TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CELAYA

DEPARTAMENTO DE INGENIERÍA MECÁNICA

TESIS

"SIMULACIÓN DE FENÓMENOS GRAVITO-INERCIALES MEDIANTE UN ROBOT SERIAL INDUSTRIAL"

PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS EN INGENIERÍA MECÁNICA

PRESENTA:

ING. GUADALUPE CARDOSO MORALES

ASESOR:

DR. HORACIO OROZCO MENDOZA

SEPTIEMBRE 2024

CELAYA, GUANAJUATO

ÍNDICE

ÍNDICE DE FIGURAS

AGRADECIMIENTOS

Aunque pareciera que es meramente un asunto individual, la realidad es que muchas personas han agregado parte de sí mismos en este proyecto. Este agradecimiento, es a todas aquellas personas que, sin darse cuenta, ayudaron a que este proyecto saliera a flote.

Agradezco a mis padres, Guillermina y Ricardo, que siempre han sido el soporte de mi vida y que gracias a ellos estoy aquí. Me apoyaron y no me dejaron caer en todo el proceso. Me dieron el espacio que necesitaba para culminar este proyecto, nunca se quejaron por a veces no estar al 100% con ellos.

A mis hermanos Cristian y Ricardo que me brindaron una cátedra de sus conocimientos en temas de programación. Sin ellos me hubiera tomado mucho más tiempo aprender lo que se necesitaba para abordar este proyecto.

Agradezco a mis amigos que estuvieron ahí todo el tiempo apoyándome y resolviendo todas mis dudas. A mi mejor amiga Korya, que me dio ánimos y que me ha estado apoyando a lo largo de mi maestría y en mi vida en general.

Por supuesto agradezco a mi asesor Horacio que siempre me estuvo guiando durante todo el proyecto y que sin él el proyecto no hubiera siquiera existido.

RESUMEN

Dentro de la tecnología de simulación se tienen proezas muy competitivas a nivel mundial, utilizadas por reconocidos pilotos de vehículos y con importantes misiones respectivas a su trabajo. Pero, por otro lado, estas tecnologías avanzadas son muy costosas.

En este trabajo se desarrolla el diseño y construcción de un simulador de fenómenos gravito-inerciales, tanto en su forma física como de software. El simulador es capaz de hacer simulaciones de carreras virtuales en tiempo real. Dicho simulador vincula un videojuego con un robot industrial.

Se propone una alternativa tecnológica para realizar el simulador, usando la tecnología estratégica, la cual disminuye los costos de su desarrollo. La tecnología empleada son un robot industrial, una computadora, visores y volante. Estas son las herramientas requeridas para brindar un simulador que obtenga realismo y versatilidad.

Se realizan tres opciones para abordar este problema, cada opción es diferente y se abordan utilizando sus herramientas específicas, pero utilizan el mismo principio, el vínculo del movimiento del volante con el movimiento del robot industrial. Se utiliza conexión serial, Arduino y ethernet en las distintas opciones, abordando cada una un punto de vista diferente del proyecto.

Python y *RAPID* son las herramientas de programación que soportan el proyecto, mediante ellas se configura y se vincula todo el hardware que se utiliza para el movimiento del robot industrial.

ABSTRACT

Within simulation technology, there are highly competitive feats on a global scale, used by renowned vehicle drivers and for important mission-specific tasks. However, on the other hand, these advanced technologies are very costly.

A technological alternative is proposed to create the simulator, using strategic technology, which reduces development costs. The technology employed includes an industrial robot, a computer, headsets, and a steering wheel. These are the required tools to provide a simulator that achieves realism and versatility.

This work involves the design and construction of a gravito-inertial phenomena simulator, both in its physical form and in software. The simulator is capable of performing real-time virtual racing simulations. This simulator links a video game with an industrial robot.

Three different approaches are taken to address this problem, each using specific tools but based on the same principle: linking the movement of the steering wheel with the movement of the industrial robot. Serial connection, Arduino, and Ethernet are used in the various approaches, each addressing a different aspect of the project.

Python and RAPID are the programming tools that support the project, through which all the hardware used for the industrial robot’s movement is configured and linked.

INTRODUCCIÓN

El objetivo de un simulador es poder recrear de la mejor manera el ambiente por el cuál fue desarrollado. En cuanto a los simuladores de movimiento, estos pretenden imitar los efectos sensoriales del entorno, para hacer una experiencia inmersiva que se compare a la realidad. Estos simuladores de movimientos tienen una parte física y una de control.

Hay diferentes tipos de simuladores de movimiento como los simuladores de carreras, que permiten experimentar la sensación de conducir un vehículo. El sistema físico junto con el sistema de control, replican los efectos de movimientos, vibraciones, aceleraciones, posiciones, velocidades, etc., que se presentan al conducir una automóvil.

Los simuladores han ido evolucionando al paso del tiempo, desde un aparato suspendido al aire libre recreando los movimientos de un avión solo con la ayuda de las corrientes de aire; hasta una inmersión total dentro de un simulador que recrea de forma precisa y realista un vuelo en avión.

Sin embargo, simuladores así de realistas llegan a ser muy costosos y su complejidad técnica es muy avanzada, por lo que se opta por simuladores menos realistas pero que permitan practicar al piloto en un entorno controlado y seguro.

En el presente trabajo, se busca abordar esa misma problemática, se propone un diseño y desarrollo de un simulador que vincule un videojuego con un robot industrial. Por medio de una comunicación serial o ethernet.

Con esto, se busca darles un nuevo uso a robots industriales desactualizados que no están operando completamente por las limitaciones de desarrollo y tecnologías nuevas, poder hacer uso de estos y equiparlos para que sean capaces de convertirse en simuladores de movimiento. Así como también, si se tienen robots

con mejores tecnologías, poder hacer también uso de la tecnología de la simulación en ellos.

Para lograr este objetivo se exponen diversas opciones para abordar el problema y realizar las conexiones entre robot y videojuego. Incluyendo así una combinación entre los lenguajes de programación *Python* y *RAPID* que tienen la tarea de controlar todo el sistema. Dentro de estas opciones se incorpora lo que es la programación en *Arduino* que ayuda a mejorar la respuesta del sistema. A todo esto, se incorporan visores de realidad virtual y videojuegos.

A través de este trabajo, se espera ofrecer un sistema integrado tanto para el entretenimiento como para la investigación de este campo, dando así nuevas oportunidades para la exploración de los fenómenos gravito- inerciales y los usos en las industrias.

CAPITULO 1.

MARCO DE REFERENCIA

1.1. Antecedentes

Los primeros intentos de simulación de vuelo según Allerton, D. (2009) fueron en dispositivos de entrenamiento, donde los pilotos podían experimentar los efectos del viento. Este simulador comprendía una cabina que solamente se activaba cuando había viento muy fuerte, la cual se movía en respuesta al manejo del piloto. Estos sistemas fueron desarrollados por Walters y Antionette en 1910 como se puede observar en la Figura 1.1.

Figura 1-1 El simulador de entrenamiento de vuelo de Antionette alrededor de 1911, este simulador utilizaba las ráfagas de viento para simular los movimientos de un avión. Allerton D. (2009)

.

Posteriormente aparece el Link (Figura 1-2), que permitía a los pilotos practicar en vuelo, los instructores podían evaluar las habilidades de vuelo, se simulaba el entorno de la cabina, su comportamiento dinámico y el rendimiento de forma neumática. De acuerdo con Allerton, D. (2009) “Se usaron actuadores para inyectar

movimiento en una pequeña cabina y los instrumentos se manejaron para parecerse a la respuesta y las características de una aeronave.”

Figura 1-2 Link trainer simulador de vuelo que permitía a los pilotos practicar en vuelo, los instructores podían evaluar las habilidades de vuelo, se simulaba el entorno de la cabina, su comportamiento dinámico y el rendimiento de forma neumática. Allerton D. (2009)

.

Cuarenta años después, hubo otra importante aportación al estudio de las simulaciones (simulación de fenómenos gravitatorios), fue realizada por el físico estadounidense J. Halcombe Laning en 1950, quien utilizó un ordenador analógico para simular el movimiento de dos cuerpos bajo la influencia de la gravedad. Fraser (2019)

En cuanto a los simuladores de carreras, estos sistemas permiten experimentar la sensación de conducir un vehículo. Es la combinación de un modelo físico y un sistema mecánico que replica los movimientos y vibraciones de un automóvil real.

Por otro lado, Kyb (2006) y Aristizabal (2015) basaron sus investigaciones en una plataforma Stewart para simular las fuerzas G que experimenta un conductor durante una carrera. Como se puede observar en la Figura 1-3, esta plataforma

dispone de seis grados de libertad controlados por seis actuadores fijados al suelo, que simulan con precisión los movimientos de un automóvil virtual.

Figura 1-3 Simulador de conducción que utiliza una plataforma Stewart para recrear los movimientos de un automóvil. Suikat, R (2005)

Para dar una alternativa a las plataformas Stewart se realizaron simuladores de movimiento, que están basados en robots industriales como lo hizo el Cybermotion y el DLR Robotic Motion Simulator.

El CyberMotion (Figura 1-4), desarrollado por el Max Planck Institute for Biological Cybernetics de Tübingen, en Alemania, simula los movimientos del automóvil dentro de un videojuego. Utiliza un *Robocoaster*, con una capacidad de carga de 500 kg. La persona es levantada y el brazo replica el movimiento que realiza el conductor en el ambiente de un videojuego. Neiuwenhuizen, F. et al (2013)

El CyberMotion Simulator tiene montada una cabina para el conductor en el extremo del brazo, además cuenta con un riel que le da un rango de movimiento extra. El CyberMotion ha demostrado ser un simulador de movimiento de última generación

para investigación psicofísica, que van desde experimentos de percepción pasiva hasta tareas de control activo, como conducir un automóvil o volar un helicóptero. Neiuwenhuizen, F. et al (2013)

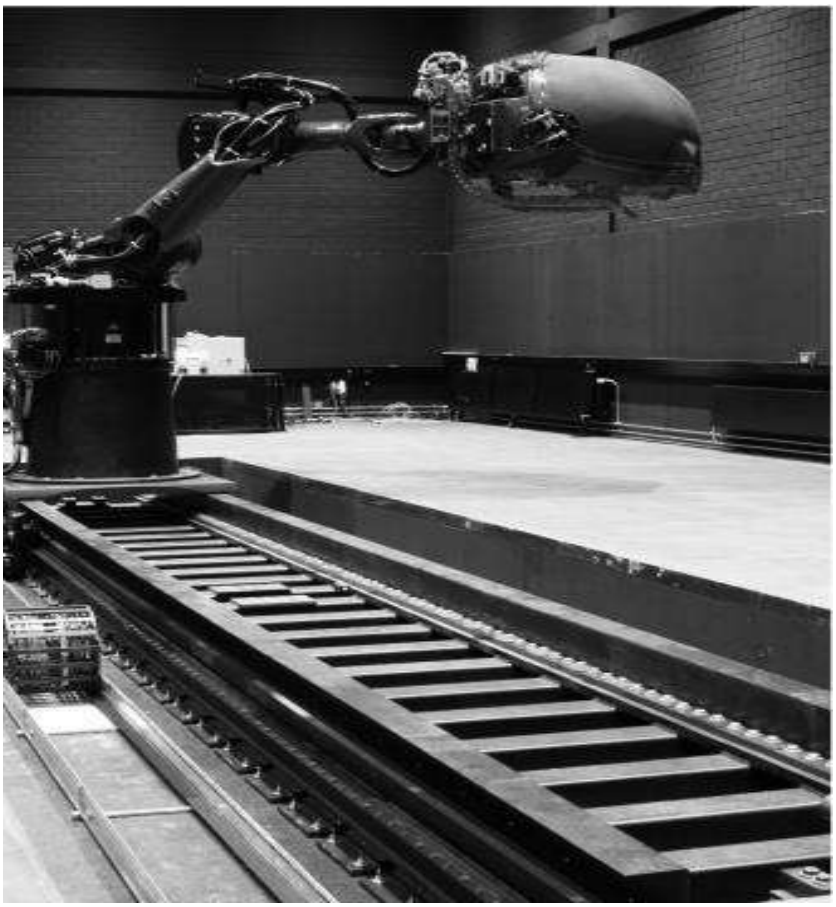


Figura 1-4 CyberMotion, simulador de movimiento de 7 grados de libertad que tiene montada una cabina para el conductor en el extremo del brazo, además cuenta con un riel que le da un rango de movimiento extra. Neiuwenhuizen, F. et al (2013)

Siguiendo este enfoque existe el DLR Robotic Motion Simulator (Figura 1-5). Donde su investigación relata el diseño y la configuración del simulador basado en cinemática en serie. Utilizan un KUKA Robocoaster que cuenta con 7 grados de libertad. Su campo objetivo es el transporte y la interacción con personas. (Bellmann, 2011)

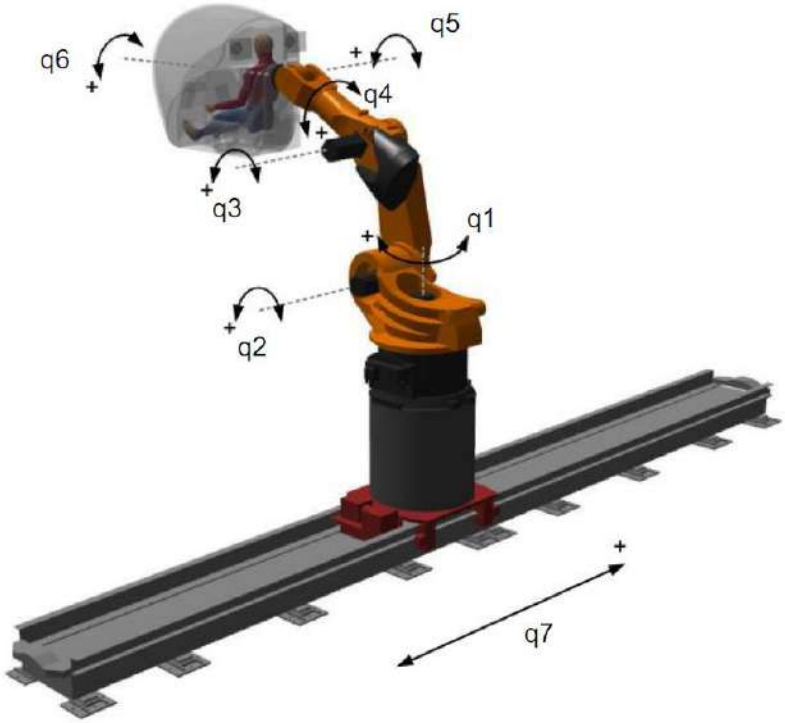


Figura 1-5 Grados de libertad del DLR Robot Motion Simulator. (Bellmann, 2011)

Actualmente hay simuladores de carreras de videojuegos que utilizan una combinación de gráficos de alta calidad, sonido envolvente y movimientos de asiento para simular la sensación de conducir un automóvil en una pista de carreras.

El presente proyecto toma como base la investigación del Cybermotion y DLR, ya que su desarrollo se asemeja a la problemática que se tienen en este proyecto. En las dos se busca crear un simulador de movimiento que sea dirigido por un robot industrial, aunque ninguno de los dos aborda conexión serial, sí se tiene una conexión ethernet en ambos.

Los sistemas de seguridad que maneja el DLR son realmente interesantes, los cuales se basan en una protección global, proteger tanto al piloto como a sus alrededores, además la protección se extiende al hadware; se tiene una valla y topes que protegen el entorno.

1.2. Planteamiento y justificación del problema

En el mercado los simuladores de movimiento son muy especializados y por ende costosos, lo que los hace poco asequibles para las investigaciones. La metodología de desarrollo está orientada a recrear el movimiento del automóvil dentro de la simulación.

La gravedad es una de las fuerzas fundamentales que rige el movimiento. Al simular estos fenómenos, se puede explorar y comprender el movimiento en diferentes entornos. La simulación de carreras, vuelos y gravedad proporciona entrenamiento efectivo, análisis de investigaciones científicas y el desarrollo tecnología avanzada.

La simulación permite a los pilotos experimentar las mismas fuerzas y condiciones dinámicas que encontrarían en una situación real. Con lo que se pueden practicar maniobras y procedimientos en un entorno seguro y controlado, además de perfeccionar sus habilidades y técnicas de vuelo y conducción.

Los simuladores de vuelo también pueden simular condiciones gravitatorias extremas, como despegues, lo que proporciona a los pilotos la oportunidad de familiarizarse y adaptar sus habilidades de vuelo en entornos únicos.

1.3. Hipótesis

El uso de un robot serial industrial para simular fenómenos gravitatorios proporciona una herramienta más precisa y flexible en comparación con otras plataformas existentes, permitiendo la realización de simulaciones más complejas y detalladas en diferentes escenarios y condiciones.

1.4. Objetivos

1.4.1. General

Diseñar y desarrollar un sistema de simulación de fenómenos gravitacionales utilizando un robot industrial permitiendo la simulación de movimiento de forma virtual, logrando que sea asequible y fácil de usar para cualquier interesado en realizar investigaciones referentes al tema.

1.4.2. Específicos

1. Investigar los conceptos básicos de los fenómenos gravito-inerciales y especificaciones técnicas de un robot serial industrial.
2. Aprender sobre la comunicación serial y TCP, para la conducción de datos del videojuego al robot industrial, logrando un movimiento en tiempo real.
3. Seleccionar el videojuego que permita ser usado con mayor facilidad y sea compatible con el plan del proyecto.
4. Crear una estructura que resista el peso del usuario y los movimientos de la simulación.
5. Diseñar y desarrollar un software para el robot industrial, que vincule el videojuego con el robot industrial y programe los movimientos.
6. Elaborar una simulación dentro de *RoboDK* para imitar el comportamiento y los movimientos que tendrá la estructura, garantizando la precisión y la estabilidad de la simulación.
7. Integrar el sistema de control del robot mediante *Python* y *RoboDK* simulando los movimientos del automóvil dentro de la carrera.
8. Realizar pruebas y experimentos para la validación del sistema de comunicación

1.5. Alcances y limitaciones

Como alcance, se pretende diseñar y programar el sistema de control del robot para simular el movimiento de un vehículo terrestre, lo que incluirá la selección y configuración de los componentes necesarios.

A su vez, integrar mediante programación en *Python* el sistema de control del robot con un software de simulación de carreras, lo que implica el desarrollo de algoritmos y uso apropiado de protocolos de comunicación entre los sistemas.

Se propone diseñar y construir un simulador de fenómenos gravito-inerciales, tanto en su forma física como de software. El sistema será capaz de hacer simulaciones de carreras virtuales en tiempo real.

Como limitadores, se tiene el envío de datos, entre el controlador y el software de simulación, ya que está limitado a una conexión serial de RS232, la cual es mucho más lenta en transferencia de datos, debido a la capacidad del robot disponible. El suplementar una conexión ethernet implica actualizar los controladores, lo cual requiere de recursos tecnológicos y económicos no disponibles.

CAPITULO 2

MARCO TEÓRICO

2.1. Simulación de efectos gravito-inerciales

En este proyecto, se refiere como simulador de realidad virtual a un sistema con interacción hombre-máquina donde la máquina envuelve al usuario en una especie de burbuja realista y por medio de estímulos sensoriales y visuales, la máquina puede recrear todo tipo de escenarios, llevando al usuario a una experiencia inmersiva que sea difícil de distinguir con la realidad. Bellmann (2011)

Los simuladores de movimiento tienen como objetivo crear la sensación de operar o estar en un vehículo real dentro de un entorno de simulación, para esto los sentidos humanos deben recibir información sensorial lo más real posible. Normalmente, las simulaciones se centran en la visión y el oído y, en el caso de los simuladores de movimiento, también incluyen señales de movimiento. Bellmann (2011)

En este caso el objetivo es recrear un simulador de carreras. Para que esto sea efectivo, se debe de representar el comportamiento del vehículo de la mejor manera, que los movimientos se sientan reales y la interacción con esta máquina se sienta natural. El ambiente en el que se rodea esta máquina debe de tener gráficos de calidad y el comportamiento del ambiente debe ser realista.

2.2. Realidad virtual

Es un entorno de escenas u objetos de apariencia real, que crea en el usuario la sensación de estar inmerso en el espacio físico que representa. Es una realidad simulada, donde las aplicaciones sumergen al usuario en un entorno artificial, generado por computadora, que simula la realidad mediante el empleo de

dispositivos interactivos, que envían y reciben información, mediante el empleo de sensores y actuadores.

La realidad virtual necesita de soportes tecnológicos que puedan aislar al usuario completamente del mundo real, creando una sensación total de inmersión. Permite simular una experiencia sensorial completa dentro de un ambiente artificial. Estos dispositivos interactivos, habitualmente son los denominados gafas o cascos de realidad virtual, si bien pueden existir otros como auriculares, guantes o trajes especiales, que permiten la percepción de diferentes estímulos sensoriales (vista, oído y, en menor medida, tacto; aún sin apenas desarrollar olfato y gusto). Ordóñez, L. (2020)

La realidad virtual involucra soluciones a problemas reales en ingeniería, medicina, usos militares, etc. El grado en que una aplicación es capaz de resolver un problema particular, es decir, el grado en que una simulación funciona bien, depende en gran medida de la imaginación humana. Ordóñez, L. (2020)

Los simuladores de los que se hablará son aquellos que se hacen llamar simuladores de movimiento, los cuales proporcionan un entorno seguro, flexible y controlable para poder entrenar a pilotos en diferentes tipos de vehículos de forma segura. Por si fuera poco, los investigadores tienen la facilidad de analizar el comportamiento de los pilotos de una forma mucho más controlada.

2.3. Movimientos traslacionales y rotacionales

Cada objeto tiene un lugar en el espacio, para poder saber su ubicación y conocer cómo se mueve, se utilizan las coordenadas traslacionales y rotacionales, estás se dividen en tres componentes de desplazamiento y tres de rotación en los diferentes ejes.

Las tres componentes de desplazamiento en los tres ejes, X, Y y Z; indican cómo se mueve el objeto en el espacio. Donde el eje X representa el movimiento horizontal, el eje Y el movimiento vertical; y el eje z representa la profundidad.

Las tres componentes de rotación en cada uno de estos ejes describen como cambia la orientación sin alterar la posición física. Normalmente se considera que, si la rotación se hace sobre el eje X, se le conoce como Alabeo, si es en el eje Y, es Cabeceo, mientras que, si es en el eje z, este es Viraje. Casas, S. (2014)

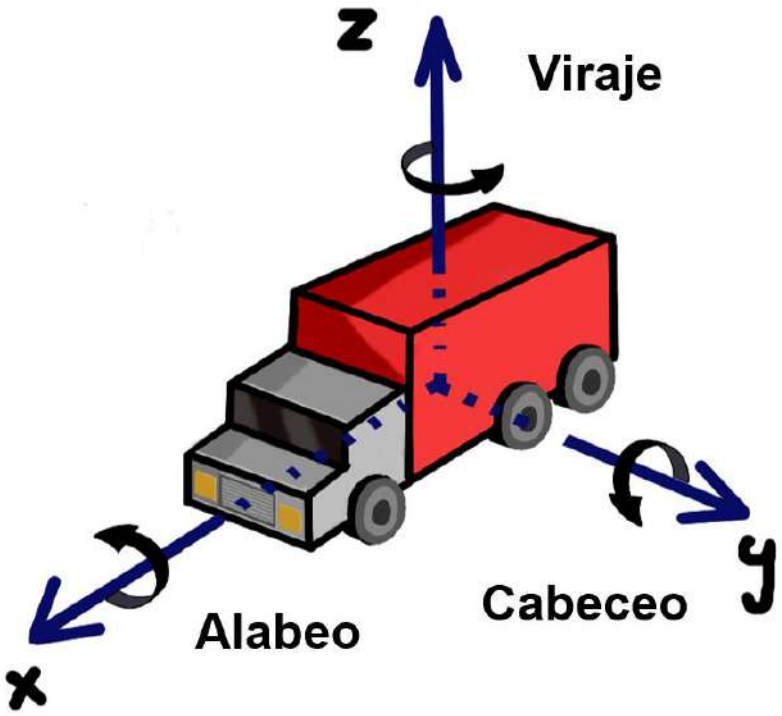


Figura 2-1 Representación de los movimientos traslacionales y rotacionales

En la Figura 2-1, se representa cada una de las coordenadas y sus respectivas rotaciones, donde se ejemplifica el movimiento que puede tener un camión de carga.

2.4. Robot industrial

Según la ISO 8373:2012, un robot es un manipulador multifuncional reprogramable, el cual es capaz de mover materiales, piezas, herramientas o dispositivos especiales, esto realizando movimientos que son programados por un usuario

Los robots son clasificados como industriales, no industriales o para usos especiales. En este proyecto solo se describen los robots industriales. International Organization for Standardization. (2012).

“El objetivo de los robots industriales es el de servir a un propósito universal y de mano de obra no calificada o semicualificada, por ejemplo, para soldar, pintar, realizar mecanizados, etc.”. Subir Kumar (2010)

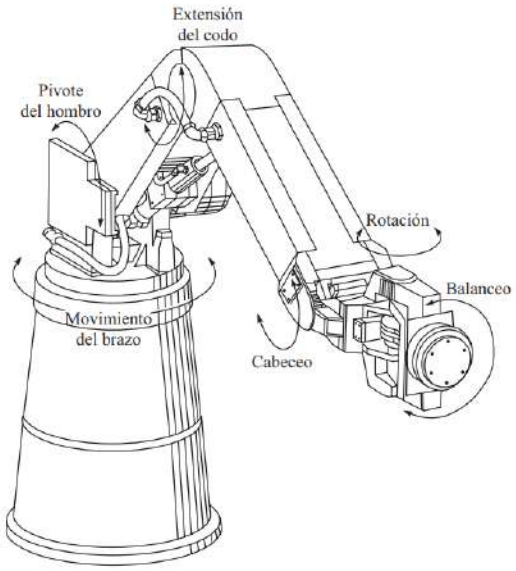


Figura 2-2 Representación de los movimientos de traslación y rotación en un robot industrial. Subir Kumar (2010)

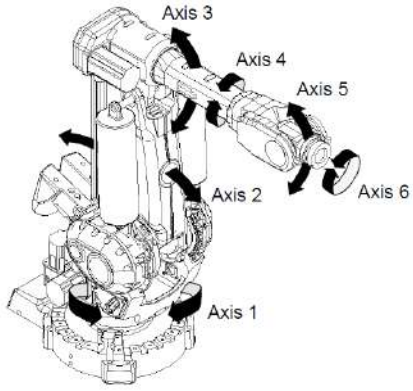
El robot utilizado en este trabajo es un robot modelo ABB IRB 6400 utilizado en diversas tareas industriales. Es conocido por su alta confiabilidad y repetibilidad, lo que es esencial en aplicaciones donde se requiere un alto grado de precisión, como la soldadura, el pintado y el manejo de materiales. El robot tiene una capacidad de carga de 250 kg y una velocidad máxima de trabajo de 15 m/s. *ABB Robotics Products* AB. (1993)

Datos generales:

Preciso: Calidad uniforme de las piezas. (Repetibilidad=1.0 mm), y con el *TrueMove*1 de ABB, el robot sigue siempre el mismo recorrido, independientemente de la velocidad

Robusto: Fabricación para un entorno exigente. El IRB 6400 está fabricado completamente en acero con materiales de gran resistencia. Los brazos están equilibrados mecánicamente y dotados de cojinetes dobles. Tiene una alta resistencia a la corrosión (*IP67)* para todo el brazo mecánico y se puede lavar con vapor a alta presión, lo que lo hace ideal para trabajar en ambientes agresivos. ABB Robotics Products ABB. (1993)

En la Figura 5-3 se indican los rangos de los ángulos de cada una de las articulaciones del manipulador



Joint	Motion range
Axe 1	+180° to -180°
Axe 2	+140° to +10°
Axe 3	+155° to +47°
Axe 4	+300° to -300°
Axe 5	+120° to -120°
Axe 6	+300° to -300°

Figura 2-3 Rango de los ángulos de las articulaciones del robot ABB IRB 6400. ABB Robotics Products ABB. (1993)

2.5. Fenómenos gravito-inerciales

El efecto de la gravedad se basa en la atracción que se produce entre dos objetos, cuanto mayor sea su masa mayor es la fuerza de atracción entre ellos. En cuanto a

la Inercia, es la propiedad que tienen los cuerpos de mantener su estado de reposo o movimiento si no es perturbado por la acción de una fuerza.

