UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de un sistema de localización para el Rover UVG empleando el sistema de captura de movimiento OptiTrack

Trabajo de graduación presentado por Santiago Enrique Fernández Matheu para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de un sistema de localización para el Rover UVG empleando el sistema de captura de movimiento OptiTrack

Trabajo de graduación presentado por Santiago Enrique Fernández Matheu para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

Vo.Bo.:

(f) _____

MSc. Miguel Zea

Tribunal Examinador:

(f) _____

MSc. Miguel Zea

(f) _____

Ing. Jose Eduardo Morales

(f) _____

Ing. Kurt Kellner

Fecha de aprobación: Guatemala, 5 de enero de 2023.

Índice

Lista de figuras

Resumen

En el presente trabajo de graduación se muestra el desarrollo de un sistema de localización para el Rover UVG empleando el sistema de captura de movimiento OptiTrack e implementado mediante el "Robot Operating System" o también conocido como ROS2. El sistema de captura de movimiento OptiTrack está conformado por 6 cámaras las cuáles se encargan de capturar la pose de cualquier marker dentro de la plataforma, y luego la información de cada cámara se conecta a través de un SWITCH para luego ser enviada a una computadora y ser procesada. La información de la pose de cada marker se actualiza continuamente y se coloca en una tabla con un ID para cada marker.

El sistema creado en ROS2 se realizó mediante un nodo programado en lenguaje de Python con la ayuda de la librería *rclpy*. El nodo se encargaba de conectarse mediante la red de comunicación TCP llamada "Robotat" a la tabla con la información la pose de todos los markers, y le solicita la información de algún marker en específico. Esta a su vez le devuelve la información para que el nodo pueda utilizarla a su conveniencia. Con esto, el nodo luego de esto, procesa la información de la pose, la asigna a una variable de tipo *odometry* la cuál encapsula en un solo objeto toda la información de la pose. Luego este objeto se publica en un topic que forma parte del workspace y al cual se puede acceder para tener la información procesada.

Como resultado principal, se tiene un nodo capaz de tomar la información de la pose de cualquier marker dentro de la plataforma del OptiTrack y publicarla en un topic para su fácil acceso. Con ello, fue posible la realización de un control punto a punto y un SLAM que se encarga de realizar un mapa del entorno del Rover UVG. Esto permitió que el Rover UVG se convirtiera en un robot más autónomo que no requiriese de controles de movimiento para moverse de un lado a otro o para esquivar obstáculos, sino que es capaz de realizarlo de manera autónoma gracias al nodo de localización implementado.

Abstract

This graduation work shows the development of a location system for the Rover UVG using the OptiTrack movement capture system and implemented through the "Robot Operating System" or also known as ROS2. The OptiTrack motion capture system is made up of 6 cameras which are responsible for capturing the pose of any marker within the platform, and then the information from each camera is connected through a SWITCH and then sent to a computer and be processed. The pose information for each marker is continually updated and placed in a table with an ID for each marker.

The system created in ROS2 was made using a node programmed in Python language with the help of the *rclpy* library. The node was in charge of connecting through the TCP communication network called "Robotat" to the table with the pose information of all the markers, and it requests the information of a specific marker. This in turn returns the information so that the node can use it at its convenience. With this, the node after this, processes the information of the pose, assigns it to a variable of type *odometry* which encapsulates all the information of the pose in a single object. Then this object is published in a topic that is part of the workspace and which can be accessed to have the information processed.

As a main result, there is a node capable of taking the information of the pose of any marker within the OptiTrack platform and publishing it in a topic for easy access. With this, it was possible to carry out a point-to-point control and a SLAM that is in charge of making a map of the environment of the Rover UVG. This allowed the Rover UVG to become a more autonomous robot that does not require movement controls to move from one side to another or to avoid obstacles, but is capable of doing so autonomously thanks to the localization node implemented.

CAPÍTULO 1

Introducción

En la Universidad del Valle de Guatemala, se desarrolló un proyecto llamado "Rover UVG". Desde el 2005 que se empezó a desarrollar y conforme han pasado los años, le han implementado nuevas estructuras y características. El rover es un robot explorador modular. Durante este trabajo de graduación se muestra la implementación de un sistema de localización para dicho robot. La manera de implementarlo de manera que este pudiera manejar la información de su ubicación y que cualquier otro módulo del robot tuviera acceso fácil a ella, fue mediante un sistema operativo especializado para robótica llamado ROS (Robot Operating System). Este sistema, permmite la creación de nodos que se encargan de manejar distintas características del robot. Para el caso de este trabajo, se creó un nodo que fuera capaz de tomar la ubicación del rover y enviar dicha información a un topic para que fuera de fácil acceso para toda la demás estructura del mismo.

La manera en que se mide la ubicación es mediante un sistema de captura de movimiento denominado *OptiTrack*, el cuál consiste en un conjunto de 6 cámaras previamente calibradas que toman la información de un dispositivo llamado *marker* con tan solo capturarlo mediante las cámaras. Esta información se envía a través de un ecosistema de red TCP llamado *Robotat* que funciona como una red WiFi a través de un enrutador. El nodo creado en ROS es capaz de conectarse a la red del Robotat para recibir la información a una frecuencia de 20Hz. Y así poder brindar esta información a la computadora principal del Rover UVG. La información de la pose del Rover lleva su posición tridimensional (x,y,z) y su orientación en cuaterniones unitarios.

Finalmente se midió la latencia, precisión y exactitud de dicho sistema para evaluar su efectividad y así también poder dar recomendaciones a futuros proyectos que se realicen para el Rover UVG. Se comparó también el sistema de captura de movimiento OptiTrack con el sensor DWM1001 mediante las gráficas de 2 trayectorias, y se encontró que el OptiTrack generaba trayectorias más suaves con menos picos.

CAPÍTULO 2

Antecedentes

Para comprender el proceso que se debe llevar a cabo para esta investigación, se deben conocer las investigaciones realizadas anteriormente respecto al sistema de captura de movimiento *OptiTrack* y al Rover. A continuación se detalla más sobre cada antecedente.

2.1. Rover

Desde el 2005 [1], en la Universidad del Valle de Guatemala, se propuso un megaproyecto de un robot explorador denominado rover [2]. A partir de ese megaproyecto, se han hecho varios avances y mejoras al robot explorador. Inicialmente, se tenía la idea de que el robot fuera capaz de recolectar desechos sólidos en las playas de Guatemala. Sin embargo, a lo largo de los años se han ido realizando severas mejoras y modificaciones a dicho robot. En el año 2021 se realizó la última modificación y mejora al denominado rover. En [3] Sagastume implementó las siguientes mejoras al proyecto:

- Se utilizaron 3 controladores: Arduino, Raspberry Pi y Pixhawk, en donde el arduino fue el encargado de los sensores también implementados.
- Se implementaron los sensores: GPS, giroscopio, acelerómetro, barómetro, sensores ultrasónicos de proximidad, HC-SR04, un sensor DHT11 para medir humedad, temperatura y sensación térmica. Se realizó el respectivo código para que todos los sensores funcionaran independientemente y al mismo tiempo
- Se cambiaron los motores a unos con menor torque, mayor velocidad y más pequeños para optimizar espacio. Se diseñó la base que los une a la estructura del robot.
- Se mantuvo el sistema de oruga de las llantas por sus ventajas respecto a la movilidad del robot en terrenos complicados.

- Se rediseñó la estructura principal de robot que se adaptara al sistema de las llantas de oruga, y con el objetivo de reducir peso y altura del robot y brindar protección a los componentes electrónicos dentro del mismo.

2.2. Sistema de captura de movimiento *OptiTrack*

En [4] se detalla la importancia y los diferentes usos que se le han otorgado a los sistemnas de captura de movimiento en diferentes industrtias. Se adentra en mayor profundidad en el sistema de captura denominado *OptiTrack*, debido a que este es el que la universidad adquirió para ser utilizado dentro de los laboratorios de robótica. En dicha tesis se habla sobre la instalación, calibración y utilización del *OptiTrack*.

Como primer punto se tiene el montaje de todos los componentes, el cuál se divide en la instalación del hardware e instalación del software. Primero, se realizó la instalación del hardware empezando con el primer componente que sería la tarjeta de red. Luego se instalaron los 6 trípodes con sus respectivas cámaras. Se utilizaron abrazaderas en la colocación de las cámaras ya que permitían un mayor control en su posicionamiento e inclinación. Para tener la correcta configuración de las cámaras, se tomaron en cuenta 3 aspectos clave: la elevación, el enfoque y la inclinación. Para la elevación de las cámaras, la compañía recomendaba utilizar elevaciones altas para maximizar la cobertura de captura en volumen. El enfoque se realizó con ayuda de la visualización de la imagen en escala de grises, y se encontró el mejor enfoque girando el anillo de enfoque a favor de las agujas del reloj hasta su valor máximo y el anillo F-Stop hasta el valor de 4. La inclinación de las cámaras se realizó a 30^o, mientras que la colocación se realizó en forma rectangular para capturar con mayor cobertura el espacio posible. Por último, se instaló el switch NETGEAR, el cual sería la fuente de alimentación de las cámaras y el que permitiría obtener y leer la información de las mismas y trasladarla a la computadora. Las conexiones entre cámaras, switch y computadora se realizaron con un cable Ethernet categoría 6.

Luego se procedió a instalar y configurar el software. Para ello, se requería de la instalación de los siguientes softwares: programa Motive para captura de movimiento óptico, kit de desarrollo de software NatNet. Motive es el encargado de leer e interpretar la información que capturan las cámaras y guardar los archivos del sistema de captura de movimiento. El software NatNet se utilizó para poder usar la información del OptiTrack en otra aplicación y unirla con la red de comunicación WiFi desarrollada. El lenguaje de programación que se utilizó con el kit de desarrollo de la aplicación fue Python debido a que permitía realizarla en tiempo real. La arquitectura de NatNet funciona con un servidor y un cliente. En este caso, el servidor es el programa Motive y el cliente es la aplicación desarrollada en Python. El cliente creado en la aplicación de Python se conecta al servidor, y recibe de Motive el paquete de información que contiene 2 vectores de dimensiones 3 y 4, los cuáles indican la posición (en metros) y la orientación en formato de cuaterniones unitarios respectivamente.

Ya con el hardware y software correctamente instalados, se realizó un proceso de calibración para asegurarse de obtener medidas verídicas del sistema de captura. Para ello, se realizó un proceso enmascarado y de *wanding*. El de enmascarado elimina cualquier reflejo que pueda afectar la medición. El de *wanding* consistió en pasar un instrumento varias veces

frente a las cámaras para que estas capturen marcos de muestreo con las que puedan calcular su posición y orientación en el espacio 3D.

Ya que se tenía el sistema de captura de movimiento, se requirió realizar una red de comunicación para formar el ecosistema Robotat, de manera que fuera posible comunicarse y transmitir la información del sistema de captura de movimiento a los diferentes agentes y que estos pudieran también transmitir información importante. Para ello, se utilizó una red WiFi con la ayuda del microcontrolador ESP32. Para instalar el hardware necesario para la comunicación se utilizó un router NETGEAR MBR624GU que generara la red. Para conectar la computadora a esta red, fue necesario instalarle un adaptador WiFi a la misma y que de esa manera, estuviera en la misma red que los agentes.

