este acabó siendo más popular de lo que se esperaba, hasta incluso vendiéndose fuera del mercado objetivo para usos como robótica.

Debido a su potencia, es posible utilizarla para un sinfín de propósitos. Desde estaciones de entretenimiento como hasta plataformas de aprendizaje tanto de actividades educativas básicas como desarrollo de proyectos robóticos. Para esto, se acompaña de diversos sistemas operativos (principalmente distribuciones de GNU/Linux) elegido por el usuario de acuerdo a sus necesidades. En la sección de Fundamentos de programación se explica de manera superficial sobre el uso de Ubuntu y la integración con ROS.

Electrónica de potencia

Desafortunadamente, las placas de desarrollo, procesadores o computadoras de placa única ofertan señales de control con voltaje y/o corriente baja, por lo que es necesario el uso de dispositivos capaces de tomar esas señales y amplificarlas de acuerdo al actuador a controlar. Pueden utilizarse desde transistores, TRIACs, relevadores, contactores, drivers, etc. La elección dependerá del diseñador del sistema, siendo la prioridad de separa la etapa de control con la etapa de potencia, por lo que generalmente se utilizan optoacopladores para realizar dicha división en caso de cortos o sobrecargas.

Drivers

En lo particular, el uso de drivers ha aumentado dada su simplicidad en uso y las grandes ventajas dentro de su construcción, ya que bien otorga la capacidad de aumentar el voltaje y corriente del sistema, como también la capacidad de “dividir” el sistema de control con el de potencia gracias a los optoacopladores integrados.

Dichos drivers permiten controlador motores con menor número de señales, permitiendo aumentar el número de actuadores posibles a controlador por la unidad de control. Para el caso de drivers de motores a pasos, pueden encontrarse infinidad de modelos; los más comunes son aquellos basados en la arquitectura del driver A4988 presentado en la Figura 5.

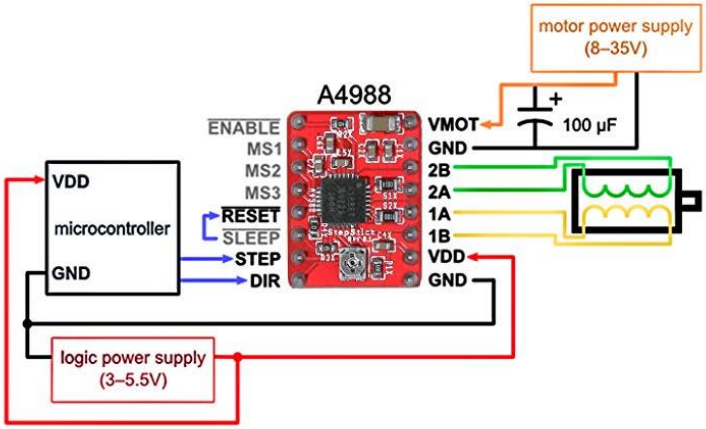


Figura 5. Driver A4988 para motor a pasos

Como puede notarse, la arquitectura del driver permite el control del motor con sólo dos pines, STEP y DIR, encargados de generar el pulso y dirección de giro respectivamente. Además, cuenta con modalidad de microstep, permitiendo mejorar la resolución de pasos del mecanismo o sistema. Respecto a la parte de potencia, el motor se conecta en los cuatro pines en función a sus dos bobinas, lo que facilita el uso de voltajes máximos de 35V siempre y cuando el motor lo soporte. Por lo que se recomienda revisar la hoja de datos del fabricante.

Fundamentos de programación

Ubuntu

Es una distribución de código abierto basada en Debian, otro sistema operativo, cuyo punto común es Linux, la madre de todos los softwares gratuitos que actúan como sistemas operativos en las computadoras de medio mundo.

La empresa responsable de su creación y de su mantenimiento, actualizaciones y desarrollo de nuevas versiones incluidas, es Canonical, fundada por el empresario sudafricano Mark Shuttleworth. Actualmente, se cuenta con la versión 20.04 lanzada en abril de 2020.

Si se visita la página oficial, notaremos que Ubuntu no requiere de un hardware muy potente para su uso, por lo que comúnmente se instala en computadoras de placa reducida como la Raspberry Pi. Permitiendo el uso de gran cantidad de programas y paquetes para diferentes rubros. Dentro de ellos podemos encontrar a ROS[3].

ROS (Robot Operating System)

ROS es una colección de frameworks para el desarrollo de software para robots de alto nivel mediante la interacción de redes de nodos mediante segmentación de sistemas. Su objetivo principal es ofertar herramientas para el desarrollo sustancial en el área de robótica mediante el uso de paquetes de código abierto para la implementación en robots sin importar el fabricante (Open Robotics, 2020).

ROS se desarrolló originariamente en 2007 bajo el nombre de *switchyard* por el *Laboratorio de Inteligencia Artificial de Stanford* para dar soporte al proyecto del Robot con *Inteligencia Artificial de Stanford (STAIR2).* Desde 2008, el desarrollo continuó principalmente en *Willow Garage*, un instituto de investigación robótico con más de veinte instituciones colaborando en un modelo de desarrollo federado. Actualmente, el proyecto se encuentra soportado por *Open Robotics*.

[3] Robot Operating System

Entorno

ROS provee herramientas muy poderosas para la programación y control de robots, permitiendo desarrollar complejas redes de nodos y comprimirlos en paquetes para realizar diversas actividades en un proyecto. Siendo así, una excelente oportunidad para su implementación y estudio en universidades con ramas en robótica y automatización.

Aunque ROS no sea directamente un sistema operativo, otorga todas las ventajas de uno convencional:

- Abstracción de hardware
- Control de dispositivos de bajo nivel
- Comunicación entre procesos mediante mensajes
- Mantenimiento de paquetes

Su desarrollo está centrado en sistemas GNU/LINUX, principalmente para las distribuciones más reconocidas:

- Ubuntu
- Debian
- Fedora
- Raspbian

Sin embargo, su popularidad ha permitido su adaptación experimental (algunas distribuciones de ROS) en sistemas operativos más comerciales como MAC OS X y Microsoft Windows.

Arquitectura

Está basado en una arquitectura de grafos con los siguientes elementos:

- *Nodos* – Scripts para el procesamiento de datos para su emisión (nodo publicador) o solicitud (nodo suscriptor).
- *Temas* – Clasificación de los mensajes.
- *Mensajes* – Datos para la comunicación unidireccional entre nodos.
- *Servicios* – Datos para la comunicación bidireccional entre nodos.
- *Paquetes* – Centralización de nodos para una tarea o funcionalidad.
- *Stacks* – Integración de paquetes para la ejecución de proyectos muy complejos.
- *Espacio de trabajo* – Administración de paquetes durante la ejecución de un proyecto con el fin de mantener un control total.

Aplicaciones

Dada su versatilidad es sencillo implementar diversas áreas en un mismo robot, permitiendo el diseño de sistemas complejos, pero aprovechando las bondades de segmentación para el desarrollo particular de cada funcionalidad y mantener un flujo de trabajo organizado. Dentro de las áreas más implementadas, se encuentra:

- Visión artificial
- Inteligencia artificial
- Simulación y visualización 3D en tiempo real
- Geolocalización
- Control de actuadores
- Planificación de movimiento
- Reconocimiento de voz

Hardware

Como todo desarrollo mecatrónico, requiere diversas etapas de diseño, fabricación, ensamble y pruebas necesarias para la validación de prototipos. A continuación, se presentan cada una de las etapas necesarias para la construcción del robot, así como de la tarjeta de control requerida.

Diseño mecánico

Con base al proyecto desarrollado por Skyentific (Skyentific, 2019), se retomó el diseño mecánico desarrollado en Fusion360, para esto se realizó la conversión a SolidWorks para permitir ciertas modificaciones necesarias. Dichas modificaciones están enfocadas al tipo de correa utilizada, así como el espacio para los rodamientos; puesto que muchas de las piezas no son comerciales en México (recordar que el diseño original está pensado con piezas de Suiza). En la Figura 6 puede observarse el diseño final, junto con las adecuaciones necesarias, además se agregó una base para el efector final y una para la cámara para el posterior uso de visión artificial.