Los fenómenos gravito-inerciales son aquellos relacionados con los dos efectos anteriores, tanto la gravedad, como la inercia.

Estos fenómenos pueden ser detectados por el humano a través de biosensores que se encuentran en el sistema vestibular, que está ubicado en el oído interno y a través del cuerpo. A estos efectos sensoriales, se les llaman claves gravito-inerciales, ya que están relacionadas con la percepción del movimiento y la orientación del cuerpo. Casas, S. (2014)

El sistema vestibular puede conocer la orientación del cuerpo humano, proporcionando información de los movimientos rápidos y poder mantener el equilibrio ante estas alteraciones. Tienen dos sistemas sensoriales: los canales semicirculares, para detectar la rotación; y los oídos, que son sensibles a la aceleración lineal. García, J. (2016)

2.6. Entornos virtuales para simulación de movimientos

El videojuego que se va a utilizar, en sí no es muy importante, ya que la programación se puede utilizar con cualquiera de los videojuegos en el mercado. Aun así, si se requiere la utilización de telemetría (descrita más adelante) se debe de seleccionar algún videojuego del cual se pueda extraer los datos de movimiento en tiempo real.

Para esto se seleccionan los juegos *BeamNG.teach* y *EuroTruck2*. *BeamNG.teach* cuenta con las herramientas suficientes para modificar todo el entorno, ofrece docenas de vehículos refinados y completamente personalizables para experimentar, además cuentan con una API de código abierto.

El videojuego *Eurotruck*, tiene un espacio abierto de buenos gráficos y recrea los movimientos del volante y los pedales de manera muy precisa.

2.7. Control de robots industriales

Típicamente los robots industriales tienen varias formas de control; la forma manual, la forma semi-manual y la programada. ABB Robotics Products AB. (1993)

- La forma manual, simplemente busca controlar al robot con una palanca. El robot se mueve siguiendo los movimientos del *teach pendant.*
- La semi-manual, permite programar los puntos a donde se desea dirigir el robot, ya sea usando coordenadas específicas o moviendo el robot con la palanca y guardando las coordenadas del punto definido a donde se quiere mover.
- La forma programada, aquí se programan los movimientos que hará el robot dentro de su lenguaje de programación, ya sea en un software externo o dentro del mismo robot.

2.8. Lenguaje de programación *RAPID*

Es un lenguaje de programación de alto nivel desarrollado por la empresa *ABB,* para generar rutinas de movimiento mediante instrucciones particulares, tipos de datos específicos y subprogramas.

A continuación, se incluye un código ejemplo de ABB Robotics Products (2007) para ver la estructura que puede tener un programa en el lenguaje RAPID:

MODULE MainModule # Se abre lo que es el módulo para empezar los procedimientos y declaración de variables

    VAR num length; # Se declaran variables tipo numérica

```
    VAR num width;
    VAR num area;

    PROC main()  #Inicio del procedimiento
    length := 10;  #Se asigna un 10  a esta variable
    width := 5; #Se asigna un 5  a esta variable
    area := length * width; #El producto de las dos anteriores valores se le asigna
a área
    TPWrite "The area of the rectangle is " \Num:=area;
    END PROC
ENDMODULE
```

Este programa calculará el área de un rectángulo y la escribirá en el *FlexPendant*:

The area of the rectangle is 50.

2.9. *Python*

Python es un lenguaje de programación de propósito general, de alto nivel, interpretado y dinámico (se pueden utilizar enfoques diferentes para resolver problemas). Es apreciado por su sintaxis clara y legible y su gran comunidad de usuarios. Su sintaxis es muy clara, además cuenta con una variedad muy grande de bibliotecas. (Lutz, M 2017)

Dentro del proyecto se utilizan algunas librerías. Estas librerías son colecciones predefinidas de funciones, clases y variables para poder expandir la función del lenguaje y no tener que escribir el código desde cero para poder realizar tareas específicas.

A continuación, se describen tres de las bibliotecas disponibles, empleadas en este proyecto

2.9.1. *Pyserial*

Es una librería que se enfoca en la comunicación serial de cualquier hadware. En este proyecto se utiliza para obtener una conexión con *Arduino* y con el robot, de forma serial.

2.9.2. *Pygame*

Esta librería se enfoca en la creación de videojuegos 2D. Esta librería es utilizada en este proyecto para hacer la conexión del control manual o un volante de *Playstation 4* (*PS4)* hacia la computadora y se puedan leer los movimientos que este realiza.

2.9.3. Robolink

El módulo *Robolink* es el puente entre *RoboDK* y *Python*. Cada objeto en el árbol de elementos de *RoboDK* se puede recuperar y está representado por el elemento objeto. Un elemento puede ser un robot, un marco de referencia, una herramienta, un objeto o un proyecto específico. Es posible realizar diferentes operaciones en ese elemento según la clase. *RoboDK*. (s.f.)

2.10. Microcontroladores *Arduino*

Arduino es una placa o plataforma de hardware libre basada principalmente en un microcontrolador, y un entorno de desarrollo (software), diseñado para facilitar el uso de la electrónica en los diversos proyectos multidisciplinares. *Arduino* es una tecnología con el uso de directo de hardware y software. En cuanto a hardware se

compone por varias partes e interfaces las cuales esta reunidas en una placa de circuito. Vital, M. (2021) En la Figura 2-4 se muestra la placa Arduino UNO.

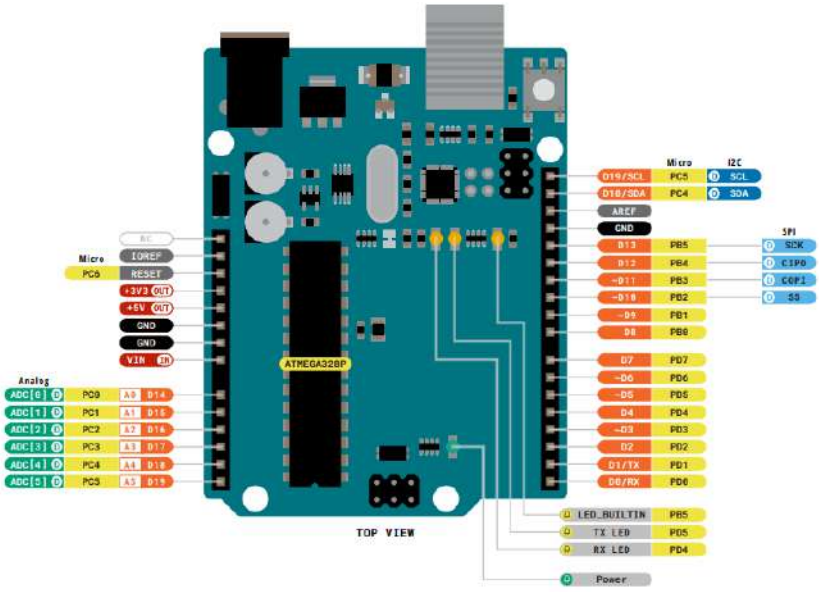


Figura 2-4 Arduino UNO Arduino(s.f.)

2.11. Programación y simulación gráfica de robots. *RoboDk*

RoboDK es un software de simulación y programación de robots. La programación de robots (programación fuera de línea) significa que los programas para el robot pueden ser creados, simulados y generados fuera de línea (desde un ordenador) para un brazo robot y un controlador robot específicos. *RoboDK* permite programar brazos robóticos para operaciones de fabricación.

La Figura 2-5 presenta una demostración del programa RoboDK, se puede observar la interfaz y el árbol de operaciones, así como el robot utilizado y sus operaciones de movimiento. RoboDK. (s.f.)

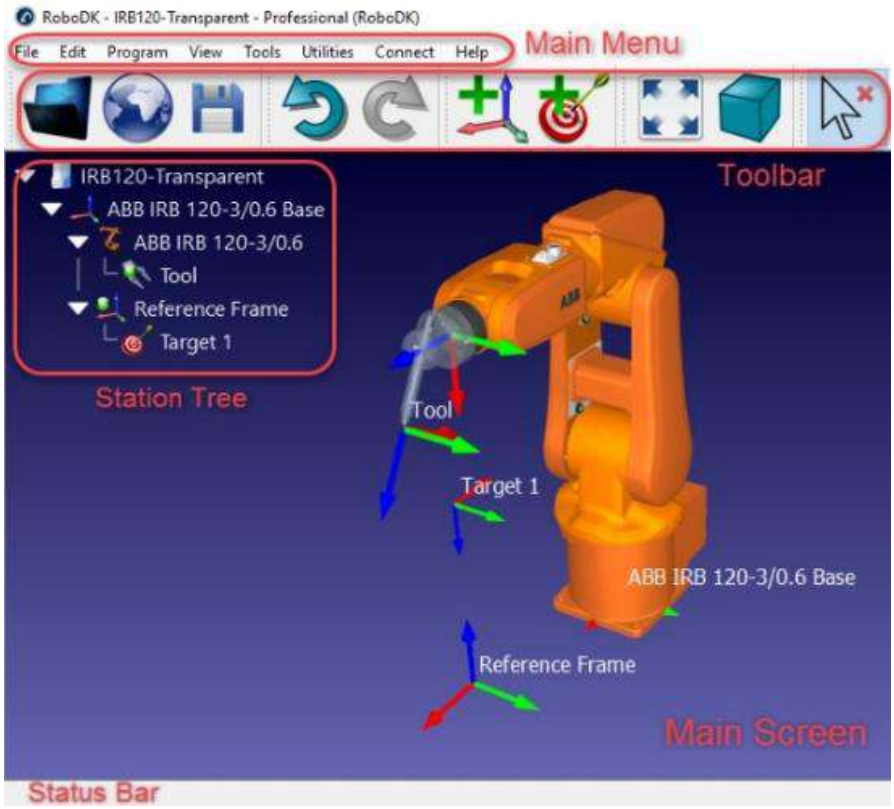


Figura 2-5 Demostración RoboDK, se puede observar la interfaz y el árbol de operaciones, así como el robot utilizado y sus operaciones de movimiento RoboDK. (s.f.)

2.12. Dispositivos periféricos auxiliares

Se utiliza un control de *PS4* y un volante *Logitec G29* para la generación de señales, como la programación es muy parecida entre los dos, se optó por utilizar el control de *PS4* en la etapa de desarrollo, mientras que el volante se utiliza en las prácticas y en el entorno final.

Para poder reconocer las señales que se mandan por el control o el volante, se tienen configuraciones establecidas por la librería de *Pygame* en *Python*, las cuales detectan el botón o palanca (*stick*) que se está moviendo.

Figura 2-6 Volante Logitec G29 y control PS4

Se utilizaron lentes de realidad virtual marca *Steren*, los cuales tienen integrados audífonos. Funcionan por medio de un teléfono celular, el cual es introducido dentro de estos lentes para visualizar de manera bifocal lo que se transmite por la pantalla. Se debe de considerar que el formato de imagen que se proyecta debe de ser bipartido, ya que los visores funcionan mediante dos lentes convexos que al unir las imágenes permiten tener visión estereoscópica.

Figura 2-7 Lentes de realidad virtual, los cuales solo se le puede adaptar al visor una pantalla de teléfono celular

2.13. Mapa del controlador para *Playstation* 4 en *Python*

Para poder detectar el control o el volante en *Python*, se necesita la ayuda de una librería llamada *Pygame*, esta tiene preestablecida la configuración que se debe de seguir para llamar a cada botón o *stick* que es pulsado.

En la Tabla 1 se muestra el mapa de configuración del control de *PS4*, donde se muestra el cómo se debe de llamar la función dentro del lenguaje de programación *RAPID* para acceder a los botones.

Tabla 1 Mapa del controlador PS4

Ejes		
Left Stick	Left -> Right	Axis 0
	Up -> Down	Axis 1
Right Stick	Left -> Right	Axis 2
	Up -> Down	Axis 3
Left Trigger	Out -> In	Axis 4
Right Trigger	Out -> In	Axis 5

Botones	
X	Button 0
Circulo	Button 1
Cuadrado	Button 2
Triángulo	Button 3
Share	Button 4
Botón PS	Button 5

Opciones	Button 6
L. Stick In	Button 7
R. Stick In	Button 8

Volante y pedales	
Volante	Joystick(0)
Acelerador	Joystick(1)
Freno	Joystick(2)

Botones analógicos

Se tienen dos tipos de botones en el *PS4*, los analógicos y los digitales. Los botones analógicos cambian su valor de a cuerdo a la presión ejercida, por ejemplo los botones R2 y L2 pueden ir de -1 a 0 o de 0 a 1 (según se programe) comienza en 0 pero mientras más se presiona el gatillo, más aumenta su valor hasta llegar a 1.

En cambio los botones digitales como el X o el circulo solo tienen dos estados, el 0 para cuando no está activo y el 1 para cuando está presionado.

En cuanto a los *sticks* o las palancas, estas se mueven de -1 a 1, incrementando en referencia a dónde se encuentren, igual que un botón analógico

2.14. Telemetría

La telemetría en los videojuegos son datos que se extraen del juego, cuyos valores cambian de acuerdo con la interacción del usuario, por ejemplo: la posición del jugador, los comentarios, elecciones, estado, etc. Estos datos contenidos dentro del mismo juego se procesan para que tengan un formato accesible. Darias, A. (2008)

CAPITULO 3

DESARROLLO DEL PROYECTO

3.1. Integración para el desarrollo del proyecto

El proyecto está distribuido de acuerdo con diferentes estrategias que se abordaron dentro del proyecto. En la Figura 3-1, se muestra un diagrama donde se puede observar las etapas que se siguieron para llegar a las diferentes soluciones, en este caso un total de 5 opciones.

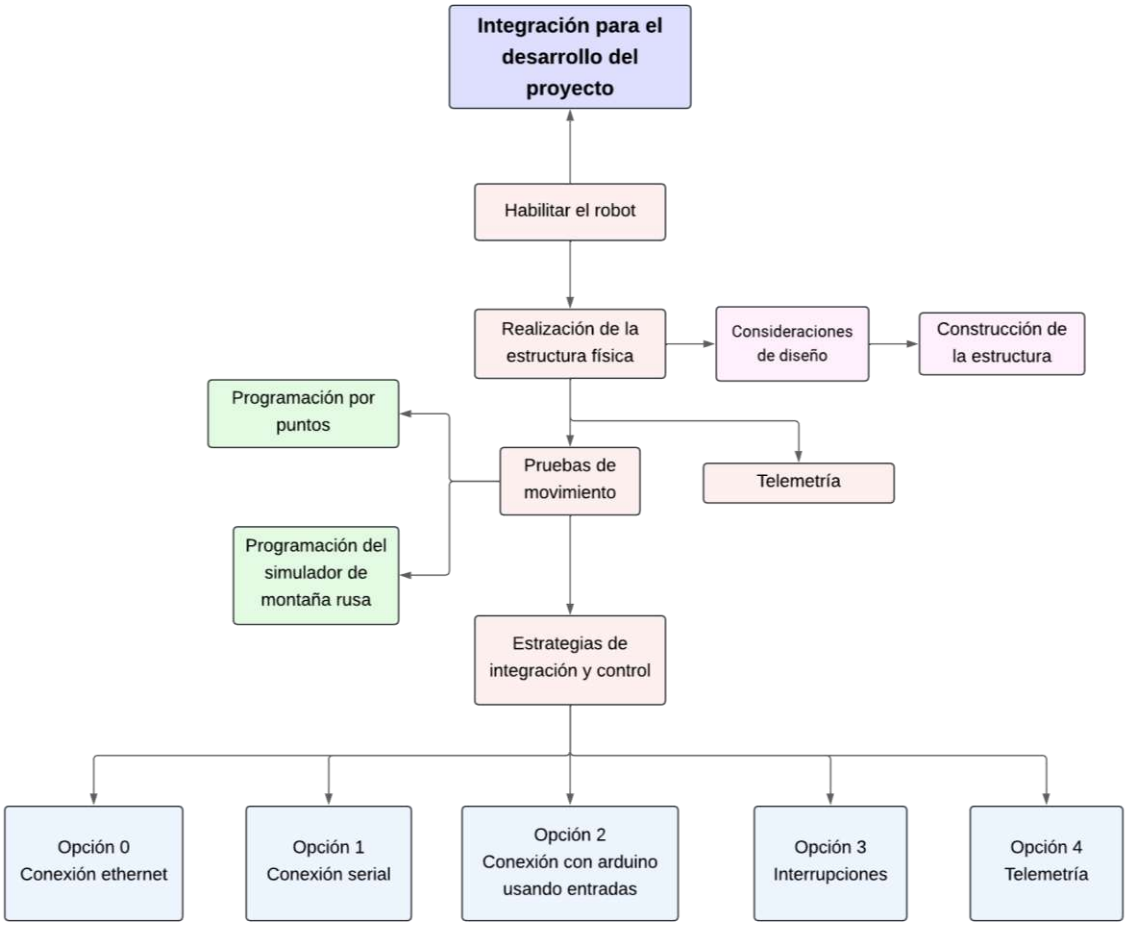


Figura 3-1 Mapa conceptual del desarrollo del proyecto y las opciones que se desarrollaron dentro del proyecto

3.2. Habilitar brazo robótico

Para poder habilitar el brazo robótico, primero se le dio mantenimiento; se hizo el acomodo de las conexiones dentro del controlador, revisión de baterías y se realizó un inició en frío del controlador (Ver Anexo 1).

Se revisaron las baterías con multímetro para asegurar su funcionamiento, estas, ya estaban dañadas, así que no daban ninguna medición de voltaje, por lo que se optó por hacer el cambio por unas nuevas.

La importancia de las baterías radica en la configuración del robot, ya que, al momento de introducir la base de datos, este necesita que siempre esté en funcionamiento la parte del inicio del robot, así que, si estas no funcionan, el robot no guarda la base de datos y se debe de configurar cada vez que se enciende.

3.2.1. Calibración

Para poder calibrar el robot, lo que se hace es mover los ejes a su punto de casa o su *home*, (cada robot tiene un punto *home*). En este caso el robot *ABB*, tiene marcas en cada uno de sus ejes para ubicar el punto home.

Para su calibración es necesario llevar todos los ejes a esos puntos *home* de referencia, para después calibrarlo utilizando el *teach pendant* y su configuración (ver Anexo 2).

En la Figura 3-2 se muestra un ejemplo de calibración de los ejes 5 y 1, en donde se lleva la posición a las líneas de cero.

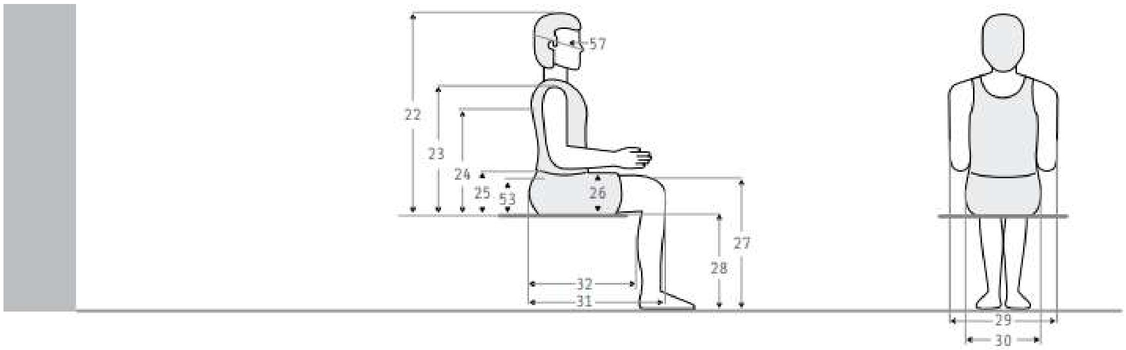
Figura 3-2 Calibración del eje 5 y 1, se llevan las líneas más gruesas a tocarse entre sí, eso indicaría el cero

3.5. Construcción de la estructura

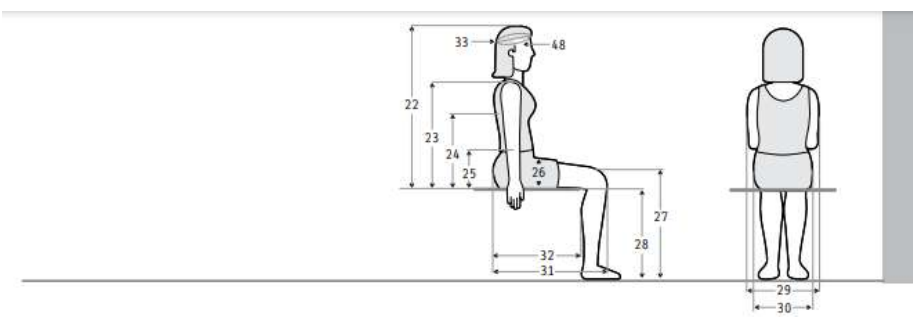
Al momento de diseñar la estructura, se tomaron en cuenta varias consideraciones, como lo son las dimensiones antropométricas de la población mexicana, estas extraídas de Hernández (2015). En su artículo nos señala las dimensiones promedio que tiene la población mexicana en diferentes posiciones, todo esto caracterizado por diferentes rangos de edad.

En la Figura 3-3 se muestra las dimensiones de hombres y mujeres adultos de 18 a 65 años y 19-24 años respectivamente. Donde se señala las opciones tomadas para la realización de la estructura.

Con la finalidad de estudiar a fondo la ergonomía, se utilizaron los datos obtenidos de las dimensiones antropométricas dentro de un rango de edad de 18 a 65 años, en una posición sentado. Las medidas que se utilizaron son un promedio de las más pequeñas y las más grandes, esto para tener un mínimo y un máximo a considerar en el largo de la estructura.



Dimensiones		18 - 65 años (n=396)				
				Percentiles		
		x̄	D.E.	5	50	95
28	Altura poplítea	412	25.65	374	412	453
29	Anchura codos	531	54.90	443	529	620
30	Anchura cadera sentado	374	31.26	328	372	423
31	Longitud nalga-rodilla	583	33.41	537	582	640



Dimensiones		19 -24 años (n=187)				
				Percentiles		
		x̄	D.E.	5	50	95
28	Altura poplítea	360	22	325	363	396
29	Anchura codos	500	52	415	495	586
30	Anchura cadera sentado	387	42	318	380	456
31	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
32	Longitud nalga-popliteo	463	26	420	465	506
33	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
48	Perímetro cabeza	544	17	517	545	572
56	Altura lumbar	190	19	158	189	222

Figura 3-3 Dimensiones de hombres adultos de 18 a 65 años y 19-24 años respectivamente. Donde se señala las opciones tomadas para la realización de la estructura. Hernández (2015).

Ávila et al (2007) establecieron en su trabajo de investigación las medidas antropométricas en posición sentado, (Figura 3-4). La consideración de las dimensiones antropométricas es un factor importante para la planeación, diseño y fabricación de estaciones de trabajo.

			Trabajadores de la industria Rosalío Ávila		
	ID	Descripción	5	50	95
Posición sentado	22	Altura normal sentado	790	854	927
	23	Altura hombro sentado	511	566	638
	24	Altura omoplato	377	434	486
	25	Altura codo sentado	201	248	293
	26	Altura máx. muslo	126	152	185
	27	Altura rodilla	435	493	556
	28	Altura poplítea	338	393	453
	29	Anchura codos	411	509	620
	30	Anchura cadera sentado	328	387	472
	31	Longitud nalga-rodilla	534	579	640
	32	Longitud nalga-popliteo	432	474	526
	53	Altura cresta-iliaca	158	200	236
	57	Diámetro a-p cara	192	217	235

				Percentiles		
ID	Descripción	Media (mm)	Desv. Std. (mm)	5	50	95
	Edad (años)	36.2	8.1	22.9	36.2	49.5
1	Peso (kg)	68.0	14.0	45.0	68.0	90.9
2	Estatura (mm)	1579.6	64.3	1473.8	1579.6	1685.5
6	Altura hombre	1315.8	61.8	1214.0	1315.8	1417.5
7	Altura codo	980.4	58.1	884.9	980.4	1075.9
12	Altura rodilla	473.5	45.0	399.5	473.5	547.5
18	Alcance brazo frontal	641.2	35.7	582.4	641.2	699.9
19	Alcance brazo lateral	608.8	35.6	550.3	608.8	667.4
23	Altura hombro sentado	557.7	34.4	501.0	557.7	614.4
25	Altura codo sentado	256.5	37.8	194.3	256.5	318.8
28	Altura poplítea (piso a muslo)	442.7	39.4	378.0	442.7	507.4
30	Anchura cadera sentado	445.4	57.9	350.1	445.4	540.7
44	Longitud pie	241.7	12.6	220.9	241.7	262.5

Figura 3-4 Medidas antropométricas en posición sentado Ávila et al (2007)

Las medidas de interés dentro de esta investigación son aquellas en posición sentada, longitud de nalga -rodilla y altura poplítea, ya que esto nos ayuda a conocer el largo de las piernas de las personas y de acuerdo con estas referencias comenzar con el diseño de la estructura.

Referente al ángulo en el que deben de estar posicionados los pies (ángulo formado por la rodilla), es importante considerar que no debe de ser incómodo para el usuario mantener largos periodos de tiempo esa posición.

Delgado, A. et al. (2012) en su artículo de investigación hacer referencia a las medidas que se deben de tener en la postura de conducción, llegando a los valores mostrados en la Figura 3-5.

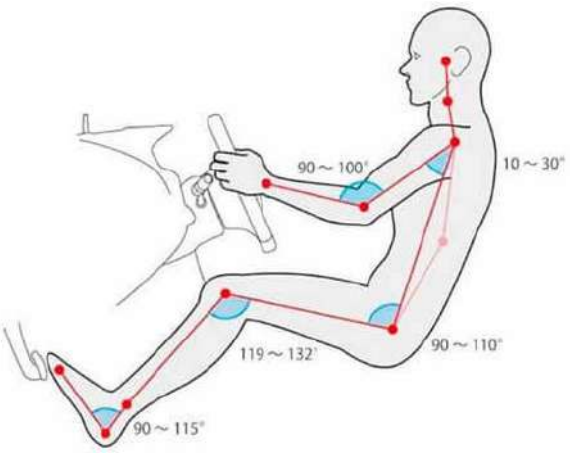


Figura 3-5 Ángulos correctos para una buena postura en la conducción

Todo esto se tomó en cuenta para poder trazar una guía que sirva para poder definir el mínimo y máximo en el largo de la estructura, así como la altura adecuada.

En la Figura 3-6, se definen en conjunto tanto el máximo como el mínimo que llevará la estructura. De fondo se tiene el modelo 2D de la base de la estructura, donde se puede ver que el largo máximo es el considerado con las mediciones más altas de adultos hombres de 18 a 65 años, encontradas por Hernández (2015) desde la planta de los pies hasta el glúteo. Las medidas mínimas se consideran en la flexión de la rodilla, ya que la estructura puede no dejar flexionar la pierna adecuadamente.