Por último, la tesis explica el desarrollo un programa que se encargara de que los agentes pudieran recibir la información proporcionada por el OptiTrack. La topología utilizada para dicha comunicación fue la de bus, en donde el publicador manda la información de la pose a todos los agentes del ecosistema. Luego de esto, se descargaron todos los componentes de software necesarios para implementar la aplicación: Eclipse Mosquitto para inicializar el broker, Anaconda y la librería *paho mqtt* para implementar el protocolo MQTT. El protocolo MQTT se utilizó para la red de comunicación. En el ESP32 se desarrollaron librerías con el lenguaje de programación en C y con el SDK ESP-IDF que permitía habilitar diferentes módulos del microcontrolador. El ESP32 debía estar conectado también a la misma red que el broker para así poder recibir la información. Dicha información se reibía en formato *string*, por lo que fue necesario decodificarla en el código. Además de eso, el ESP32 también determinaba si el *id* recibido no estaba en uso y, en ese caso, se le enviaba un mensaje a la computadora indicándole que se estaría recibiendo la información mediante ese *id*. Aparte, también guarda en la memoria flash el *id* escogido, evitando así cualquier posible desconexión, permitiendo de esta manera que la librería fuera más robusta y eficiente.

CAPÍTULO 3

Justificación

El robot Rover que se ha trabajado en la Universidad del Valle de Guatemala funciona por medio de un control remoto. Los sensores implementados únicamente le permiten identificar objetos cerca del mismo, para luego poder esquivarlos mediante el movimiento a control remoto. Se pretende que el Rover sea un robot autónomo que pueda moverse independientemente y sepa identificar mediante sus sensores los obstáculos que se encuentren en su camino para así poder esquivarlos sin ayuda de un usuario. A su vez, también es necesario que el Rover sea capaz de realizar tareas en donde se mueva dentro de un espacio y pueda identificar dónde se encuentra para poder regresar al punto de origen o ir a lugares específicos.

Para ello, es necesario que el Rover tengo un sistema de localización que le permita saber su ubicación en cualquier instante de tiempo. Existen varias maneras de implementar este sistema de localización. Por ello, se desea aprovechar el sistema de captura de movimiento *OptiTrack* para implementarlo mediante marcadores debidamente colocados.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar un sistema de localización para el robot Rover UVG mediante el sistema de captura de movimiento *OptiTrack*.

4.2. Objetivos específicos

- Acoplar el sistema de captura de movimiento *OptiTrack* al Robot Operating System (ROS).
- Establecer un intercambio de comunicación adecuada entre el *OptiTrack* y la computadora principal del Rover UVG.
- Evaluar la latencia, precisión y exactitud del sistema de localización propuesto.

CAPÍTULO 5

Alcance

Este trabajo de graduación se enfoca en crear un sistema de localización para el Rover UVG mediante varios procesos para su correcta implementación. Como primer punto se tiene el desarrollo de un programa dentro del sistema operativo de ROS2. El enfoque es encontrar la manera de poder emplear el sistema de captura de movimiento en ROS2 de manera sencilla, sin que el servidor de dicho sistema necesariamente esté corriendo ROS2. Es decir, se pretende crear una conexión mediante el protocolo TCP entre el nodo de ROS y el ecosistema Robotat [4] para que ambos funcionen independiente mediante el servidor.

Un gran obstáculo que se tenía durante el desarrollo del proyecto era la poca dispoibilidad del laboratorio para realizar pruebas, ya que al trabajar con el sistema de captura de movimiento se requería trabajar en horarios con luz natural, apagar las luces y tener pocas personas dentro del laboratorio. Además, se tuvo una limitante por el tiempo en que se tardó levantar el server de Robotat. Esto más que todo debido a los atrasos causados por el departamento de IT de UVG.

CAPÍTULO 6

Marco teórico

6.1. Sistemas de captura de movimiento

Los sistemas de captura de movimiento son una tecnología que permite capturar el movimiento de personas u objetos en cierto espacio determinado, tomando cierta información como posición y/o velocidad. Esta información es procesada y enviada a otro dispositivo para su uso dependiendo del propósito. Existen muchas aplicaciones de los sistemas de captura de movimiento, tales como terapia deportiva, agricultura, cuidado de la salud, cine, videojuegos y robótica.[5]

Los primeras tecnologías implementadas de captura de movimiento fueron realizadas por el animador Lee Harrison III en los 60's, las cuáles se utilizaron en el mundo del cine animado capturando el movimiento de los actores, y procesandolos para crear animaciones. Aunque inicialmente no se tuvo tanta precisión, esta tecnología empezó a generar popularidad con los años, por lo que la tecnología comenzó a evolucionar. Ahora es utilizada en muchas otras industrias y existen varios tipos de sistemas de captura.[5]

Las principales técnicas de captura que se utilizan para desarrollar esta tecnología son las siguientes:

- **Óptico pasivo:** utiliza markers reflectivos que se colocan en los objetos a analizar. Los markers reflejan la luz emitida cerca de las cámaras y con dicho reflejo son capaces de calcular la posición y orientación de dicho marker.

- **Óptico activo:** funciona igual que el optico pasivo, pero en este caso los markers no son reflectivo, sino que ellos mismos emiten la luz que capturan las cámaras. Por esto, los markers requieren de una fuente de energía.

- **Marker-less (sin markers):** esta técnica no utiliza markers de ningún tipo, en cambio, utiliza cámaras especiales que sensan la profundiad. Por ello, pueden detectar

objetos o cuerpos en movimiento y calcular su posición y orientación. Por lo general tienen menor precisión.

- **Inerciales:** esta técnica no requiere de cámaras para funcionar, en cambio, utiliza unidades de medición inercial (IMU's por sus siglas en inglés). Las IMU's contienen sensores que miden las velocidades rotacionales de los objetos en cuestión. Los más utilizados son los giroscopios, magnetómetros y acelerómetros.

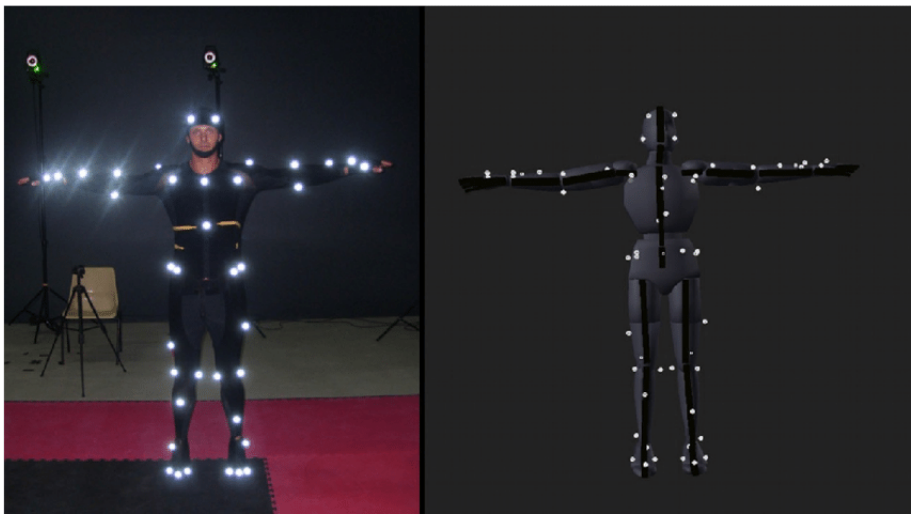

Figura 1: Captura de movimiento mediante técnica "óptico activo"[6]

6.1.1. Sistema de captura de movimiento OptiTrack

OptiTrack es un proveedor de sistemas de captura de movimiento, ofreciendo las cámaras especiales que capturan la información, así como el software que recibe y procesa dicha información. Este sistema es el que fue adquirido por el Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica de la Universidad del Valle de Guatemala. Las cámaras que se están utilizando son las $Prime^X41$. Las especificaciones de dicha cámara son las siguientes [7]:

- Resolución del sensor de imágen: 2048x2048.
- Rango de markers pasivo de 30m y de markers activos de 45m.
- Precisión 3D: $\pm$ 0.1mm.
- Latencia de 5.5ms.
- Velocidad de fotogramas de 180Hz.
- Tamaño: 12.6x12.6x13.2 cm.
- Conexiones: puerto de datos GigE, sincronización de cámara Ethernet, energía de alimentación PoE o PoE+.

En los sistemas de captura de movimiento, la colocación de las cámaras es un aspecto muy importante a tomar en cuenta. Para lograr una gran cobertura y así poder capturar el mayor volumen, las cámaras deben tener una altura suficiente. La posición del conjunto de cámaras debe de permitir capturar los markers por al menos 2 de las cámaras del conjunto. Y para evitar redundancias, la separación entre las cámaras no debe ser tan pequeña, de manera que no existan 2 o más cámaras capturando imágenes muy similares. Ya con las cámaras bien colocadas, es necesario calibrarlas mediante 2 procedimientos: *masking* y *wanding*. EL primero se utiliza para poder ignorar reflejos presentados por ventanas u otros objetos dentro del cuarto (incluso por las mismas cámaras). El segundo consiste en mover una vara de calibración frente a las cámaras que les permita calcular la posición y orientación 3D de las mismas. El software que se utiliza para guardar la información de las cámaras previamente colocadas y calibradas recibe el nombre de *Motive*.

6.2. Robot Operating System (ROS)

ROS es una plataforma de herramientas y librerías de software que permiten contruir aplicaciones para el manejo de robots. Este sistema, facilita el manejo de todos los componentes que conforman un robot, como lo son sensores, actuadores y el sistema de control. ROS permite una fácil comunicación entre cada uno de los componentes. Para ello, utiliza topics y mensajes entre cada sistema que compone al robot. Los mensajes se pueden grabar usando registros para asegurar una buena calidad, facilitar las pruebas y realizar mejoras en la comunicación. [8]

6.2.1. Workspace, paquetes y nodos

La manera de trabajar de ROS es mediante un workspace, paquetes y nodos. A continuación se explica qué es cada uno de ellos:

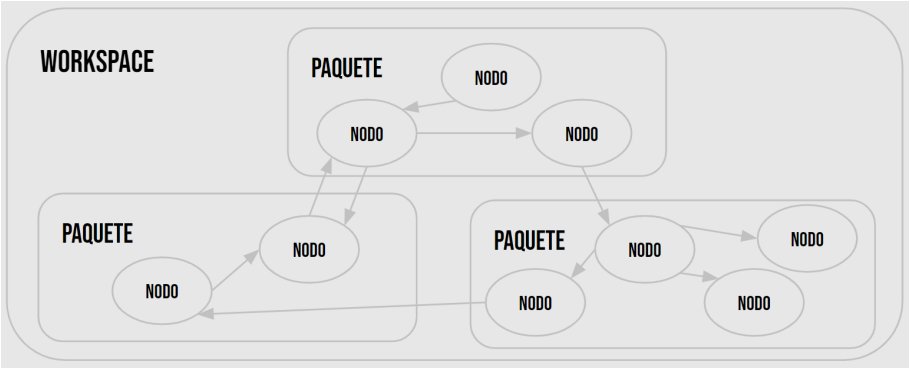

Figura 2: Diagrama de representación del Workspace, paquetes y nodo

- **Workspace:** el workspace es el área de trabajo de cualquier proyecto desarrollado en ROS. Es básicamente una carpeta en la cuál estarán incluidos todos los paquetes del proyecto, de manera que puedan tener comunicación entre sí.

- **Paquetes:** estos, como se dijo anteriormente, van incluidos dentro del workspace. Los paquetes se encargan cada uno de distintan área del robot sobre el cuál se imlementa el sistema ROS. Los paquetes pueden tener comunicación entre sí e incluyen dentro de ellos a los nodos.

- **Nodos:** los nodos, incluidos dentro de los paquetes, son la fuente del código que controlará las partes más específicas del robot en cuestión. Cada nodo realiza una sola acción importante y es capaz de comunicarse con otros nodos ya sea del mismo paquete o de otro. Estos nodos pueden publicar información en Topics. Cualquier otro nodo puede suscribirse a dichos topics para capturar la información deseada.