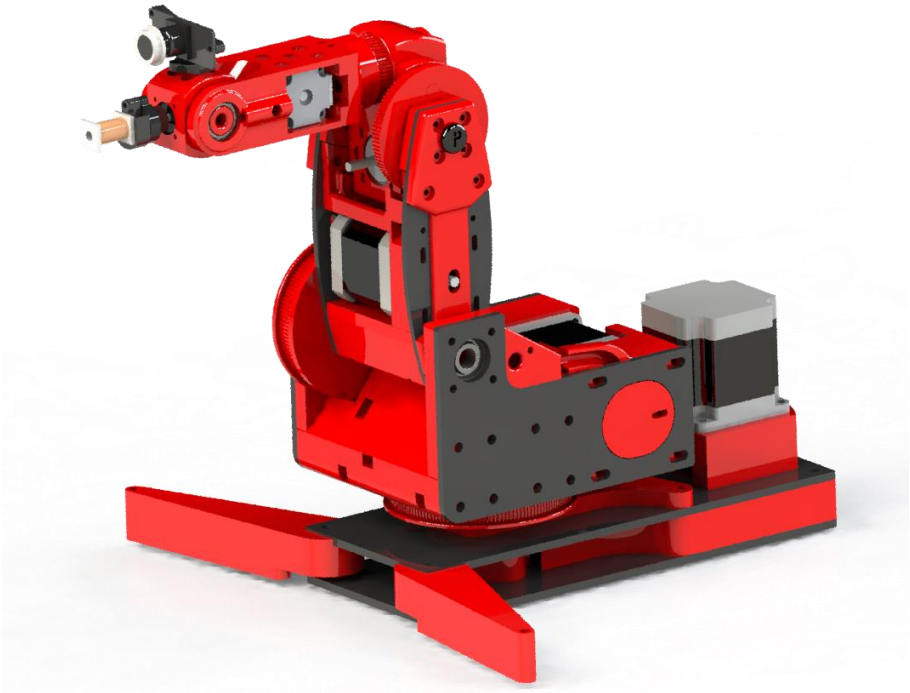


Figura 6. Diseño mecánico del robot

Diseño electrónico

Respecto al diseño electrónico, el principal enfoque fue el obtener una placa de control y potencia con la intención de aumentar su portabilidad. Se consideró el uso de una ESP32 como procesador de control, mientras que los drivers A4988 y DRV8825 cumplen con la etapa de potencia. En total se utilizaron seis drivers para cada uno de los motores de cada articulación. En la Figura 7 se presenta el diagrama electrónico utilizado para su posterior fabricación y ensamble.

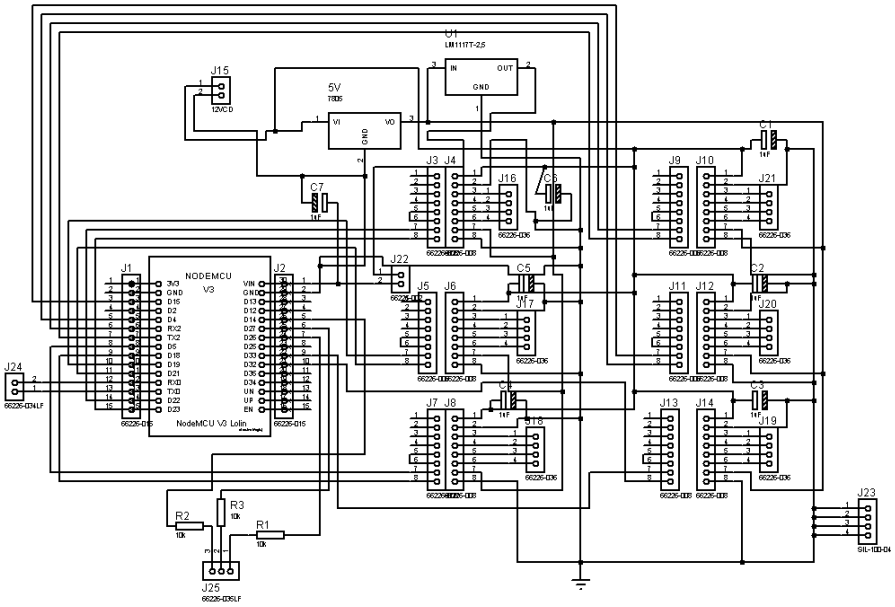


Figura 7. Diagrama de electrónico

Es importante recalcar que el diseño del circuito electrónico se basó en centralizar las señales a la ESP32 para posteriormente activar cada uno de los drivers de acuerdo a la programación. Igual se consideró el uso de un transistor BJT para la amplificación de la corriente para activar al efector final. También se han dejado dos pines de entrada, en caso se desee utilizar botones para el control del brazo, principalmente para la construcción de modos de operación.

Fabricación

De acuerdo a la etapa de fabricación, se enfocó principalmente a la impresión 3D de cada una de las piezas necesarias para el posterior ensamble del prototipo. Así como la compra de los elementos comerciales necesarios (motores, ejes y rodamientos). En la Figura 8 y Figura 9 se presentan algunas piezas en impresión.

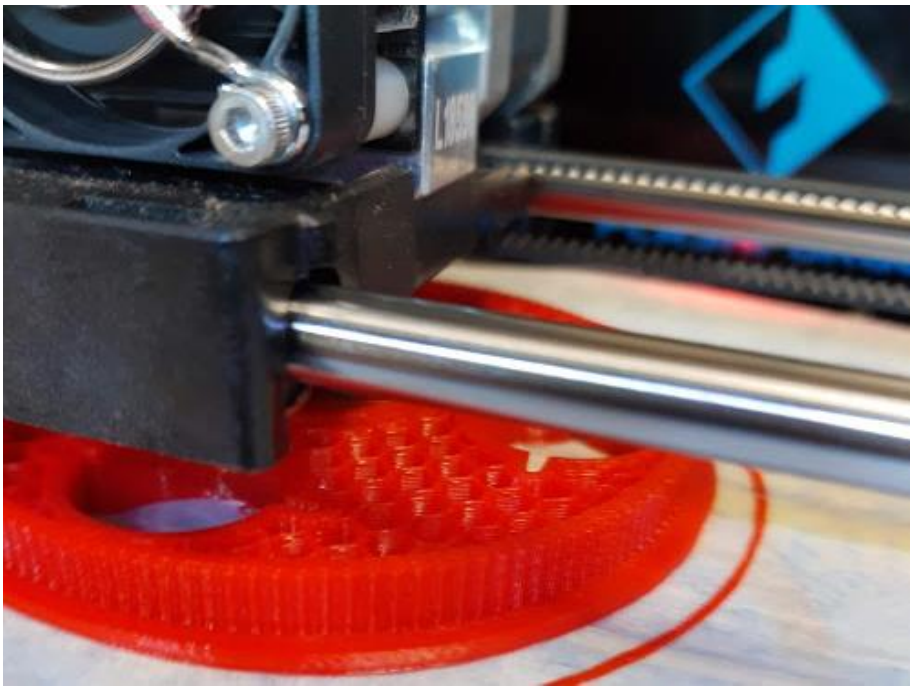


Figura 8. Impresión 3D de polea para articulación dos

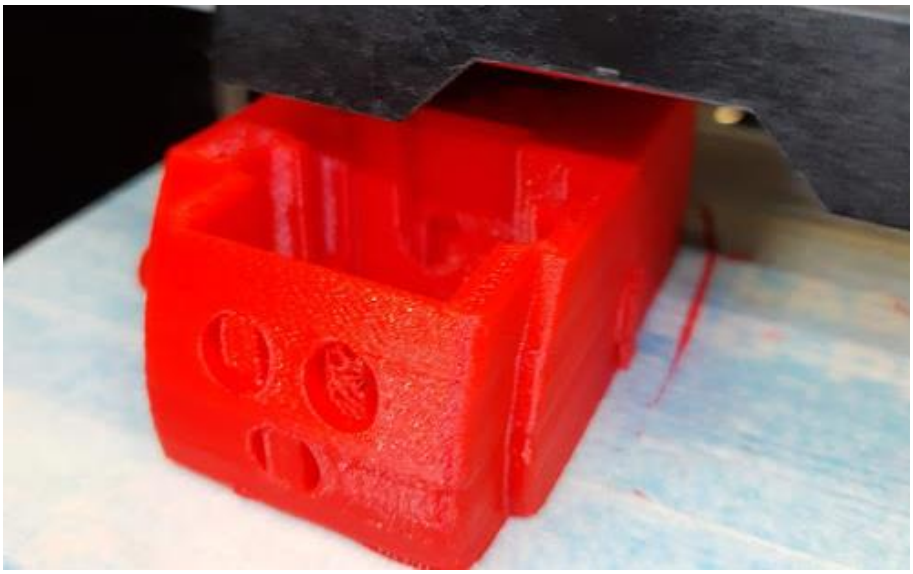


Figura 9. Impresión 3D de pieza para el tercer eslabón

Mientras que en la Figura 10 pueden observarse algunas de las piezas utilizadas para el ensamble.

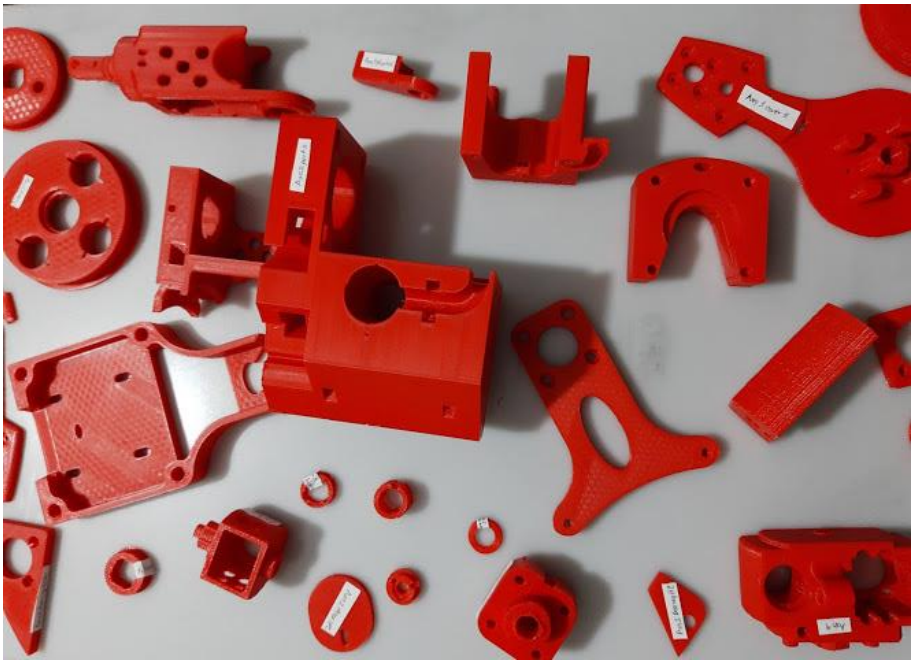


Figura 10. Piezas impresas

El filamento utilizado fue PLA debido a su costo y fácil acceso, así como el proceso de impresión es mucho más sencillo que otros materiales. Además, el prototipo no está diseñado para soportar grandes esfuerzos, por lo que el material cumplirá fácilmente con las exigencias mecánicas.