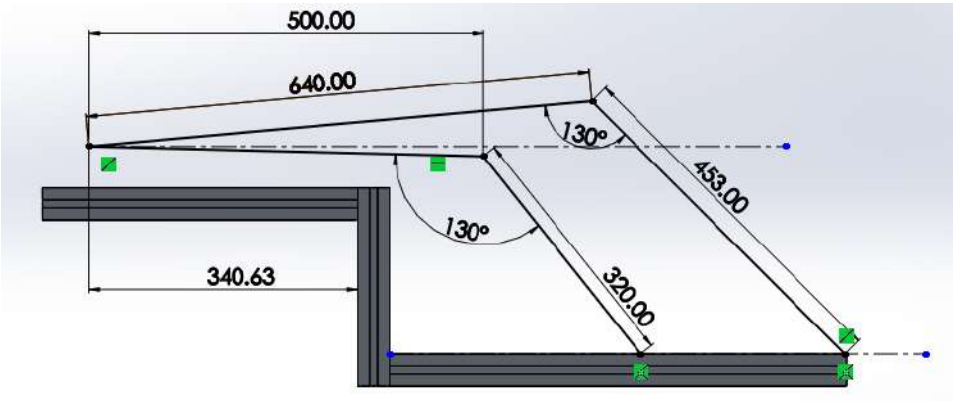


Figura 3-6 Representación del máximo y mínimo en el largo de las piernas de los adultos

3.6. Creación de bocetos

Dentro de la creación de los primeros bocetos se consideraron los valores establecidos en los párrafos anteriores.

Las partes principales para considerar son: el largo de la estructura, la posición donde se colocará el volante y pedales, y la seguridad del usuario.

Dentro de la seguridad se considera como primer diseño, el utilizar un sistema de topes como los utilizados en juegos mecánicos. También se opta por utilizar arneses o cinturón de seguridad.

La postura más utilizada en simuladores de carrera es aquella con reclinación, tanto del respaldo como del asiento, pero al ser una posición con ángulos muy pronunciados, esto causará que nuestra estructura esté muy larga, lo que provoca que no sea muy estable, aumentando el valor de las cargas torsionales y su tendencia a flexionarse en el extremo más alejado al robot.

Como material de construcción para la estructura, se eligió el perfil estructural de aluminio de 40mm, ya que es resistente y a la vez ligero, aunque es difícil de soldar, por lo que las uniones se limitarán a ser rectas y utiliza escuadras entre ellas.

Figura 3-7 Perfil estructural de aluminio Bosh de 40mm x 40mm

Teniendo como consideración lo anterior, como primera opción, se consideró que la parte que sostiene el volante se pudiera mover hacia delante y hacia atrás. El soporte donde se colocará el volante forma parte del conjunto que servirá como seguridad para el usuario.

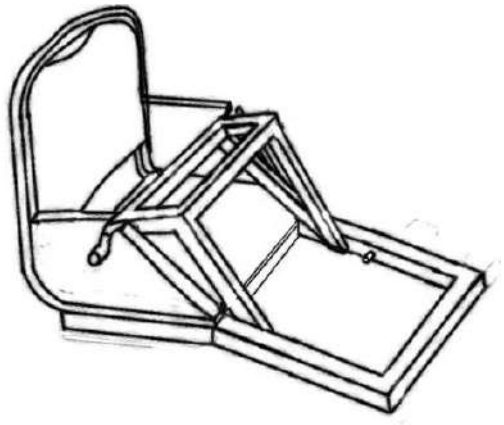


Figura 3-8 Primer boceto de la estructura

El principal inconveniente de este diseño es que la parte donde se colocarán los pedales no tiene ningún refuerzo y la inclinación que tiene en la zona donde reposan los ´pies no se puede realizar de manera sencilla con un perfil empleado. Además, la estructura del volante es muy inestable al solo tener un punto de contacto en cada lado, por lo que se rechazó la propuesta

Después de varios diseños y correcciones para cada uno de los factores descritos, se llegó al diseño mostrado en la Figura 3-9. Los perfiles crearán ángulos de 90° que se deberán unir con pequeñas escuadras.

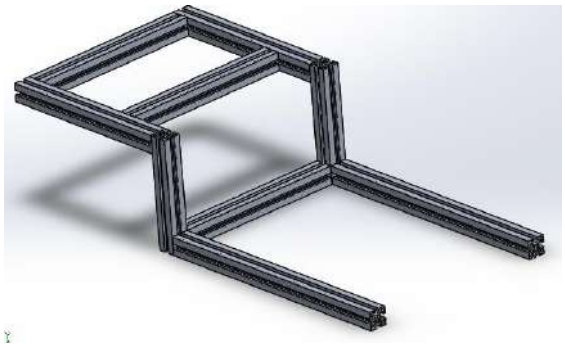


Figura 3-9 Modelado de la estructura

En la Figura 3-10, la silla sienta en la estructura de una mejor manera y se tienen refuerzos de solera a cada lado para impedir lo más que se pueda la flexión.

Figura 3-10 Modelado de la estructura con la silla

En cuanto a la seguridad, la primera opción fue utilizar cinturones de seguridad de 4 puntos con cierre rápido; este, por medio de dos arneses que salen de atrás de la silla, sujetan al usuario desde el pecho y estómago. (Figura 3-11)

Figura 3-11 Primer sistema de seguridad utilizado

Posteriormente se mejoró el sistema de sujeción empleando un arnés *Aces Racing* de 5 puntos con relleno de 2 pulgadas y una certificación E4. Este arnés cuenta con un ajuste universal y está diseñado para carreras de automoviles.

Figura 3-12 Segundo sistema de seguridad, este con 5 puntos de apoyo para aumentar la seguridad del usuario

Para la parte superior donde se ajusta el volante, se instaló una estructura rectangular, considerando un espacio apropiado entre esta y la silla para que se pueda entrar y salir fácilmente. (Figura 3-13)

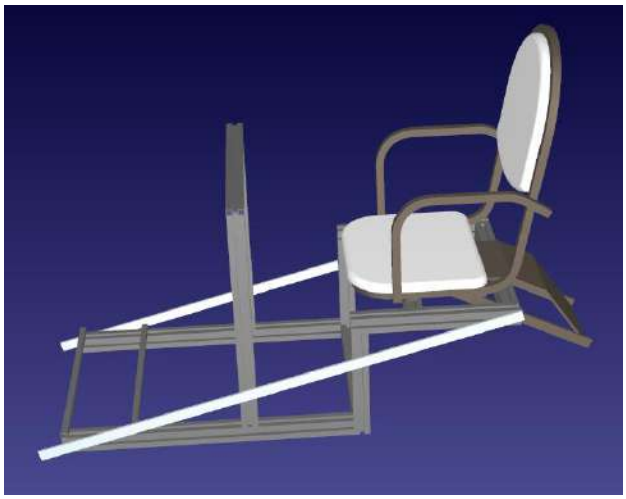


Figura 3-13 Representación 3D de la estructura

Por último, el soporte de los pedales. Esta parte se puede deslizar hacia delante y atrás, siendo guía los perfiles paralelos de la parte inferior de la estructura, dando oportunidad a un fácil cambio de las distancias que el usuario puede alcanzar. (Figura 3-14)

Figura 3-14 La estructura podrá deslizarse hacia delante y atrás, siendo guía los perfiles paralelos de la parte de debajo de la estructura, dando oportunidad a un fácil cambio de las distancias que el usuario puede alcanzar

3.7. Construcción

Se comenzó con el corte de los perfiles de aluminio, de acuerdo con las dimensiones establecidas en el diseño. En este proceso fue importante asegurarse que los acabados y ángulos de los cortes fueran apropiados.

Las escuadras se construyeron de perfiles rectos de acero de 5/16 x 1 pulgada, con barrenos para tornillos M8, cuidando de igual manera su acabado y dimensiones (Figura 3-15).

Figura 3-15 Escuadras de acero

La Figura 3-16 muestra de manera parcial, el proceso de ensamble de la estructura. Los perfiles también fueron barrenados.

Figura 3-16 Conexión de los perfiles estructurales de aluminio

Se utilizó solera de aluminio para los soportes laterales. (Figura 3-17)

Figura 3-17 Refuerzo de solera, que conecta los perfiles superiores con los inferiores

Para el ensamble de la estructura y la silla, se utilizan dos perfiles acoplados a la base de la silla (ver Figura 3-18).

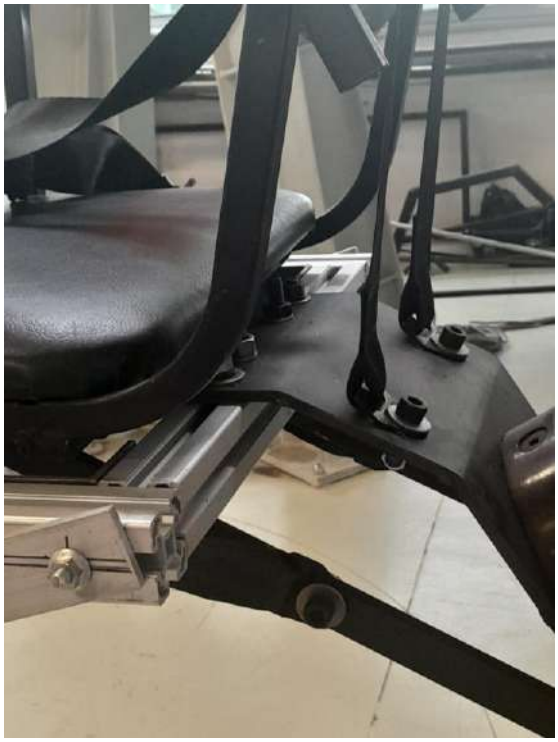


Figura 3-18 La silla de acero, está conectada por medio de los perfiles de aluminio y una estructura de acero unidas con placas

En la Figura 3-19 se muestra la estructura final acoplada al robot, incluyendo los soportes para el volante y los pedales.

Figura 3-19 Estructura terminada

3.8. Prueba de movimiento libre

Se realizaron pruebas de movimiento libre del brazo del robot. Movimiento libre se refiere a que solo se mueve el robot en modo manual, utilizando la palanca del *teach pendant* para mover el control. Se comprueban los movimientos en todas las direcciones, así como el funcionamiento de cambio de ejes y la vibración de la estructura acoplada, debida al cambio rápido de movimientos. (Figura 3-20)

Figura 3-20 Se comprueban los movimientos en todas las direcciones, así como el funcionamiento de cambio de ejes evaluando cualitativamente la vibración debida al cambio rápido de movimientos

Para las pruebas de movimiento, primero se hicieron pruebas estáticas de flexión, con el propósito de ver la resistencia a la flexión de la estructura. Se realizaron movimientos bruscos encima de la silla, se probó la resistencia que tienen los perfiles donde reposan los pedales y se ejerció fuerza en el extremo de manera exagerada, para asegurar que soporta cualquier peso (dentro del rango establecido). (Figura 3-21)

Figura 3-21 Se realizaron movimientos bruscos encima de la silla, se probó la resistencia que tienen los perfiles donde reposan los pedales y se aplicó una fuerza en el extremo de manera exagerada

3.9. Análisis de la estructura

El análisis estructural se realizó considerando los siguientes parámetros para el perfil de aluminio estructural.

- **Material:** Aluminio de alta resistencia.
- **Dimensiones:** Perfiles de 40 mm.
- **Módulo de Elasticidad (E):** 69 GPa (69,000 MPa).
- **Límite de Fluencia:** 240 MPa.
- **Densidad:** 2,700 kg/m³.

Se realizó un análisis estático, utilizando herramientas de elementos finitos (FEA). Este método permite modelar la estructura y aplicar las cargas previstas para evaluar su comportamiento bajo condiciones reales de operación. Durante el análisis, se consideró el peso de una persona de 100 kg, incluyendo el peso de la estructura misma de 25 kg, la aceleración y desaceleración del robot, así como las posibles fuerzas laterales y de torsión que podrían ocurrir durante el movimiento. (Figura 3-22)

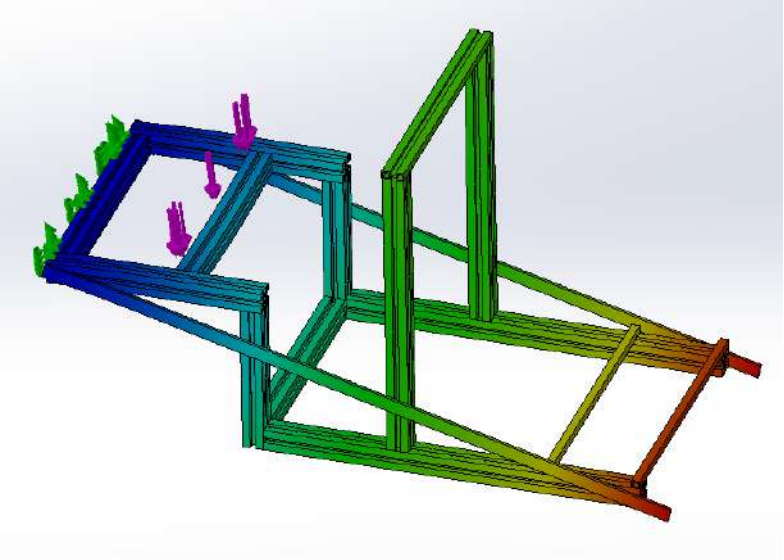


Figura 3-22 La simulación por elementos finitos (FEA), permite modelar la estructura y aplicar las cargas previstas para evaluar su comportamiento bajo condiciones reales de operación

Utilizando un software de modelado 3D, se representó la estructura conforme a las medidas del modelo físico. Se considera el perfil estructural, pero la estructura se simplifica para poder analizarlo estáticamente. Se ejecuta la simulación por elementos finitos con dos softwares distintos, los cuales nos dan resultados similares, con variaciones de décimas de milímetros, con referencia a la deformación.

Para las condiciones frontera se colocaron restricciones fijas donde se ancla la estructura a la silla y posteriormente al robot. Se le coloca la carga y se realiza el análisis.

3.10. Programación por puntos

Se hicieron dos pruebas de programación por puntos, la primera, es seguir una secuencia de cuatro puntos, tomando de referencia una estructura rectangular, marcando los puntos por las esquinas.

El robot se mueve desde el punto 1 al 2 y así sucesivamente hasta llegar nuevamente al punto 1 (Figura 3-23). Para esto, lo que se hace es programar cada punto manualmente, primero se mueve el robot al punto donde queremos iniciar (punto 1) se le da un tipo de movimiento y se guarda el punto. Se hace esto para los siguientes puntos y se corre el programa. (Véase Anexo 3)

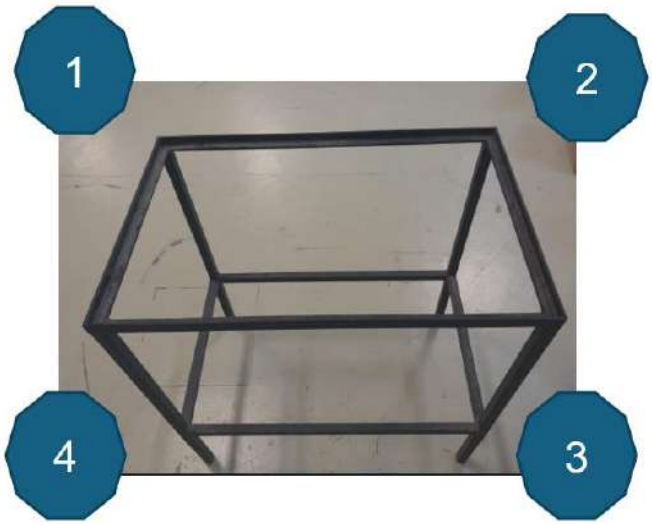


Figura 3-23 Representación de los puntos del recorrido de la prueba uno de programación

La segunda prueba consiste en simular el movimiento de una montaña rusa. Para esta prueba se recrearon los movimientos, con pocos puntos (Ver Anexo 4). Más adelante se representará un recorrido completo.

El programa básicamente mueve el robot a una serie de puntos definidos en el espacio, programados mediante el software *RoboDK.* Cada punto tiene su respectiva velocidad y orientación para recrear el recorrido de la montaña rusa. De

un fragmento de video de una montaña rusa se identifican los puntos de cambio de movimiento, esto para poder dar una dirección para el siguiente movimiento.

Igualmente se mide el lapso que pasa de un movimiento a otro, para poder estimar la velocidad que se requiere y la duración del movimiento.

3.11. Pruebas de movimiento. Montaña rusa

La finalidad de esta prueba es evaluar la capacidad del robot para poder realizar movimientos fluidos. Para esto se establece una serie de ejercicios basados en los movimientos de una montaña rusa. Esto con el propósito de observar los movimientos que realiza el robot, cuáles son los límites de los giros, la velocidad máxima a la que puede moverse, además de cómo se comporta el robot al realizar varios movimientos en un corto periodo de tiempo.

Se consiguió un video que simula una montaña rusa, este video debe de ser específico para visores de realidad virtual (ver Anexo 5). Se ubicaron todos los cambios de movimiento que hay en el video, y se especifica el tiempo en el que transcurre ese cambio y el movimiento específico que se realiza.

Considerando los ejes de movimientos definidos en la Figura 7-24, se recabó la información necesaria de cada movimiento, para el trayecto completo.

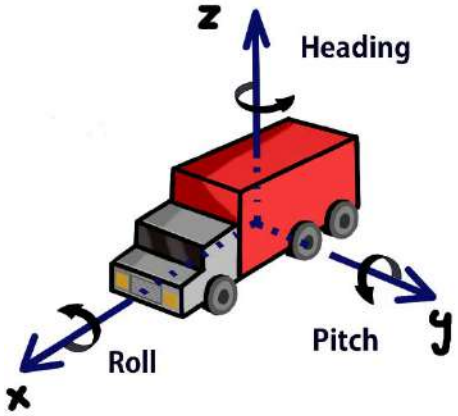


Figura 3-24 Representación de los movimientos traslacionales y rotacionales

A continuación en la Tabla 2, se listan los movimientos definidos y sus tiempos correspondientes

Tabla 2 Lista de movimientos realizados y sus tiempos

Tiempo	Movimiento	Tiempo	Movimiento
00:08	Movimiento en el eje y negativo mientras se realiza un *heding* de 20°	02:21	Subida inclinada a la izquierda
00:14	Se mantiene con la posición anterior	02:34	*Pitch* y r*oll de* 0°
00:17	Se realiza un p*itch* negativo de 45°	02:37	*Pitch* positivo de 10° con roll positivo 5°
01:31	Mantiene la posición	02:39	*Pitch* positivo de 15° con roll negativo de 10°
01:34	Regresa el p*itch* a 0°	02:42	*Pitch* negativo de 15° con roll negativo de 10°
01:37	Se realiza un p*itch* positivo de 45°	02:48	*Pitch* y r*oll* de 0°
01:39	*Roll* positivo de 15°	02:49	*Pitch* positivo de 25°
01:04	*Pitch* negativo de 30°	02:05	Desplazamiento en Z positivo
01:42	*Pitch* negativo de 30° con roll negativo de 10°	02:51	Desplazamiento en Z negativo
01:46	*Pitch* 0° *rol*l negativo de 10°	02:52	Desplazamiento en Z positivo
01:49	*Roll* 0° y *pitch* positivo de 25°	02:55	Desplazamiento en Z negativo
01:51	*Pitch* positivo de 25°	02:54	Desplazamiento en Z positivo
01:58	*Pitch* positivo 10° con *roll* positivo de 5°	02:55	Desplazamiento en el eje y positivo
02:01	*Pitch* negativo de 10° con *roll* positivo de 25°	02:57	*Pitch* positivo de 25°
02:08	*Pitch* negativo de 30°	02:59	*Pitch* negativo de 15° con roll positivo de 10°
02:01	*Pitch* negativo de 25°	03:05	*Pitch* y *Roll* de 0°
02:12	*Pitch* de 0° y r*oll* de 0°	03:09	*Pitch* negativo 15° con roll positivo de 10°

02:15	*Pitch* negativo de 30°	03:11	*Pitch* negativo 25° con roll negativo de 10°
02:16	*Pitch* positivo de 15° con roll positivo de 10°	03:02	*Pitch* y roll de 0°
02:17	*Pitch* 0° y r*oll* de 0°	03:24	Desplazamiento negativo en el eje Y
02:19	*Roll* negativo de 10°		

Mediante un analisis de los movimientos expuestos en el video, se elige el ángulo y la velocidad con la que se realizará cada uno de los movimientos. (Figura 3-25)

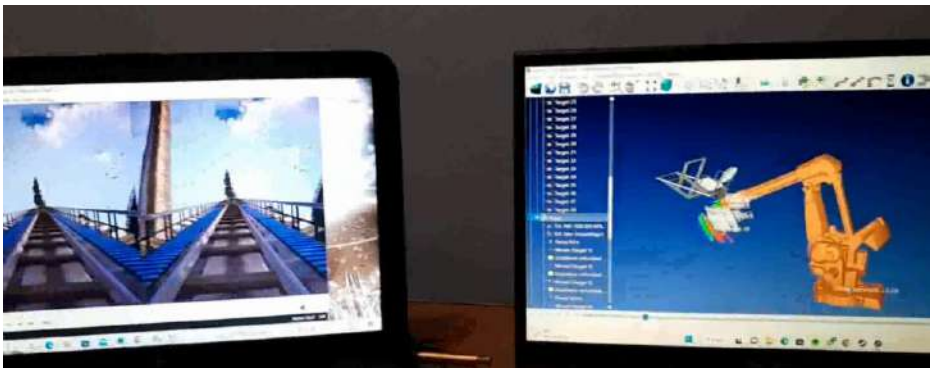


Figura 3-25 Sincronización de la programación punto a punto con los movimientos en el video que se utiliza.

Las velocidades de los movimientos del robot se determinaron con base en el tiempo de duración de los movimientos en el video. Los valores estimados se ajustan con referencia a la secuencia del video, considerando además que mover el robot a velocidades muy altas y con cambios repentinos es muy peligroso para las personas. En general se optó por disminuir las velocidades, son considerados siempre los movimientos del video.

El uso de la plataforma de simulacion requiere usar visores con lentes especiales inmersivos en los que se coloca un celular para proyectar el video de los movimientos a simular. La computadora y el celular se conectan via remota para proyectar simultáneamente el video, a su vez la computadora manda una señal al

controlador del robot *ABB*, para sincronizar el inicio del video y la secuencia programada de los movimientos en el robot.

Para esto se utiliza un programa en *Python* que manda la señal en el momento que se toca el teclado, este activa el programa del robot que solo espera la señal, mediante la entrada serial para poder activarse. Como se tiene un ligero retraso entre la pantalla que se utiliza en los visores y la computadora, se ajusta el tiempo en el que se manda la señal al robot. En el anexo 6 se incluyen los códigos en *Python* y en *RAPID* correspondientes.

3.12. Telemetría

El uso de telemetría tiene como objetivo que el videojuego proporcione en tiempo real, datos de movimiento del automóvil que se está simulando, para que dependiendo de donde se encuentre el automóvil en el videojuego, el robot se mueva a esa misma posición

En este punto fue necesario realizar una investigación y recopilación de videojuegos de carreras que cumplan con las especificaciones necesarias para utilizarlos en la simulación. Las características que se necesitan son: buenos gráficos, compatibilidad con el volante *Logitech G29*, que tenga la opción de configuración para acoplar los pedales, tener opción de telemetría, de manera preferente de código libre o que tenga una API para su uso.

Se consideraron los siguientes videojuegos que se ajustan a las especificaciones:

- *Torcs*
- *Life for Speed*
- *BeamNG.teach.*
- *Speed dreams*
- *Truck simulator*

- *Forza horizon*
- *Need for speed*

Comprobando las especificaciones, precio y facilidad de uso, se descartaron 4 videojuegos, reduciendo la lista a las siguientes opciones:

- *Truck simulator*
- *BeamNG.teach.*
- *Life for speed*

Se probaron los videojuegos para ver su compatibilidad con los pedales y el volante, los tres funcionaron realmente muy bien, en los tres videojuegos seleccionados; el embrague, freno y acelerador (de los pedales) funcionaron muy bien, los tres manejaban la función de configuración sin pedales o con pedales. En cuanto al volante se tenían muchas opciones para adaptar el estilo de botones que se quisiera utilizar, permitiendo adicionar estilos de manejo.

El programa *Life for speed* no se logró conectar de manera confiable con telemetría, por lo que se descartó, ya que se necesita esta conexión para la transferencia de datos. En cuanto al videojuego *Truck simulator*, se tuvieron problemas en la conexión del volante después de ejecutarlo durante periodos de tiempo prolongados.

Se decidió seleccionar como primera opción a *BeamNG.teach*, ya que cuenta con muchas herramientas para modificar todo el entorno, de acuerdo con su página oficial, *BeamNG* ofrece docenas de vehículos refinados y completamente personalizables para experimentar, permitiendo ajustar todas las partes móviles para crear casi cualquier experiencia de conducción deseada. ruedas, suspensión, motores y etc. BeamNG.teach (2023)

Este programa, cuenta con una licencia especialmente diseñada para estudiantes que estén haciendo alguna investigación que tenga que ver con el videojuego.

Mediante una solicitud directa a la compañía BeamNG, se obtuvo una licencia por un año para posteriormente obtener una licencia extendida.

Este programa tiene además una API de código abierto, con la cual es posible controlar cada vehículo que está dentro del juego También permite configurar su IA para hacer que un vehículo conduzca hasta un cierto punto de ruta, siguiendo a otro vehículo, cubriendo un mapa o siguiendo una trayectoria definida por el usuario. BeamNG.teach (2023)

CAPITULO 4

ESTRATEGIAS DE INTEGRACIÓN Y CONTROL

4.1. Opción 0. Integración mediante ethernet

Para realizar la conexión entre el software de simulación y el controlador del robot, se consideró instalar una tarjeta ethernet (Figura 4-1) en el controlador del robot ABB. y así poder conectar el robot con RoboDk directamente. RoboDk tiene un apartado para comunicarse con otro dispositivo por medio de una IP (Internet protocol), además de que tiene integrado el lenguaje *Python* para programar directamente los movimientos.

Figura 4-1 Tarjeta ethernet para el controlador ABB

En investigaciones y trabajos anteriores, se intentó utilizar esta tarjeta ethernet, sin éxito pues se requiere de actualizaciones del firmware del robot. La solución fue contactar a la compañía *RoboDK* para obtener un drive paracontroladores *ABB* modo serial, que pudiera conectar el programa *RoboDK* con el robot utilizando la conexión RS232.

Para esto se debe instalar el drive dentro del controlador *ABB,* utilizando una memoria USB y cargar el programa en el robot. Esto permite utilizar *RoboDK*, sin necesidad de la tarjeta ethernet de *ABB*.

En la Figura 4-2 se muestra un esquema de interacción entre los componentes del sistema de comunicación al utilizar la tarjeta ethernet, o el drive serial.