6.3. Protocolos de comunicación

[9]Para que exista comunicación entre 2 dispositivos a través de la red, se requiere establecer protocolos que se basan en distintas normas que permiten que la información pase de un punto a otro sin perderse o distorsionarse. Existen distintos protocolos de comunicación que se han ido desarrollando y mejorando a través de los tiempos para permitir este intercambio de información entre dispositivos. A continuación se muestran algunos ejemplos de protocolos de comunicación:

- Protocolos punto a punto
- Comunicación entre redes
- Protocolos de transmisión de paquetes
- Protocolo TCP/IP

6.3.1. Protocolos punto a punto

Son de los primeros protocolos utilizados para realizar una comunicación entre dos únicos dispositivos. Sus normas básican se basan en lo siguiente: cada ordenador tiene un papel en la comunicación y debe identificarse al comienzo de una sesión de comunicación, el que inicia la sesión se le conoce como çomandoz al que responde se le conoce como respuesta". Tienen ciertas reglas que controlan la correcta recepción de los datos, como por ejemplo colocar la suma total de BITs recibidos al final de cada mensaje. Existe un tiempo máximo que no se debe sobrepasar entre el envío de un mensaje y la recepción del mismo en el dispositivo receptor. Y finalmente se tiene un número máximo de veces que se puede repetir un mensaje en caso de que pasado el tiempo correspondiente, no se logre la comunicación deseada.

Figura 3: Protocolo de comunicación punto a punto [10]

6.3.2. Comunicación entre redes

Para este tipo de comunicaciones, se deben establecer las mismas normas del apartado anterior además de las que se explican a continuación. Si las máquinas que se están comunicando entre sí son servidores de una misma red local (LAN), entonces debe existir una forma de identificar a las terminales de la red en la cuál se está realizando la comunicación; por ejemplo, asignando un número a cada terminal. Los sistemas de polling son los que controlan la comunicación mediante una computadora central que le pregunta a cada una de las computadoras de la red, si desean realizar una sesión de comunicación y en caso afirmativo, les ordena que lo hagan.

6.3.3. Protocolos de transmisión de paquetes

Estos se basan en empaquetar la información que se desea transmitir junto con la información de receptor y destinatario. En los protocolos descritos anteriormente, el buen funcionamiento de la comunicación dependía de los equipos que enviaban y recibían la información. En este caso, el buen funcionamiento depende del propio mensaje, ya que este se organiza en paquetes como si fueran cartas de correo ordinario, en donde los paquetes van de equipo en equipo hasta que llegan a el equipo destinatario. Los equipos por donde va pasando la información únicamente tienen permitido leer las direcciones para así poder seguir transmitiendo el paquete de equipo en equipo hasta llegar a su destino.

CAPÍTULO 7

Levantamiento del server de Robotat

A continuación se muestra un diagrama de lo que se realizó para el levantamiento del robotat y con el cuál se puede explicar el funcionamiento del mismo. En el desarrollo de este capítulo, se explicará detalladamente cada parte del diagrama.

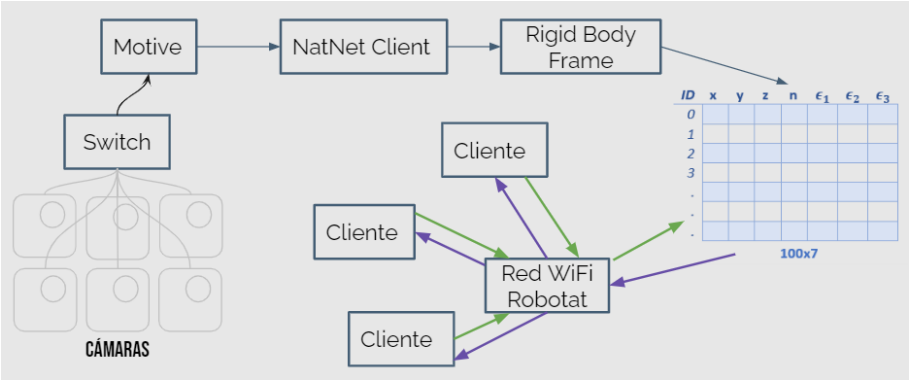

Figura 4: Diagrama de flujo para el levantamiento del server de Robotat

Para que la computadora principal del Rover UVG pudiera recibir la información de su pose, fue necesario cumplir con algún sistema de comunicación con el cuál se pudiera transmitir dicha información. Por ello, se utilizó el ecosistema de Robotat. Este ecosistema es una red de comunicación Wifi que funciona por medio de un router NETGEAR MBR624GU. El router opera a una banda de frecuencias de 2.4 GHz y utiliza algoritmos de encriptación WPA2-PSK. El nombre de esta red es *Robotat* y la contraseña es *iemtbmcit116*.

La información de la pose de los markers que captura cada una de las 6 cámaras que conforma el OptiTrack, pasa a través de cables que se conectan a un switch (Figura 5). El switch funciona como una clase de unificador. Reúne la información de varias entradas (las

cámaras) y la junta en un solo bus de datos mediante un cable que luego se conecta a la computadora principal del cit-116.

Figura 5: Switch NETGEAR ProSafe GS728TPPv2 y conexión de cámaras (en negro) y computadora (en azul).

Tomado de [4]

La computadora que se conectó directamente con el Optitrack, también se conectó a la red del Robotat mediante un adaptador WiFi, de manera que estuviera en la misma red de los agentes.

En la computadora se recibe la informacion de todas las cámaras y se procesa mediante el programa Motive. La información capturada incluye la pose de cada cuerpo rígido, posición de cada punto que conforma cada cuerpo rígido, información de tiempos de procesamiento, etc. Lo que se necesitaba capturar era únicamente la información de la pose de cada cuerpo rígido, por lo que Motive luego de capturar todos los datos, realiza un streaming de los mismos y estos son procesados por una función llamada "NatNetClient". Esta es una función que viene dentro del software del OptiTrack y es la que se encarga en decodificar, procesar, filtrar y ordenar toda la información que se recibe del streaming de Motive. Con esto, se publican únicamente los datos que se requieran según la configuración de la función. En este caso, únicamente se necesitaba tener la pose de los cuerpos rígidos.De esta manera, ya no se estaría utilizando toda la información que no se necesitaba.

Luego de que la información de la pose de todos los markers dentro de la plataforma es procesada por "NatNetClient", pasa por un proceso llamado "Rigid Body Frame". Este es el proceso en el cual se genera la tabla mostrada en la Figura 4. Todos los datos de la pose de cada cuerpo rígido asociado a cada marker se van colocando en dicha tabla continuamente. La información capturada por la tabla incluye el número de ID que identifica a cada marker (va del 1 al 10), la posición tridimensional (x,y,z) y la orientación en cuaterniones unitarios $(n, \epsilon_1, \epsilon_2, \epsilon_3)$ dentro del espacio cerrado entre las 6 cámaras dentro de la plataforma del OptiTrack.

Como se dijo anteriormente, la tabla se actualiza continuamente con la información

correspondiente para cada cuerpo rígido, por lo que si un cuerpo rígido (marker) dentro de la plataforma se mantiene en movimiento, entonces la tabla estará cambiando los valores para dicho marker conforme se va moviendo. La frecuencia a la cuál se actualizan los datos es de aproximadamente 120Hz. Luego, mediante la red de comunicación TCP del ecosistema Robotat, es posible transitir la información de dicha tabla a cualquier cliente que lo desee. Se establece entonces mediante el servidor del Robotat un socket en el puerto 83. Se asocia dicha conexión con la tabla de manera que el servidor recibe un array conformado por la lista de los ID's correspondientes a los markers que el cliente desea conocer su pose. Luego, genera un array conformado por la pose de cada uno de los markers solicitados (uno seguido de otro), por lo que el tamaño del array es 7n, donde n es la cantidad de markers solicitados. Se codifica dicho array en formato json para poder ser enviado mediante el socket a través del ecosistema robotat al cliente que lo solicitó.

La manera en que *Motive* publica los datos no siempre es la misma que como los muestra en el programa. Se tomó el eje vertical como el eje y, y los ejes del plano horizontal como x y z.

En resumen, lo que realiza el programa *mocap_ server* es lo siguiente.

1. Se conecta con NatNet para recibir la información de las poses de los cuerpos rígidos.
2. Coloca toda la información en tablas para cada uno de los cuerpos rígidos, con identificación para cada uno.
3. Se conecta mediante las librerías de *socket* al servidor del *Robotat*.
4. Recibe la información enviada por los agentes con el listado de markers de los cuales tiene que enviar la pose.
5. Captura la pose de los markers solicitados por el agente a partir de la tabla de datos. Y los codifica para poder ser enviados.
6. Envía los datos de la pose al agente que los solicitó, identificándolo mediante su IP.

Todo esto, se realizó mediante un método de código *multi-threading*. De manera que se tiene un hilo por cada agente que desee conectarse, y el programa pueda enviarle la información requerida a cada agente al mismo tiempo, sin ningún problema. Esto implica que se pueden conectar varios clientes al servidor de manera simultánea.

CAPÍTULO 8

Desarrollo del nodo en ROS2

8.1. Instalación y configuración de ROS2

Lo primero que se tuvo que hacer para poder realizar un nodo en ROS2 fue instalar una máquina virtual, ya que este sistema presenta una funcionalidad completa en Linux. Existen diversas máquinas virtuales que pueden ser útiles para este proyecto, sin embargo, la máquina utilizada y que presentó menores problemas fue la Oracle VM VirtualBox [11]. Para la creación de la máquina virtual se tomaron en cuenta las siguientes características:

Sistema operativo	Linux
Versión	Ubuntu (64-bits)
Imagen	Ubuntu 20.04
Memoria	4096 MB
Procesadores	3 CPUs
Memoria de video	16 MB

La versión actual, con soporte, de ROS2 se va actualizando cada cierto tiempo. La versión de ROS2 que se utilizó para este trabajo fue ROS2 Foxy. Para ello, se requería instalar en la máquina virtual la imagen de la versión 20.04 de Ubuntu. La instalación de ROS2 se realizó en la terminal de comandos siguiendo los comandos dados por la página oficial para la instalación de ROS2 Foxy [12].

Para poder utilizar ROS2 fue necesario realizar una operación en la terminal de comandos dentro de la carpeta de Foxy ubicada en `/opt/ros/foxy` en la cual se encuentran todos los archivos correspondientes a ROS Foxy. Dentro de esta carpeta se encuentra un archivo llamado setup.bash. Para utilizar ROS2 en la terminal, se debe hacer un `source` a dicho archivo de la siguiente manera:

```
$ source /opt/ros/foxy/setup.bash.
```

Esto se debía realizar cada vez que se abría una nueva ventana de la terminal de comandos, de lo contrario, no se podía utilizar ROS2 en dicha terminal. Para poder omitir este paso cada vez que se abría una terminal, y poder utilizar todas las funciones de ROS2 sin problema, se realizó lo siguiente. Se abrió el código fuente *.bashrc* que se ejecuta cada vez que se abre la ventana de comandos y determina como este se mira y funciona. Al final del código, se agregó la línea desrita anteriormente de manera que cada vez que se abre una ventana de la terminal, se ejecuta dicha instrucción y se pueden utilizar las funciones de ROS2 sin problema.

Ahora para poder crear el primer programa de ROS2, fue necesario tener una herramienta para levantar el programa que fuera específica para ROS2. Esta herramienta se llama *colcon* y se instaló en la consola mediante el comando

```
$ sudo apt install python3-colcon-common-extensions
```

Esta herramienta permitió crear todos los elementos necesarios para el programa de ROS, como lo son el workspace, los paquetes y los nodos.