Ensamble

Posteriormente, en la etapa de ensamble se utilizaron todas las piezas impresas y los elementos adquiridos para la construcción del brazo robótico de seis grados de libertad. Por lo que se comenzó a armar los eslabones por separado para después conectarlos entre ellos. Como medio de transmisión mecánica, se consideró el uso de correas con perfil GT2. Asimismo, se cuidó mucho el paso del arnés que transmite las señales eléctricas para activar a los motores a pasos, principalmente para evitar daños ante el movimiento del robot y no limitar tanto su movimiento.

En la Figura 11 se presenta al primer eslabón en el proceso de ensamble, mientras que en la Figura 12 se muestra el resto de eslabones para ensamblar.

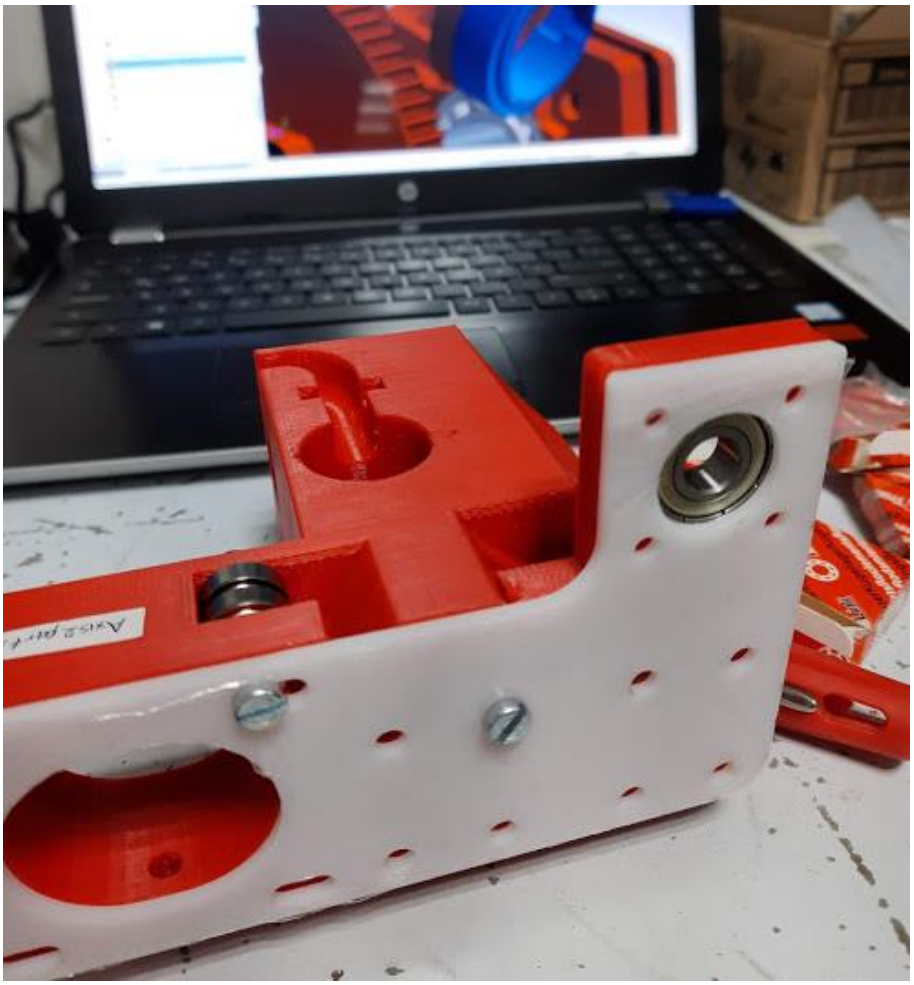


Figura 11. Ensamble del primer eslabón

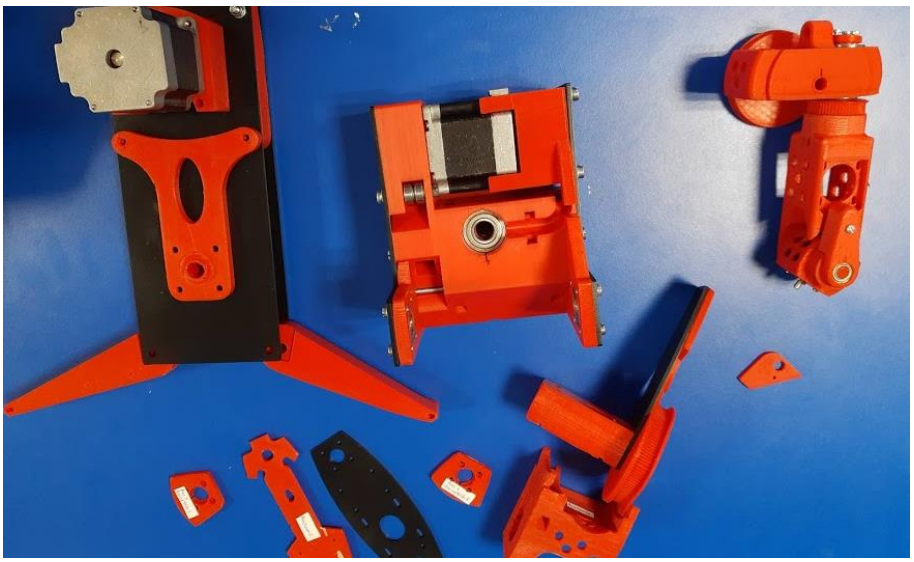


Figura 12. Eslabones separados

De esta manera, en la Figura 13 se observa la centralización de los cables necesarios para conducir las señales de activación a cada uno de los motores. Este arreglo permitió el uso de un conector DB25 para facilitar la conexión del robot con la placa de control y potencia fabricada.

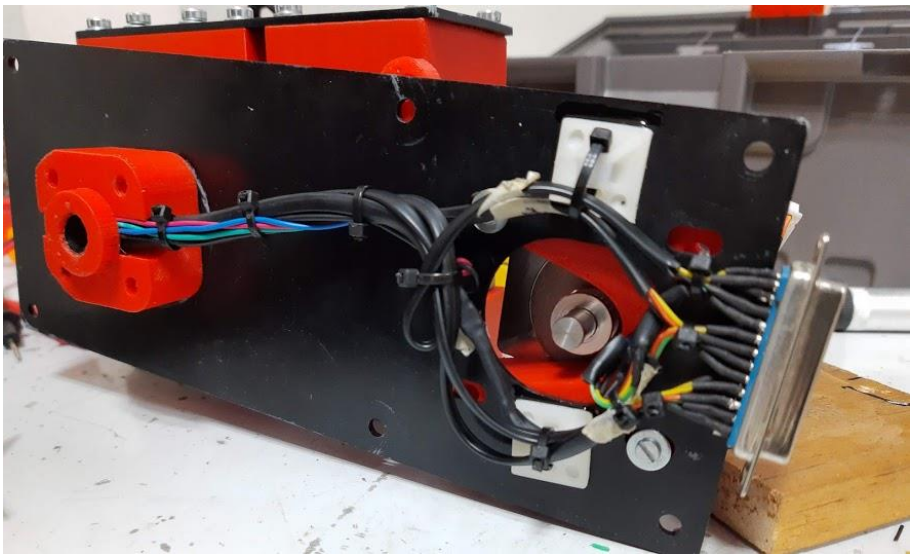


Figura 13. Centralización de arnés del robot

Concluyendo con el ensamble principal, se obtuvo un prototipo robótico de seis grados de libertad presente en la Figura 14

Figura 14. Unidad mecánica

Posteriormente, se instaló el arnés necesario para el control del efector final, así como de la cámara para la adquisición de imágenes (Figura 15).