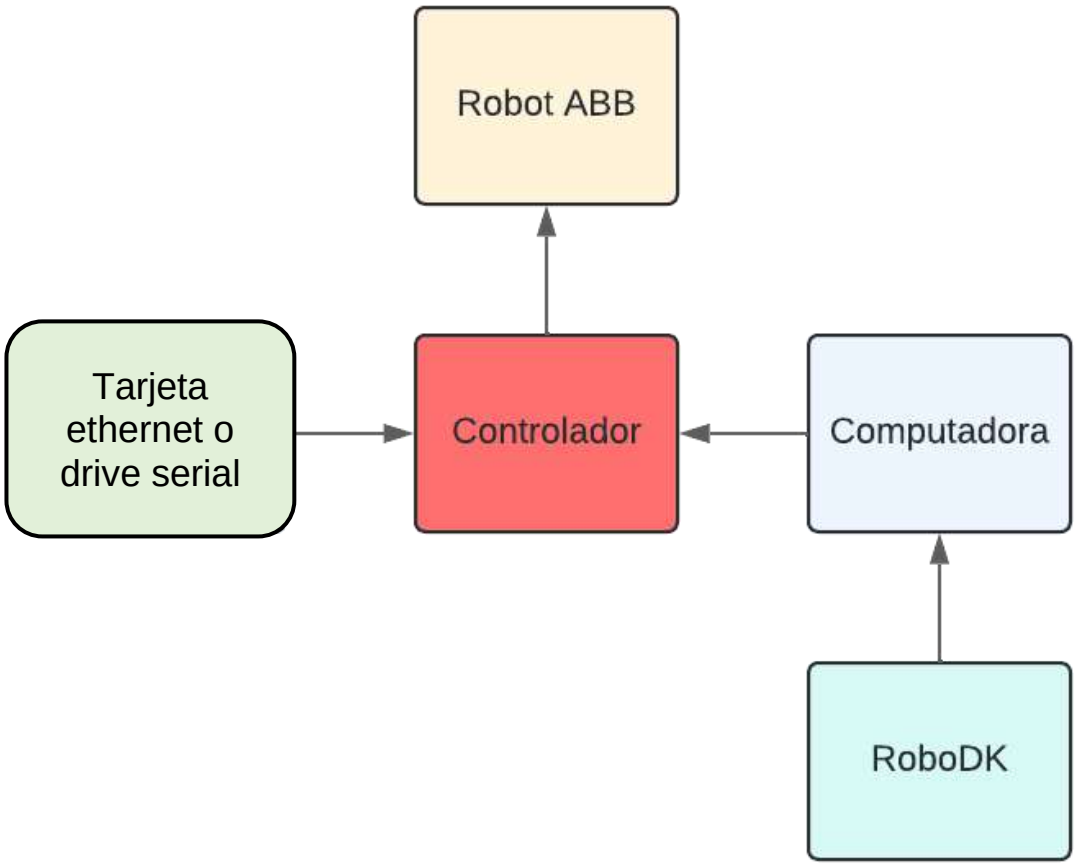


Figura 4-2 Estructura del sistema de comunicación software-controlador al utilizar la tarjeta o el driver serial.

Para poder mover el robot, se necesita vincular al sistema el control de *PS4*, el volante y los pedales *Logitec G29*, permitiendo que *RoboDK* se sincronice en tiempo real con el Robot *ABB*. Para este fin se realizaron dos programas.

Programa de verificación

El primer programa es para comprobar la operación de los botones del control de *PS4*, conocer si están dañados y si se pueden vincular de forma correcta a *Python*, así como mandar las señales correctas.

El programa tiene una ventana donde se muestra un rango de movimiento de los botones que se quieren utilizar, en este caso el botón X, el círculo, stick izquierdo, stick derecho y los botones R2 y L2. (Figura 4-3)

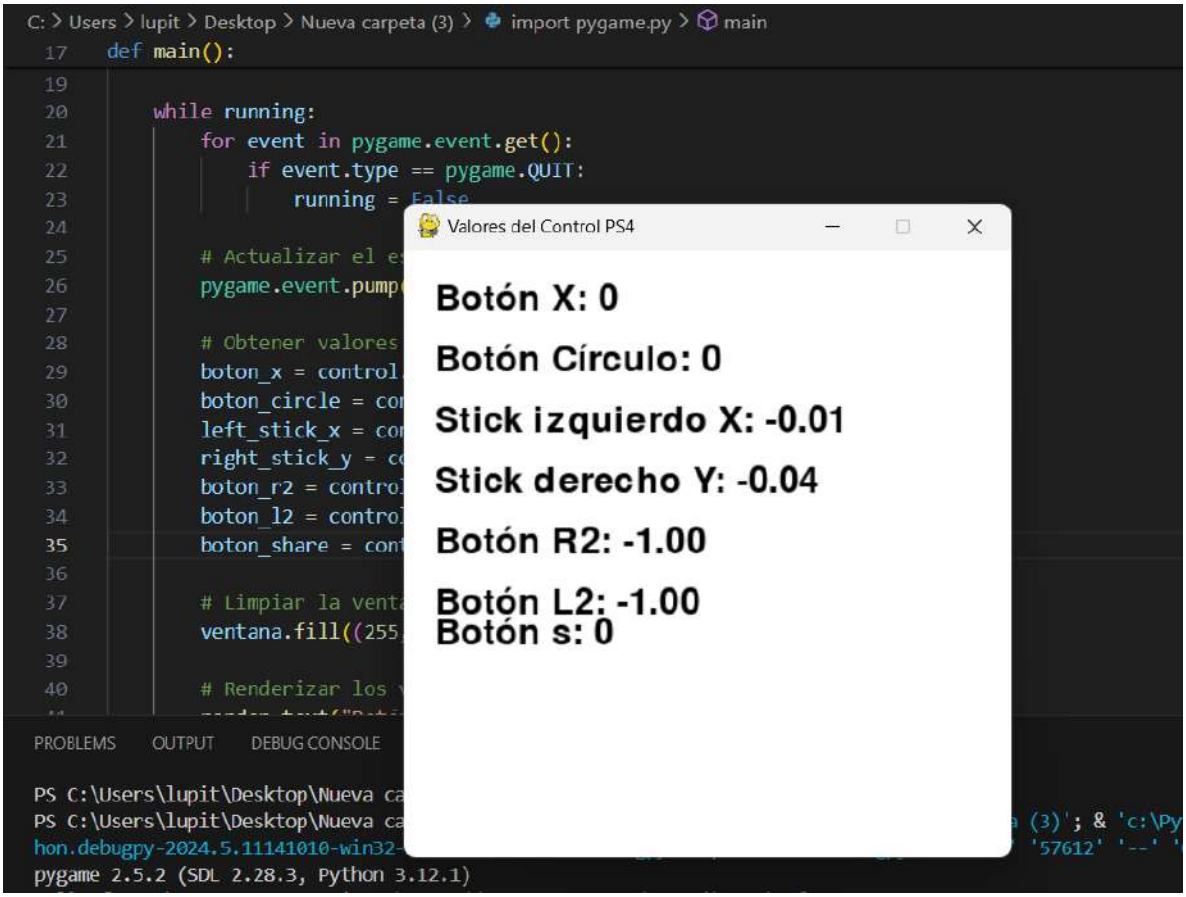


Figura 4-3 Ventana donde se muestran los rangos de valores de los controles que se quieren utilizar, en este caso el botón X, el círculo, stick izquierdo, stick derecho y botones R2 y L2

El código del programa se incluye en el anexo 7 de este trabajo.

En terminos generales el programa abre una ventana , donde se muestran en tiempo real los valores de los controles utilizados, sus valores se actualizan en tiempo real de acuerdo a las acciones realizadas.

Despues de inicializar la biblioteca *Pygame* y el *joystick*, se configuró de la fuente de texto y de la ventana. Se inicia un ciclo, donde se van actualizando en tiempo real dentro de la ventana los valores de cada botón programado. Esto se activa cada vez que se detecta una entrada de señal física del control de *PS4*.

Programa de movimientos

El objetivo de este programa es mover el robot de acuerdo a las acciones realizadas en el control de *PS4*, para esto primero se inicia *Pygame* para vincular el control y se establece la conexión con el programa *RoboDK* por medio de *Robolink*. (Ver Anexo 8)

Posteriormente se ejecuta un ciclo continuo que está recibiendo en tiempo real las señales del control, obteniendo los valores del *stick* izquierdo y los gatillos R2 y L2. Se calcula el desplazamiento en X con el valor del movimiento del *stick*, la rotación en el plano Rx con el gatillo R2 y la rotación en el plano Ry con el gatillo L2.

El programa calcula la nueva posición del robot usando las rotaciones y traslaciones mencionadas antes. El robot se mueve a la nueva posición calculada con una instrucción *Move L*. Al final, para salir del ciclo solo es necesario presionar Ctrl + C.

Figura 4-4 Movimiento del robot virtual guiado por medio del control de PS4

De esta manera se consigue que el control de *PS4* pueda ser vinculado al robot dentro de *RoboDK*. Se utilizó este mismo método para los pedales y el volante (Ver Anexo 9)

4.2.1 Consideraciones acerca de RoboDK

La programación en *Python* se realiza dentro del programa *RoboDK,* ya que este tiene un apartado especifico para vincular *Python* con *RoboDK. Aunque RoboDK* cuenta con su propia version de *Python* instalada dentro de *RoboDK*, esta no será de utilidad para lo que se necesita realizar, ya que no se le pueden instalar bibliotecas externas.

Debido a lo anterior fue necesario vincular un módulo *Python* externo (instalado de forma independiente) a *RoboDK* para poder instalar todas las bibliotecas neccesarias, como*: Pyserial, Pygame y Robolink*

Si la programación se realiza en un *Python* externo que no esté vinculado, este será mucho más difícil de programar en *RoboDK*. Se debe de recordar

que el *Python* vinculado, se debe abrir dentro de *RoboDK*, no de forma independiente.

Conexión con robot *FANUC*

Como ya fue mencionado, por cuestiones de desactualización de los equipos de los robots ABB, no se puede realizar la simulación en tiempo real utilizando conexión ethernet, por lo que, a manera de prototipo se utilizó un robot *FANUC* modelo LR Mate 200iC con controlador R-30iA Mate. Este es un robot pequeño de aproximadamente 1 metro de alcance, por lo que no se puede montar la estructura. En el diagrama de la Figura 4-5, se muestra cómo se relacionan cada uno de los componentes del sistema para acceder al movimiento del robot.

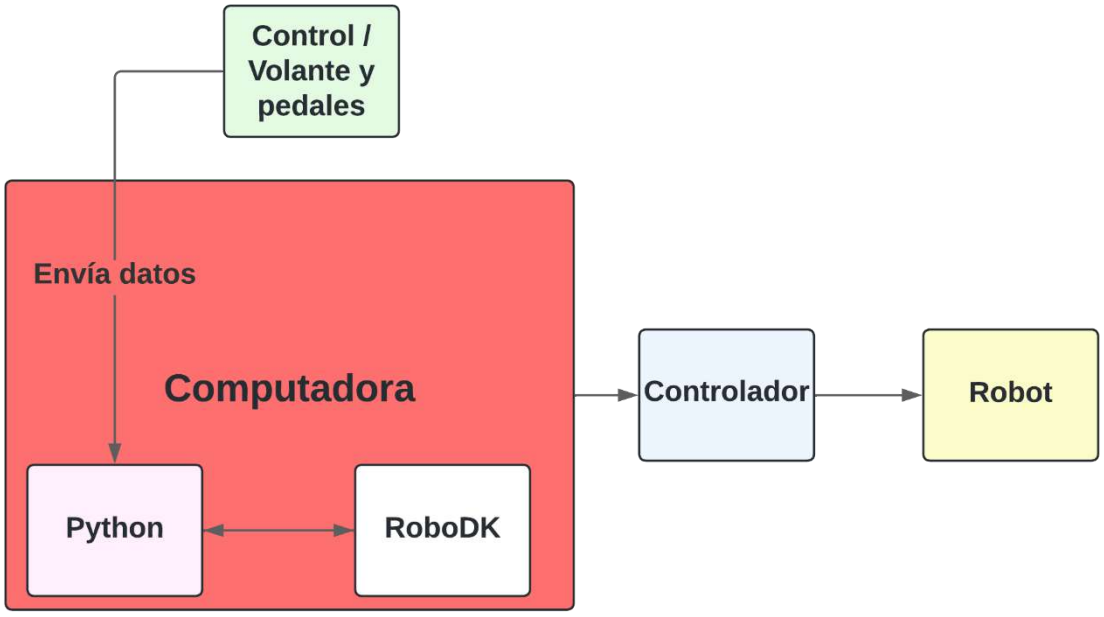


Figura 4-5 Estructura de las relaciones entre cada uno de los componentes para poder acceder al movimiento del robot.

De la figura puede verse que es necesario en la computadora tener instalado *Python* y *RoboDK*, tener conectado el control *PS4* por USB o *Bluetooth*, y conectar mediante ethernet el controlador del robot FANUC.

En *Python* se escribe el programa mencionado anteriormente (anexo 8), el cual calcula la nueva posición del robot usando las rotaciones y traslaciones enviadas por el control, y realizar el movimiento correspondiente mediante el comando *MoveL*.

El programa de *Python* está vinculado a la simulación de *RoboDK* como se muestra en la Figura 4-6, al correr este programa, no se debería tener problema para mover el robot *FANUC* dentro de la simulación

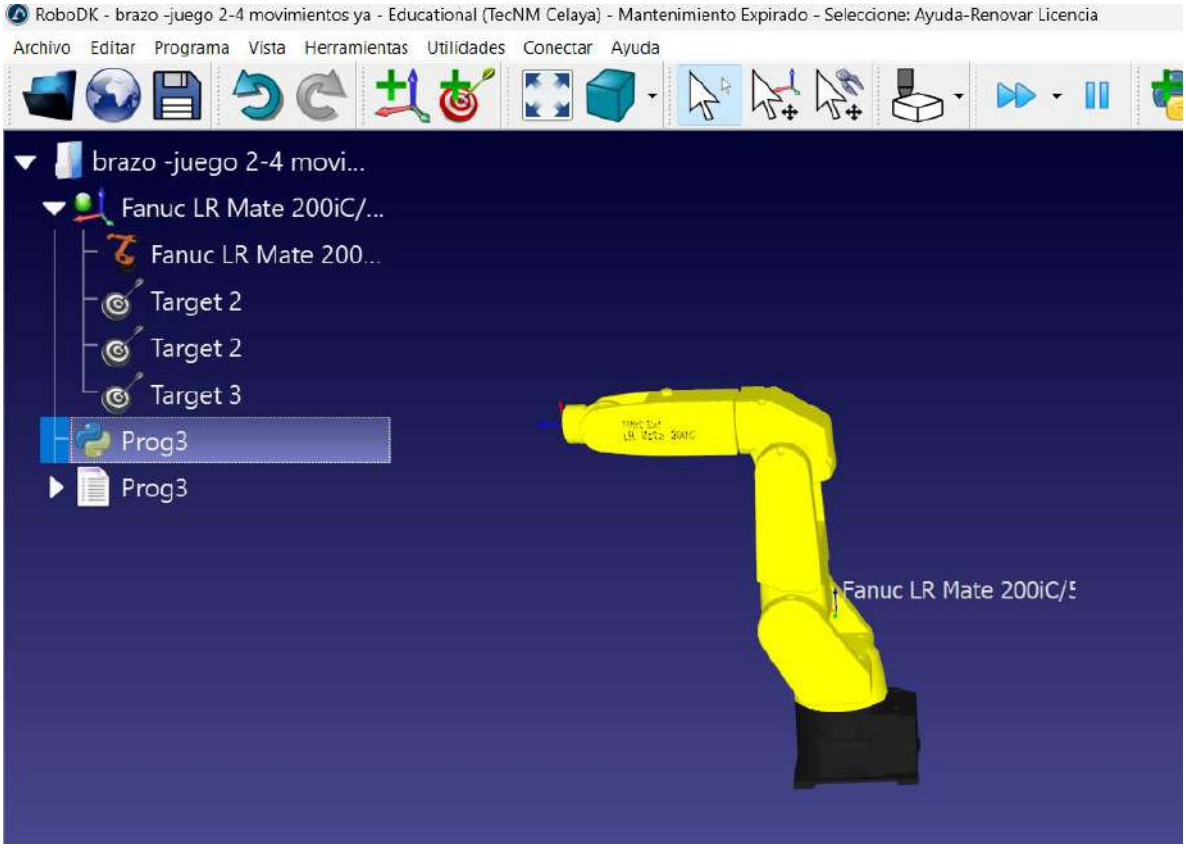


Figura 4-6 Programa de Python vinculado a la simulación de RoboDK . Al ejecutar este programa, no se debería tener problema para mover el robot FANUC dentro de la simulación.

Para mover el robot *FANUC* en la simulación y en físico, en tiempo real se debe de realizar una conexión ethernet entre la computadora y *el robot.*

Para la conexión entre computadora y el robot *FANUC,* se deben seguir los siguientes pasos. Ver anexo 10

- Tener los drivers que se introducirán en el robot FANUC (estos se introducen por una memoria USB, la cual debe de ser menor a 1 GB)
- Establecer una IP en la computadora
- Obtener la IP del robot
- Conectar *RoboDK* con el robot (se exlplica en anexo 10)
- Ejecutar un comando ping desde al computadora al robot
- Ejecutar los drivers y ejecutar un comando ping desde el *teach pendant* a la computadora

4.2. Opción 1. Integración mediante comunicación serial

Esta opción se desarrolló en dos partes, la que manda las señales y la que las recibe.

Primera parte

En esta parte se utilizó el lenguaje de programación *Python* de forma integradora, para conectar todos los elementos utilizados y enviar las señales necesarias al controlador del robot. El programa en *Python* se encarga de establecer la comunicación serial con el controlador del robot ABB, para después detectar las señales del volante o control de *PS4* y enviarlas al controlador en forma de bits.

Las librerías utilizadas para la realización de estas interconexiones son *Pyserial*, para establecer la conexión serial, y *Pygame*, para la conexión con el volante y el control de *PS4*.

En cuanto a la segunda parte, la programación se desarrolló en el lenguaje de programación del propio controlador, este es el lenguaje *RAPID*. El programa está compuesto por un conjunto de instrucciones que describen la actividad del robot.

Existen instrucciones específicas para los distintos comandos, por ejemplo, una para mover el robot, otra para seleccionar una salida, etc. Por lo general, las instrucciones cuentan con un conjunto de argumentos asociados que definen qué debe ocurrir con una instrucción concreta. Lutz, M. (2017)

En el diagrama de la Figura 4-7, se muestra cómo se relaciona cada uno de los componentes para poder llegar al movimiento del robot.

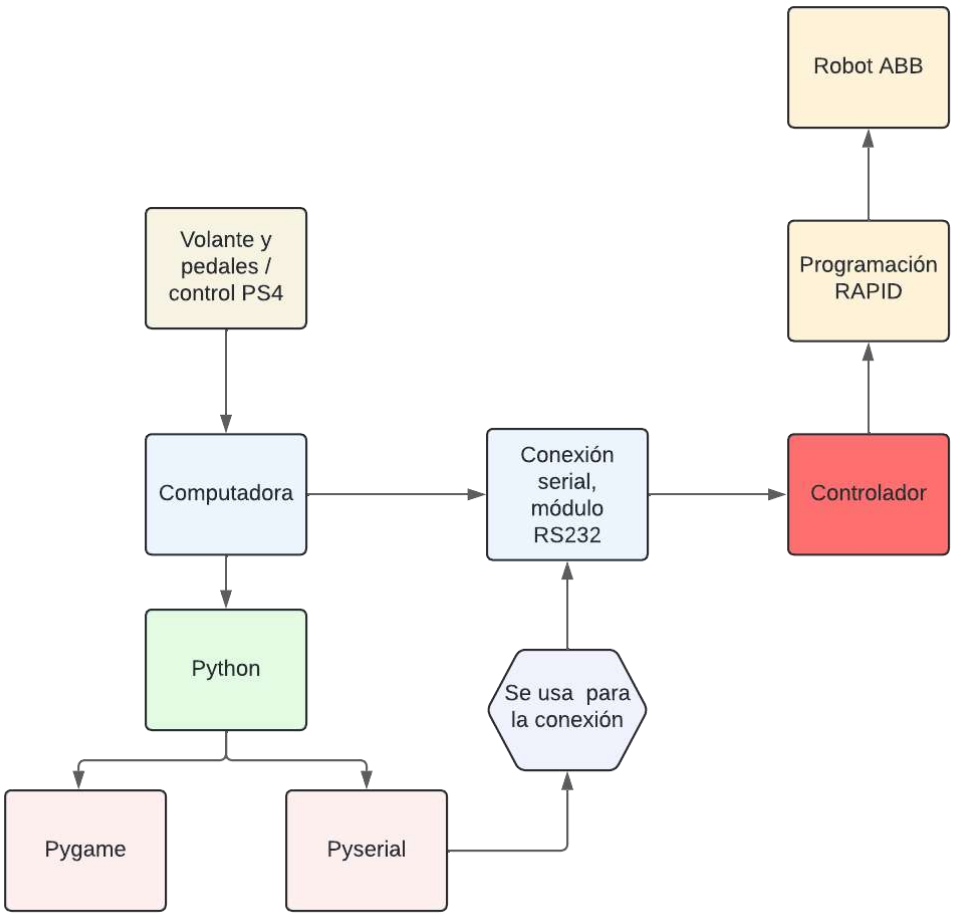


Figura 4-7 Relaciones entre cada uno de los componentes para poder generar el movimiento del robot.

El controlador ABB IRB 6400 cuenta con dos módulos seriales, protocolos RS232 y RS485. Se utilizó el puerto RS232, realizando la conexión entre la entrada de 12 pines del SIO 1 del controlador y un conector DB9 (Figura 4-8), para después con un adaptador USB-RS232 conectarse a la computadora.

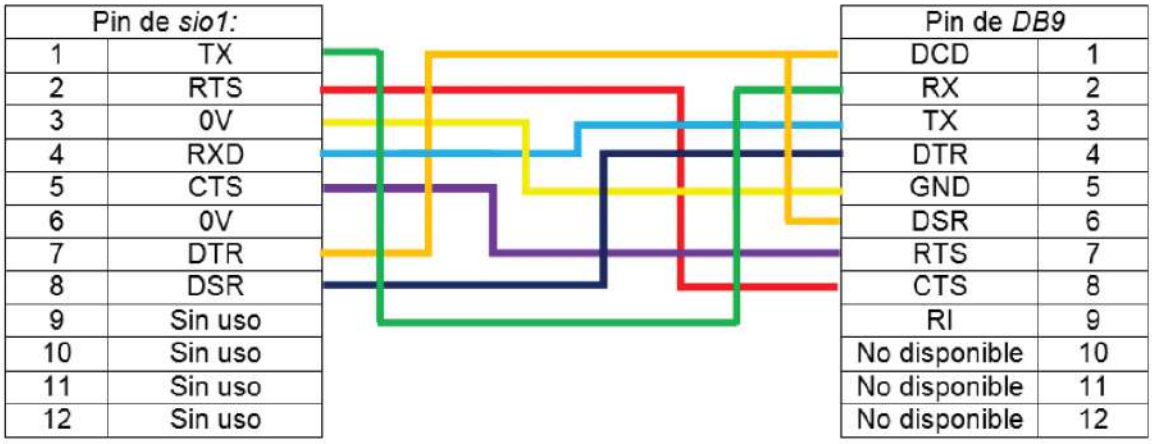


Figura 4-8 Diagrama de cable SIO1 a DB9. Carreón Villal, J. R. (2014)

Se utiliza como centro de operación la computadora, en ella se controla todo por medio del programa *Python*. Empleando las librerías mencionadas anteriormente.

Lo que hace *Python* es primero establecer una conexión serial con el controlador del robot, para después comprobar la conexión del control *PS4* o el volante.

La configuración del puerto serial presentado utiliza el puerto COM6, pero se debe considerar que este puede cambiar de acuerdo con los puertos disponibles en la computadora.

Se crea una interfaz gráfica tkinter para enviar y recibir datos. Los datos estarán en la misma ventana, donde tendrá un acceso de datos por el usuario, que serán los que se enviarán por la conexión al controlador. (ver Anexo 11)

Se debe tener en cuenta que los datos que se van a enviar deben formar una cadena de valores, con la siguiente estructura: número + * + letra (a, b o c). Primero, el número indica los grados que se va a mover un eje, el asterisco es un separador la letra marca el eje del robot a mover.

Programa en *RAPID*

Este programa inicia estableciendo la comunicación serial, lee la cadena de texto que se envía desde *Python* hasta encontrar un asterisco, convierte esto en un valor numérico y el carácter adicional que recibe después del asterisco es para identificar a qué articulación del robot se va a aplicar el valor.

El caracter adicional consta de tres posibles opciones; a, b o c correspondientes a las articulaciones rax1, rax2 o rax5. Finalmente, se cierra la comunicación y mueve el robot a la posición que se le asignó. (ver Anexo 12)

4.2.1. Conexión del control *PS4*

En esta parte lo que se pretende es integrar un control de *PS4*, para enviar los datos, en lugar de estar introduciendo manualmente los números.

4.2.2. Programación en *Python*

En *Python* se programan las opciones referentes a la activación de botones o joystick. De acuerdo con el tipo de botón pulsado, se manda una señal por vía serial que es recibido por el controlador y este al programa en *RAPID*. (Figura 4-9)

Se comienza llamando las librerías, inicializando *Pygame* y la comunicación serial. Se configura una ventana, donde se mostrarán los valores que se enviarán al

controlador. Esta ventana se integra para saber qué es lo que se está mandando, aunque se puede omitir, ya que también la terminal lo muestra.

Se inicializa el controlador del control de *PS4* y un ciclo, en donde si se presionan los botones X, circulo o R2, se envían caracteres específicos a través del puerto serial (1*a, 1 *b y 1*c) respectivamente, esto se muestra en una ventana para comprobar lo que se está enviando. (ver Anexo 13)

Se agregó un retraso de 0.5 segundos para esperar el envío de datos.

Figura 4-9 Este programa se encuentra en el anexo 13, lo que se muestra en la imagen es la ventana resultante de dicho programa, donde se representa lo que se está enviando al robot de forma serial por medio del lenguaje de programación Python.

Se agregaron condiciones para enviar los datos, se verifica el estado del botón correspondiente y el intervalo de tiempo transcurrido desde el último envío de datos, esto para evitar enviar datos con demasiada frecuencia y que no se sature el puerto serial.

Se agregó el manejo de la tecla r, como medida de seguridad, cuando se detecta la tecla r se envía un comando específico, que termina el programa *RAPID*. (ver Anexo 14)

4.2.3. Programación en *RAPID*

El programa en el lenguaje *RAPID* toma las señales que *Python* manda de forma serial y las convierte en caracteres. De acuerdo con que tipo de señal es recibida actúa de forma diferente para mover los ejes del robot ABB.

Dentro de una rutina *while* se establecen condiciones de acuerdo con el botón que sea oprimido en el control o, si se ha movido el volante o apretado el acelerador o el freno. Dependiendo de la señal que reciba, se manda una señal diferente al controlador del robot. Como se explica en el diagrama de la Figura 4-10.

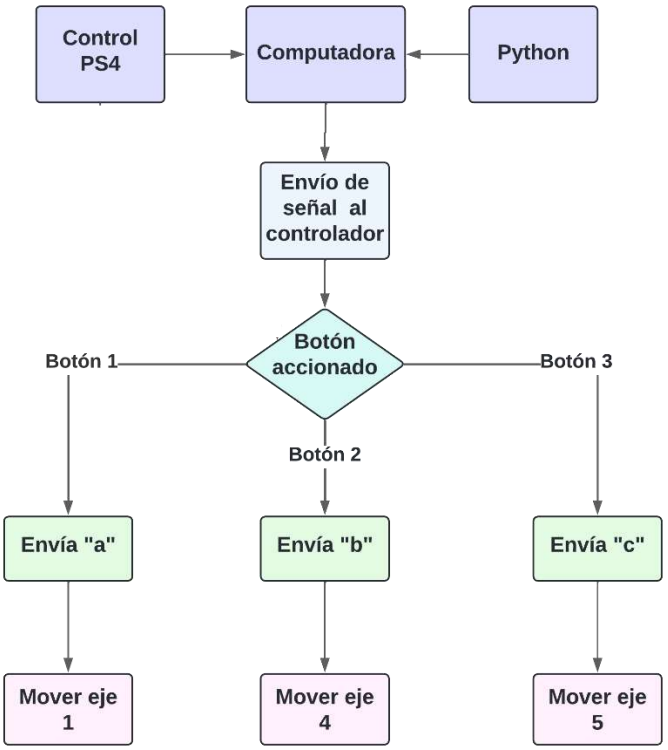


Figura 4-10 Diagrama de flujo para representación del programa de control de movimientos mediante control PS4.