8.2. Creación del workspace, paquetes y nodos

Lo primero que se tuvo que hacer antes de crear el primer programa de ROS2 fue crear un espacio de trabajo (*workspace*). Dentro de este estarían todos los paquetes y nodos que controlarían al Rover UVG. La creación del espacio de trabajo fue sencilla. Se ejecutó el comando `mkdir ros2_ws` dentro del directorio principal en la línea de comandos de la terminal. Esto generó un nuevo folder llamado *ros2_ ws*. Dentro de este folder, se creó un nuevo folder llamado *src* el cual contendría todos los códigos y paquetes mencionados anteriormente. Para ese momento, estos folderes no significaban nada en ROS2, para ello, se utilizó la herramienta *colcon* previamente instalada para poder definir este folder como el correspondiente *workspace* de ROS2. Se ejecutó el comando `colcon build` dentro de la carpeta *ros2_ ws* para realmente crear el espacio de trabajo.

Luego de crear el espacio de trabajo, se debía crear dentro del mismo, el paquete con el que se estaría trabajando la localización del rover UVG con el Optitrack. El paquete se creo dentro del folder *src* en el espacio de trabajo, como se mencionó anteriormente. Se ejecutó el siguiente comando:

```
$ ros2 pkg create rover_location_optitrack_pkg --build-type ament_python
--dependencies rclpy
```

Dicho comando, creó un paquete de ROS2 dentro del espacio de trabajo con el nombre *rover_ location_ optitrack_ pkg* y con lenguaje de programación asociado de *Python*. Esto quiere decir que los nodos creados dentro de este paquete se debían programar en lenguaje *Python*. Se incluyó la dependencia *rclpy* que asocia el código de *Python* con ROS2, de

manera que este se pueda compilar. Al crearse el paquete, se incluyó dentro de la carpeta del *WorkSpace*, una carpeta con el nombre del paquete recién creado. Dentro de dicho paquete, se tiene el siguiente directorio:

```
package.xml resource rover_location_optitrack_pkg setup.cfg setup.py test
```

El archivo *package.xml* lleva las dependencias que tiene el nodo, es decir, las librerías que necesita el nodo para funcionar.

```
1  <?xml version="1.0"?>
2  <?xml-model href="http://download.ros.org/schema/package_format3.xsd"
3  <package format="3">
4    <name>rover_location_optitrack_pkg</name>
5    <version>0.0.0</version>
6    <description>TODO: Package description</description>
7    <maintainer email="fer18171@todo.todo">fer18171</maintainer>
8    <license>TODO: License declaration</license>
9
10   <depend>rclpy</depend>
11   <depend>socket</depend>
12   <depend>json</depend>
13   <depend>nav_msgs</depend>
14   <depend>geometry_msgs</depend>
15
16
17
18   <test_depend>ament_copyright</test_depend>
19   <test_depend>ament_flake8</test_depend>
20   <test_depend>ament_pep257</test_depend>
21   <test_depend>python3-pytest</test_depend>
22
23   <export>
24     <build_type>ament_python</build_type>
25   </export>
26 </package>
27
```

Figura 6: Código del archivo package.xml con dependencias del nodo.

Como se observa en la Figura 6, se colocan las dependencias en un lenguaje de programación HTML. En este caso, el nodo de ROS para este proyecto requería de las siguientes librerías: rclpy, socket, json, nav_msgs y geometry_msgs.

El archivo *setup.py* es donde se colocan los ejecutables. Es decir, los archivos de códigos que se crearon para ser ejecutados. En este caso, fue necesario colocar en este código, el nombre del nodo que se estaría ejecutando. Se utilizó el mismo nombre para el ejecutable que para el nodo. Se ejemplifica en la línea 23 del código mostrado en la Figura 7:

Como se observa en el código, para generar el ejecutable, primero se colocó el nombre del ejecutable (es decir, el nombre con el cual se puede accionar el nodo desde la consola). En este caso, se nombró "optitrack_client", igual que el nombre del nodo. Luego se colocó un signo "=" para luego indicarle el nodo al cuál se deseaba asociar el ejecutable. Para nombrar el nodo, primero se indicó el nombre del paquete, seguido por el nombre del nodo dentro de dicho paquete.

Para crear el nodo principal de ROS en el cual se tiene todo el código base de este proyecto, se ubicó el directorio con el mismo nombre del paquete, dentro del paquete recién creado en la línea de comandos de la terminal. Es decir, se realizó la siguiente instrucción:

```
$ cd /ros2_ws/src/rover_location_optitrack_pkg/rover_location_optitrack_pkg
```

```python
from setuptools import setup

package_name = 'rover_location_optitrack_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='fer18171',
    maintainer_email='fer18171@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "optitrack_client = rover_location_optitrack_pkg.optitrack_client:main"
        ],
    },
)

```

Figura 7: Código del archivo *setup.py* con nombres de ejecutables

Ya dentro del directorio del paquete, se ejecutó la siguiente instrucción:

```
$ touch optitrack_client.py
```

Dicha instrucción, creó un archivo de Python en el folder del paquete. Sin embargo, este archivo y cualquier código que se colocara en él, no se podría ejecutar ya que aun no era un archivo ejecutable. Para convertir el archivo en ejecutable, se colocó la siguiente instrucción en la línea de comandos:

```
$ chmod +x optitrack_client.py
```

Con esto último, ya se tenía creado el nodo de ROS encargado de tomar la ubicación del Rover UVG mediante el sistema de captura de movimiento *OptiTrack*. Ya solo faltaría generar el código para que este funcione.

8.3. Programa para el nodo de localización del Rover UVG

8.3.1. Formato del código y librerías utilizadas

El código correspondiente para el nodo se realizó en el lenguaje de Python y se utilizó el método de Programación Orientada a Objetos. Además, se utilizaron las siguientes librerías para su buen funcionamiento (se explica también lo que realiza cada una de ellas):

- **rclpy**: esta librería es el API de Python en ROS2. Permite programar un nodo de ROS2 en Python.

- **socket**: permitió crear una conexión con la red y de esa manera, conectar el programa con el ecosistema Robotat.
- **json**: esta librería permitió codificar y decodificar los mensajes enviados por la red mediante un formato específico.
- **odometry**: con esta librería se pudo encapsular toda la información de la pose del cuerpo rígido, para ser publicada en un topic de manera más fácil y compacta.

8.3.2. Explicación del código

Código básico para un nodo de ROS

Para realizar un nodo de ROS en Python, se necesitó utilizar la librería rclpy, como se explicó anteriormente. El código básico para el main fue el siguiente:

```
def main(args=None):
    rclpy.init(args=args)
    node = OptiTrackNode()
    rclpy.spin(node)
    rclpy.shutdown()

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

En el código anterior, se definió primero la función del main con ningún argumento (args=None). Luego, se inicializó la conexión entre ROS y Python con rclpy.init, nuevamente sin argumentos. La siguiente línea fue la encargada de crear el nodo, asociándolo con una clase. En este caso, se asoció con la clase `OptiTrackNode()` que se explicará más adelante. Luego de crear el nodo y asociarlo a la variable `node`, se utilizó la función *spin* con el argumento *node* para que el programa ejecute el código de la función asociada al nodo de manera continua. Esto en la línea de comandos de la terminal hace que se ejecute el código y si en dado caso se terminan de ejecutar todas las instrucciones del mismo, la línea de comandos se queda atascada. Esto sucede porque el programa sigue ejecutando la instrucción *spin* aunque el nodo ya no esté realizando ninguna instrucción. Para dejar de ejecutar la instrucción *spin* en la línea de comandos, se presiona la combinación de teclas `ctrl+C`. Al dejar de ejecutarse dicha instrucción, lo siguiente en ejecutarse es la función *shutdown*, que desactiva el nodo y por ende se habilita de nuevo la línea de comandos.

Código asociado al nodo

Luego de escribir el código base para crear el nodo, se realizó el código de la clase asociada al nodo. Se creó la clase de la siguiente manera:

```
class OptiTrackNode(Node):
```

En donde se definió el nombre del nodo y se le atribuyó el objeto *Node* proveniente de la librería *rclpy*. Dentro de la clase se definió la función `__init__(self)`, que funcionó como

si fuera el main (es decir, es el cuerpo principal del código). Este se ejecuta al realizar un spin del nodo. A continuación se muestra el código y la explicación del mismo.

```
def __init__(self):
    super().__init__("optitrack_connection")
    # Start sochet connection with Robotat
    self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.s.connect(('192.168.50.201',1883))
    self.fullMsg = self.s.recv(1024).decode("utf-8")
    self.get_logger().info(self.fullMsg) #Verify Robotat has
    connected

    #Create publisher to publish the data in a topic
    self.publisher_=self.create_publisher(Odometry, "/OptiTrack", 50)
    self.timer_ = self.create_timer(0.1,self.publish_coordinates)
    self.get_logger().info("The number publisher has been started")
```

En la primera línea del código, se llamó a la función `__init__` del objeto *Node* y se le colocó el nombre del nodo, que en este caso es *optitrack_ connection.*

La siguiente parte del código se encargó de realizar la conexión con la red del ecosistema Robotat mediante la librería *socket*. Para ello, se creó un objeto de tipo socket denominado `self.s` y luego se realizó la conexión de dicho socket con el Robotat mediante la función `connect(IP, port)`. El primer parámetro de la función es la IP de la red a la cuál se quiere conectar, en este caso, la IP es 192.168.50.201 correspondiente al Robotat. El segundo parámetro es el puerto, que se definió como 1883. Luego de conectarse correctamente con el servidor de Robotat, este automáticamente envía un mensaje para confirmar que la conexión se realizó con éxito. Realmente esa información no es relevante y por lo tanto se requiere eliminarla del buffer de entrada. Por ello, se ejecuta la instrucción `recv(size)` la cual lee los datos del buffer con la cantidad de bits indicada por el parámetro *size*, luego lo decodifica por el formato *utf-8* y lo guarda en la variable `self.fullMsg`. Luego, para verificar que realmente se realizó la conexión con el Robotat, se imprime dicho mensaje en la terminal mediante la instrucción `self.get_logger().info(message)`. Esta función imprime el texto colocado en el parámetro *message* como información en la terminal.

Luego de tener el nodo conectado exitosamente con la red del Robotat, y de haber vaciado el buffer de entrada, se prosiguió a crear el publisher. Este estaría publicando en una topic, los datos de la pose recibidos desde el socket. La función utilizada fue la siguiente:

```
self.create_publisher(msg_type, topic, qos_profile)
```

El primer parámetro es el tipo de mensaje que se estará publicando con dicho publisher. En este caso, como se explicó anteriormente, el tipo de mensaje se llama *Odometry*. El parámetro *topic* es el nombre del topic en el cuál se estará publicando el mensaje para que luego un suscriptor pueda suscribirse a dicho topic con ese mismo nombre y recibir la información. El último parámetro es la profundidad del buffer para el publisher.

Luego se creó un timer, el cuál estaría ejecutando una función cada cierto tiempo. El tiempo establecido fue de 0.1 segundos, ya que se tiene que la frecuencia a la cual se capturan los datos del optitrack es de aproximadamente 100Hz, y tomando en cuenta los tiempos

perdidos al transmitir el mensaje mediante la red y demás se puede tomar una frecuencia de 10Hz, que equivale a un período de 0.1 segundos. La función asociada al timer se denominó `self.publish_coordinates` y se explica a continuación.

Función encargada de recibir y publicar la pose del marker

El título de esta sección hace referencia a que la pose se recibirá mediante el socket proveniente desde la red del robotat y luego se publicará en el topic de ROS para que la información pueda ser extraída por otros nodos del Rover UVG que lo requieran.

Como se explicó en el capítulo anterior, el server del ecosistema Robotat tiene la información de los cuerpos rígidos correspondientes a todos los markers dentro del Optitrack. Sin embargo, para este nodo, solo fue necesaria la información de 2 cuerpos rígidos: el del cuerpo central del Rover UVG, y el del efector final de la mano robótica creada en la tesis de Rodrigo Díaz. Por ello, el servidor de Robotat no puede enviar información sin antes recibir un listado de los ID's correspondientes a los cuerpos rígidos de los cuales tiene que enviar la información de su pose. Esta información debía enviarse en formato *json*, de manera que fuera más fácil de enviarse y que el programa del servidor pudiera interpretarla correctamente. Luego de pedirle la información de 2 markers en específico, el server de Robotat envía dicha información en una lista con 14 valores. Los primeros 7 correspondientes al primer marker y los otros 7 al segundo marker. De los 7 valores de cada marker, los primeros 3 son la ubicación (x,y,z) del marker y los otros 4 son la orientación como cuaterniones unitarios. Esta información es enviada también en formato *json*, por lo que es necesario decodificar dicha información para poder ser utilizada. Esto se realiza mediante la función `json.loads()`. A continuación se muestra el código correspondiente que realiza el proceso recién explicado:

```
def publish_coordinates(self):
        #Request the server the coordinates from markers 1 and 2
        self.markerNumbers = [2,5]
        self.s.sendall(json.dumps(self.markerNumbers).encode())

        #Receive the coordiantes
        self.coordinates = self.s.recv(1024).decode("utf-8")
        self.jsonCoordinates = json.loads(self.coordinates)
        self.M1_Pos = self.jsonCoordinates[0:3]
        self.M1_Ori = self.jsonCoordinates[3:7]
        self.M2_Pos = self.jsonCoordinates[7:10]
        self.M2_Ori = self.jsonCoordinates[10:13]
```

Del código mostrado, la variable `self.markerNumbers` es la que guarda los ID's de los markers requeridos en una lista. Luego la función `self.s.sendall` (correspondiente al objeto *s*), es la encargada de enviar la variable `self.markerNumbers` mediante el socket al servidor Robotat. Como se explicó anteriormente, esto se hizo en formato *json*. Por ello, se utilizó la función `json.dumps().encode()`. Luego de que el Robotat envía la información requerida, esta se guarda en el buffer de entrada del socket. Para guardarla en una variable y poder utilizarla, se creó primero la variable `self.coordinates`. La función utilizada de la librería socket para asignar la información recibida a la variable `self.coordinates`, fue `self.s.recv(size).decode('utf-8')`. El parámetro de *size* se colocó en 1024 bits debido a

que la información recibida ocupaba un espacio de bits menor a dicha cantidad, y con ello se aseguraba recabar la totalidad de la información requerida. Como se explicó anteriormente, esta información se recibía en formato *json*, por lo que era necesario decodificarla. Para ello, se definió la variable `self.jsonCoordinates` a la cuál se le atribuyó el resultado de la función `json.loads()` aplicada a la variable `self.coordinates`.

Ya que se tenía toda la información decodificada, se separó en diferentes variables para posición y orientación del marker 1 y del marker 2. Finalmente, para poder publicar toda esa información en el topic, se creó el objeto de tipo Odometry y se guardó en él, toda la información requerida para la pose del primer marker. A continuación se muestra el código utilizado:

```
#Start Odometry
odom=Odometry()
odom.header.frame_id = "odom"

# set the position
odom.pose.pose.position.x = self.M1_Pos[0]
odom.pose.pose.position.y = self.M1_Pos[1]
odom.pose.pose.position.z = self.M1_Pos[2]
odom.pose.pose.orientation.x = self.M1_Ori[0]
odom.pose.pose.orientation.y = self.M1_Ori[1]
odom.pose.pose.orientation.z = self.M1_Ori[2]
odom.pose.pose.orientation.w = self.M1_Ori[3]
```

Como se puede observar en el código anterior, el objeto odom creado a partir de Odometry tiene los parámetros para la posición en x, y y z; y para la orientación en cuaterniones unitarios. Toda esa información se encapsula en ese objeto llamado *odom* de tipo *Odometry* para poder ser publicado en el topic.

El código para publicar dicho objeto en el topic con el publisher que se creó anteriormente es:

```
self.publisher_.publish(odom)
```

Esta toma el objeto publisher que se había creado y que ya está asociado al topic en donde se desea publicar la información, y luego llama a la función publish con el parámetro del objeto que se desea publicar. Que en este caso es el objeto *odom*.

Luego de implementar el código explicado en esta sección, se tendría el nodo completo para poder solicitar y recibir la información de la pose del Rover UVG. En la siguiente sección, se detalla la manera en que puede correrse el programa del nodo en la terminal para poder verificar su funcionamiento y recibir la pose del Rover UVG desde la consola.

8.4. Resultados del nodo en la consola de comandos

8.4.1. Requerimientos previos

Dado que se trabajó mediante una red de comunicación con un protocolo TCP dentro del ecosistema de Robotat, para poder hacer funcionar el código del nodo fue necesario cumplir con cierto requerimiento de red dentro de la máquina virtual. A continuación se explica el requerimiento y cómo se pudo cumplir con él dentro de la máquina virtual.

El ecosistema Robotat funciona como una red de internet local. Por lo tanto, para poder acceder a él y establecer una conexión de cualquier tipo con el ecosistema, fue necesario lo siguiente: que el servidor del ecosistema estuviera encendido y que la computadora principal del Rover UVG estableciera una conexión con dicha red. Esto quiere decir que para correr el nodo en cualquier computadora, se debe activar el servicio WiFi y conectarse con el enrutador del ecosistema Robotat. El nombre de la red, como se mencionó en el capítulo anterior, es *Robotat*. Y la constraseña para poder acceder a él es *iemtbmcit116*. Sin embargo, como se utilizó una máquina virtual para poder utilizar ROS2 y realizar el nodo dentro de ella, es necesario que la red se conecte tanto en la máquina física como en la virtual. Para ello, se debe seleccionar una configuración específica en la máquina virtual, de manera que esta se conecte directamente a la misma red que la máquina física en donde se está corriendo esta. Para poder seleccionar dicha configuración, se realizaron los siguientes pasos:

1. Se abrió el programa de la máquina virtual *Oracle VM virtual box* dando doble click en el ícono del escritorio.
2. A continuación, se abrió la ventana del *Oracle Manager* como se muestra en la Figura 8:

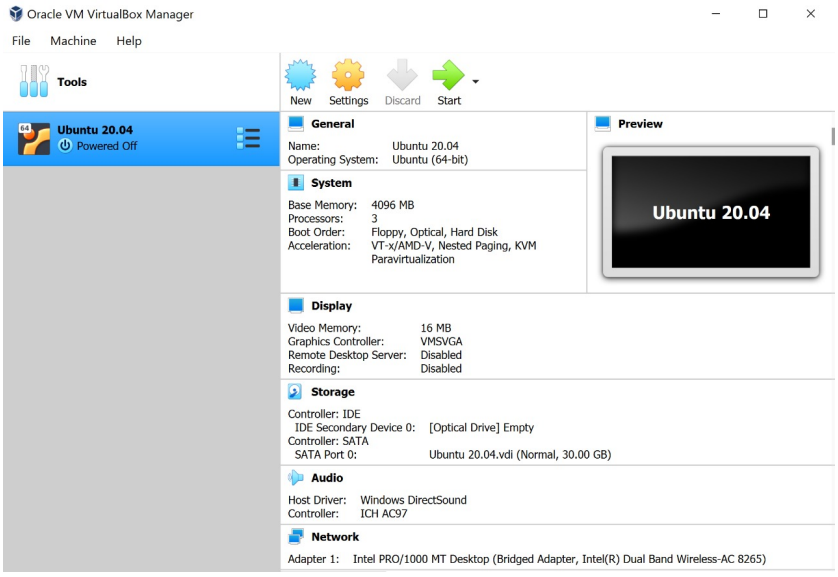

Figura 8: Ventana del Oracle Manager

3. Se seleccionó la máquina virtual en la que se trabajó el presente trabajo de graduación. Esta tiene el nombre “Ubuntu 20.04”.

4. Antes de ingresar a la máquina virtual, se seleccionó la opción *Setting* representada con un ícono de tuerca amarilla.

5. Se abrió la ventana de *Settings* en donde se seleccionó la pestaña de *Network* como se muestra en la Figura 9:

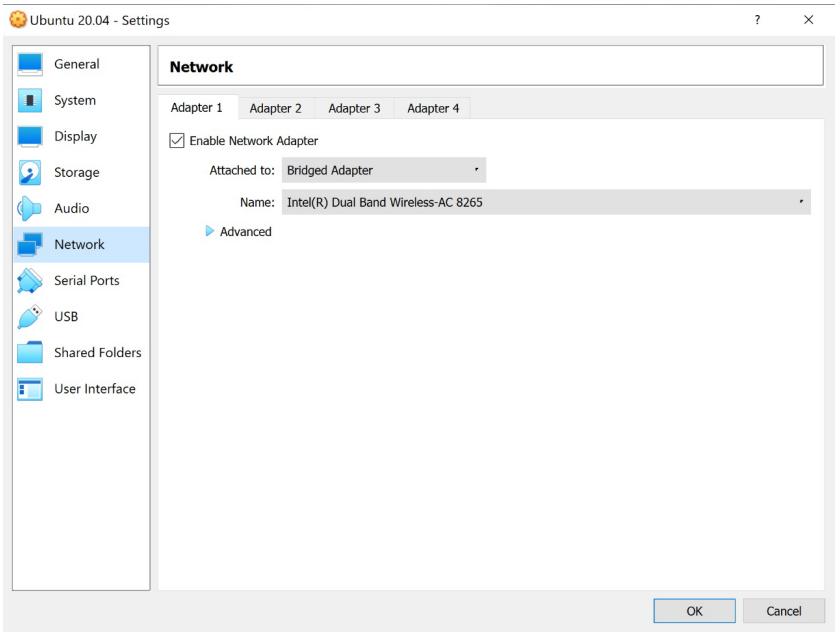

Figura 9: Ventana de *Settings* para la máquina virtual *Ubuntu 20.04*

6. Luego, se seleccionó dentro de la pestaña *Adapter 1* la opción *Attached to* y se mostraron las siguientes alternativas:

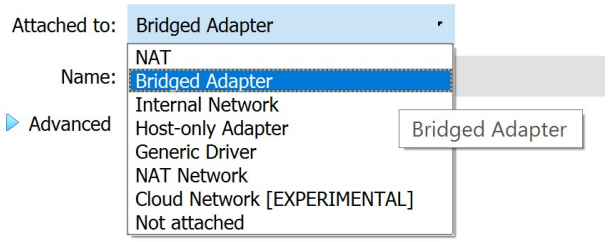

Figura 10: Alternativas del adapter para la conexión de red de la máquina virtual

7. De todas las alternativas mostradas, la que nos permitiría crear un puente entre la conexión de red de la máquina física con la máquina virtual era *Bridged Adapter*. De manera que la misma red WiFi que esté conectada en la máquina física, será la que se conectará en la máquina virtual. Se seleccionó dicha alternativa.

8. Ya que se tenía la configuración correcta de red, se cerraron todas las ventanas de configuración y se inició de nuevo la máquina virtual.

8.4.2. Resultados en la consola de comandos

Para observar los resultados de lo que realiza el nodo desde la consola de comandos, se puede utilizar la siguiente instrucción:

```
$ ros2 run pkg_name node_name
```

Dicha instrucción corre el nodo *node_ name* que se encuentra dentro del paquete *pkg_ - name*. El nodo trabajado dentro de este trabajo de graduación tenía el nombre *optitrack_ - client* y se encontraba dentro del paquete *rover_ location_ optitrack_ pkg*. Por lo tanto, la instrucción dada para correr el nodo en la consola quedó de la siguiente manera:

```
$ ros2 run rover_location_optitrack_pkg optitrack_client
```

Al enviar dicha instrucción, el nodo creado y explicado en la sección anterior empezó a correr. Lo primero que realizó fue establecer la conexión con el servidor del Robotat. Para indicar que la conexión se pudo establecer de manera correcta, se imprimió el siguiente mensaje en la consola:

```
Connected to the Robotat server... Type EXIT to stop
```

Lo cual indicaba que la conexión se había realizado con éxito. Lo siguiente que se realizaba en el código de manera continua, era pedirle al Robotat la pose de 2 markers en específico y luego de recibir dichas poses, empaquetar la pose del cuerpo central del Rover en un objeto de tipo Odometry y enviarlo mediante un publisher a un topic con el nombre de "OptiTrack". El código no tiene instrucciones de imprimir nada más en la consola de comandos. Por lo tanto, lo único que está haciendo el nodo es publicar la información en el topic. Cualquier otro nodo que requiriese la información, podía suscribirse a dicho topic para tomar la información y realizar con ella lo que necesitase. Por el momento, lo único que estaba activo era el nodo "optitrack_client". El topic, para activarse, debía tener por lo menos un *publisher* y un *subscriber*. Una manera gráfica y sencilla de observar los nodos que están activos es mediante la herramienta `rqt`. Dicha herramienta abre una ventana con distintas funcionalidades para visualizar información de los paquetes, nodos, topics, etc. Al momento de enviar el comando `rqt` en la consola, se abre la siguiente ventana mostrada en la Figura 11:

Para abrir el diagrama de los nodos y topics activos, se seleccionó la pestaña *plugins>introspection>node graph*, y con esto se abrió una ventana gráfica con los nodos y topics activos y la manera en que estos se relacionan.

Para poder suscribirse al topic de una manera sencilla y así recibir los datos publicados en este, se puede utilizar la siguiente instrucción:

```
$  ros2 topic echo /wheel/odometry
```

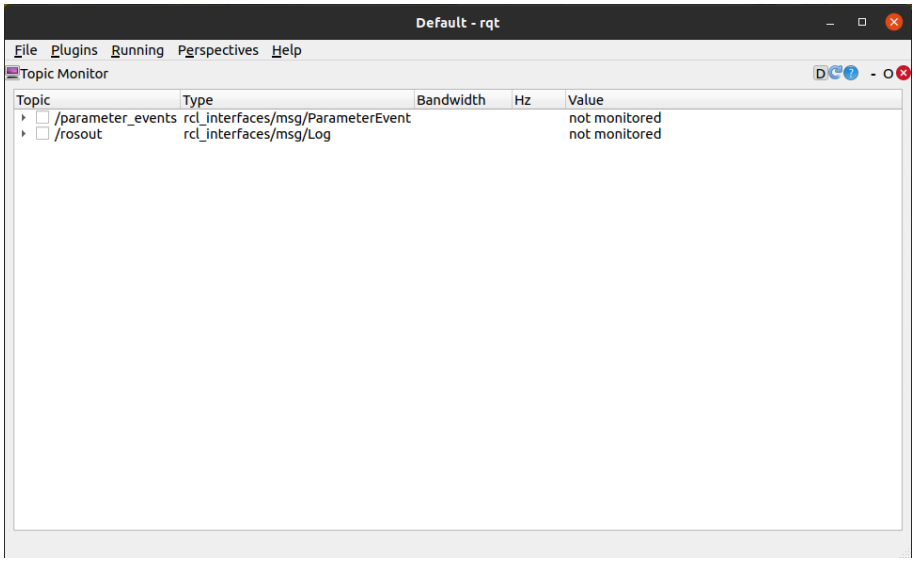

Figura 11: Ventana rqt

Esto comienza a imprimir en la consola de una manera muy desordenada, toda la información contenida dentro del objeto de tipo *odometry*. Como este objeto contiene bastante información empaquetada de distintas formas, al intentar imprimirse en la consola no es posible visualizar los datos que interesan de manera sencilla. Para ello, se utilizó la herramienta `rqt` de nuevo para visualizar de manera ordenada los datos del objeto *odometry*. Al abrirse la ventana de `rqt` como se muestra en la Figura 11, aparecen los topics activos en ese momento. Se muestra en la Figura 12 una imagen de cómo se pueden observar los datos:

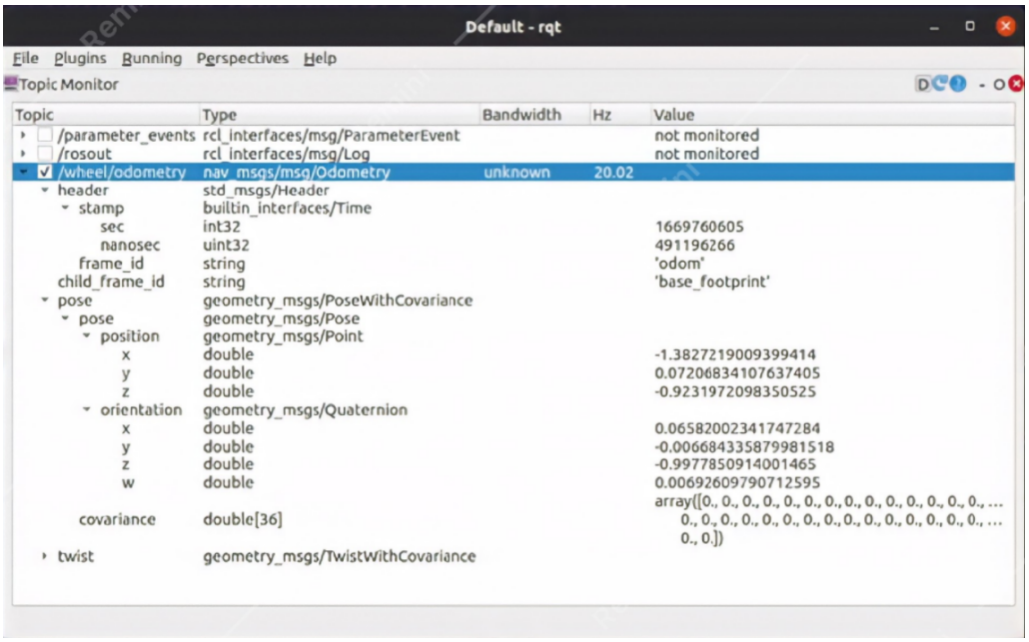

Figura 12: Visualización del objeto odometry publicado por el nodo en la ventana rqt

8.5. Validación del nodo

8.5.1. Fuente de odometría para control punto a punto del Rover UVG

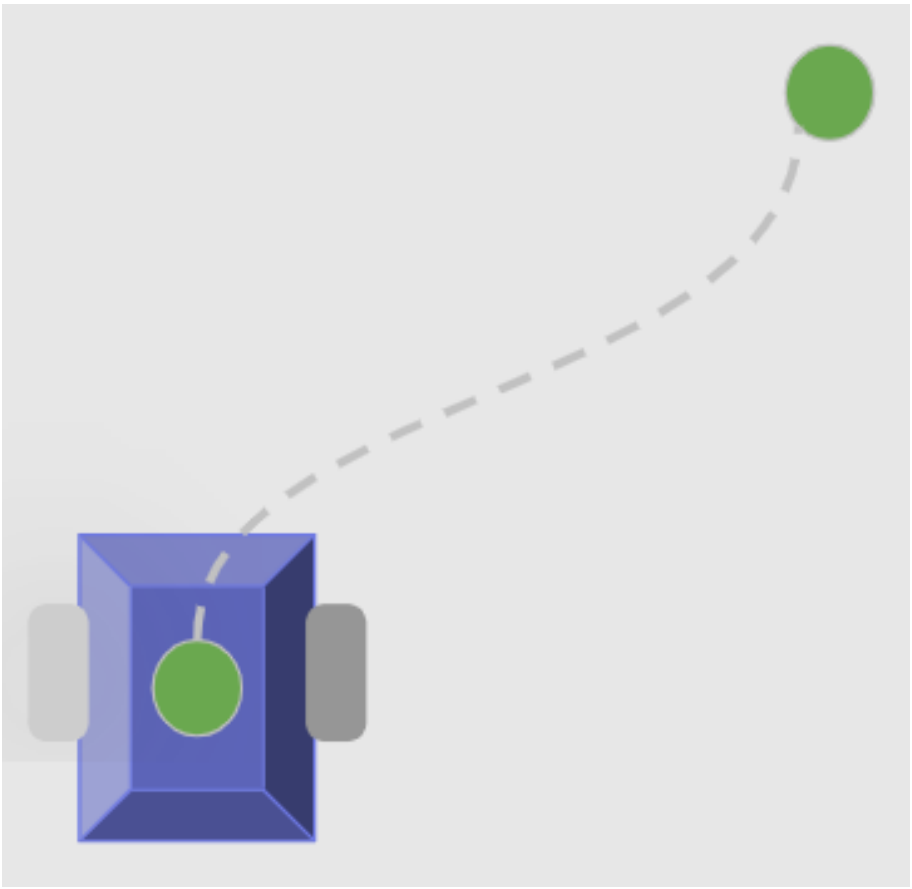

Figura 13: Representación del control punto a punto

El nodo creado anteriormente es muy importante para el Rover UVG ya que gracias a dicho nodo y a su funcionamiento, se puede tener una fuente de odometría con la cuál es posible realizar un control punto a punto para el Rover y de esa manera lograr que este se convierta en un robot autónomo. Sin este nodo dentro del sistema del Rover, este no sería capaz de realizar un control punto a punto ya que si el Rover no sabe dónde se encuentra, ¿como será capáz de direcccionarse para avanzar a la localización deseada?. El control punto a punto se programó en un nodo de ROS que formaba parte del WorkSpace del Rover UVG. Se hizo en conjunto con Diego Gerardo Mencos Caal en su proyecto de graduación titulado Ïntegración de una computadora central en el Rover UVG para la ejecución de ROS".

8.5.2. Fuente de odometría para SLAM del rover UVG

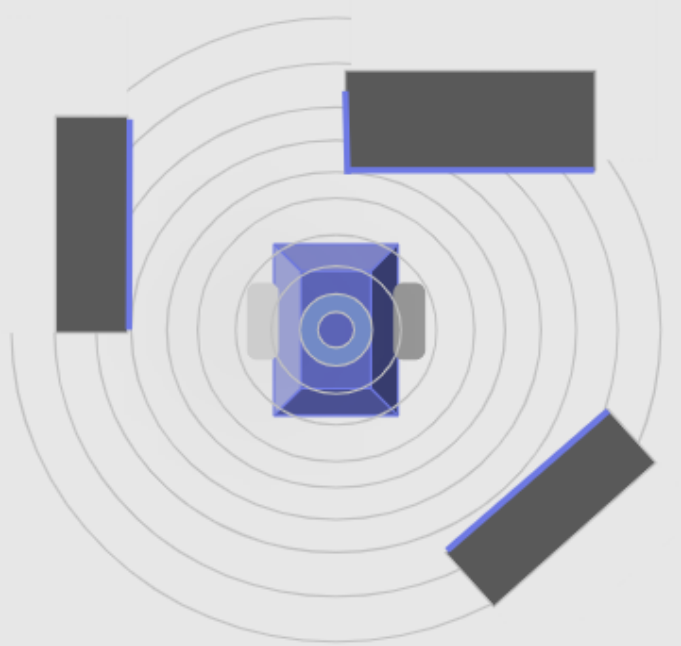

Figura 14: Representación del SLAM

Otra de las funcionalidades importantes que se le implementó al rover UVG durante el trabajo de graduación titulado Ïmplementación de un sistema de localización y mapeo simultáneo mediante un escáner Lidar Hokuyo como un nodo en ROS para el Rover UVG"de Katharine Senn fue un SLAM con la ayuda de un sensor LiDAR. Lo que realiza el SLAM implementado en el Rover UVG es generar mapas de lo que tiene alrededor. El sensor LiDAR es capaz de capturar la distancia de todo lo que se encuentra a su alrededor en un ángulo de apertura de 180^o. Esto es capaz de crear un mapa incompleto de lo que el LiDAR captura en esos 180^o de apertura. Para poder crear un mapa detallado y completo, el robot debía ser capaz de capturar la información conforme se traslada, e ir guardando dicha información en el mapa para que este se fuera actualizando. Para ello, era necesario que el Rover tuviera información de su pose continuamente para que con ello pudiera guardar la posición de cualquier obstáculo alrededor de él dentro del mapa, y que este se fuera moviendo junto con él.

8.5.3. Pose del efector final del brazo robótico del Rover

Este sistema de localización funciona técnicamente para cualquier marker dentro de la plataforma, y es capaz de publicar en diferentes topics los diferentes markers. Por ello, con este nodo de ROS es posible obtener la pose del efector final para el brazo robótico desarrollado en el trabajo de graduación de Jose Rodrigo Díaz Díaz titulado "Diseño, implementación, simulación y control de un brazo robótico para el Rover UVG".

8.5.4. Validación del nodo por sí solo

Dado que el presente proyecto de graduación, presenta la posibilidad de tener la pose de cualquier marker dentro del sistema de captura de movimiento OptiTrack. Es posible utilizarlo para cualquier otro Robot o sistema que lo requiera y no únicamente para el Rover UVG. Es decir, el nodo de ROS2 correspondiente a la pose de los markers es capaz de publicar la información en un topic de manera que cualquier Robot que utilize ROS2 sea capaz de tomar la información y utilizarla a su convenciencia. Para demostrarlo, se realizó la medición de 3 trayectorias mediante un marker del OptiTrack y un objeto de tipo Path en Rviz. El objeto de tipo path se suscribía al topic de odometría correspondiente y tomaba la información de la pose del objeto de tipo Odometry, y de esa manera era capaz de graficar en un espacio tridimensional, la trayectoria que el marker iba tomando en tiempo real. De esta manera, se puede comprobar que la pose que brinda el topic asociado al nodo desarrollado en el presente trabajo de graduación, se puede utilizar en un futuro para otro tipo de proyectos robóticos que así lo requieran.

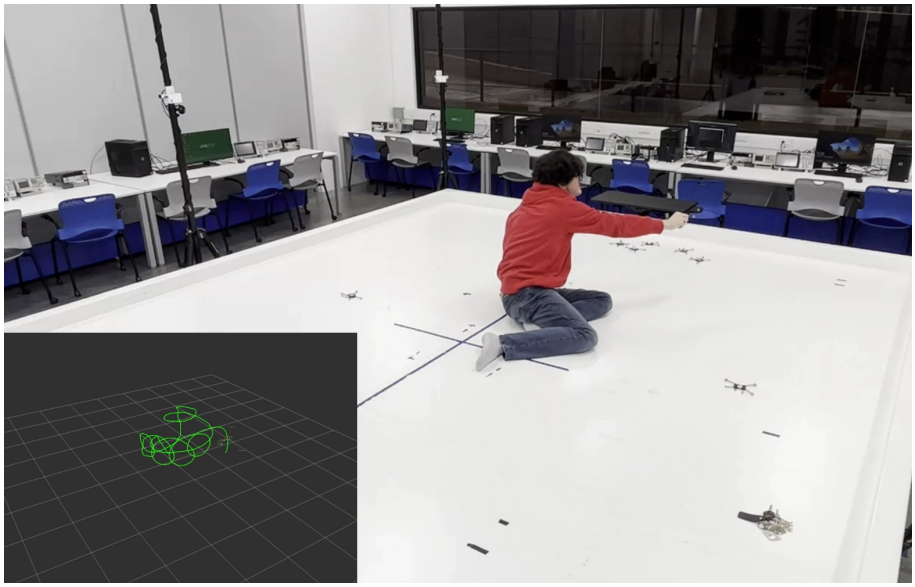

Figura 15: Toma de trayectorias tridimensionales con un marker

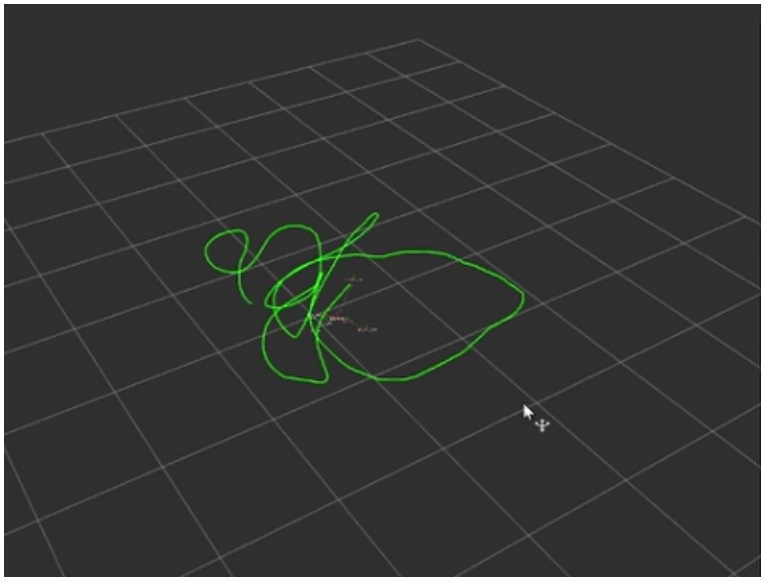

Figura 16: Trayectoria tridimensional 1: movimientos aleatorios

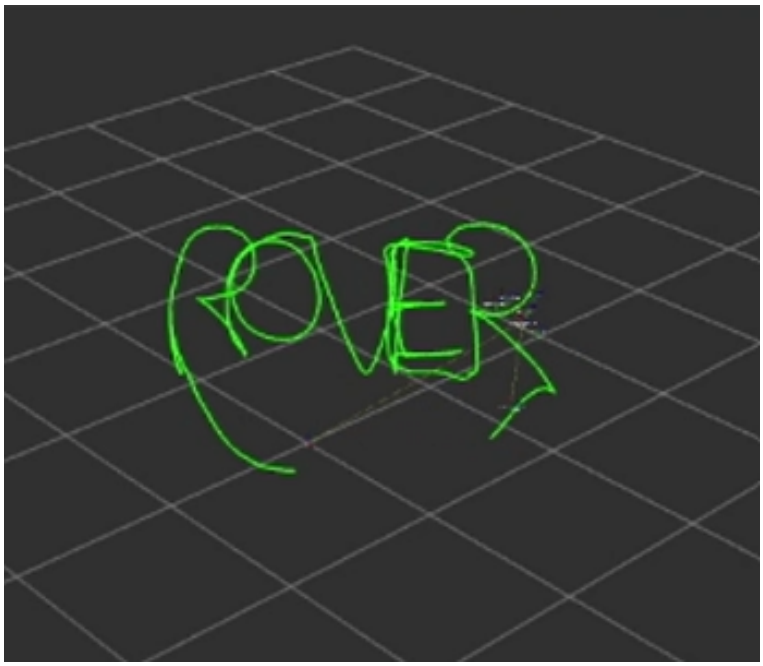

Figura 17: Trayectoria tridimensional 2: escritura de la palabra ROVER"

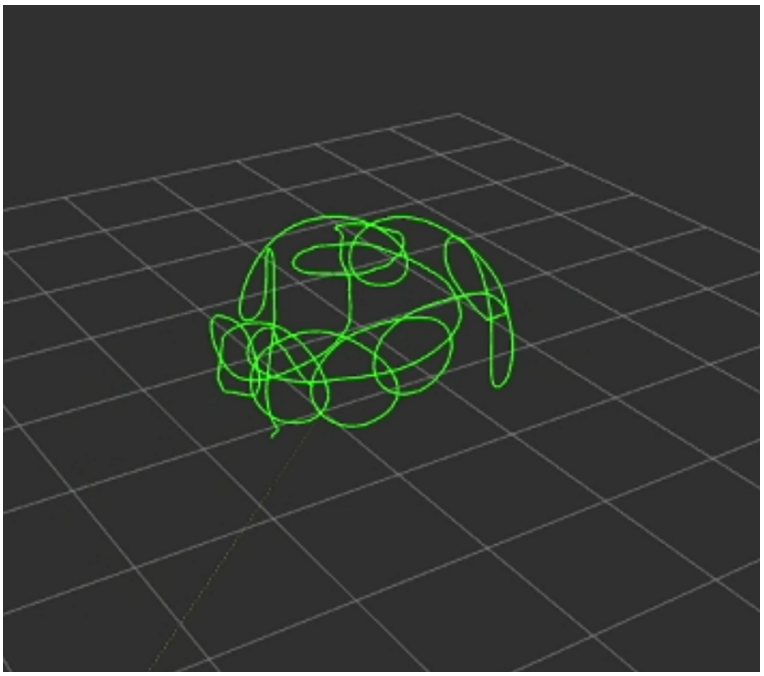

Figura 18: Trayectoria tridimensional 3: movimientos circulares aleatorios

CAPÍTULO 9

Medición de latencia, precisión y exactitud

9.1. Medición de trayectorias

Para medir la precisión y exactitud del sistema de localización propuesto del OptiTrack, se realizaron mediciones continuas a un marker en específico y con él se hicieron 2 trayectorias que luego se graficaron mediante el uso de excel. La primera trayectoria corresponde a un rectángulo. La segunda trayectoria corresponde a un "4". Estas mediciones se hicieron tanto con el sistema de captura de movimiento OptiTrack, como con el sensor de localización DWM1001 utilizado por Natalia de León.

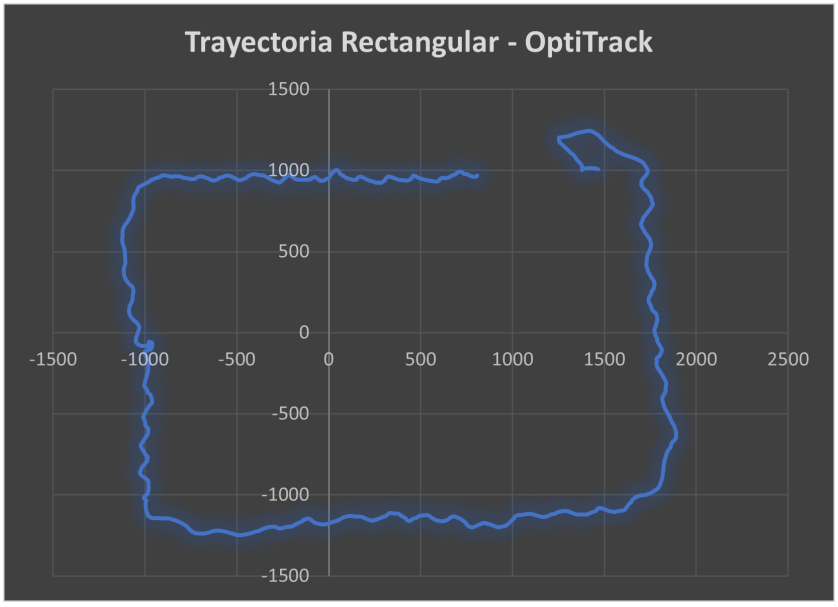

Figura 19: Trayectoria de rectángulo medido con el OptiTrack