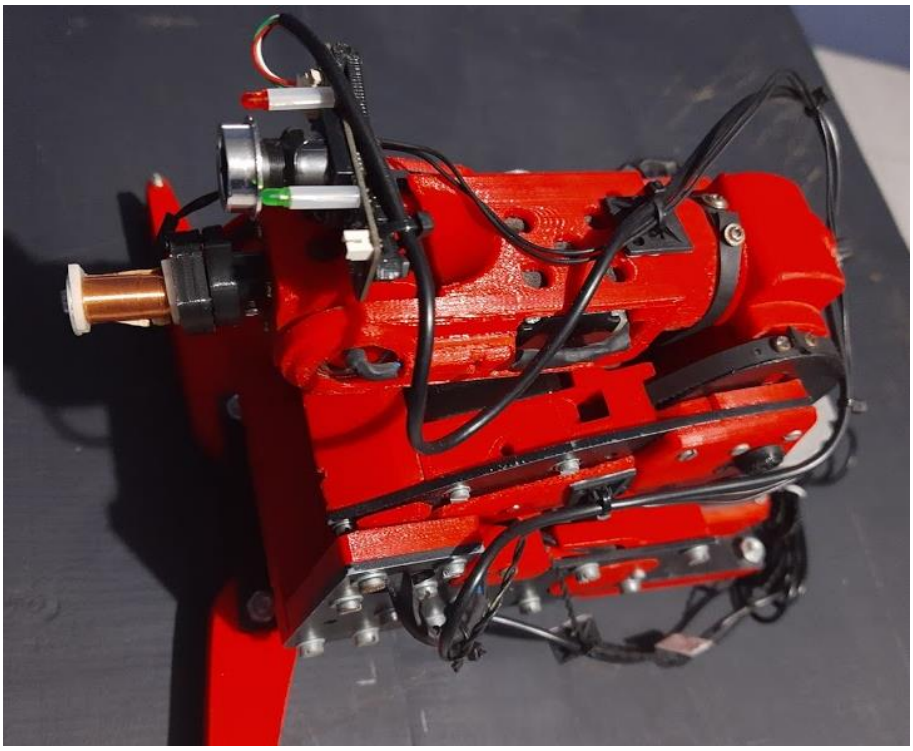


Figura 15. Instalación de arnés de efector final y cámara

Respecto a la parte de electrónica, se fabricó y soldó cada componente de acuerdo a la Figura 16. De esta manera, se permitió un acceso sencillo a cada elemento en caso de mantenimiento. Al igual de proveer buen flujo de aire para el sistema de enfriamiento otorgado por un ventilador en la parte superior.

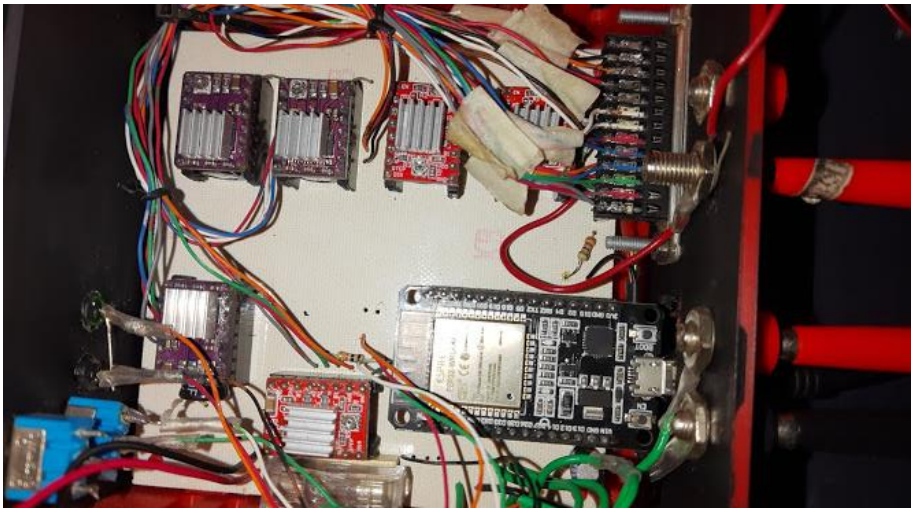


Figura 16. Placa de control y potencia

Software

Con el fin de tener bases firmes para el desarrollo del proyecto, se consideró por tres áreas principales de investigación: Git, ROS y OpenCV. Dentro de Git, el objetivo fue comprender e implementarlo dentro de la construcción del proyecto, principalmente por las condiciones de emergencia sanitaria para trabajo remoto. En la Figura 17 puede observarse los tres temas centrales.

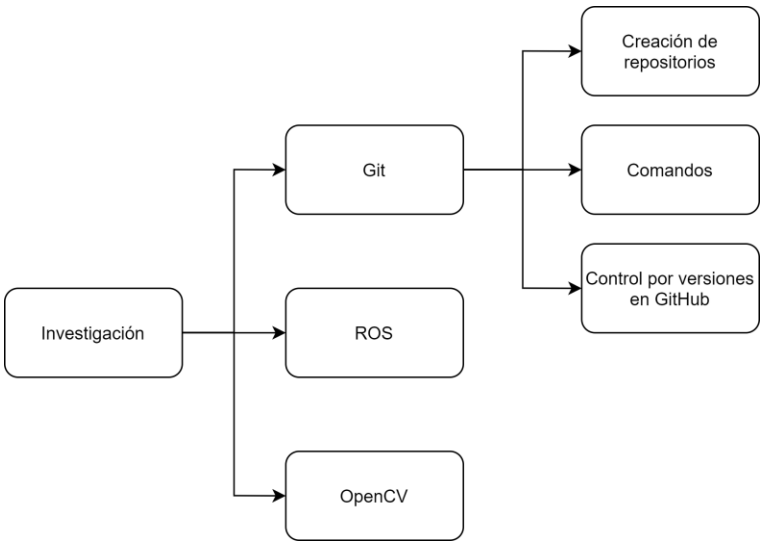


Figura 17. Temas de investigación en Git

Con base a las diversas investigaciones, se generó el repositorio de la Figura 18 en GitHub con el fin de recopilar todos los archivos del proyecto.

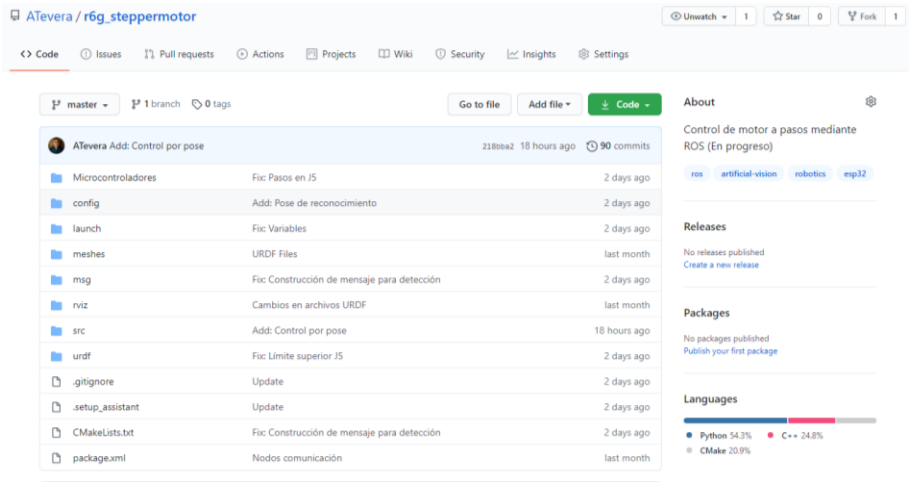


Figura 18. Repositorio en GitHub

Además, permite tener un control de versiones sumamente preciso; por lo que el desarrollo del proyecto.

Implementación de ROS

En paralelo, las investigaciones, las investigaciones también se enfocaron en ROS; principalmente por ser el primer contacto con este tipo de herramientas. En la Figura 19 y Figura 20 pueden observarse las tres secciones de la investigación.