Originalmente el programa estaba limitado a hacer movimientos rápidos. Si se estaba realizando un movimiento y se oprimía otro botón para cambiar de movimiento, el nuevo movimiento no se realizaba de forma inmediata, ya que se

necesitaba terminar el movimiento anterior para poder realizar en siguiente. (ver Anexo 15)

Para esto se modificó el código, reduciendo la distancia recorrida se redujo a movimientos muy pequeños, lo que permitió cambiar de dirección en cualquier momento. Para esto se incluyó un ciclo que esté recibiendo información de la conexión constantemente y detecte el cambio de dirección. (Ver Anexo 16)

4.3. Opcion 2 Conexión microcontrolador *Arduino*

Para aumentar la velocidad de reacción del robot, se hizo uso de un controlador *Arduino UNO* que registra las entradas y manda una señal al robot en el momento de un cambio de movimiento.

Fue necesario construir un circuito que mande el voltaje correspondiente al controlador, cada vez que se oprima un botón.

Figura 4-11 Circuito utilizando un controlador Arduino UNO, para enviar señales de voltaje al controlador del robot, se utilizaron 3 relevadores y 3 resistencias de 1000 ohms

Circuito

El circuito consiste en utilizar, tres relevadores mediante tres botones pulsadores, simulando los botones del control de PS4 y un microcontrolador. Se debe tener en cuenta que se utilizaron los leds para simular las entradas del controlador y una fuente 5VCD para simular la entrada de voltaje que genera el controlador. En la Figura 4-12 se muestra el circuito correspondiente.

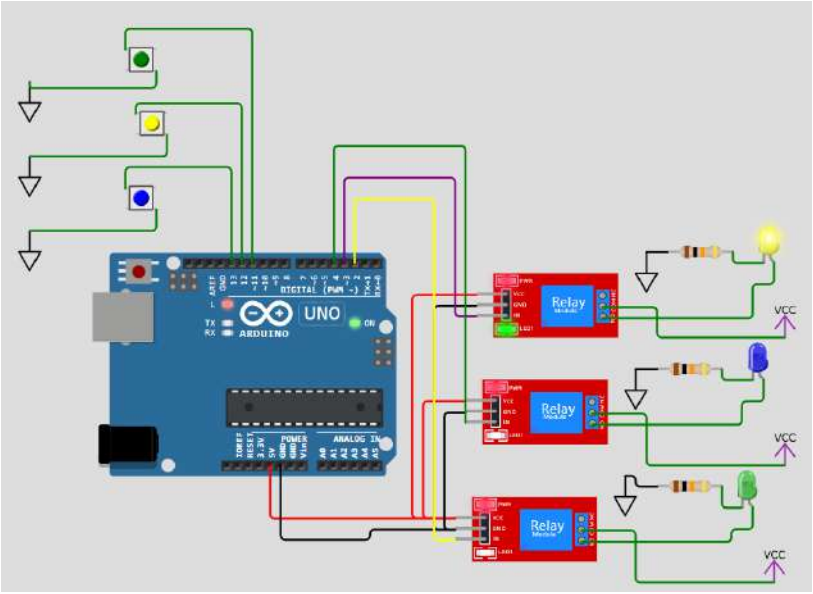


Figura 4-12 Representación del circuito. Se utilizan tres relés mediante tres botones pulsadores, simulando los botones del control de PS4 y un microcontrolador

Programación

En esta opción de comunicación, también se desarrollaron tres programas. Uno para la *PC*, escrito en *Python*, otro para el controlador Arduino, en su lenguaje propio y el tercero en el lenguaje *RAPID*, para el controlador del robot.

La computadora se comunica con el control *PS4* haciendo uso de la biblioteca *Pygame*, para recibir información relativa a las acciones realizadas por el usuario sobre el control. La computadora procesa esta información y genera los comandos

correspondientes que envía al puerto serial al Arduino, haciendo uso de la biblioteca *Pyserial.*

Finalmente, el *Arduino* se encarga de enviar la información correspondiente al controlador del robot (Anexo 18)

Antes de esto se debe cargar un programa a *Arduino* que detecte la señal que *la computadora* manda, en este caso la señal del botón elegido. Para esto, se utiliza el propio software de *Arduino*, dependiendo del botón que se toque esta manda una señal de entrada a los diferentes pines que tiene el robot. (Ver Anexo 17)

Cuando el robot está realizando el primer movimiento con la programación en *RAPID* y se pulsa cualquier otro botón del control de *PS4*, se detecta inmediatamente la señal, y deja de ejecutarse el movimiento actual para realizar el nuevo movimiento programado en esa señal de interrupción. Este proceso continuo de forma secuencial, con una serie de interrupciones que generan movimientos siguientes (Figura 4-13).

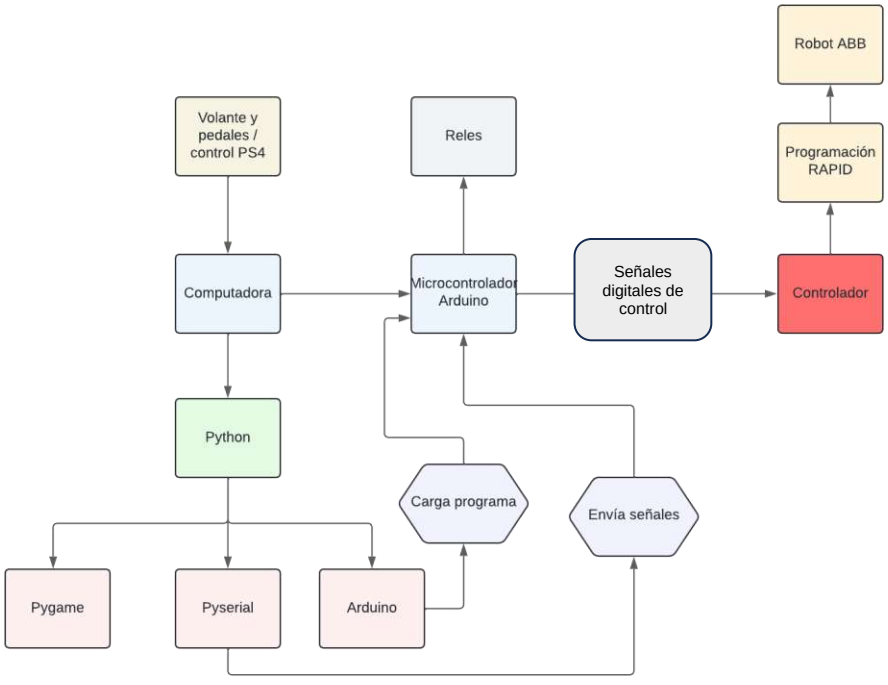


Figura 4-13 Estructura del sistema desarrollado para implementar la opción de integración

El programa en lenguaje *RAPID*, desarrollado con auxilio del *teach pendant* del controlador, primero genera la interrupción correspondiente a las señales de entrada y selecciona los comandos a la interrupción

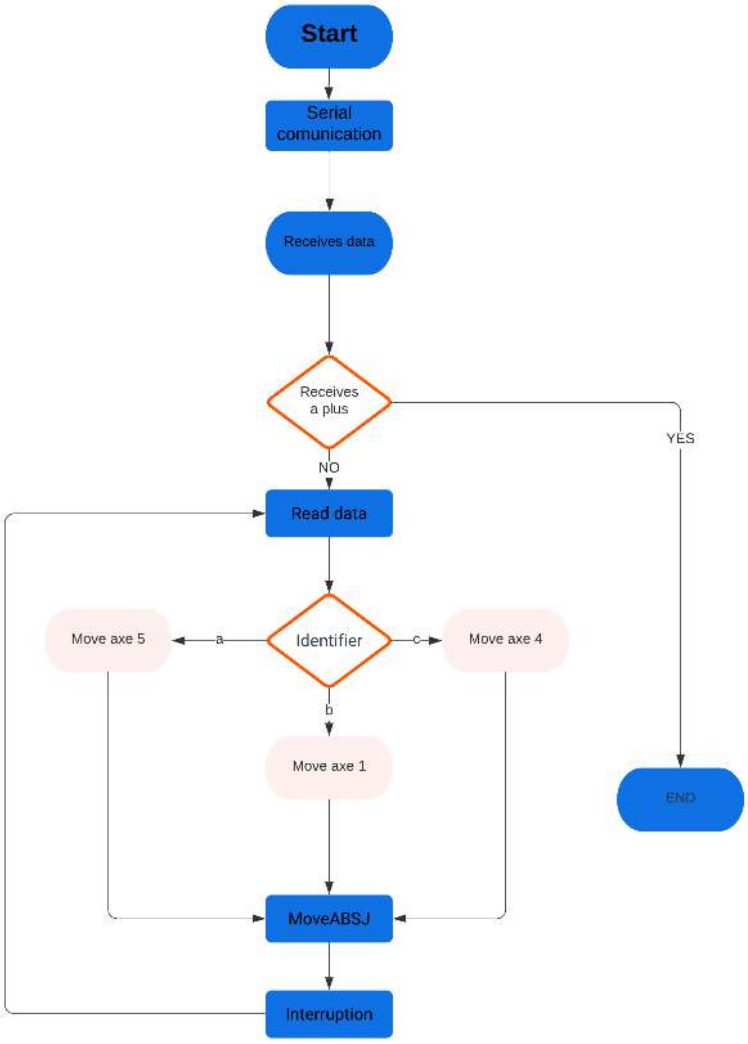


Figura 4-14 Diagrama de flujo para representación del programa en *RAPID*

4.4. Opción 3. Interrupciones

En el lenguaje *RAPID* del robot, las interrupciones funcionan como una regla fuera del programa original que siempre se debe de cumplir, sin importar donde se encuentre el robot.

Por ejemplo, cuando el robot está en un ciclo, pero se requiere que cada cierto tiempo se cambie la herramienta, se programa una interrupción, activada por una señal de entrada digital, generada por ejemplo mediante un botón externo.

De esta manera, al momento de presionar el botón, se genera la interrupción para que el robot mediante otra función externa se salga de su circuito a una estación, se le cambie la herramienta y vuelva a regresar al circuito.

Con base en esto, se planeó utilizar las interrupciones para el cambio rápido de movimientos. Se hizo uso de la estructura desarrollada en la opción 2. Mediante relevadores se generan las señales. Cada relevador corresponde a un botón del control de *PS4.*

Para cada acción realizada en el control *PS4*, se programa con un movimiento de un eje diferente, por lo que se le asigna a cada uno una interrupción. Lo primero que se hizo fue probar si funcionaba con 2 movimientos en secuencia.

El programa comienza con un ciclo de movimientos lineales entre dos puntos A y B, cuando se activa una señal de entrada, se dirige el movimiento hacia un punto C y se sale del programa. (Ver Anexo 24)

4.5. Opción 4. Telemetría

En esta opción se parte de la base de la opción 1 (conexión serial). La actualización al plan anterior consta de extraer los datos de posición directamente del videojuego y convertirlos en movimiento para el robot.

Explicando más a detalle lo anterior, se tiene como núcleo la computadora, donde se tiene el videojuego en ejecución, al mismo tiempo se está corriendo un programa

de extracción de datos (telemetría) para poder recabar los datos que el videojuego nos está produciendo, Figura 4-15 (ver Anexo 19).

La información de interés es la posición actual del automóvil dentro del videojuego, así como su velocidad. Por el momento solo se busca la extracción de lo que es la velocidad, pero fácilmente se pueden extraer otros datos.

Estos datos se procesan mediante un programa en *Python* y se mandan por conexión serial al controlador del robot, donde por medio de un programa en *RAPID*, entran como una variable tipo *char*, se convierte a *string* y posteriormente a un valor numérico para poder leer el dato en su forma original. Por último, este dato se da como coordenada de un movimiento establecido en el robot, se guarda la posición anterior y se introduce el nuevo dato de velocidad.

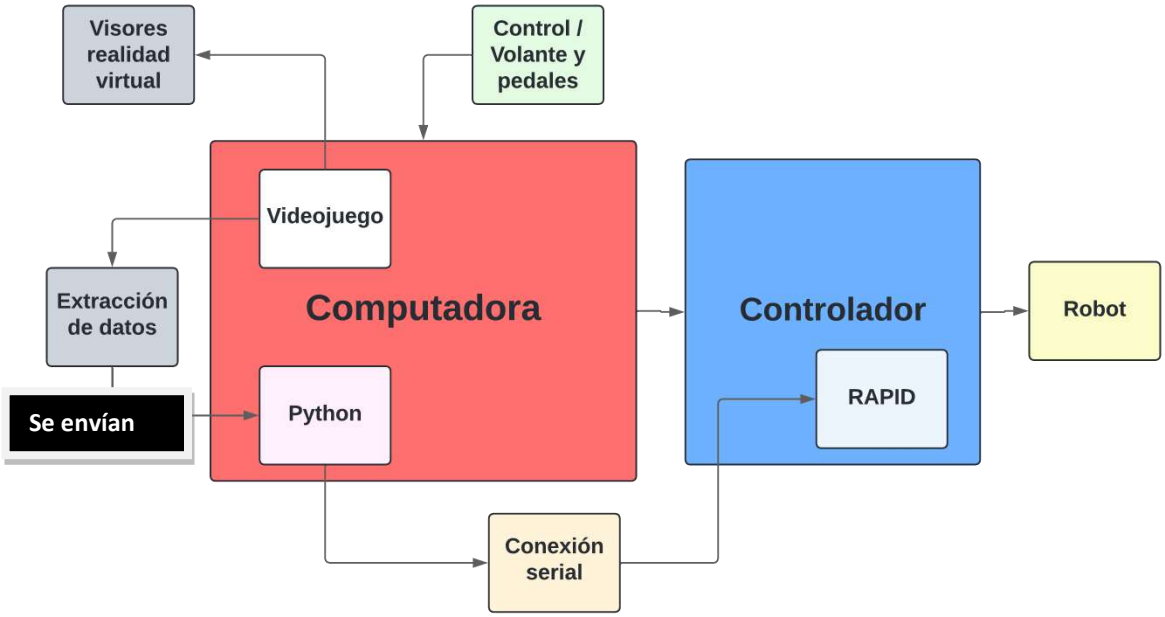


Figura 4-15 Esta estructura implementada la opción 4, se tiene como núcleo la computadora, donde se tiene el videojuego en ejecución, al mismo tiempo se está corriendo un programa de extracción de datos (telemetría) para recabar los datos que el videojuego nos está produciendo

Entre mayor velocidad tenga el vehículo, el eje 3 y el 5 se mueven de forma positiva, si estos disminuyen también lo hace la posición del robot. Los ejes se mantienen

limitados tanto de forma positiva como de forma negativa. Si el vehículo virtual alcanza una velocidad muy alta y los ejes no puedan alcanzarla, estos se queden en su posición máxima.

Lo primero que se debe de hacer es abrir el videojuego *EUROTRUCK* 2, para poder sincronizar el *API* de telemetría. Se abre la *API* de telemetría y se sincroniza con el videojuego (Ver Anexo 19). Como resultado de una buena sincronización, la *API* arrojará una página de valores que se obtienen del juego, estos valores son los que serán necesarios para el movimiento del robot.

Figura 4-16 Pantalla del programa EUROTUCK 2, se pueden observar los buenos los gráficos del juego

Se prosigue ejecutando en la *PC* en *Python*, Este código primero establece la comunicación serial con el controlador y se inicia un ciclo donde accesa mediante un comando *GET* la *URL*, que es la de los datos de la *API REST*. Esta *URL* proporciona datos tipo *JSON* para obtener información específica.

Al acceder a los datos se escogen los necesarios para el movimiento, en este caso los movimientos de posición x,y,z y los de orientación, heading, pitch y roll. Se imprimen los valores y se envían por conexión serial. Es muy importante antes de

mandar los datos, convertirlos en cadena de caracteres para asegurar la correcta transmisión de datos.

Después de todo esto, se debe de subir al controlador un programa que reciba los datos que se mandan desde el programa en *Python*, para esto por medio de la *USB*, se envía un programa que recibe estos datos.

Figura 4-17 Carga al controlador de un programa que reciba los datos que se mandan de Python, mediante una USB

Dentro del programa en *RAPID*, se comenzó declarando todas las variables, se programó una función para mover al robot dependiendo del dato que se manda por la conexión serial. Cabe destacar, que se utiliza el comando *Offs*, el cual se utiliza para mover los ejes.

Se inicia un ciclo donde se abre la conexión serial, se lee un *byte* y se convierte a cadena de caracteres, se prosigue convirtiendo esta cadena a un valor numérico y se envía a la función que se menciona anteriormente para mover al robot a posicione específicas.

Después de tener todo en su estado inicial, es decir, el videojuego iniciado, al igual que telemetría y conectado el control de *PS4*, se inicia la ejecución del programa mediante el *teach pendant* del robot *y se* ejecuta el programa en *Python.* Se debe tener en cuenta, que es recomendable ejecutar primero el programa en *RAPID* y colocar la velocidad mínima dentro del *teach pendant*, después de esto ejecutar el programa en Python.

Lo que hace el robot es moverse dependiendo de la velocidad que se tiene, a mayor velocidad el eje 5 se mueve hacia arriba y a menor velocidad este se mueve hacia abajo.

Figura 4-18 Sincronización de PC-Robot

CAPITULO 5

RESULTADOS Y TRABAJOS FUTUROS

5.1. Resultados

5.1.1. Análisis de la estructura

De los resultados del análisis con elementos finitos para la estructura, se obtuvo un factor de seguridad de 4.8, que indica que la estructura puede soportar hasta 4.8 veces la carga máxima esperada antes de llegar al punto de falla. Este margen adicional es crucial para garantizar la seguridad del usuario y compensar cualquier variabilidad en los materiales o condiciones de operación no previstas. (Figura 5-2)

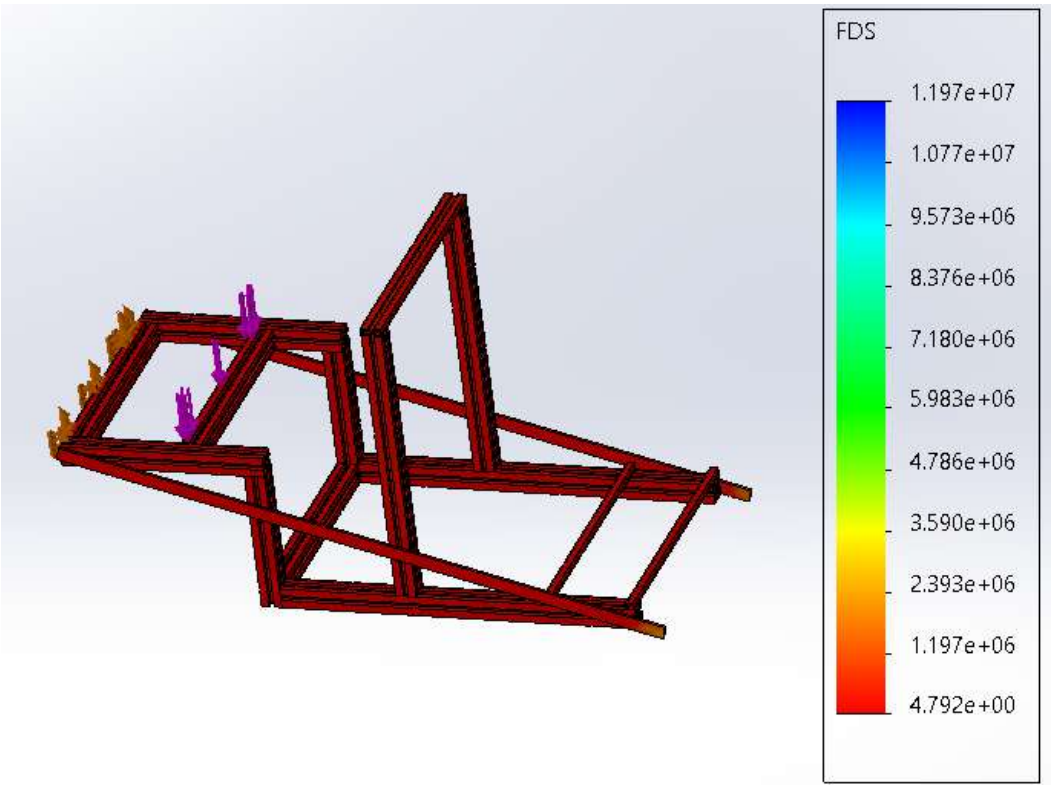


Figura 5-1 El análisis de la estructura reporta un factor de seguridad de 4.8, que indica que la estructura puede soportar hasta 4.8 veces la carga máxima esperada antes de llegar al punto de falla

Se obtuvo, igualmente una deformación máxima de 2.8 mm, que es lo suficientemente baja para asegurar que la estructura mantenga su forma y función sin afectar la comodidad del usuario ni la precisión del robot. (Figura 5-3)

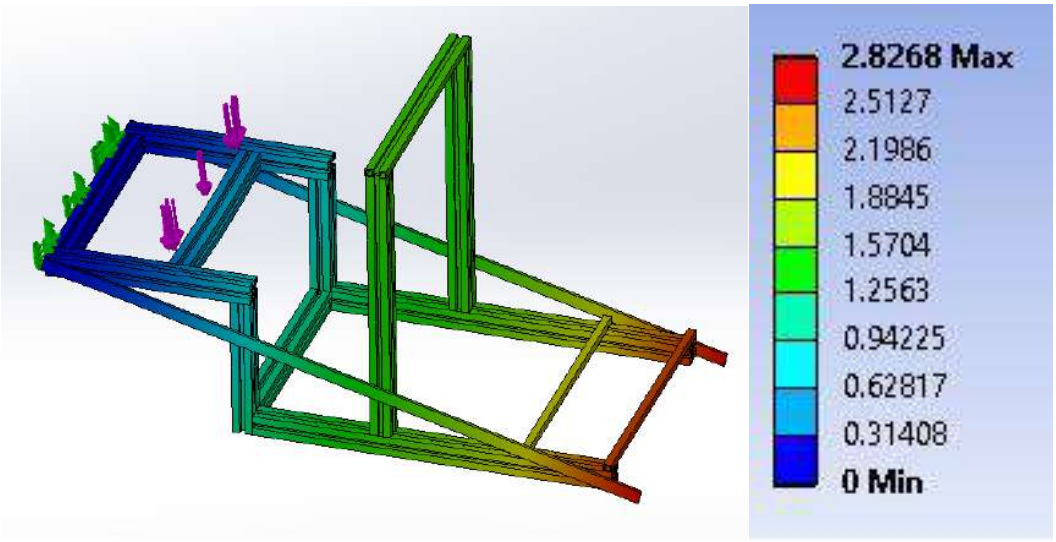


Figura 5-2 La deformación máxima de 2.8 mm, es lo suficientemente baja para asegurar que la estructura mantenga su forma y función sin afectar la comodidad del usuario ni la precisión del robot

El esfuerzo máximo de Von Mises reportado de 4,740,000 N/m² se encuentra dentro de los límites permisibles para perfiles de 40 mm de aluminio de alta resistencia. Este material fue elegido por su excelente relación peso-resistencia, lo que minimiza la masa añadida al robot sin comprometer la resistencia estructural. El uso de estos perfiles también facilita el ensamblaje y la personalización de la estructura, permitiendo adaptaciones.

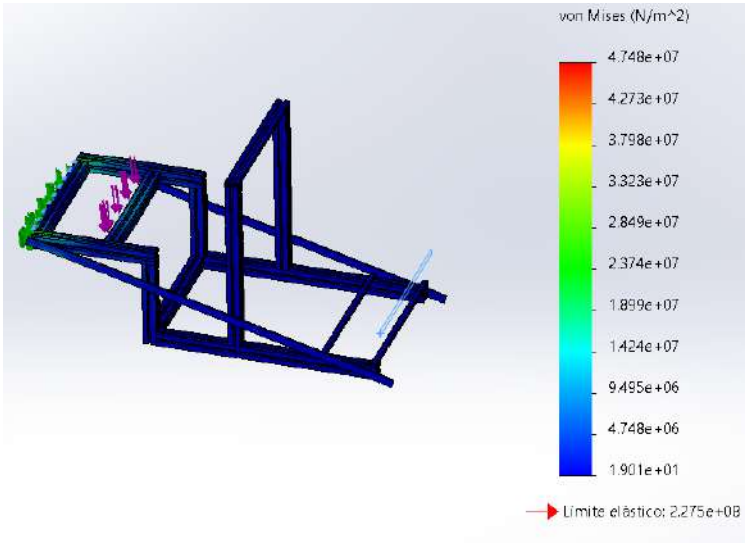


Figura 5-3 La tensión de Von Mises de 47,400,000 Pa se encuentra dentro de los límites permisibles para los perfiles Bosch de 40 mm, los cuales están hechos de aluminio de alta resistencia.

Contraste con los límites de falla

El factor de seguridad calculado nos indica que la estructura puede soportar 4.8 veces la carga máxima esperada antes de fallar. Este factor de seguridad es considerable y sugiere que la estructura es extremadamente robusta y segura. Normalmente factores menores a la unidad se consideran críticos.

En el caso de deformación, se consideran valores críticos el 0.1% de su longitud máxima. Sin embargo, la deformación de 2.8 mm que se tiene, aunque es mayor que el límite teórico, se considera aceptable para perfiles de aluminio estructural, que permiten cierta flexibilidad sin comprometer la seguridad. Esta deformación no afecta significativamente la funcionalidad o la integridad estructural

El esfuerzo máximo de Von Mises calculado de 47.4 MPa, está muy por debajo del límite de fluencia del aluminio, que es de 240MPa. La estructura opera por debajo de la capacidad máxima, asegurando que no habrá deformaciones plásticas ni fallas bajo condiciones de cargas previstas.

Las aceleraciones que alcanza tanto la simulación de la montaña rusa como el simulador de carreras tienen un promedio de 2.5 G´s, por lo que no son aceleraciones peligrosas para los pasajeros. Se tienen en cuenta los cambios bruscos de velocidad dentro del simulador de montaña rusa, por lo que se limita la velocidad a 100 m/s y los cambios de velocidad se realizan de forma más lenta y controlada, buscando no sobrepasar los 3 G´s como máximo.

5.1.2. Comunicación Ethernet

Aunque no se consiguió realizar comunicación por ethernet, debido a problemas con la transferencia al archivo del robot *FANUC*, se evaluó la calidad de la simulación de movimientos utilizando pruebas previamente realizadas por otros usuarios. Rodríguez, C. (2023).

Durante la comparación de la simulación física, se pudo notar que los movimientos del robot y los movimientos en la simulación dentro de *RoboDK* tienen un desfase casi imperceptible y se mueven con mucha sincronización, lo cual es muy importante para el simulador de carreras.

Durante la simulación en *RoboDk*, se registraron las trayectorias del robot, las señales del control de *PS4* y los datos de comunicación ethernet. Se rectificaron las respuestas que tenía el robot a las señales de entrada del control.

El control de *PS4* funciona apropiadamente, como sustituto de la palanca del *tech pendant*, el movimiento es muy fluido.

La velocidad para la simulación se calculó dependiendo de los botones seleccionados, como la variación en los movimientos es continua (van incrementado dependiendo la posición en la que se encuentre), este valor se ajusta por dos factores; el desplazamiento y la rotación, logrando así un movimiento más preciso y fluido.

Los controles se configuraron de la forma siguiente:

Stick izquierdo

- Desplazamiento en x (valor del *stick* izquierdo) *(10)
- Ángulo de rotación del eje 1 (valor del *stick* izquierdo) * (0.2)
- Rotación *ry*: 0

Botón R2

- Rotación *ry* :1

Botón L2

- Rotación ry: -1

Con esta configuración se puede simular el giro (cabeceo), la aceleración y el frenado del automóvil del videojuego.

5.1.3. Opción 1 Conexión serial

Como se describió en la sección 4.1, la estrategia para mejorar la velocidad de respuesta fue programando movimientos de menor duración, para monitorear con mayor frecuencia si se solicitaban cambios en ellos

Un problema asociado con esta estrategia es que las señales se acumulan en una lista de espera, esperando su turno para ser enviadas. Por lo tanto, incluso si se envía un cambio de movimiento, esta señal no se recibe hasta que las señales pendientes se ejecuten.