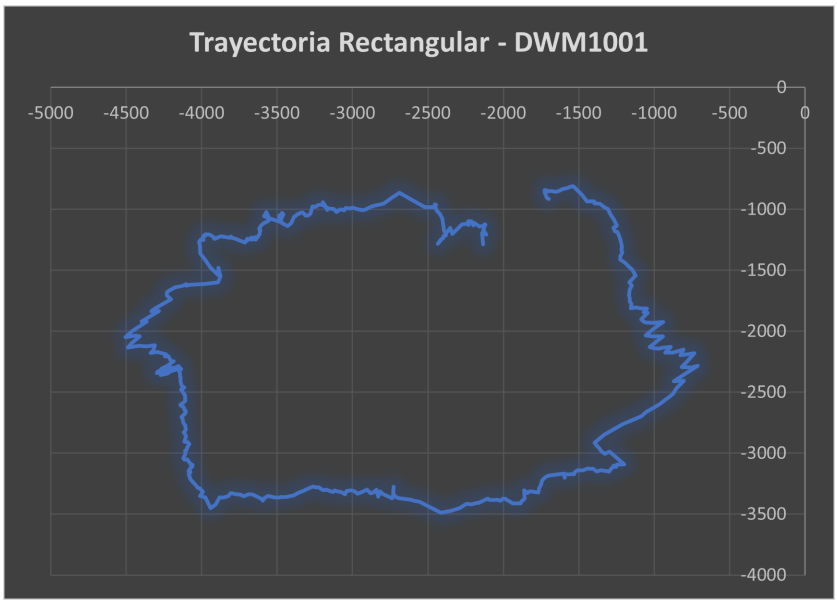

Figura 20: Trayectoria de rectángulo medido con el sensor DWM1001

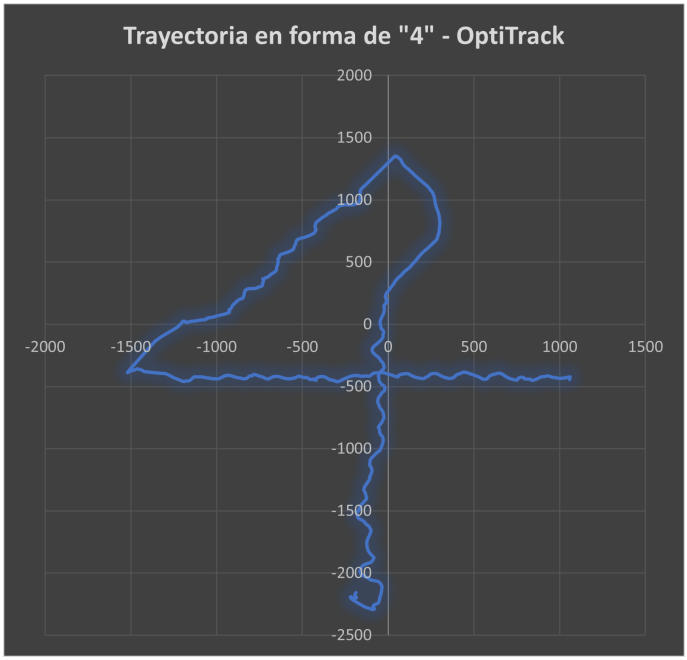

Figura 21: Trayectoria en forma de "4"medido con el OptiTrack

Se puede observar en las imágenes de las Figuras 19,20,21, 22 que las trayectorias tomadas mediante el OptiTrack se ven más suavizadas a comparación de las tomadas con el sensor DWM1001, las cuáles contienen mayor cantidad de *picos*.

Para poder realizar las tomas y asegurarse de que ambos sensores sí tomaran las mismas trayectorias y así realizar una comparación fidedigna. Se colocaron ambos dispositivos de medición como se muestra en la Figura 23:

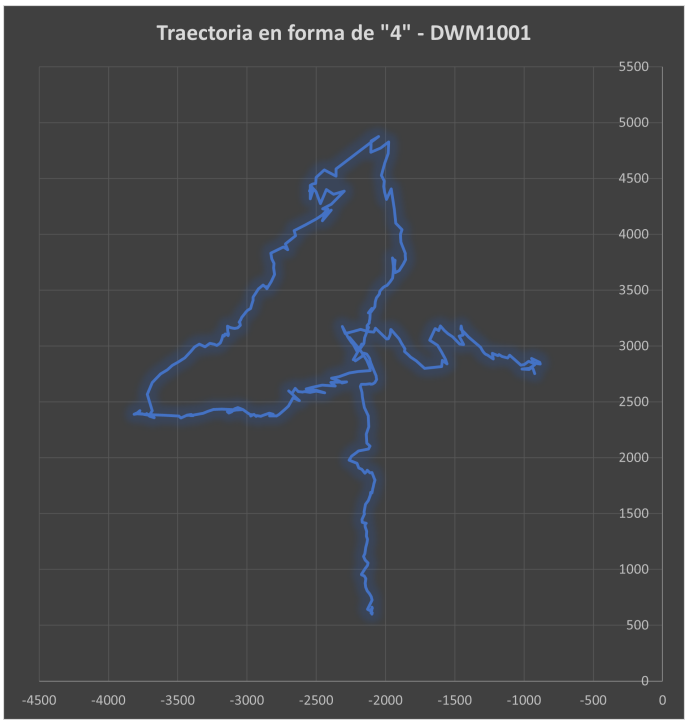

Figura 22: Trayectoria en forma de "4"medido con el sensor DWM1001

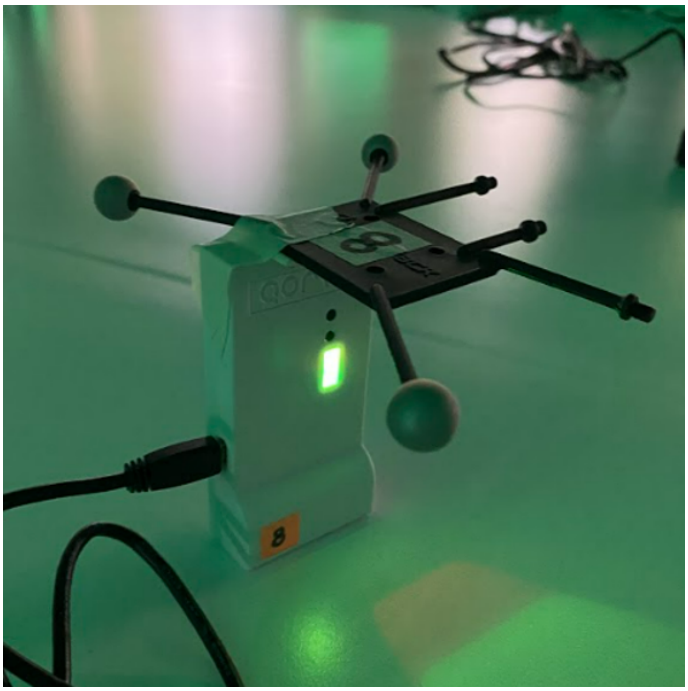

Figura 23: Marker del OptiTrack junto a sensor DWM1001 para medición de trayectorias

De esta manera, se aseguró que las mediciones se realizaran sobre los mismos espacios durante la trayectoria.

CAPÍTULO 10

Conclusiones

- Se desarrolló un nodo de ROS2 capaz de obtener información del sistema de captura de movimiento OptiTrack siendo completamente independiente de él, mediante el protocolo de comunicación TCP establecido con el ecosistema Robotat.
- La pose del Rover UVG fue capaz de transmitirse desde el OptiTrack hasta la computadora principal del Rover mediante el ecosistema Robotat y el desarrollo de un nodo en ROS2 que publica dicha información en el topic `/wheel/odometry` a una tasa de 20Hz.
- Se determinó la eficiencia de las mediciones de localización para cualquier marker dentro del OptiTrack mediante la toma de datos de una trayectoria formada por un marker, en la cuál se hacía una trayectoria en forma de rectángulo y otra en forma de "4".
- Se determinó que las mediciones realizadas por el OptiTrack tienen una mayor precisión que las tomadas por el sensor DWM1001 debido a que las gráficas de las trayectorias del primero son más suavizadas y tienen menos picos que el segundo, lo cual indica que las mediciones tienen una menor incertidumbre y por lo tanto una mayor precisión.

CAPÍTULO 11

Recomendaciones

- La precisión del sistema de movimiento OptiTrack es muy alta, sin embargo, la rapidez con la que se transmiten los datos desde que son tomados por el OptiTrack hasta que llegan a la computadora principal del Rover es muy baja. Esto puede provocar que el Rover reciba información precisa de una ubicación en la que el Rover ya no se encuentra y por lo tanto generar errores al momento de utilizar dicha ubicación para procesos de movimiento o trayectoria. Por ello, se recomienda alguna de las siguientes opciones:
  - Controlar el movimiento del Rover a bajas velocidades, de manera que la actualización de su ubicación en la computadora del mismo sea lo más cercana posible a su ubicación real.
  - Implementar un 2do método de localización que transmita la información de manera rápida sin importar que no sea muy preciso, y generar un filtro de Kalman que procese la información de ambos métodos y brinde una ubicación tanto rápida como precisa.
- Se recomienda implementar un sistema para la colocación del marker en la infraestructura del rover UVG (o cualquier otro que utilize el nodo) en una posición de manera que sea siempre visible por las cámaras del OptiTrack sin importar los obstáculos que se encuentren en el camino.
- Se recomienda verificar las regiones dentro de la plataforma del OptiTrack en las cuáles funciona y se captura correctamente la pose del Robotat, ya que se logró identificar que en las esquinas habían puntos en donde ya no era posible identificar el cuerpo rígido correspondiente al marker.
- Se recomienda y sería una buena mejora para este proyecto el tener una plataforma más grande con mayor cantidad de cámaras para el OptiTrack de manera que se puedan realizar más pruebas con mayor apertura y un mapa más amplio.

CAPÍTULO 12

Bibliografía

[1] L. Rivera, *Trabajo de graduación en modalidad de Megaproyecto: Sistema Explorador Robotizado y Autárquico*, 2005.

[2] M. Guzmán, *Diseño y construcción del vehículo para un sistema explorador robotizado y autárquico*, 2005.

[3] H. Sagastume, *Diseño mecánico, selección de motores e implementación de sensores para un robot explorador modular*, 2021.

[4] C. Perafán, *Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica*, 2021.

[5] M.-S. Academy. "What is motion capture and how does it work?" (2020), dirección: `https://optitrack.com/applications/robotics/` (visitado 20-05-2022).

[6] J. Hordyj. "Inverse Kinematics of anthropomorphic structures for vision systems applications." (oct. de 2015), dirección: `https://www.researchgate.net/figure/Actor-wearing-suit-adjusted-for-optical-motion-capture-on-the-left-Virtual-model_fig1_283152771` (visitado 20-05-2022).

[7] I. NaturalPoint. "Prime^{X}41." (2022), dirección: `https://optitrack.com/cameras/primex-41/specs.html` (visitado 23-05-2022).

[8] ROS. "ROS - Robot Operating System." (2021), dirección: `https://www.ros.org/` (visitado 23-05-2022).

[9] M. Suyama. "Protocolos de comunicación." (2004), dirección: `https://desarrolloweb.com/articulos/1617.php` (visitado 16-09-2022).

[10] S. Sánchez. "Topología y Arquitectura de red." (), dirección: `https://sites.google.com/site/redesehistoria/tipos-de-redes/protocolo-tcp-ip/topologia-y-arquitectura-de-una-red` (visitado 16-09-2022).

[11] Oracle. "Virtual Box." (2022), dirección: `https://www.virtualbox.org/wiki/Downloads` (visitado 23-05-2022).

[12] S. Macenski, T. Foote, B. Gerkey, C. Lalancette y W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, n.º 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. dirección: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.