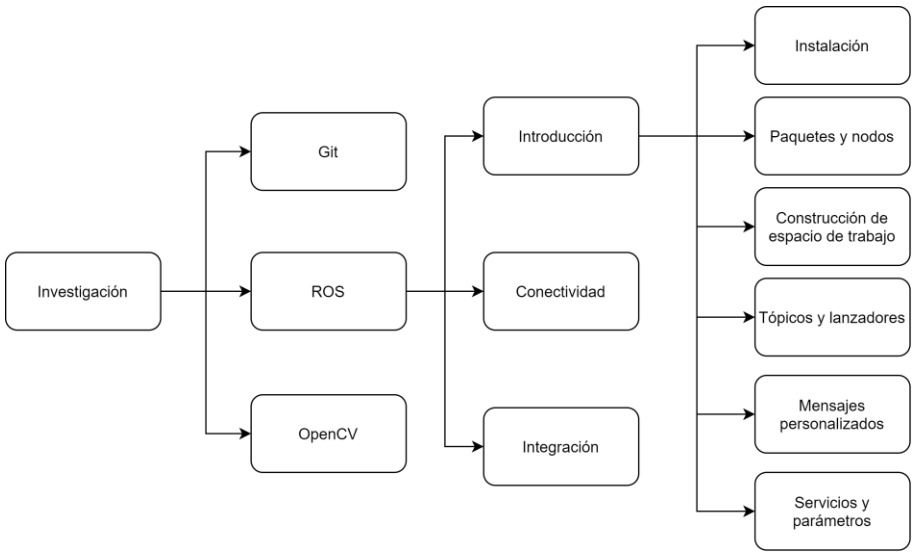


Figura 19. Introducción a ROS

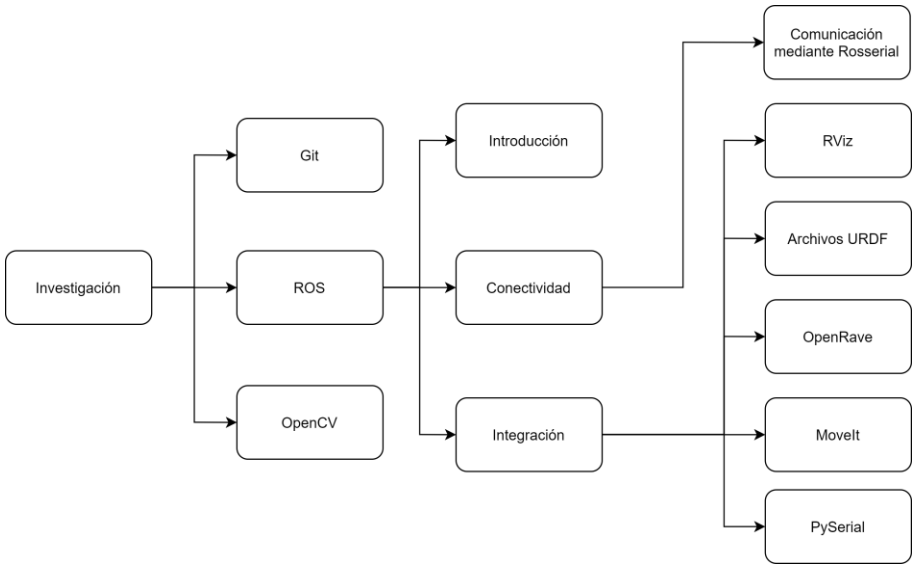


Figura 20. Conectividad e integración de ROS

La investigación de cada uno de los temas permitió tener un concepto mucho más claro sobre el uso de ROS en proyectos de robótica. Por lo que se procedió a la instalación de todas las herramientas necesarias para el desarrollo del proyecto.

Partiendo del hecho de estar utilizando Ubuntu 20.04 LTS en la Raspberry Pi 4 Tipo B, se eligió la versión Noetic de ROS (principalmente por ser la más nueva y por tener soporte hasta mayo de 2025). Para su instalación se aplicaron los siguientes comandos en la terminal de Ubuntu:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654

$ sudo apt update

$ sudo apt install ros-noetic-desktop-full

$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
```

Posteriormente se construyó el espacio de trabajo para el desarrollo de todos los paquetes y dependencias necesarias para el uso de ROS en el proyecto. Para esto, bastó con ejecutar los siguientes comandos para crear la carpeta del espacio de trabajo, así como la carpeta src en home. Ya el comando *catkin_make* permite construir y compilar el espacio de trabajo.

```
$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/

$ catkin_make
```

Ahora queda generar el paquete donde se desarrollarán cada uno de los nodos encargados de controlar al brazo robot. Como parte de las reglas del compilador de ROS, es necesario ejecutar el comando dentro de la carpeta src del espacio de trabajo.

```
$ catkin_create_pkg r6g_steppermotor std_msgs rospy roscpp
```

Analizando la estructura del comando, r6g_steppermotor es el nombre del paquete del robot, mientras que los siguientes tres elementos son dependencias útiles para el uso de mensajes, programación en Python y programación en C++ respectivamente. Como último paso, se debe compilar todo el espacio de trabajo y declarar el paquete dentro del entorno de ROS:

```
$ cd ~/catkin_ws

$ catkin_make

$ . ~/catkin_ws/devel/setup.bash

$ echo "source . ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

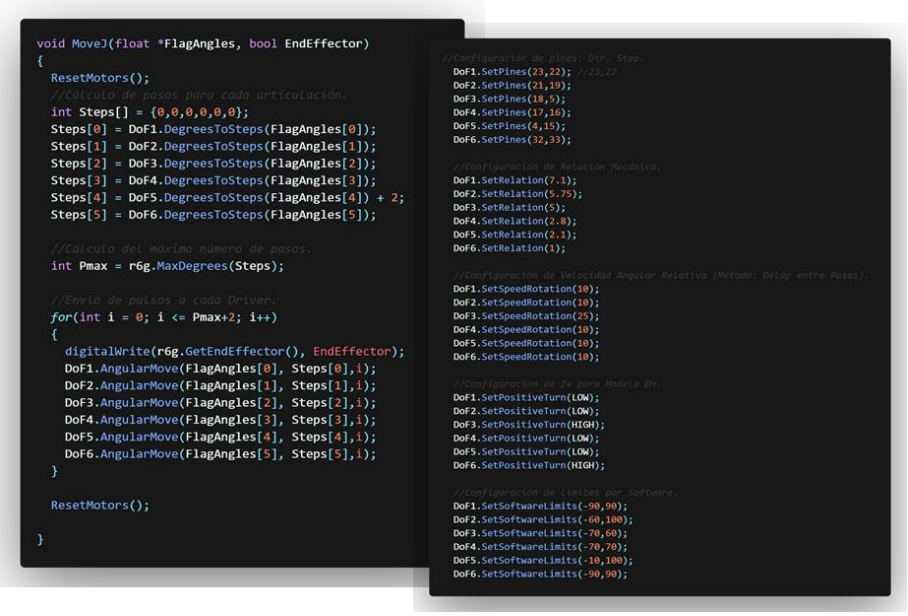
Teniendo la plataforma de desarrollo, se construyeron diversos nodos encargados de procesar la información. Sin embargo, el procesamiento de la información era un poco lenta por parte de la ESP32, por lo que se optó por desarrollar una librería exclusiva para el control de los seis motores a paso. En la Figura 21 puede observarse algunas secciones del código.

```
#include <Arduino.h>
#include "r6gController.h"

Joint::Joint() {}

int Joint::DegreesToSteps(float New)
{
  int pulses = 0;
  pulses = (abs(New - _Old)*_Relation)/1.8;
  return pulses;
}

bool Joint::CheckSafetyMove(float J)
{
  return J >= _Min && J <= _Max;
}

bool Joint::AngularMove(float New, int steps, int count
{
  if (CheckSafetyMove(New))
  {
    if (New > _Old) digitalWrite(_PDir, _Positive);
    else digitalWrite(_PDir, not _Positive);

    if (count <= steps)
    {
      digitalWrite(_PStep, HIGH);
      delay(_Speed);
      digitalWrite(_PStep, LOW);
    }
    else _Old = New;

    return true;
  }
  return false;
}
```

```
void Joint::Shutdown()
{
  pinMode(_PDir,OUTPUT);
  pinMode(_PStep,OUTPUT);
  digitalWrite(_PStep,LOW);
  digitalWrite(_PDir, LOW);
}

Robot::Robot() {}

int Robot::MaxDegrees(int Pulsos[6])
{
  int max1, max2, max3, max4;
  max1 = max(Pulsos[0],Pulsos[1]);
  max2 = max(Pulsos[2],Pulsos[3]);
  max3 = max(Pulsos[4],Pulsos[5]);
  max4 = max(max1,max2);
  return max(max3,max4);
}

void Robot::InitEndEffector(int EndEffector)
{
  _EndEffector = EndEffector;
  pinMode(_EndEffector, OUTPUT);
  digitalWrite(_EndEffector, LOW);
}
```

Figura 21. Secciones de librería para el control de motores a paso (Tevera-Ruiz, 2020)

Esta acción, permitió reducir las líneas de código en un 50% por lo que la tarjeta de procesamiento de la placa de control y potencia permitió trabajar de mejor forma. Además, se preparó para la comunicación serial con la Raspberry mediante mensajes con estructura JSON[4], permitiendo tener todo organizado. En la Figura 22 puede observarse las ventajas de haber construido la librería, permitiendo tener una codificación mucho más limpia.