Para resolver el problema anterior, se agregaron condiciones para enviar los datos. Se verifica el estado del botón correspondiente y el intervalo de tiempo transcurrido desde el último envío de datos, esto para evitar enviar datos con demasiada frecuencia y prevenir que el puerto serial se sature.

El último problema que se presentó fue el movimiento discontinuo. Al tener desplazamientos muy cortos, los intervalos de envío de señales son más lentos y la velocidad del robot disminuye, haciendo que el robot no se mueva de forma continua, sino que lo haga a pasos, resultando en desplazamientos no continuos.

Para mejorar la continuidad en los movimientos, se variaron tres factores; la velocidad, el intervalo de envío y un factor de cambio. Mediante prueba y error se determinaron tres conjuntos de parámetros con los cuales el movimiento a lapsos se reduce significativamente, estos son los siguientes:

Opción 1		**Opción 2**		**Opción 3**	
Velocidad	5	Velocidad	30	Velocidad	20
Intervalo	.2	Intervalo	.1	Intervalo	.07
Factor	10000	Factor	8000	Factor	8000

5.1.4. Comunicación con Arduino

En esta opción se tuvieron problemas similares a los del caso anterior, aunque los movimientos son mucho más rápidos.

Se observó que el movimiento discontinuo persiste, aunque es más fluido que en la conexión serial. Esto se comprueba al medir los lapsos de tiempo y las velocidades que acepta para disminuir el movimiento vibratorio y la percepción de este.

Al usar una velocidad de 30 mm/s, tanto en la opción de conexión serial como en esta opción, se muestra una reducción significativa de la vibración, pero no suficiente para no percibirla.

5.1.5. Uso de Interrupciones

Después de implementar esta opción y realizar muchas pruebas de movimientos, se encontró al mismo problema. Las interrupciones funcionan de manera similar a un paro, la diferencia es que se pueden programar.

En el primer programa realizado, donde solo había dos movimientos, no hubo problema en los movimientos, pero cuando se termina la interrupción, automáticamente el programa sigue en la misma línea en la que se detuvo antes de activar la interrupción, lo cual genera un problema con la continuación de la secuencia.

No se puede salir de la interrupción sin que continúe el movimiento que se interrumpió. Por lo tanto, si hay un tercer movimiento y se sale de la primera interrupción, al activar la segunda interrupción, esta no seguirá la secuencia, ya que continuaría el movimiento interrumpido.

La solución desarrollada fue crear una secuencia de sub-interrupciones, pero esto no es práctico, ya que el número de interrupciones crecerá sin límite, debiendo ser anidadas unas dentro de otras, resultando en una alternativa complicada.

5.1.6. Uso de telemetría

Esta opción es la más adecuada para la integración del sistema. Consiste en extraer del juego los datos de posición del vehículo, reconfigurarlos y enviarlos al controlador del robot.

El problema enfrentado está relacionado con los límites impuestos por el espacio de trabajo del robot, los cuales pueden ser fácilmente rebasados después de varios movimientos.

La solución empleada fue escalar el valor de la velocidad que viene desde el videojuego, con una proporción 1:3, para que el límite de movimiento del robot no se rebase demasiado rápido, mientras aumenta la velocidad del automóvil virtual.

El robot se configura a una velocidad estándar de 100mm/s. El rango de movimiento del eje 5 del robot es de -120° a 120°, pero consideradas las dimensiones de la estructura, la coordenada vertical mínima (piso) y la seguridad del usuario, el rango de movimiento tiene que reducirse a valores de -50° a 50° aproximadamente.

Posterior a esto, se limita también la velocidad recibida del auto virtual, que está ligada al rango de movimiento dicho anteriormente. Este límite se coloca en los 150 km x hr del auto virtual. Lo que hace es que cuando se alcanza ese límite, el robot se estanca en ese movimiento, es decir, aun cuando la velocidad del auto virtual aumente, el robot se quedará en esa misma posición, hasta que se reduzca por debajo de esta velocidad.

Hasta este punto solamente se realizó la sincronización con lo que es el eje 5 del robot y la velocidad del auto virtual. Pero perfectamente se pueden ligar los demás tipos de movimiento dentro de esta programación, que sería muy parecida a la ya realizada.

5.2. Trabajos futuros

Como trabajos futuros, se puede abordar las opciones siguientes.

> Utilizar un robot con capacidad de carga equivalente a la del robot empleado en este proyecto, pero con comunicación ethernet disponible. Este tipo de comunicación permitirá integrar el sistema, de manera más sencilla y eficiente.

También se debe verificar el desarrollo del programa *RoboDK*, el cual sigue actualizándose y ofreciendo mejores rutinas y opciones de integración.

En el área de telemetría falta desarrollar la opción de tener todos los ejes trabajando al mismo tiempo, sincronizados con el videojuego. En este momento, ya se tienen las bases para poder realizar esto, simplemente es necesario desarrollar mucho más la misma idea.

Debe iniciarse el uso de filtros de movimiento, para recuperar el espacio de trabajo del robot, durante movimientos suaves, para no rebasar los límites físicos.

Además, es muy conveniente adquirir unos visores de realidad virtual de mejor calidad, que se conecten directamente con la *PC*, ya que los que se utilizaron dentro de este proyecto, estaban limitados al tipo de video.

Se puede hacer una mejora a la estructura y equiparle un mejor asiento, uno mucho más cómodo para el usuario y una cabina que recubra la estructura.

CONCLUSIONES

El análisis de carga de la estructura construida con perfiles de aluminio estructural ha demostrado tanto física como teóricamente ser una solución eficaz y segura para soportar las cargas previstas dentro de la operación del sistema. Los resultados obtenidos a través de simulaciones por elementos finitos indican un factor de seguridad de 4.8, lo que garantiza una capacidad de carga muy superior a la esperada.

La deformación máxima registrada de 2.8 mm, la cual, aunque es un poco alta, debe considerarse pese el aluminio estructural se caracteriza por su alta resistencia y flexibilidad. Esta solución no compromete la comodidad del usuario ni la precisión del robot, validando la elección de materiales y diseño.

La evaluación de las diferentes opciones que se realizaron a lo largo del proyecto reveló importantes consideraciones sobre la eficiencia y precisión en el movimiento del robot.

La conexión serial y las interrupciones presentaron limitaciones significativas en términos de fluidez y respuesta del sistema. Sin embargo, la telemetría permitió tener una solución más prometedora, para la sincronización precisa de los movimientos del robot con los datos extraídos del vehículo virtual. Esta integración garantiza que el robot pueda operar de manera coherente y precisa en aplicaciones exigentes.

La validación de la simulación frente a estudios previos y la implementación de mejoras en el código a lo largo de todo el proyecto ayudaron a abordar los desafíos operativos y optimizar el desempeño del robot.

Las modificaciones en la programación, como la adición de condiciones para el envío de datos, los periodos de tiempo de los envíos y restricciones de seguridad, mejoraron significativamente la capacidad de respuesta y la precisión en los movimientos.

La combinación de materiales de alta resistencia, las técnicas de simulación y el enfoque teórico proporciona una base sólida para el desarrollo de soluciones que demuestran que es posible lograr un equilibrio entre robustez estructural y precisión en el control.

El proyecto ha demostrado que es posible diseñar una estructura robusta y un sistema de control eficiente para la realización de una simulación de efectos gravito-inerciales, utilizando un robot. Dando paso a simulaciones mucho más complejas, por ejemplo, simulaciones de vuelo.

La implementación mediante telemetría fue muy importante, ya que ayudó a superar las limitaciones de otras opciones y proporcionar un movimiento preciso y fluido. Esto no solo asegura la viabilidad de un sistema funcional, sino que también ayuda a ver futuras mejoras y aplicaciones en la robótica y la automatización.

Este proyecto ha sido un gran desafío, con sus respectivos altibajos, me ha dejado un aprendizaje profundo y gran crecimiento personal, enseñándome la importancia que tiene la perseverancia y la adaptación al cambio.

Estoy orgullosa de lo que he logrado y tengo la satisfacción de que lo que hice tiene el potencial suficiente para algo mucho más grande.

REFERENCIAS

1. Nieuwenhuizen, F. M., & Bulthoff, H. H. (2013, June 30). The MPI CyberMotion Simulator: A Novel Research Platform to Investigate Human Control Behavior. Journal of Computing Science and Engineering. *Korean Institute of Information Scientists and Engineers.* https://doi.org/10.5626/jcse.2013.7.2.122
2. Allerton, D. (2009) *Principles of Flight Simulation*, Chichester, UK: John Wiley and Sons Ltd
3. Carreón Villal, J. R. (2014). *Diseño Y Operación De Una Estación Robótica Tipo Industrial* [Tesis de Licenciatura]. Instituto Tecnológico de Celaya.
4. Granados, S (2016). Análisis estructural de una plataforma Stewart-Gough para la aplicación de un simulador de vuelo [Tesis de Licenciatura]. Instituto Politcnico Nacional
5. Gómez, J. A., & Guacaneme, J. V. (2018). *Diseño e Implementación de una Plataforma de Stewart* [Tesis de Licenciatura]. Universidad Militar Nueva Granada.
6. Aristibal, D. (2015) *Control de una plataforma Stewart* [Tesis de ingeniería]. Universidad Tecnológica de Pereira Facultad de Tecnologías Tecnología Mecatrónica Pereira Risaralda
7. Kyb, C. (2006) *Control of a Dynamic Driving Simulator: Time-Variant Motion Cueing Algorithms and Prepositioning* [Tesis de diploma]. Institut für Verkehrsführung und Fahrzeugsteuerung
8. López, A, (2017) *Análisis dinámico de la plataforma Gough-Stewart* [Tesis de maestro en ciencias y tecnología en la especialidad de mecatrónica] Posgrado Interinstitucional de Ciencia y Tecnología

9. Nogareda C. et al. (2008) *Ergonomía. 5ª edición*. Madrid: INSHT
10. Ávila, R., Prado, & González, E. L. (2007). Dimensiones antropométricas de la población latinoamericana: México, Cuba, Colombia. *ResearchGate*. Recuperado desde: researchgate.net
11. Hernández, G. (2014). Uso de medidas antropométricas para el diseño de estaciones de trabajo enfocado a operadoras de las industrias de la ZMG. *Ciateqdigital Repositorio*. Recuperado 19 de mayo de 2023, de http://ciateq.repositorioinstitucional.mx/jspui/handle/1020/161
12. Delgado, A. et al. (2012) Influencia de los patrones posturales en la conducción y la antropometría en la carga biomecánica del raquis. *Repository UPB*. Recuperado 19 de mayo de 2023 de http://hdl.handle.net/20.500.11912/7333
13. *BeamNG.teach.* (2023, 29 marzo). Recuperado 12 de abril de 2023, de https://beamng.tech/
14. *Python* API for BeamNG.tech (2023) *Github.* Recuperado 15 de abril de 2023, de https://github.com/BeamNG/BeamNGpy
15. Fraser, D. (2019) Memorial Tributes: National Academy of Engineering Volume 22. *Nacional academies* Recuperado el 16 de marzo del 2023 de https://www.nae.edu/219823/J-HALCOMBE-LANING-19202012#publicationContent
16. Bellmann, T., Heindl, J., Hellerer, M., Kuchar, R., Sharma, K., & Hirzinger, G. (2011). The DLR Robot Motion Simulator Part I: Design and setup. 2011 IEEE International Conference on Robotics and Automation. doi:10.1109/icra.2011.5979913
17. Subir Kumar (2010) Introducción a la robótica. MC GRAW Hill
18. Ordóñez, L. (2020) Realidad Virtual y Realidad Aumentada. CEDRO
19. Serrano, D. (2017) Simulador de movimientos lineales basado en un robot con 6 grados de libertad. Pistas Educativas Vol. 39 - ISSN: 2448-847X
20. Lee YH, Meng YS, Heng YH. (2008) Experimental characterizations of an air to land channel over sea surface in C band. In: Proc. XXIXth URSI General Assembly

21. International Organization for Standardization. (2012). Robots and robotic devices -- Vocabulary (ISO 8373:2012)
22. Casas, S. (2014) Mejoras en la generación de claves gravito-inerciales en simuladores de vehículos no aéreos. Tesis doctoral. Universidad de Valencia
23. García, J. (2016) Fisiología del sistema vestibular. SEORL
24. Matlacuatzi R. (2014) Diseno del simulador dinamico para pilotos
25. como un sistema biomecatronico
26. ABB Robotics Products AB. (1993). *User's guide* (3HAC 0930). Suecia: s.n.
27. ABB Robotics Products AB. (1993). *Reference manual* (3HAC 0941-1). Suecia: s.n.
28. ABB Robotics Products AB. (1993). *Product manual IRB 6400* (3HAC 0849-1). Suecia: s.n.

29. ABB Robotics Products (2007) Manual del operador Introducción a RAPID (3HAC029364-005). Suecia: s.n.
30. ABB. ABB México. [En línea] ABB, enero de 2013. http://www.abb.com.mx.
31. Lutz, M. (2017). Learning Python (3rd ed.). O'Reilly. ISBN: 978-0-596-15806-4.
32. RoboDK. (s.f.). *RoboDK API - Python API*. Recuperado de https://robodk.com/doc/es/RoboDK-API.html#PythonAPI
33. Vital, M. (2021) Introducción a Arduino. Vida Científica boletín científico de la escuela preparatoria No. 4. Vol. 9 No. 17(2021) 4-8. ISSN: 2007-4905
34. Arduino(s.f.) UNO R3. Recuperado https://docs.arduino.cc/hardware/uno-rev3/
35. Suikat, R (2005) The new dynamic driving simulator. Institude of transportation systems. German aeroespace centre (DLR)
36. Darias, A. (2008) Predicción de abandonos y pagos en videojuegos Freemium para móviles. Máster en Economía, Finanzas y Computación. University of Huelva & International University of Andalusia
37. Rodríguez, C. (2023) RoboDK to FANUC. (Video) https://youtu.be/WQg-NgXyI3Q?si=TNT4Uk0TF4We_21y
38. Noguera, T. (2018) 7 aplicaciones para disfrutar de la mejor realidad virtual en iOS y Android. Recuperado el 13 abril del 2024. https://www.xatakamovil.com/aplicaciones/7-apps-para-disfrutar-de-la-mejor-realidad-virtual-en-ios-y-android
39. Timothy (2021) Connect to Fanuc Robot for beginners. RoboDK API. RoboDK. Recuperado 16 de abril del 2024. https://robodk.com/forum/Thread-CONNECT-TO-FANUC-ROBOT-for-beginners

ANEXO 1. HABILITACIÓN DEL BRAZO ROBÓTICO

Dentro del mantenimiento inicial del robot, se revisaron las conexiones del controlador, asegurando que todo estuviera conectado correctamente.

Se revisaron las baterías, de respaldo del controlador ya que su buen estado es importante para garantizar el almacenamiento de datos dentro del controlador. Al momento de introducir nuevos programas, calibrar y configurar, estos datos se guardan al apagar el robot y se mantienen al estar energizado. Sin las baterías, se tendría que configurar el robot cada vez que se enciende. (Figura A1-1)

Para esto, se comprobó el voltaje y la corriente de las baterías, para descartar que no estuvieran deterioradas. Sin embargo, se tuvieron que remplazar, ya que no marcaban voltaje. Las baterías que se le colocaron se seleccionaron de baterías de repuesto. Finalmente, se colocaron dentro del controlador.

Figura A1-1 Batería de repuesto para el controlador del Robot ABB

Antes de encender el brazo, se verifica que el interruptor que energiza al robot no se haya desconectado, si es así, se debe de regresar a su posición original, de otro modo este no encenderá.

Figura A1-2 Caja de interruptores, se debe de revisar si todos los interruptores estén en la posición de encendido

El proceso para rehabilitar el robot se basa en el trabajo de Carreón (2014), el cual propone el procedimiento mostrado en la Figura A-3 que indica paso a paso lo que se debe de hacer para habilitar el robot.

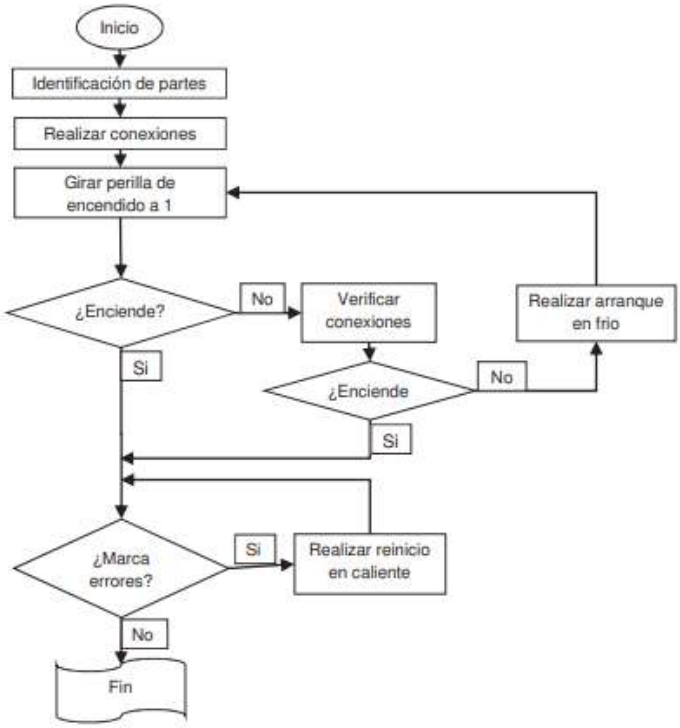


Figura A1-3 Diagrama de flujo para inicio en frio

Para que los robots funcionen correctamente, la compañía ABB proporciona configuraciones especiales para cada uno. Estas configuraciones anteriormente se introducían al robot con el uso de disquetes. Sin embargo, esto fue sustituido por un convertidor de disquetes, en el cual se puede sustituir los disquetes por una memoria, donde deben almacenarse cada uno de los disquetes.

Se dispone de nueve disquetes para cada robot. El primero es el disquete serial, el cual se identifica por el número de serie de cada robot. Los otros ocho disquetes corresponden al software *Baseware*. El contenido de los disquetes debe transferirse a una memoria de manera específica, como se mencionará más adelante.

Instalación de floppys

Para almacenar el contenido de los disquetes a la memoria USB, es necesario utilizar un software apropiado, como *Floppy Manager*. Este programa crea áreas en la memoria USB, fragmenta la capacidad de la memoria en varias partes, las cuales sirven para reemplazar digitalmente a los disquetes. Su función es colocar los datos de cada disquete dentro de estas partes, simulando que cada parte es un disquete. Primero, se debe tener una memoria USB completamente vacía y se selecciona la memoria que se está utilizando, como se muestra en la Figura A-4

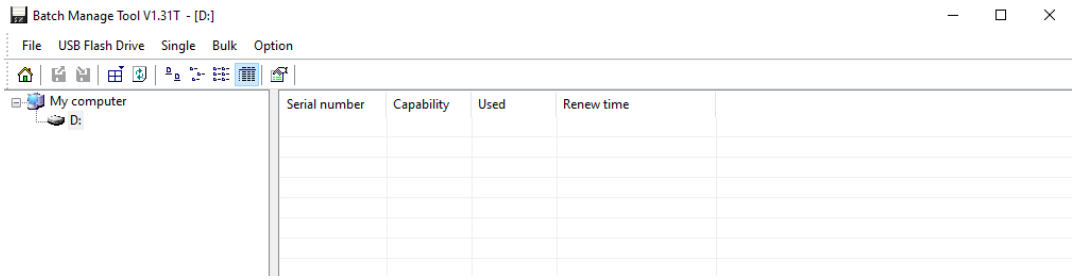


Figura A1-4 Ventana del programa Floppy Manager. Se inicia el software y se debe visualizar la memoria en la parte derecha

Se le da clic derecho señalando la memoria. Se selecciona formatear. Se muestra una nueva pantalla, en el recuadro de particiones (the number of floppy disk) se puede cambiar el número de particiones en las que se va a dividir la memoria.

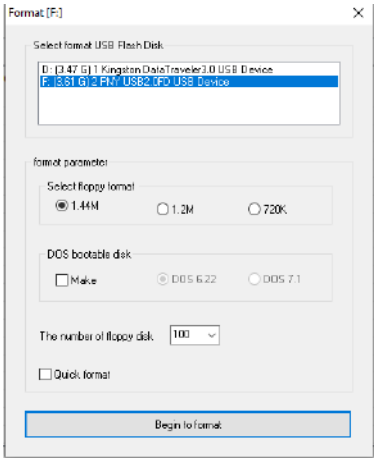


Figura A1-5 Cuadro donde se configura las particiones de la memoria

En cada una de esas divisiones, se introduce la información de cada disquete que se vaya a utilizar, para esto, se mostrará una pantalla por cada división (Figura A-6). Tenemos particiones que van del 000 hasta el número de particiones hechas, para introducir información, solo damos doble clic en la partición y se abrirá una carpeta para introducir los archivos.

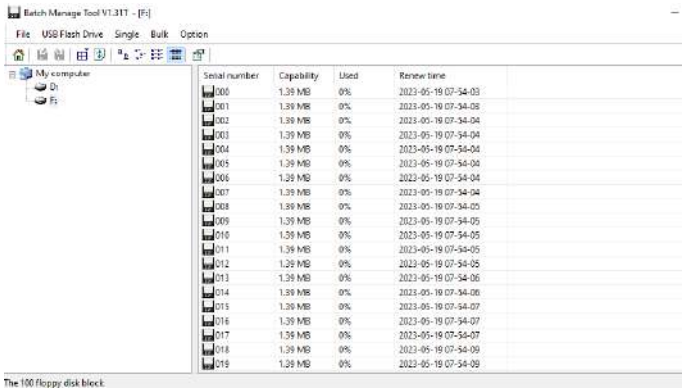


Figura A1-6 Numero de particiones señalizadas dentro de la USB

Dentro de los disquetes se busca el número de serie del robot a utilizar, en este se utilizó el robot con el número de serie que tenemos es el 64-06195 y con el software *Baseware* de 3.0 (Figura A-7). Estos se introducen dentro de la memoria, cada uno en una partición diferente.

Figura A1-7 Número de serie y Baseware del robot ABB

Para encender el robot, se gira la perilla de 0 a 1. Se coloca la memoria en el convertidor de disquetes, se observa que se marca 00, indicando la posición actual o carpeta activa. Se debe cambiar el contador, dependiendo del disquete que va solicitando el robot.

Figura A1-8 Convertidor de disquetes, alimentar las configuraciones de cada robot ABB

En el *teach pendant* se mostrará un mensaje especificando insertar el Disco flexible "key" con el número de serie correspondiente al controlador (ver Figura A-9) en la unidad de disco.

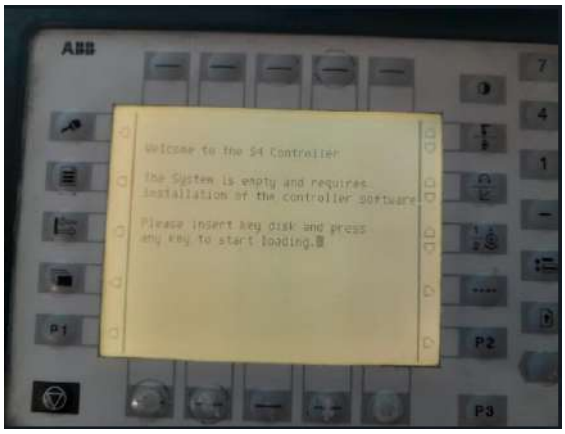


Figura A1-9 Se debe de colocar el key disk referente al robot utilizado

Después de introducir los disquetes, en algún momento de la instalación, se elige el modo de instalación, se muestran tres opciones, normalmente se selecciona Silent:

- Silent = instalación rápida, no pide información extra.
- AddOpt = Permite elegir idioma, configurar hora y fecha.
- Query = Pide archivos de respaldo del manipulador, datos de calibración y de parámetros de sistema. Seleccionar Silent en caso de no tener respaldos del manipulador. Carreón (2014)

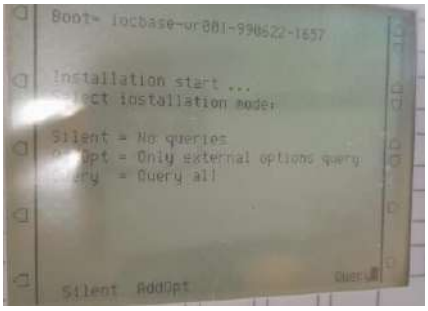


Figura A1-10 Modo de instalación, de debe de escoger entre tres modos de instalación

En tipo de DC link se selecciona DC2 que corresponde al modelo del controlador.

El *teach pendant* se iniciará y puede mostrar los siguientes errores:

- Baterías de respaldo descargadas: Esto se soluciona dejando prendido el controlador 36 horas continuas.
- Ejes no encontrados: Hay que calibrar el manipulador.
- Error fatal: se necesita un reinicio caliente. Carreón (2014)

ANEXO 2. CALIBRACIÓN DEL ROBOT

Cada vez que se instala o reinstala el sistema operativo es necesario que el manipulador se calibre y siempre se debe de verificar que esté calibrado, ya que de otra manera no se podrá ejecutar ningún programa, (Figura A-11). Dentro de la ventana *jogging* o movimiento libre en el *teach pendant* se puede ver que no se muestran las coordenadas en el lado derecho.

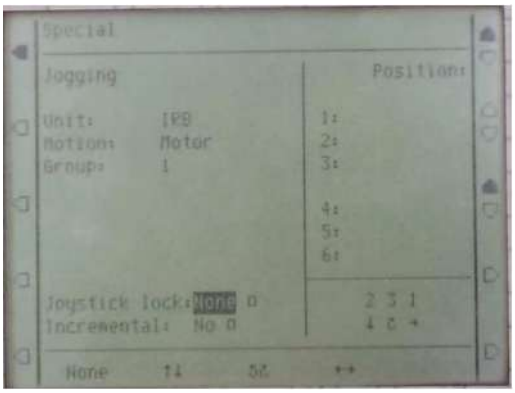


Figura A2-11 Dentro de la ventana jogging o movimiento libre en el teach pendant se puede ver que no se muestran las coordenadas en el lado derecho, ya que necesita calibrarse.

Se comienza a mover los ejes desde el 1 al 6, posicionándolos en sus marcas de inicio, estas se encuentran a un lado de cada uno de los ejes, como se muestra en la Figura A-12.

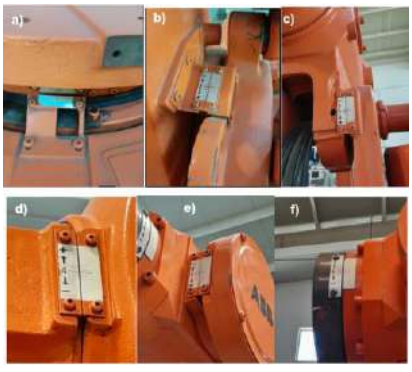


Figura A2-12 Se comienza a mover los ejes desde el 1 al 6, posicionándolos en la marca de inicio, estas se encuentran a un lado de cada uno de los ejes a) Eje, 1 b) Eje 2, c) Eje 3, d) Eje 4, e) Eje 5, f) Eje 6

Cuando se tienen todos los ejes en sus posiciones de referencia inicial, mediante la siguiente ruta:

Menú de otras ventanas > Service > View > Calibration > Fine Calibrate,

Se calibran todos los ejes a la vez. El robot obtiene la posición de calibración, como se ve en la Figura A-13

Figura A2-13 Si se calibran todos los ejes, se llega a la posición de calibración del robot

ANEXO 3. PROGRAMACIÓN DEL ROBOT POR PUNTOS

Se hicieron dos pruebas de programación por puntos, la primera, es seguir una secuencia de cuatro puntos, tomando de referencia una estructura rectangular, marcando los puntos por las esquinas.