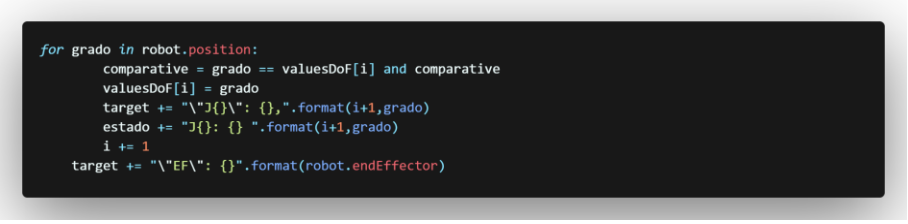


```
void MoveJ(float *FlagAngles, bool EndEffector)
{
  ResetMotors();
  //Cálculo de pasos para cada articulación.
  int Steps[] = {0,0,0,0,0,0};
  Steps[0] = DoF1.DegreesToSteps(FlagAngles[0]);
  Steps[1] = DoF2.DegreesToSteps(FlagAngles[1]);
  Steps[2] = DoF3.DegreesToSteps(FlagAngles[2]);
  Steps[3] = DoF4.DegreesToSteps(FlagAngles[3]);
  Steps[4] = DoF5.DegreesToSteps(FlagAngles[4]) + 2;
  Steps[5] = DoF6.DegreesToSteps(FlagAngles[5]);

  //Cálculo del máximo número de pasos.
  int Pmax = r6g.MaxDegrees(Steps);

  //Envío de pulsos a cada Driver.
  for(int i = 0; i <= Pmax+2; i++)
  {
    digitalWrite(r6g.GetEndEffector(), EndEffector);
    DoF1.AngularMove(FlagAngles[0], Steps[0],i);
    DoF2.AngularMove(FlagAngles[1], Steps[1],i);
    DoF3.AngularMove(FlagAngles[2], Steps[2],i);
    DoF4.AngularMove(FlagAngles[3], Steps[3],i);
    DoF5.AngularMove(FlagAngles[4], Steps[4],i);
    DoF6.AngularMove(FlagAngles[5], Steps[5],i);
  }

  ResetMotors();

}
```

```
//Configuración de pines: Dir, Step.
DoF1.SetPines(23,22); //23,22
DoF2.SetPines(21,19);
DoF3.SetPines(18,5);
DoF4.SetPines(17,16);
DoF5.SetPines(4,15);
DoF6.SetPines(32,33);

//Configuración de Relación Mecánica.
DoF1.SetRelation(7.1);
DoF2.SetRelation(5.75);
DoF3.SetRelation(5);
DoF4.SetRelation(2.8);
DoF5.SetRelation(2.1);
DoF6.SetRelation(1);

//Configuración de Velocidad Angular Relativa (Método: Delay entre Pasos).
DoF1.SetSpeedRotation(10);
DoF2.SetSpeedRotation(10);
DoF3.SetSpeedRotation(25);
DoF4.SetSpeedRotation(10);
DoF5.SetSpeedRotation(10);
DoF6.SetSpeedRotation(10);

//Configuración de Sentido Modelo DH.
DoF1.SetPositiveTurn(LOW);
DoF2.SetPositiveTurn(LOW);
DoF3.SetPositiveTurn(HIGH);
DoF4.SetPositiveTurn(LOW);
DoF5.SetPositiveTurn(LOW);
DoF6.SetPositiveTurn(HIGH);

//Configuración de Límites por Software.
DoF1.SetSoftwareLimits(-90,90);
DoF2.SetSoftwareLimits(-60,100);
DoF3.SetSoftwareLimits(-70,60);
DoF4.SetSoftwareLimits(-70,70);
DoF5.SetSoftwareLimits(-10,100);
DoF6.SetSoftwareLimits(-90,90);
```

Figura 22. Funciones de envío de señales

Respecto a la estructura de comunicación entre la Raspberry y la ESP32, se optó por diseñar un nodo en ROS dedicado a la construcción de mensajes con estructura JSON (Figura 23) y su envío mediante PySerial (Liechti, 2015).



```
for grado in robot.position:
        comparative = grado == valuesDoF[i] and comparative
        valuesDoF[i] = grado
        target += "\"J{}\": {},".format(i+1,grado)
        estado += "J{}: {} ".format(i+1,grado)
        i += 1
    target += "\"EF\": {}".format(robot.endEffector)
```

Figura 23. Construcción de mensaje JSON

[4] JavaScript Object Notation

Por el lado del procesador, la ESP32 realiza el proceso de deserealización mediante una librería desarrollada por *Benoît Blanchon.* En la Figura 24 se presenta el proceso utilizado para el procesamiento del mensaje y el envío de información de los ángulos a cada grado de libertad, así como el estado del efector final mediante la función *MoveJ*.

```
//Obtención de valores del mesaje tipo JSON.
deserializeJson(doc, ROSmessage);
JsonObject Angles = doc.as<JsonObject>();
//Inicializa y asigna los valores al array de envio para movimiento.
float FlagAngles[6] = {};
for(int i = 0; i < 6; i++)
{
  String index = "J" + String(i+1);
  FlagAngles[i] = Angles[index].as<float>();
}

//Actualiza el estado del efector final.
bool State_EF = Angles["EF"].as<int>() == 1;
//Movimiento del robot.
MoveJ(FlagAngles, State_EF);
```

Figura 24. Deserealización de mensaje JSON

En paralelo, se generó el resto de nodos para el control del robot. Parte medular fue la construcción de un archivo URDF[5], encargado de describir al robot en un archivo XML; de esta manera la traducción a un modelo cinemático integrado a ROS, como el propuesto por MoveIt es más sencillo (PickNik Robotics, 2020). En la Figura 25 puede observarse algunas de las configuraciones utilizadas para exportar el modelo 3D de SolidWorks a URDF mediante una extensión (ROS, 2020).

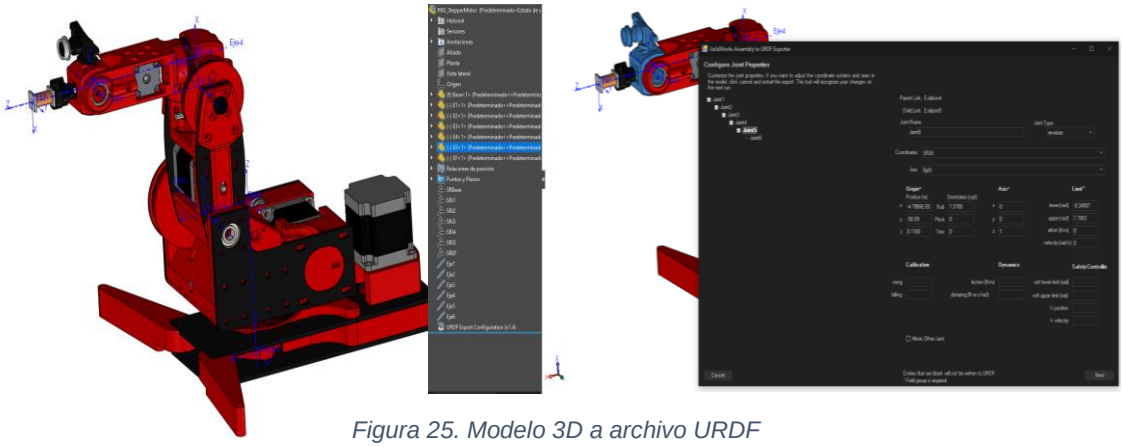


Figura 25. Modelo 3D a archivo URDF

[5] Unified Robot Description Format

Es importante recalcar que este método requiere que el ensamble esté correctamente definido, principalmente las relaciones mecánicas. Además, debe verificarse que la extensión está posicionando correctamente los sistemas de referencia de cada grado de libertad.

Posteriormente, con apoyo de MoveIt se estableció toda la configuración cinemática necesaria. Para esto, se siguieron los siguientes pasos:

```
$ sudo apt install ros-noetic-moveit

$ sudo apt install ros-noetic-franka-description

$ roslaunch moveit_setup_assistant setup_assistant.launch
```

En consecuencia, se presentará una interfaz gráfica para definir cada uno de los aspectos del robot con base al archivo URDF. Para este caso, sólo fue necesario establecer la cadena cinemática y definir poses al robot. Al finalizar, se obtuvo un simulador con base al RViz (Figura 26), permitiendo tener planificadores de trayectorias, detección de colisiones por software y solucionadores de cinemática directa e inversa.

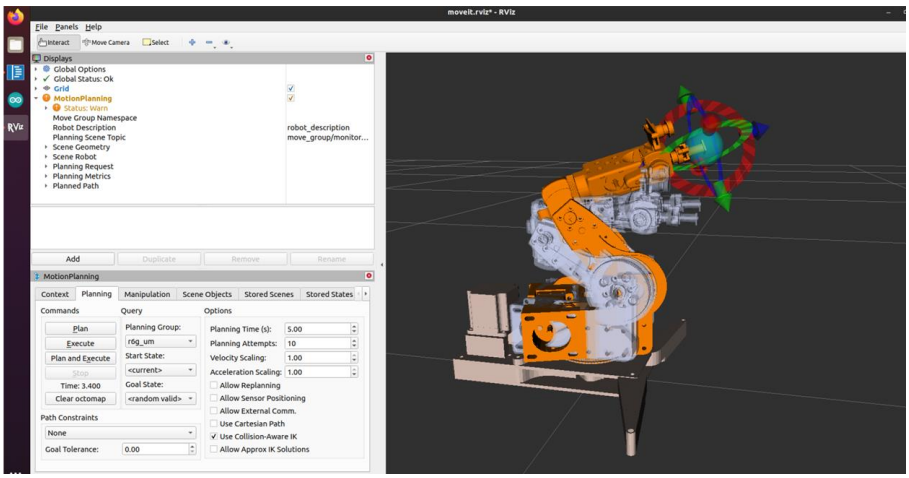


Figura 26. Simulación del robot en RViz

Obteniendo los beneficios de MoveGroup (Sucan, 2017) dentro del entorno de ROS, se construyeron dos nodos para los cálculos de cinemática directa (Figura 27) y cinemática inversa (Figura 28). Como puede observarse, la codificación es muy sencilla y práctica.

```
moveit_commander.roscpp_initialize(sys.argv)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
move_group = moveit_commander.MoveGroupCommander("r6g_um")
joints = move_group.get_current_joint_values()
print(joints)
rate = rospy.Rate(10)

while not rospy.is_shutdown():
    for i in range(0,len(joints)):
        joints[i] = math.radians(float(input("J{}: ".format(i+1))))
    move_group.go(joints, wait = True)
    print(move_group.get_current_pose())
    move_group.stop()
```