El robot se mueve siguiendo la secuencia de puntos L desde el punto 1 al 2 y así sucesivamente hasta llegar nuevamente al punto L→2→3→4→1. Para programar cada punto manualmente, primero se mueve el robot al punto donde se va a iniciar, se define un tipo de movimiento y se guarda el punto. Se hace esto para los siguientes puntos y se ejecuta el programa.

Para generar el programa meidante el *teach pendant*, se abre la ventana del programa y se designa un nuevo programa en la parte superior. Luego, mueve el robotal primer punto y se le asigna un tipo de movimiento *MoveL* o *MoveJ*, guardándolo con “mode pose”. De esta manera, el usuario puede mover el robot y guardar los puntos. Si se programan puntos en el mismo sitio, el programa no se ejecutará.

El programa resultante debe ser similar a lo siguiente:

```
MoveL P1, v200, z10, tool0;  ! Mover al punto P1
MoveL P2, v200, z10, tool0;  ! Mover al punto P2
MoveL P3, v200, z10, tool0;  ! Mover al punto P3
MoveL P4, v200, z10, tool0;  ! Mover al punto P4
MoveL P1, v200, z10, tool0;  ! Volver al punto P1
```

Donde:

- MoveL: Comando para movimientos lineales.
- V200: Velocidad del movimiento
- z10: Precisión del movimiento
- tool0: Herramienta

Para ejecutar el programa, se debe accionar el botón *enabling device*, y a su vez, oprimir *start* en la pantalla para que inicie el programa.

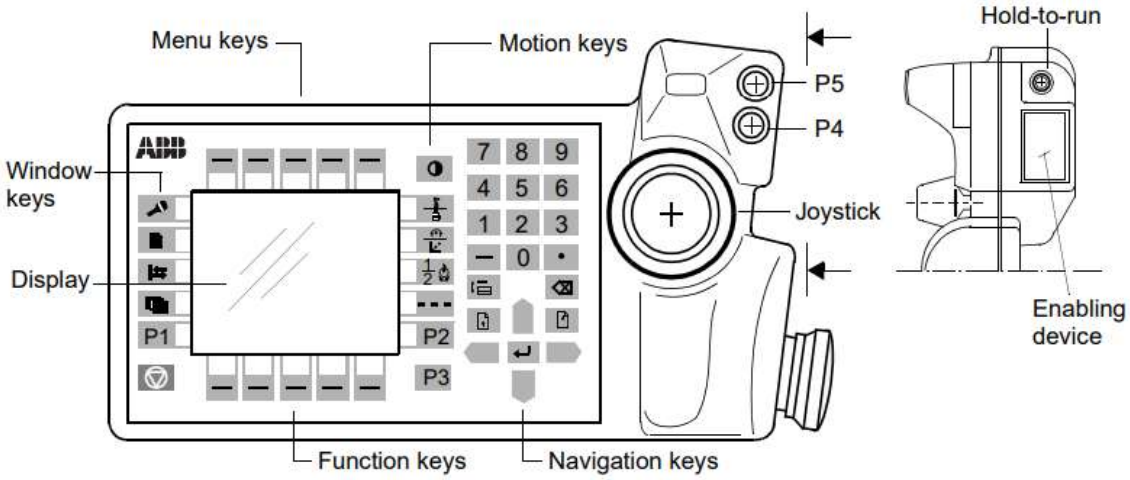


Figura A3-1 Partes del teach pendant

ANEXO 4. PROGRAMACIÓN PARA SIMULACIÓN MONTAÑA RUSA

Este programa mueve al robot a una serie de puntos definidos en el espacio, programados mediante el programa *RoboDK*. Cada punto tiene su respectiva velocidad y orientación para recrear el movimiento de la montaña rusa. De un fragmento de video de una montaña rusa se identifican los puntos de cambio de movimiento, esto para poder dar una dirección para el siguiente movimiento.

Igualmente se mide el lapso que pasa de un movimiento a otro, para estimar la velocidad que se requiere y la duración del movimiento.

Programación *RAPID*

```
MODULE MOD_Prog2

    PROC Main ()

        MoveL [[2181.090,0.000,165.118],[0.57357644,-
        0.00000000,0.81915204,-0.00000000],[0,0,-1,1],
        [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v500,z1,tool0
        \WObj:=wobj0;

        WaitTime 6.000;

        MoveL [[2136.540,297.215,119.036],[0.46213573,-
        0.13496925,0.87635601,-0.01462898],[0,0,-1,1],
        [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v200,z1,tool0
        \WObj:=wobj0;

        WaitTime 1.000;

        MoveL [[2204.760,314.133,308.066],[0.58707093,-
        0.13225944,0.79865728,-0.00130615],[0,-1,-1,1],
```

```
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v300,z1,tool0
\WObj:=wobj0;

WaitTime 1.000;

MoveL [[2188.670,307.597,260.205],[0.49878203,-
0.06041088,0.86391581,0.03487824],[0,0,-1,1],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100,z1,tool0
\WObj:=wobj0;

WaitTime 5.000;

MoveL [[2113.800,297.075,92.100],[0.38175123,-
0.06444658,0.92162901,0.02669465],[0,-1,0,1],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v200,z1,tool0
\WObj:=wobj0;

WaitTime 1.000;

MoveL [[2196.640,308.718,275.682],[0.53599077,-
0.05883201,0.84133699,0.03748013],[0,-1,0,1],[9E+09,9E+09,
9E+09,9E+09,9E+09,9E+09]],v200,z1,tool0 \WObj:=wobj0;

WaitTime 2.500;

MoveL [[2179.380,306.292,245.490],[0.46062382,-
0.06187475,0.88485012,0.03220996],[0,-1,0,1],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v200,z1,tool0
\WObj:=wobj0;

ENDPROC

ENDMODULE
```

ANEXO 5. PROYECCIÓN DE VIDEO EN EL VISOR, BIFOCAL

Para usar el visor con el video, se debe tener un video que sea bifocal, es decir que la pantalla esté dividida en dos partes: una que muestre lo del ojo derecho y otra lo del ojo izquierdo, Figura A5-1.

Figura A5-1 Se debe tener un video que sea bifocal, es decir que la pantalla esté dividida en dos partes Noguera, T. (2018)

ANEXO 6. PROGRAMAS PARA PRUEBAS DE MOVIMIENTO EN MONTAÑA RUSA

Sincronización de un video con los movimientos del robot utilizando un programa en *Python* que manda la señal en el momento que se toca el teclado, este activa el programa de *RAPID* que solo espera la señal de la entrada serial para poder activarse. Como se tiene un ligero retraso entre la pantalla que se utiliza en los visores y la computadora, se ajusta el tiempo en el que se manda la señal a *RAPID*.

El programa en *RAPID* corresponde a los movimientos de un video de montaña rusa.

Programación en *Python*

```
import serial
import keyboard
import time

# Configuración del puerto
ser = serial.Serial(
    port='COM6',
    baudrate=9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS
)

# Defnir una función para enviar datos a través del puerto serial
def enviar_datos():
    time.sleep(0)
    # Enviar el byte 's' a través del puerto serial
```

```
    ser.write(b's')
    # Imprimir un mensaje
    print("Señal enviada")

# Asocia la función enviar_datos() al evento de presionar la tecla espacio
keyboard.on_press_key('space', lambda _: enviar_datos())
# Espera hasta que se presione la tecla 'esc' (Escape) para salir del programa
keyboard.wait('esc')
# Cierra la conexión serial cuando se sale del bucle principal
ser.close()
```

Programación en RAPID

```
MODULE MOD_Prog1
  VAR iodev iodev1;
  VAR BYTE letra;
        PERS tooldata Ensamblaje1:=
[TRUE,[[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[1,
[0,0,20],[1,0,0,0],0,0,0.005]];
        PERS wobjdata ABB:= [FALSE, TRUE, "",
[[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[[0,0,0],[1,
0,0,0]]];
        PROC Main ()
              Open "sio1:", iodev1\Bin;
                    letra: =ReadBin(iodev1);
               IF letra <> 42 THEN
                     WaitTime 0.000;
                     MoveAbsJ [[10.233900,51.638800,62.259300,18.054000,-
               28.008700,-10.142900],
               [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100,z1, Ensamblaje1
               \WObj:=ABB;
```

MoveAbsJ [[-10.000000,50.631100,63.174400,-0.000013,-28.174400,0.000007], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v100,z1,Ensamblaje1 \WObj:=ABB;

MoveAbsJ [[-10.000000,43.607300,54.140200,-0.000014,-49.140200,0.000008], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,z1,Ensamblaje1 \WObj:=ABB;

WaitTime 2.000;

MoveAbsJ [[-10.000000,43.607300,54.140200,0.000007,-19.140200,-0.000005], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v10,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 5.000;

MoveAbsJ [[-10.000000,45.736300,56.825200,-0.000044,-8.825170,0.000036], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,z1,Ensamblaje1 \WObj:=ABB;

MoveAbsJ [[-10.000100,50.631200,63.174400,-30.144200,-10.154200,43.885400], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1,Ensamblaje1 \WObj:=ABB;

WaitTime 2.000;

MoveAbsJ [[-10.000000,47.144600,58.627100,-0.000005,-33.627100,0.000003], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v40,z1,Ensamblaje1 \WObj:=ABB;

MoveAbsJ [[-9.261370,44.005900,54.395900,-0.019819,-32.348800,-31.009300], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v50,z1, Ensamblaje1 \WObj:=ABB;

MoveAbsJ [[-9.999980,45.028800,55.927700,1.247050,-33.885200,6.934720],

[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v70,z1,Ensamblaje1 \WObj:=ABB;

MoveAbsJ [[0.000000,49.242200,61.348200,0.000000,-28.348200,-0.000000], [9E+09, 9E+09,9E+09,9E+09,9E+09,9E+09]], v30,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 3.000;

MoveAbsJ [[0.000000,47.144700,58.627100,0.000000,-38.627100,-0.000000], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 2.500;

MoveAbsJ [[8.000010,47.144600,58.627100,2.156490,-23.717400,-11.937500], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v5,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 2.000;

MoveAbsJ [[8.000020,43.607300,54.140300,-2.060250,-36.052800,-8.260880], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 2.000;

MoveAbsJ [[8.000000,43.607300,54.140200,0.000003,-24.140200,-0.000002], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,z1, Ensamblaje1 \WObj:=ABB;

```
                WaitTime 2.000;
                MoveAbsJ [[7.999990,47.144700,58.627100,-0.000018,-
13.627100,0.000014],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,z1,Ensamblaje1 \WObj:=ABB;
                WaitTime 1.000;
                MoveAbsJ [[7.999990,43.607300,54.140200,-0.000009,-
29.140200,0.000006],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v20,z1,Ensamblaje1 \WObj:=ABB;
                WaitTime 1.000;
                MoveAbsJ [[7.999990,43.607300,54.140200,-0.000021,-
19.140200,0.000015],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v20,z1,Ensamblaje1 \WObj:=ABB;
                WaitTime 2.500;
                MoveAbsJ [[8.000010,48.544900,60.438700,0.000019,-
10.438700,-0.000014], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,z1,
Ensamblaje1 \WObj:=ABB;
                WaitTime 1.000;
                MoveAbsJ [[8.000020,48.544900,60.438800,-16.148200,-
11.123900,25.379800],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v20,z1,Ensamblaje1 \WObj:=ABB;
                WaitTime 1.500;
                MoveAbsJ [[7.999980,48.544900,60.438800,16.148200,-
11.123900,-25.379800], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v40,z1,
Ensamblaje1 \WObj:=ABB;
                MoveAbsJ [[8.000010,45.736200,56.825100,0.000565,-
28.824600,-10.000400], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1,
Ensamblaje1 \WObj:=ABB;
                WaitTime 3.000;
                MoveAbsJ [[8.000000,45.736200,56.825100,-0.000003,-
23.825100,0.000002],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1,Ensamblaje1 \WObj:=ABB;
```

```
WaitTime 13.500;
MoveAbsJ [[8.000000,50.631100,63.174300,-4.698290,-19.368900,14.312800],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1,Ensamblaje1 \WObj:=ABB;
WaitTime 1.000;
MoveAbsJ [[8.000060,50.631200,63.174400,6.931350,-19.609400,-21.356000], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v60,z1, Ensamblaje1 \WObj:=ABB;
WaitTime 1.000;
MoveAbsJ [[7.999980,47.144600,58.627100,-0.000608,-28.627000,-14.999600], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1, Ensamblaje1 \WObj:=ABB;
WaitTime 1.500;
MoveAbsJ [[8.000000,47.144700,58.627100,0.000002,-23.627100,-0.000002], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;
WaitTime 6.500;
MoveAbsJ [[8.000010,48.544900,60.438700,0.000014,-23.438700,-0.000009], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v50,z1, Ensamblaje1 \WObj:=ABB;
WaitTime 0.500;
MoveAbsJ [[8.000000,47.144700,58.627100,0.000002,-23.627100,-0.000002], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;
WaitTime 0.500;
MoveAbsJ [[8.000010,48.544900,60.438700,0.000014,-23.438700,-0.000009], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;
WaitTime 0.500;
```

MoveAbsJ [[8.000000,47.144700,58.627100,0.000002,-23.627100,-0.000002], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 0.500;

MoveAbsJ [[8.000010,48.544900,60.438700,0.000014,-23.438700,-0.000009], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 0.500;

MoveAbsJ [[8.000000,47.144700,58.627100,0.000002,-23.627100,-0.000002], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 1.000;

MoveAbsJ [[8.000000,52.013300,65.010400,3.280760,-23.213800,-17.961200], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v20,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 2.000;

MoveAbsJ [[8.000020,47.136200,58.630500,4.152560,-38.369600,11.524100], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 0.500;

MoveAbsJ [[8.000010,47.136200,58.630400,0.000021,-23.630400,-0.000013], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;

WaitTime 6.000;

MoveAbsJ [[8.000070,50.623100,63.178100,-7.308110,-16.482300,16.796800], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1,Ensamblaje1 \WObj:=ABB;

WaitTime 1.000;

MoveAbsJ [[7.999940,47.136200,58.630400,-0.683221,-30.601500,-9.406190], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1, Ensamblaje1 \WObj:=ABB;

```
                WaitTime 2.000;
                MoveAbsJ [[8.000010,47.136200,58.630400,0.000021,-
23.630400,-0.000013], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,z1,
Ensamblaje1 \WObj:=ABB;
                WaitTime 8.000;
                MoveAbsJ [[0.000000,47.136200,58.630400,0.000000,-
23.630400,-0.000000], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v20,z1,
Ensamblaje1 \WObj:=ABB;
            ENDIF
        ENDPROC
ENDMODULE
```

ANEXO 7. CONEXIÓN ETHERNET

El programa de este anexo tiene como objetivo ver cómo actúan los botones del control de *PS4*, determinar si están dañados y si se pueden vincular correctamente a un programa en *Python*, así como enviar las señales correctas.

Para esto, simplemente se programa una ventana que muestra un rango de movimiento de los botones que se desean utilizar. En este caso, se incluyen el botón X, el círculo, *stick* izquierdo, *stick* derecho, botón R2 y L2

Programación en Python

```
import pygame
pygame.init()

#ventana
tamaño_ventana = (400, 400)
ventana = pygame.display.set_mode(tamaño_ventana)
pygame.display.set_caption("Valores del Control PS4")

#controlador de control PS4
control = pygame.joystick.Joystick(0)
control.init()

# Fuente para el texto
fuente = pygame.font.Font(None, 36)

def main():
    running = True

    while running:
```

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        running = False

# Actualizar el estado del controlador
pygame.event.pump()

# Obtener valores del control
boton_x = control.get_button(0)  # Botón X
boton_circle = control.get_button(1)  # Botón Círculo
left_stick_x = control.get_axis(0)  # Eje X del stick izquierdo
right_stick_y = control.get_axis(3)  # Eje Y del stick derecho
boton_r2 = control.get_axis(5)  # Botón R2 (gatillo derecho)
boton_l2 = control.get_axis(4)  # Botón L2 (gatillo izquierdo)
boton_share = control.get_button(8)  # Botón X

# Limpiar la ventana
ventana.fill((255, 255, 255))

# Renderizar los valores en la ventana
render_text("Botón X: {}".format(boton_x), (20, 20))
render_text("Botón Círculo: {}".format(boton_circle), (20, 60))
render_text("Stick izquierdo X: {:.2f}".format(left_stick_x), (20, 100))
render_text("Stick derecho Y: {:.2f}".format(right_stick_y), (20, 140))
render_text("Botón R2: {:.2f}".format(boton_r2), (20, 180))
render_text("Botón L2: {:.2f}".format(boton_l2), (20, 220))
render_text("Botón s: {}".format(boton_share), (20, 240))

# Actualizar la pantalla
pygame.display.flip()
```

```
    pygame.quit()

def render_text(text, position):
    text_surface = fuente.render(text, True, (0, 0, 0))
    ventana.blit(text_surface, position)

if __name__ == "__main__":
    main()
```

ANEXO 8. PROGAMA PARA EFECTUAR MOVIMIENTOS MEDIANTE EL CONTROL PS4

En este anexo se incluye un programa en *Python* para mover el robot de acuerdo a las acciones del control de *PS4*. Esta simulación se lleva a cabo utilizando el robot ABB 6400 simulado en *RoboDK.* (Figura A8-1)

Figura A8-1 Movimiento del robot virtual guiado por medio del control de PS4

El programa calcula la nueva posición del robot usando las rotaciones y traslaciones del mismo. El robot se mueve a la nueva posición calculada mediante una instrucción *MoveL*. Al final, para salir del ciclo, solo es necesario presionar <Ctrl + C>.

Programación en Python

```
import pygame
from robolink import *
from robodk import *
pygame.init()
```

```
RDK = Robolink()

# Configurar el controlador de PS4
controller = pygame.joystick.Joystick(0)
controller.init()

# Coordenada donde se va a ir el robotsin
cordenada = [0, 0, 0, 0, 0, 0]
posicion inicial = None  # Guarda la posición inicial del robot



try:
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
        pygame.event.pump()

        left_stick_x = controller.get_axis(0)
        boton_r2 = controller.get_axis(5)  # Botón R2
        boton_l2 = controller.get_axis(4)  # Botón L2

        # Calcular desplazamiento en X y rotación en Rx
        displacement_x = left_stick_x * 10
        rotacion_angl = left_stick_x * .1  # Aquí se ajusta el ángulo
        rotacion_ry = 0  # Valor inicial de rotación en Ry

        if boton_r2 > 0:  # Si se presiona el gatillo R2
            rotacion_ry = 0.01  # Ajusta el ángulo
        elif boton_l2 > 0:  # Si se presiona el gatillo L2
```

```
            rotacion_ry = -0.01  # Ajusta el ángulo

        robot = RDK.Item('', ITEM_TYPE_ROBOT)
        current_position = robot.Pose()
        target_position = current_position * transl(displacement_x, 0, 0) * rotz(rotacion_angl)
        robot.MoveL(target_position)

except KeyboardInterrupt:
    print("Deteniendo la detección")
    pygame.quit()
```

ANEXO 9. PROGRAMA PARA EFECTUAR MOVIMIENTOS UTILIZANDO EL VOLANTE

Programa similar al del anexo 8, pero en este caso se usa interacción con los pedales y el volante.

Programación de *Python*

```
import pygame
from robolink import *
from robodk import *

pygame. init()

RDK = Robolink ()

volante = pygame. joystick.Joystick(0)  # Volante
acc = pygame. joystick.Joystick(1)  # acelerador
freno= pygame. joystick.Joystick(2)  # freno

volante. init()
acc. init()
freno. init()



# Coordenada donde se va a ir el robot
target_coordinate = [0, 0, 0, 0, 0, 0]

try:
    while True:
        for event in pygame.event.get ():
            if event. type == pygame.QUIT:
```

```
            pygame. quit()
            exit ()
        pygame. event.pump()

        volante_angle = volante.get_axis (0) * 180.0  # Ángulo del volante
        acc_val = acc.get_axis (0)  # acelerador
        freno_val = 1.0 – freno.get_axis (0)  #  freno

        # (Rz) ángulo del volante
        rotation_angle = volante_angle * 0.5

        #  desplazamiento en X valor del acelerador
        displacement_x = acc_val * .01
        robot = RDK.Item('', ITEM_TYPE_ROBOT)
        current_position = robot. Pose()

        # nueva posición considerando la rotación y el desplazamiento
        target_position = current_position * transl (displacement_x, 0, 0) *
rotz(rotation_angle)

        robot. MoveL(target_position)
except KeyboardInterrupt:

  print ("Deteniendo la detección")

 pygame. quit()
```

ANEXO 10. COMUNICACIÓN ROBOT FANUC/PROGRAMA ROBODK

De acuerdo con la página oficial de *RoboDK*, se deben de seguir los siguientes pasos para tener la conexión entre robot el *FANUC* y el programa *RoboDK,* Timothy (2021)

***RoboDK – FANUC* ROBOT(R-30iB)**

- PC con Robot
- Conectar el cable Ethernet al controlador R-30ib
- Asignar una IP a la PC y al Robot (en el mismo segmento).
- Seleccionar MENU->SETUP->HOST COMM para asignar la dirección IP al ROBOT (no se permite espacio antes de la dirección IP), seleccionar NEXT->INIT->YES->ROBOT para finalizar la configuración.
- Seleccionar MENU->SETUP->HOST COMM->PING para verificar que la dirección IP esté correctamente configurada.
- Configurar una dirección IP en la configuración de red de la tarjeta de red Ethernret de la PC.

Configuración de software

- ROBOT
- Debe estar instalado el USER SOCKET MESSAGING (R648)
- Seleccionar MENU->STATUS->VERSIONID->CONFIG para verificar.
- Debe estar configurado $Karel_ENB en 1.
- Seleccionar MENU->SYSTEM_Variables para verificar, si está en 0, cámbialo a 1 y reiniciar.
- Cargar los drivers de *RoboDk* para *FANUC* en el robot.
- Descargar el driver: https://robodk.com/doc/en/Robots-Fanuc-R...Fanuc.html
- Usar la versión correcta del driver (o el robot no seguirá tu código).
- Subir RDK_S3. PC, GO_MJ, GO_ML, GO_MC, GO_PROG al robot.

- Insertar el USB en el teach pendant de *FANUC*.
- MENU->FILE->FILE->UTIL->SET DEVICE->USB on TP->ENTER.
- Seleccionar '*' para mostrar todos los archivos.
- Seleccionar todos los drivers y LOAD->YES al robot.
- Inicializar el Driver del Robot (según el sitio web de *RoboDK*: https://robodk.com/doc/en/Robots-Fanuc-R...Fanuc.html).

Seguir los siguientes pasos para instalar drivers:

- Seleccionar Menu-(Next)-System-[TYPE]-Variables.
- Seleccionar $HOSTS_CFG (mantener presionada la tecla de desplazamiento mientras seleccionas hacia abajo para desplazarse más rápido).
- Seleccionar el número 3.
- Configurar $SERVER_PORT a 2000.
- ($HOSTS_CFG [3].$SERVER_PORT = 2000).
- Seleccionar Menu-Setup.
- Seleccionar [TYPE]-Host comm.
- Seleccionar [SHOW]-Servers.
- Seleccionar "S3"-Enter.
- Configurar el Nombre del protocolo en SM.
- En nombre del puerto, si hay varios puertos disponibles: Configurar el puerto a P3 (o el puerto correspondiente).
- Configurar el tiempo de Inactividad a 9999.
- Configurar el estado de inicio a [CHOICE] START.
- Configurar el estado actual a STARTED:
- Para hacerlo, seleccionar [ACTION]-DEFINE, luego [ACTION]-START.
- Iniciar el programa DRIVERRDK_S3:
- Seleccionar el botón SELECT desde el teach pendant.
- Desplazarse hacia abajo hasta el programa DRIVERRDK_S3.
- Seleccionar Enter (botón del teach pendant).

- Seleccionar Shift-Reset y Shift-Forward para iniciar el programa.
- El botón DEADMAN debe estar siempre presionado.
- Al finalizar la configuración, las señales 'busy' y 'run' deberían cambiar de amarillo a verde.

PC

- Abre *RoboDK*->file->open online library para seleccionar un modelo de robot.
- File->open->archivo del robot descargado.
- Conéctate al robot:
- Hacer clic derecho en el nombre del robot en el árbol y seleccionar connect to robot.
- Ingresar la dirección IP configurada (por ejemplo, 192.168.1.10).
- Puerto predeterminado 2000.
- Si está correctamente configurado, recibirás una señal verde 'ready'.

Teach pendant:

- Ejecutar el programa manualmente:
- Cambiar el R-30ib a T1.
- Cambiar el interruptor superior izquierdo del TP a ON.
- Seleccionar el programa (SELECT->nombre del programa (RDKS_3. PC)->ENTER).
- Presionar el botón DEADMAN (todo el tiempo en modo T1/T2).
- Presionar SHIFT+RESET para que desaparezca la señal FAULT.
- Presionar SHIFT+FWD para ejecutar el programa seleccionado.
- Si no funciona, intenta FCTN->ABORT ALL y regresar al paso de selección del programa. Timothy (2021)

ANEXO 11. PROGRAMA PARA CONEXIÓN SERIAL

El programa presentado en este anexo se divide en dos partes, la parte que manda señales y la que recibe.

El programa desarrollado en Python se encarga de conectar de forma serial con el controlador del robot *ABB*, para después detectar las señales del volante o control de *PS4* y enviarlas al controlador en forma de bits.

Programa *Python*

```
import serial
import tkinter as tk

# Configuración del puerto
ser = serial. Serial(
    port='COM6',
    baudrate=9600,
    parity=serial. PARITY_NONE,
    stopbits=serial. STOPBITS_ONE,
    bytesize=serial. EIGHTBITS
)

#  enviar datos al Robotsin
def enviar_datos ():
    datos_a_enviar = entrada.get ()
    ser. write(datos_a_enviar.encode())
    entrada. delete(0, tk.END)  # Limpiar

# recibir datos
```

```
def recibir_datos ():
    datos_recibidos = ser. readline().decode()
    salida. insert(tk.END, datos_recibidos)

# Configuración de la interfaz gráfica
ventana = tk. Tk()
ventana. title("Comunicación Serial con Robot ABB 6400")

frame = tk. Frame(ventana)
frame. pack(padx=10, pady=10)

etiqueta = tk. Label(frame, text="Enviar datos al Robot:")
etiqueta. pack()

entrada = tk. Entry(frame, width=30)
entrada. pack()

boton_enviar = tk. Button(frame, text="Enviar", command=enviar_datos)
boton_enviar. pack()

etiqueta_recibidos = tk. Label(frame, text="Datos recibidos del Robot:")
etiqueta_recibidos. pack()

salida = tk. Text(frame, height=5, width=30)
salida. pack()

boton_recibir = tk. Button(frame, text="Recibir", command=recibir_datos)
boton_recibir. pack()
ventana. mainloop()
```

ANEXO 12 PROGRAMA EN *RAPID PARA COMUNICACIÓN SERIAL*

La segunda parte de la comunicación serial, se utiliza un programa escrito en el lenguaje *RAPID*. El programa está compuesto por un conjunto de instrucciones que describen la actividad del robot.

Este programa abre la comunicación serial, lee la cadena de texto que se envía desde la computadora hasta encontrar un asterisco, convierte esto en un valor numérico y el carácter adicional que se incluye después del asterisco es para identificar a qué articulación del robot va a aplicar el valor.

El carácter adicional consta de tres posibles opciones, (a, b o c), asignando la articulación correspondiente (rax1, rax2 o rax 5), donde rax es el eje que se va a mover. Finalmente, cierra la comunicación y mueve el robot a la posición asignada.

Programación en *RAPID*

```
MODULE MO
    VAR iodev iodev1;
    VAR BYTE letra;
    VAR STRING frase;
    VAR BOOL flag1;
    VAR NUM valor;
    VAR STRING ide;
    VAR jointtarget jpos10;