Figura 27. Cinemática directa con MoveIt

```
moveit_commander.roscpp_initialize(sys.argv)
robot = moveit_commander.RobotCommander()
scene = moveit_commander.PlanningSceneInterface()
r6g = moveit_commander.MoveGroupCommander("r6g_um")
r6g.set_planner_id("BFMT")
r6g.set_planning_time(3)
r6g.allow_replanning(True)
r6g.set_pose_reference_frame('Base')
r6g.set_goal_position_tolerance(0.01)
r6g.set_goal_orientation_tolerance(0.01)
r6g.set_num_planning_attempts(3)
pose_target = geometry_msgs.msg.Pose()
coordinates = [0]*3
coordinates[0] = target.posicion[0]/100
coordinates[1] = target.posicion[1]/100
coordinates[2] = target.posicion[2]/100
pose_target.orientation.x = target.orientacion[0]
pose_target.orientation.y = target.orientacion[1]
pose_target.orientation.z = target.orientacion[2]
pose_target.orientation.w = target.orientacion[3]
r6g.set_start_state_to_current_state()
r6g.set_position_target(coordinates)
print(type(target))
r6g.plan()
rospy.sleep(5)
r6g.go(wait = True)
joints = r6g.get_current_joint_values()
print(joints)
r6g.stop()
r6g.clear_pose_targets()
```

Figura 28. Cinemática inversa con MoveIt

Ya con el simulador, se comenzaron a realizar pruebas de comunicación entre la Raspberry Pi con la ESP32. Desafortunadamente, la latencia era alta y obligaba al robot a tener ciertos movimientos bruscos. Esto era por el hecho en como ROS procesa la información; ya que, al tener procesos en paralelo, había gran número de mensajes saturando el puerto serial de la ESP32. La solución fue construir una base de datos dinámica exclusiva para gestionar los valores de cada grado de libertad y del estado del robot. Este hecho permitió centralizar el flujo de información a dicha base de datos, por lo que se optó a construir una librería en Python encargada de procesar todos esos datos (Figura 29). Ya en la Figura 30 puede observarse un ejemplo de escritura de datos por parte de la librería en un archivo CSV (Python, 2020).

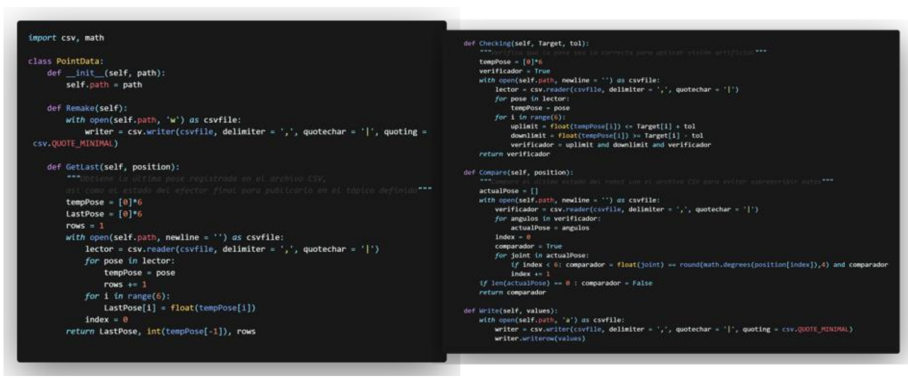


Figura 29. Librería para la gestión de la base de datos

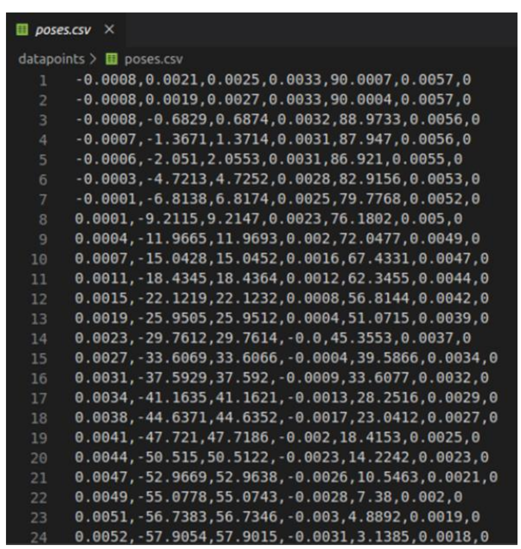


Figura 30. Ejemplo de base de datos dinámica

En conjunto, y con apoyo de la Figura 31, se desarrollaron un total de siete nodos principales para el control del robot. Como puede observarse, el uso de ROS permite tener un sistema sumamente flexible a la adición de nuevas formas de control, por lo que fácilmente se puede implementar nuevas técnicas y aprovechar las ventajas de procesos en paralelo.

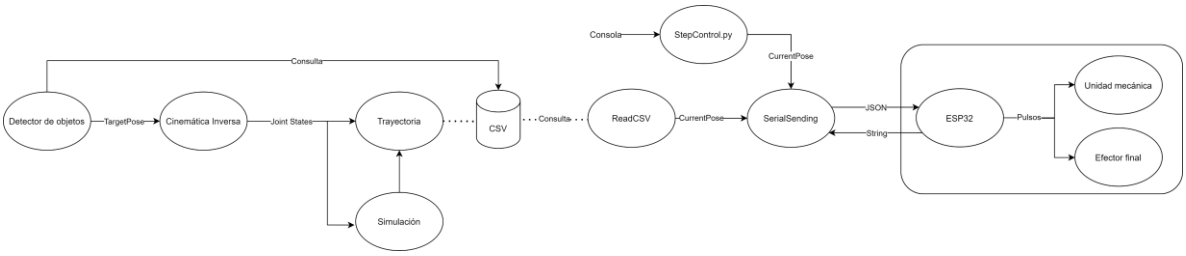


Figura 31. Arquitectura de red de nodos

Reconocimiento de piezas en espacio de trabajo

Respecto al desarrollo de visión artificial, el enfoque fue detectar una pieza en función de su color e indicar sus coordenadas en el espacio de trabajo del robot. Para esto, en la Figura 32 se presenta una sección de código encargado de declarar los trackbars para calibrar la sensibilidad de la cámara en tiempo real, con la intensión de adaptarse a las condiciones de luz. Esta técnica evita tener que modificar el código ante ajustes cotidianos.

```
def makeTrackBars():
    cv.namedWindow('Color a detectar minimos')
    cv.createTrackbar('Hue Min','Color a detectar minimos',0,360,nothing)
    cv.createTrackbar('Saturación Min','Color a detectar minimos',0,255,nothing)
    cv.createTrackbar('Valor Min','Color a detectar minimos',0,255,nothing)
    cv.createTrackbar('Area','Color a detectar minimos',0,10000,nothing)
    cv.createTrackbar('Blur','Color a detectar minimos',1,50,nothing)

    cv.namedWindow('Color a detectar maximos')
    cv.createTrackbar('Hue Max','Color a detectar maximos',0,360,nothing)
    cv.createTrackbar('Saturación Max','Color a detectar maximos',0,255,nothing)
    cv.createTrackbar('Valor Max','Color a detectar maximos',0,255,nothing)
```

Figura 32. Construcción de trackbars para calibración

Por otra parte, la ubicación de la pieza requiere tener un marco de referencia específico. Por lo que fue necesario aplicar una técnica de *transferencia de perspectiva* (Figura 33) con la intensión de poder aplicar una regla de tres posteriormente de acuerdo al ratio de la imagen adquirida.

```
def ordenar_puntos(puntos):
    n_puntos = np.concatenate([puntos[0], puntos[1], puntos[2], puntos[3]]).tolist()
    y_order = sorted(n_puntos, key = lambda n_puntos: n_puntos[1])
    x1_order = y_order[:2]
    x1_order = sorted(x1_order, key = lambda x1_order: x1_order[0])
    x2_order = y_order[2:4]
    x2_order = sorted(x2_order, key = lambda x2_order: x2_order[0])
    return [x1_order[0], x1_order[1], x2_order[0], x2_order[1]]

def roi(image, ancho, alto):
    imagen_alineada = None
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    _, th = cv.threshold(gray, 150, 255, cv.THRESH_BINARY)
    cnts = cv.findContours(th, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)[0]
    cnts = sorted(cnts, key = cv.contourArea, reverse = True)[:1]
```

Figura 33. Funciones para transferencia de perspectiva (Solano, 2020)

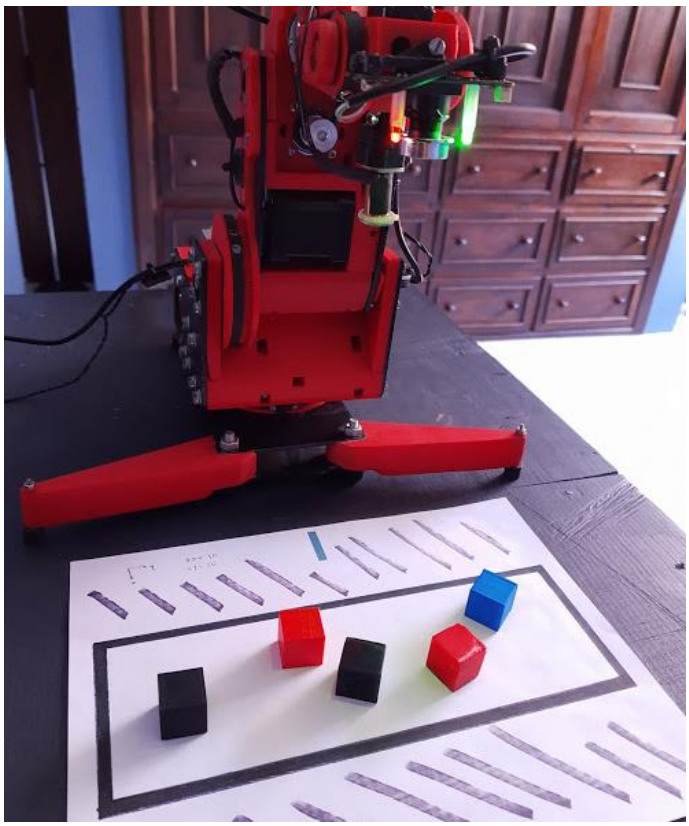


Figura 34. Espacio de trabajo para visión artificial

Dicha técnica permite enfocarse en cierta área rectangular con bordes (Figura 34). En este caso, al conocerse las dimensiones del área de trabajo, basta con hacer una regla de tres con el ratio de la imagen adquirida para conocer las coordenadas del objeto en función a la esquina superior izquierda. Sin embargo, al conocerse la posición de ese punto con respecto a las coordenadas del robot, fácilmente puede conocerse la ubicación de la pieza.

De esta manera, en la Figura 35 y Figura 36 puede observarse la interfaz gráfica de visión artificial junto con el simulador para la detección de piezas en color azul. Así como definir sus coordenadas en función del robot.

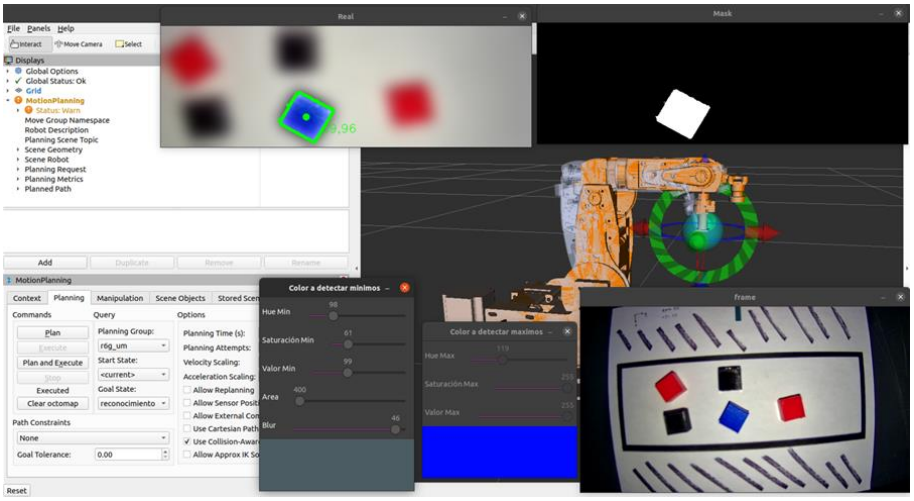


Figura 35. Detección de pieza color azul 1/2

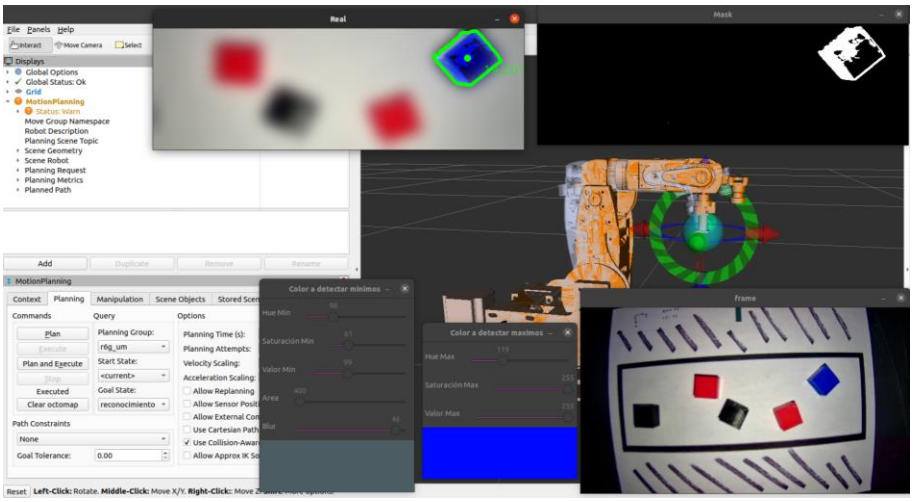


Figura 36. Detección de pieza color azul 2/2

Beneficios alcanzados

Como beneficios, se encuentra la obtención de un sistema modular dedicado al desarrollo de proyectos de robótica gracias a la implementación de ROS en el brazo robótico. Otorgando la oportunidad de utilizar las mismas herramientas en otros prototipos y potenciar sus capacidades a un bajo costo.

Además, el primer contacto con el uso de ROS permite su integración con el hardware de robots industriales comerciales mediante ROS Industrial; por lo que puede aprovecharse todo el diseño del fabricante y aplicar controles completamente diseñados por alumnos considerando periféricos ajenos a la marca sin conflicto con el fabricante.

Conclusiones

En lo particular, el proyecto cumplió en el desarrollo de la red de nodos necesaria para el control del brazo antropomórfico de seis grados de libertad y con esto, obtener una plataforma de aprendizaje para prácticas en el Laboratorio de Visión Artificial y Robótica Avanzada.

En caso requiera obtener mayores detalles del proyecto, puede visitar el repositorio (Tevera-Ruiz, r6g_steppermotor, 2020) en GitHub donde se encuentra cada uno de los nodos presentados y la estructura completa del proyecto.

Trabajos a futuro

Sería adecuado integrar sensores de posición a cada grado de libertad con la intensión de obtener un sistema de lazo cerrado y reducir el error entre cada uno de los movimientos. Puesto que, al estar a ciegas, el sistema de control no considera el error, llegando al punto de perder su conteo de ángulos completamente.

Por otro lado, la cinemática inversa funciona correctamente, pero existen varios problemas con singularidades por parte del robot, por lo que sería prudente definir correctamente el área de trabajo del robot para posicionar ahí la pieza con la que interactuará el robot.

Bibliografía

BCN3D. (2016). *BCN3D MOVEO*. Obtenido de https://www.bcn3d.com/es/bcn3d-moveo-un-brazo-robotico-de-codigo-abierto-impreso-en-3d/

Circuits4you. (31 de Diciembre de 2018). *ESP32 DevKit ESP32-WROOM GPIO Pinout*. Obtenido de Circuits4you: https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/

Kuhn, T. (28 de Octubre de 2017). *Raspberry Pi Robot Arm with simple Computer Vision*. Obtenido de Electron Dust: https://electrondust.com/2017/10/28/raspberry-pi-robot-arm-with-simple-computer-vision/

Kuhn, T. (3 de Noviembre de 2017). *ScrewPicker*. Obtenido de GitHub: https://github.com/T-Kuhn/ScrewPicker

Lecaillon, G. (11 de Julio de 2019). *Arduino Robot arm with Computer Vision*. Obtenido de Hackday.io: https://hackaday.io/project/168336-arduino-robot-arm-with-computer-vision#j-discussions-title

Liechti, C. (2015). *Documentation*. Obtenido de PySerial: https://pythonhosted.org/pyserial/

Open Robotics. (2020). *Noetic*. Obtenido de ROS: https://www.ros.org/

Open Source Computer Vision. (2020). *Modules*. Obtenido de OpenCV: https://docs.opencv.org/master/

PickNik Robotics. (2020). *Tutorials*. Obtenido de MoveIt: https://ros-planning.github.io/moveit_tutorials/

Python. (2020). *CSV*. Obtenido de Documentation: https://docs.python.org/3/library/csv.html

ROS. (22 de Octubre de 2020). *solidworks_urdf_exporter*. Obtenido de GitHub: https://github.com/ros/solidworks_urdf_exporter

Skyentific. (9 de Septiembre de 2019). *SmallRobotArm*. Obtenido de GitHub: https://github.com/SkyentificGit/SmallRobotArm

Solano, G. (19 de Octubre de 2020). *Transferencia de perspectiva*. Obtenido de Omes: https://omes-va.com/midiendo-la-distancia-entre-2-objetos-python-opencv/

Sucan, I. (24 de Julio de 2017). *MoveGroup Class Reference*. Obtenido de ROS: http://docs.ros.org/en/jade/api/moveit_ros_planning_interface/html/classmoveit_1_1planning__interface_1_1MoveGroup.html

Tevera-Ruiz, A. (1 de Diciembre de 2020). *r6g_steppermotor*. Obtenido de GitHub: https://github.com/ATevera/r6g_steppermotor

Tevera-Ruiz, A. (15 de Noviembre de 2020). *StepperMotorJoints_Library*. Obtenido de GitHub: https://github.com/ATevera/StepperMotorJoints_Library

Weisberg, J. (30 de Julio de 2018). *Moveo ROS*. Obtenido de GitHub: https://github.com/jesseweisberg/moveo_ros

Índice de figuras