    PROC Main ()
        Open "sio1:", iodev1\Bin;
            letra: =ReadBin(iodev1);
            frase: ="";
            WHILE letra <> 42 DO
```

```
            frase: =frase+ByteToStr(letra\Char);
            letra: =ReadBin(iodev1);

        ENDWHILE
        flag1: =StrToVal(frase,valor);
        jpos10: =CJointT();
        letra: =ReadBin(iodev1);
        ide: =ByteToStr(letra\Char);
        TEST ide
            CASE "a":
                jpos10.robax.rax_1: = valor;
            CASE "b":
                jpos10.robax.rax_4: = valor;
            CASE "c":
                jpos10.robax.rax_5: =valor;
        ENDTEST
        Close iodev1;
        MoveAbsJ jpos10, v200,z50,tool0;

    ENDPROC
ENDMODULE
```

ANEXO 13. EJECUCIÓN DE MOVIMIENTOS MEDIANTE CONEXIÓN SERIAL DEL CONTROL *PS4*

Con el programa en *Python de este anexo se controlan* las opciones referentes a la activación de botones o joystick. De acuerdo con el tipo de botón pulsado, se manda una señal por vía serial que es recibida por el controlador y este la envía al programa en *RAPID*.

Si se presionan los botones X, circulo o R2, se envían caracteres específicos a través del puerto serial (1*a, 1 *b y 1*c) respectivamente, también se muestra en una ventana para saber que se está enviando.

Programación en Python

```
import pygame
import serial
import time

# Inicializar Pygame y la comunicación serial
pygame. init()
ser = serial. Serial(port='COM6', baudrate=9600, parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE, bytesize=serial.EIGHTBITS)

# Ventana Pygame
tamañov = (400, 400)
ventana = pygame.display.set_mode (tamañov )
pygame.display.set_caption ("Valores del Control PS4")

# Controlador de control PS4
control = pygame. joystick.Joystick(0)
control.init()
```

```
# Fuente para el texto
fuente = pygame.font.Font(None, 36)

def main():
    running = True
    texto = ""  # Texto para mostrar los datos enviados
    ultexto = pygame.time.get_ticks()  # Tiempo en que se mostró el texto por última vez
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        # Actualizar el estado del controlador
        pygame.event.pump()

        # Obtener valores del control
        boton_x = control.get_button(0)  # Botón X del control PS4
        boton_circle = control.get_button(1)
        boton_r2 = controlador.get_axis(5)

        # Si se presiona el botón X
        if boton_x:
            # Mandar los caracteres a través de comunicación serial
            data_to_send = b'1*a\n'
            ser.write(data_to_send)
            print(data_to_send)

            # Actualizar el texto para mostrar lo que se está enviando
            texto = "Datos enviados: {}".format(data_to_send)
```

```
            ultexto = pygame.time.get_ticks()  # Actualizar el tiempo de la última actualización del texto

# Agregar un retraso de 0.5 segundos (500 milisegundos)
            time.sleep(0.5)

        # Si se presiona el botón circulo
        if boton_circle:
            # Mandar los caracteres a través de comunicación serial
            data_to_send = b'1*b\n'
            ser.write(data_to_send)
            print(data_to_send)
            # Actualizar el texto para mostrar lo que se está enviando
            texto = "Datos enviados: {}".format(data_to_send)
            ultexto = pygame.time.get_ticks()  # Actualizar el tiempo de la última actualización del texto

            # Agregar un retraso de 0.5 segundos (500 milisegundos)
            time.sleep(0.5)
# Si se presiona el botón r2
        if boton_r2:

 # Mandar los caracteres a través de comunicación serial
            data_to_send = b'1*c\n'
            ser.write(data_to_send)
            print(data_to_send)

  # Actualizar el texto para mostrar lo que se está enviando
            texto = "Datos enviados: {}".format(data_to_send)
            ultexto = pygame.time.get_ticks()  # Actualizar el tiempo de la última actualización del texto
```

```
# Agregar un retraso de 0.5 segundos (500 milisegundos)
            time.sleep(0.5)
        # Limpiar la ventana
        ventana.fill((255, 255, 255))

        # Renderizar el texto en la ventana solo si han pasado menos de 2 segundos
        current_time = pygame.time.get_ticks()
        if current_time - ultexto < 1000:  # 2000 milisegundos = 2 segundos
            text_surface = fuente.render(text, True, (0, 0, 0))
            ventana.blit(text_surface, (20, 20))

        # Actualizar la pantalla
        pygame.display.flip()
    pygame.quit()
if __name__ == "__main__":
    main()
```

ANEXO 14. PROGRAMA EN PYTHON MODIFICADO PARA CONEXIÓN SERIAL *PYTHON*

Se modificó el código, incluido en el anexo 13 agregando condiciones para enviar los datos, se verifica el estado del botón correspondiente y el intervalo de tiempo transcurrido desde el último envío de datos, para evitar enviar datos con demasiada frecuencia y que no se sature el puerto serial.

Se agregó también el manejo de la tecla *r*, como medida de seguridad, cuando se detecta la tecla *r* se envía un comando específico, que, si lo detecta el controlador, este termina el programa en *RAPID*.

Programación de *Python*

```
import pygame
import serial
import time

pygame.init()
ser = serial.Serial(port='COM4', baudrate=9600, parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE, bytesize=serial.EIGHTBITS)

tamaño_ventana = (400, 400)
ventana = pygame.display.set_mode(tamaño_ventana)
pygame.display.set_caption("Valores del Control PS4")

controlador = pygame.joystick.Joystick(0)
controlador.init()

fuente = pygame.font.Font(None, 36)
```

```
ultimo_tiempo_envio = 0  # Variable para almacenar el tiempo del último envío de señal
intervalo_envio = 0.1  # Intervalo mínimo entre envíos de señal en segundos (por ejemplo, 0.1 segundos)

def main():
    global ultimo_tiempo_envio
    corriendo = True
    texto = ""
    ultimo_texto_actualizado = pygame.time.get_ticks()
    while corriendo:
        event = None
        for e in pygame.event.get():
            if e.type == pygame.QUIT:
                corriendo = False
            elif e.type == pygame.KEYDOWN:
                event = e
                manejar_evento_tecla_presionada(event)
            elif e.type == pygame.KEYUP:
                manejar_evento_tecla_liberada(event)

        pygame.event.pump()

       boton_l2 = controlador.get_axis(4)
        boton_r2 = controlador.get_axis(5)
        boton_circulo = controlador.get_button(1)
        eje_stick_izquierdo_x = controlador.get_axis(0)

        tiempo_actual = pygame.time.get_ticks()
        if boton_l2 > 0.9 and (tiempo_actual - ultimo_tiempo_envio >= intervalo_envio * 8000):
```

```
            datos_a_enviar = b'.01*a\n'
            ser.write(datos_a_enviar)
            print(datos_a_enviar)
            ultimo_tiempo_envio = tiempo_actual
            texto = "Datos enviados: {}".format(datos_a_enviar)
            ultimo_texto_actualizado = pygame.time.get_ticks()

        if boton_r2 > 0.9 and (tiempo_actual - ultimo_tiempo_envio >= intervalo_envio
* 8000):
            datos_a_enviar = b'.01*b\n'
            ser.write(datos_a_enviar)
            print(datos_a_enviar)
            ultimo_tiempo_envio = tiempo_actual
            texto = "Datos enviados: {}".format(datos_a_enviar)
            ultimo_texto_actualizado = pygame.time.get_ticks()
        if eje_stick_izquierdo_x == 1 and (tiempo_actual - ultimo_tiempo_envio >=
intervalo_envio * 8000):

            datos_a_enviar = b'.01*c\n'
            ser.write(datos_a_enviar)
            print(datos_a_enviar)
            ultimo_tiempo_envio = tiempo_actual
            texto = "Datos enviados: {}".format(datos_a_enviar)
            ultimo_texto_actualizado = pygame.time.get_ticks()

        # Tecla 'r'
        elif event and event.key == pygame.K_r:
            datos_a_enviar = b'1*a+\n'
            ser.write(datos_a_enviar)
            print(datos_a_enviar)
            texto = "Datos enviados: {}".format(datos_a_enviar)
```

```
            ultimo_texto_actualizado = pygame.time.get_ticks()

        ventana.fill((255, 255, 255))
        tiempo_actual = pygame.time.get_ticks()
        if tiempo_actual - ultimo_texto_actualizado < 1000:
            superficie_texto = fuente.render(texto, True, (0, 0, 0))
            ventana.blit(superficie_texto, (20, 20))
        pygame.display.flip()
    pygame.quit()
def manejar_evento_tecla_presionada(evento):
    pass
def manejar_evento_tecla_liberada(evento):
    pass
if __name__ == "__main__":
    main()
```

ANEXO 15. PROGRAMA EN EL LENGUAJE *RAPID* ACTUALIZADO

El programa en *RAPID* toma las señales que el programa en *Python* envía de forma serial y las convierte en caracteres. De acuerdo con que tipo de señal reciba, actúa de forma diferente para mover los ejes de robot *ABB*.

Dentro de una rutina *while* se establecen condiciones y acciones de acuerdo con el botón que se oprima en el control, si se ha movido el volante o apretado el acelerador o el freno.

Programación *RAPID*

```
VAR iodev iodev1;
VAR BYTE letra;
VAR STRING frase;
VAR BOOL flag1;
VAR NUM valor;
VAR STRING ide;
VAR jointtarget jpos10;

PROC Main()
        Open "sio1:", iodev1\Bin;
        WHILE letra <> 43 DO
                letra:=ReadBin(iodev1);
                frase:=""; !limpia la variable
                WHILE letra <> 42 DO
                        frase:=frase+ByteToStr(letra\Char);
                        letra:=ReadBin(iodev1);
                ENDWHILE
                flag1:=StrToVal(frase,valor);
```

```
                jpos10:=CJointT();
                letra:=ReadBin(iodev1);
                ide:=ByteToStr(letra\Char);
                TEST ide
                    CASE "a":
                        jpos10.robax.rax_4:= valor;
                    CASE "b":
                        jpos10.robax.rax_4:= valor;
                    CASE "c":
                        jpos10.robax.rax_5:= valor;
                ENDTEST
                Close iodev1;
                MoveAbsJ jpos10,v500,fine,tool0;
            ENDWHILE
        ENDPROC
ENDMODULE
```

ANEXO 16. PROGRAMA EN *PYTHON* PARA MEJORAR LA VELOCIDAD DE RESPUESTA DEL SISTEMA

El programa estaba limitado para realizar movimientos rápidos. No era posible cambiar de dirección, ya que se necesitaba terminar el movimiento anterior para realizar en siguiente.

Para solucionar esto, se modificó el código. La distancia recorrida se redujo a movimientos muy pequeños, lo que ahora permite cambiar de dirección en cualquier momento. Se incluyó un ciclo que recibe información de la conexión, de forma constante detecta el cambio de dirección.

Programación en *RAPID*

```
MODULE MO
VAR iodev iodev1;
VAR BYTE letra;
VAR BYTE signal;
VAR STRING frase;
VAR BOOL flag1;
VAR NUM valor;
VAR STRING ide;
VAR jointtarget jpos10;
VAR jointtarget targetA := [ [10, 50, 63, 20, -50, -10], [9E+09, 9E+09, 9E+09, 9E+09, 9E+09, 9E+09] ];
VAR jointtarget targetB := [ [10, 50, 63, 20, -40, -10], [9E+09, 9E+09, 9E+09, 9E+09, 9E+09, 9E+09] ];
VAR BOOL stopS;
PROC Main()
    Open "sio1:", iodev1\Bin;
    stopS := FALSE;
```

```
    WHILE letra <> 43 DO
        frase := "";
         letra := ReadBin(iodev1);
        jpos10 := CJointT();
        ide := ByteToStr(letra\Char);

        WHILE  ide = "a" DO
            jpos10.robax.rax_5:=jpos10.robax.rax_5 + 0.2;
            MoveAbsJ jpos10, v500, fine, tool0;
              letra :=ReadBin(iodev1);
              ide := ByteToStr(letra\Char);
        ENDWHILE
        IF ide = "b" THEN
              jpos10 := targetB;  ! Asigna jpos10 a targetB
                  MoveAbsJ targetB, v80, fine, tool0; ! Mueve a "b"
          ENDIF
    ENDWHILE
    Close iodev1;
ENDPROC
ENDMODULE
```

ANEXO 17. PROGRAMACIÓN DE *ARDUINO*

El proyecto en *Arduino* tiene como objetivo utilizar tres relevadores y tres botones pulsadores, para simular los botones del control de PS4. Se debe de tener en cuenta que se utilizaron los leds para simular las entradas del controlador y una fuente VCC para simular la entrada de voltaje que genera el controlador, la cual se usa para energizar el común.

Se conectan los pines IN de los módulos de los relevadores con los pines digitales de *Arduino*, igualmente se conectan los pines a tierra de los módulos del relé y los pines de VCC a los módulos del relevador de 5V del Arduino. En cuanto a los pines de NC se conectan a las entradas digitales del controlador del robot ABB y el COM, a la fuente que tiene de 24 V.

Programación en *Arduino*

```
// Definiciones
#define Rele1 2
#define Rele2 3
#define Rele3 4
#define Push1 11
#define Push2 12
#define Push3 13
void setup() {
  Serial.begin(9600);

  pinMode(Rele1, OUTPUT);
  pinMode(Rele2, OUTPUT);
  pinMode(Rele3, OUTPUT);
  pinMode(Push1, INPUT_PULLUP);
  pinMode(Push2, INPUT_PULLUP);
  pinMode(Push3, INPUT_PULLUP);
}
```

```
void loop() {
  bool RUN1 = false;
  bool RUN2 = false;
  bool RUN3 = false;

  while (1) { // Haga por siempre
    // Para Push1 y Rele1
    if (Serial.available() > 0) {
      char command = Serial.read();
      if (command == 'X') {
        delay(200); // Anti-Debounce
        while (!digitalRead(Push1)) {
          delay(200); // Anti-Debounce
        }
        RUN1 = !RUN1; // Cambia el estado lógico de RUN1
      }

      // Para Push2 y Rele2
      if (command == 'R2') {
        delay(200); // Anti-Debounce
        while (!digitalRead(Push2)) {
          delay(200); // Anti-Debounce
        }
        RUN2 = !RUN2; // Cambia el estado lógico de RUN2
      }
      // Para Push3 y Rele3
      if (command == 'C') {
        delay(200); // Anti-Debounce
        while (!digitalRead(Push3)) {
          delay(200); // Anti-Debounce
```

```
        }
        RUN3 = !RUN3; // Cambia el estado lógico de RUN3
      }
    }

    // Activa o desactiva el Rele1
    digitalWrite(Rele1, RUN1 ? HIGH : LOW);
    // Activa o desactiva el Rele2
    digitalWrite(Rele2, RUN2 ? HIGH : LOW);
    // Activa o desactiva el Rele3
    digitalWrite(Rele3, RUN3 ? HIGH : LOW);
  }
}
```

ANEXO 18. PROGRAMA EN *PYTHON* PARA ENVÍO DE COMANDOS AL CONTROLADOR *ARDUINO*

Para enviar los comandos correspondientes a las acciones del control *PS4* se usa el programa en *Python* de este anexo. Se recibe la señal del control por medio de *Pyserial*, la identifica y envía el comando correspondiente al *Arduino* de manera serial

Programación *Python*

```
import pygame
import time
import serial  # Importa la biblioteca pyserial

# Inicialización del controlador de Pygame
pygame.init()
controller = pygame.joystick.Joystick(0)
controller.init()

# Inicialización de la conexión serial con Arduino
ser = serial.Serial('COM9', 9600)  # Reemplaza 'COM9' con el puerto que esté utilizando Arduino




def main():
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
```

```
            running = False

        # Actualizar el estado del controlador
        pygame.event.pump()

        # Obtener valores del control
        button_x = controller.get_button(0)  # Botón X
        button_r2 = controller.get_axis(5)
        button_circle = controller.get_button(1)

        if button_x:
            print("El botón 'X' está presionado")
            # Envía la señal al Arduino a través del puerto serial
            ser.write(b'X')  # Envía la letra 'X' como señal
        time.sleep(0.2)  # Pequeño retraso para evitar un bucle demasiado rápido

        if button_r2:
            print("El botón 'R2' está presionado")
            # Envía la señal al Arduino a través del puerto serial
            ser.write(b'R')  # Envía la letra 'X' como señal

        time.sleep(0.2)  # Pequeño retraso para evitar un bucle demasiado rápido
        if button_circle:
            print("El botón 'C' está presionado")
            # Envía la señal al Arduino a través del puerto serial
            ser.write(b'C')  # Envía la letra 'X' como señal
        time.sleep(0.2)  # Pequeño retraso para evitar un bucle demasiado rápido
if __name__ == "__main__":
    main()
```

ANEXO 19. USO DE TELEMETRÍA EN EL VIDEOJUEGO

Para iniciar, se debe ejecutar el programa *Ets2Telemetry*, en donde aparece la conexión con *Eurotruck*, y sus enlaces. Al seleccionar el último se muestran todos los datos que se van a recabar de *Eurotruck,* como se muestra en la Figura A19-1.

Figura A19-1 Listado de todos los datos que se reciben del programa Eurotruck.

ANEXO 20. PROGRAMAS PARA CONTROL DE MOVIMIENTOS EMPLEANDO TELEMETRÍA

En esta opción, se usa como base de la opción 1 (conexión serial). La modificación requerida consiste en extraer los datos de posición directamente del videojuego y convertirlos en movimiento para el robot.

De esta manera, la computadora se convierte en el elemento principal que tiene el videojuego en ejecución y al mismo tiempo, se está ejecutando el programa de extracción de datos o (telemetría) para poder recopilar la información que el videojuego produce.

Programación en Python

```
import requests
import json
import serial
import time

ser = serial.Serial(port='COM6', baudrate=9600, parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE, bytesize=serial.EIGHTBITS)

# URL de la página web que contiene los datos JSON
url = "http://127.0.0.1:25555/api/ets2/telemetry"

while True:
    # Realizar una solicitud GET a la URL
    response = requests.get(url)

    # Verificar si la solicitud fue exitosa (código de estado 200)
    if response.status_code == 200:
```

```
        # Convertir la respuesta a un objeto JSON
        data = response.json()

        # Acceder a los datos del camión
        truck_data = data.get("truck")

        # Acceder a los datos de placement
        placement_data = truck_data.get("placement")

        # Acceder a los valores específicos dentro de placement
        x_value = str(placement_data.get("x"))
        y_value = str(placement_data.get("y"))
        z_value = str(placement_data.get("z"))
        heading_value = str(placement_data.get("heading"))
        pitch_value = str(placement_data.get("pitch"))
        roll_value = str(placement_data.get("roll"))

        # Imprimir los valores
        print("Valor de x:", x_value)
        #print("Valor de y:", y_value)
        #print("Valor de z:", z_value)
        #print("Valor de heading:", heading_value)
        #print("Valor de pitch:", pitch_value)
        #print("Valor de roll:", roll_value)

        # Escribir en el puerto serie
        ser.write(x_value.encode())  # Es importante convertir x_value a una cadena
antes de escribirlo en el puerto
        #ser.write(y_value.encode())
        #ser.write(z_value.encode())
        #ser.write(heading_value.encode())
```

```
        #ser.write(pitch_value.encode())
        #ser.write(roll_value.encode())

    else:
        print("Error al realizar la solicitud:", response.status_code)

    # Esperar n segundos antes de la siguiente iteración
    time.sleep(1)
```

Programación en *RAPID*

```
MODULE MO
    VAR num x;  ! Inicializa x con un valor constante
    VAR num y := 0;  ! Inicializa y con un valor constante
    VAR num z := 0;  ! Inicializa z con un valor constante
    VAR iodev iodev1;
    VAR BYTE data;
    VAR robtarget p20;
    VAR  bool m;
    VAR STRING frase;

    !VAR robtarget targetA := [[x, y, z, xy, xz, yz], [0, 0, 0, 0, 0, 0]];
! Tool variables:
    PERS tooldata Ensamblaje1 :=
[TRUE,[[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[1,
[0,0,20],[1,0,0,0],0,0,0.005]];
! Reference variables:
    PERS wobjdata RobotBase := [FALSE, TRUE, "",
[[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[[0,0,0],[1,
0,0,0]]];
```

```
    ! Función para mover el robot a la posición recibida
    PROC Movet()

        p20 := CRobT(\Tool:= Ensamblaje1  \WObj:=RobotBase);
        p20 := Offs(p20,x,0,0);
    ENDPROC

    ! Programa principal
    PROC main()
        ! Abre el puerto serial para la comunicación
        Open "sio1:", iodev1\Bin;

        ! Bucle principal para recibir y mover a la posición recibida
        WHILE TRUE DO
            ! Espera a recibir datos de posición por el puerto serial
            data := ReadBin(iodev1);  ! Lee los datos como byte
            frase:=ByteToStr(data\Char);!transforma data de byte a string
            m := StrToVal(frase,x);  ! transforma frase en valor y se lo da a x
            Movet;

        ENDWHILE
        ! Cierra el puerto serial al finalizar
        Close iodev1;
    ENDPROC
ENDMODULE
```

ANEXO 21 SEGUNDA OPCIÓN DE TELEMETRÍA

En este anexo, se ajusta la velocidad, 1:3 y se limita por una velocidad máxima de 200 mm/s

Programación en *RAPID*

```
MODULE MO
    VAR num x;  ! Inicializa x con un valor constante
     VAR iodev iodev1;
    VAR BYTE data;
    VAR robtarget p20;
    VAR  bool m;
    VAR STRING frase;
   CONST num MAX_VEL := 200;  ! Velocidad máxima permitida del auto virtual

    PERS wobjdata RobotBase := [FALSE, TRUE, "",
[[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[[0,0,0],[1,
0,0,0]]];
    ! Función para mover el robot a la posición recibida

   PROC Movet()
       m := StrToVal(frase, x);  ! Transforma frase en valor y se lo da a x
       x := x / 3;  ! Ajusta x proporcionalmente (1:3)
       IF x > MAX_VEL THEN
          x := MAX_VEL;  ! Limita x a la velocidad máxima permitida
       ENDIF
       p20 := CRobT(\Tool:= Ensamblaje1  \WObj:=RobotBase);
       p20 := Offs(p20, x, 0, 0);  ! Aplica el desplazamiento limitado por la velocidad

    ENDPROC
```

```
    ! Programa principal
    PROC main()
        ! Abre el puerto serial para la comunicación
        Open "sio1:", iodev1\Bin;

        ! Bucle principal para recibir y mover a la posición recibida
        WHILE TRUE DO
            ! Espera a recibir datos de posición por el puerto serial
            data := ReadBin(iodev1);  ! Lee los datos como byte
            frase:=ByteToStr(data\Char);!transforma data de byte a string
            Movet; ! Llama al procedimiento para mover el robot

        ENDWHILE
        ! Cierra el puerto serial al finalizar
        Close iodev1;
    ENDPROC
ENDMODULE
```

ANEXO 23 TERCERA OPCIÓN DE TELEMETRÍA

En este anexo se ajusta la velocidad máxima del anexo anterior a 100mm/s, también se cambia de estructura para el movimiento de los ejes del robot, ya que al utilizar el *Off*, solo se limita el movimiento a 3 ejes, este se sustituye por una estructura más completa, la estructura *trans*, que representa las coordenadas de la posición en el espacio cartesiano del robot (X, Y, Z) y también puede incluir componentes que representen ejes adicionales del robot, como e1, e2, ..., e6, cosa que el *Off* no puede hacer.

Programación en *RAPID*

```
MODULE MO
    VAR num x;  ! Inicializa x con un valor constante
     VAR iodev iodev1;
    VAR BYTE data;
    VAR robtarget p20;
    VAR  bool m;
    VAR STRING frase;
   CONST num MAX_VEL := 100;  ! Velocidad máxima permitida

    PERS wobjdata RobotBase := [FALSE, TRUE, "",
[[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[[0,0,0],[1,
0,0,0]]];

   PROC Movet()
       m := StrToVal(frase, x);  ! Transforma frase en valor y se lo da a x
       x := x / 3;  ! Ajusta x proporcionalmente (1:3)
       IF x > MAX_VEL THEN
          x := MAX_VEL;  ! Limita x a la velocidad máxima permitida
       ENDIF
```

```
        p20 := CRobT(\Tool:= Ensamblaje1  \WObj:=RobotBase);
        p20.trans.e5 := x;  ! Velocidad a la posición del eje 5
        MoveL p20, v100, fine, tool0;  ! Velocidad limitada

    ENDPROC

    ! Programa principal
    PROC main()
        ! Abre el puerto serial para la comunicación
        Open "sio1:", iodev1\Bin;

        ! Bucle principal para recibir y mover a la posición recibida
        WHILE TRUE DO
            ! Espera a recibir datos de posición por el puerto serial
            data := ReadBin(iodev1);  ! Lee los datos como byte
            frase:=ByteToStr(data\Char);!transforma data de byte a string
            Movet; ! Llama al procedimiento para mover el robot

        ENDWHILE
        ! Cierra el puerto serial al finalizar
        Close iodev1;
    ENDPROC
ENDMODULE
```

ANEXO 24. PROGRAMA EN RAPID PARA EL USO DE INTERRUPCIONES

En este anexo se incluye un ejemplo del uso de interrupciones en el lenguaje *RAPID* para el control de movimientos

```
MODULE MO

VAR robtarget jpos10;
VAR robtarget targetA := [[10, 50, 63, 20, -50, -10], [0, 0, 0, 0, 0, 0]];
VAR robtarget targetB := [[10, 50, 63, 20, -40, -10], [0, 0, 0, 0, 0, 0]];
VAR Intnum int_ent;
VAR Intnum int_ent2;
VAR BOOL movingToTargetA := FALSE;
VAR BOOL movingToTargetB := FALSE;

PERS tooldata Ensamblaje1 := [TRUE,[[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[1,[0,0,20],[1,0,0,0],0,0,0.005]];
PERS wobjdata RobotBase := [FALSE, TRUE, "", [[0.000,0.000,0.000],[1.00000000,0.00000000,0.00000000,0.00000000]],[[0,0,0],[1,0,0,0]]];


PROC Main()
   WHILE TRUE DO
      IDelete int_ent;
      Connect int_ent WITH rutina_int;
      ISignalDI d1, 1,  int_ent;
```

```
        IDelete int_ent2;
        Connect int_ent2 WITH rutina_int2;
        ISignalDI d2, 1,  int_ent2;

        WHILE d1 < 2 DO
            IF NOT movingToTargetA THEN
                movingToTargetA := TRUE;
                movingToTargetB := FALSE;
                MoveToTargetA;
            ENDIF
        ENDWHILE

        WHILE d2 < 2 DO
            IF NOT movingToTargetB THEN
                movingToTargetB := TRUE;
                movingToTargetA := FALSE;
                MoveToTargetB;
            ENDIF
        ENDWHILE

        StopMove;
        movingToTargetA := FALSE;
        movingToTargetB := FALSE;
    ENDWHILE
ENDPROC

PROC MoveToTargetA()
    ! Mover el robot al objetivo A
    jpos10 := targetA;
    MoveL targetA, v100, z1, Ensamblaje1 \WObj:=RobotBase;
```

```
ENDPROC

PROC MoveToTargetB()
    ! Mover el robot al objetivo B
    jpos10 := targetB;
    MoveL targetB, v100, z1, Ensamblaje1 \WObj:=RobotBase;
ENDPROC

TRAP rutina_int
    StopMove;
    IF movingToTargetA THEN
        RETURN;
    ENDIF
    ! Código adicional si es necesario
ENDTRAP

TRAP rutina_int2
    StopMove;
    IF movingToTargetB THEN
        RETURN;
    ENDIF
    ! Código adicional si es necesario
ENDTRAP

ENDMODULE
```