Facultad de Ciencias Exactas, Ingeniería y Agrimensura

FCEIA

UNIVERSIDAD NACIONAL DE ROSARIO

Materia Optativa Robótica Móvil

Trabajo Práctico Final

Create3
Una trayectoria para grabar un dataset

Alumnos
Lautaro Rinaldi
Alejandro Rivosecchi

Docente
Taihú Pire

12 de septiembre de 2024

Índice

Resumen

Motivados por el nuevo robot Create3 del DCC quisimos involucrarnos con el y su funcionamiento, incorporarle nuevos sensores y generar datasets para que próximas/os estudiantes tengan una plataforma lista para usar cuando hagan pruebas y desarrollen algoritmos de robótica.

En este trabajo mostramos detalladamente y lo mas reproducible posible que pudimos como lo hicimos. Al robot le incorporamos una computadora a bordo y sensores, realizamos varias calibraciones tanto intrínsecas como extrínsecas de ellos y grabamos algunos datasets.

1. Introducción

En el desarrollo, implementación y testeo de algoritmos con aplicación en robótica (como de mapeo, localización o SLAM) no sólo es útil tener un robot físico, funcional y con diversos sensores sino que también resulta de mucha utilidad tener un dataset ya grabado y de calidad.

Disponer de un robot da la ventaja de poder hacer pruebas en tiempo real y de manera personalizada. Se pueden elegir y modificar el entorno del robot, la trayectoria que hace, su velocidad e incluso los sensores y accesorios que utiliza.

Aunque utilizar el robot también trae inconvenientes por las complejidades de, entre otras cosas, la logística de disponer de él, hacerlo funcionar, disponer de un lugar físico donde va a estar y adaptarlo a las necesidades además de lidiar con cuestiones como la iluminación del lugar, la batería del robot, control de sus movimientos y calcular sus trayectorias.

Por todo esto un primer objetivo o fase de este trabajo es:

- Asegurar del buen funcionamiento del robot y hacer posible la conexión, lectura de sensores internos y su navegación,
- incorporar sensores externos y una computadora a bordo que sirva para administrar y sea intermediaria entre los sensores y el robot,
- que el robot y los sensores externos incorporados utilicen ROS2 para las comunicaciones
- que sea razonablemente fácil iniciar el robot, sus sensores y controlarlo.

Un segundo objetivo o fase es generar un dataset con los datos de todos los sensores, grabado en un entorno adecuado y que recorra trayectorias interesantes. Esto resulta muy ventajoso ya que permite hacer pruebas sin desplegar nada físico ni requerir de ninguna logística más que tener el dataset. Se pueden correr distintas versiones de un mismo algoritmo sobre los mismos datos para así ver las diferencias de performance entre cada versión, hacer múltiples pruebas en simultaneo y a velocidades mas lentas o mas rápidas que la que fue grabado.

Una clara desventaja de los datasets es que una vez son grabados los datos quedan fijos por lo que no es posible modificar entornos, trayectorias, iluminaciones, sensores y es necesario hacer una nueva grabación si se requiere alguna modificación.

Es por esto es que el grabado de un dataset debería, al menos, tener las siguientes propiedades:

- Contener grabaciones de múltiples trayectorias distintas y que a su vez sean recorrida más de una vez,
- contar con la medición de todos los sensores, tanto internos como externos, priorizando los datos crudos (raw),
- estar grabado con buena iluminación,

- incorporar elementos y técnicas que ayuden a localizar al robot y los objetos en el entorno para poder generar un ground-truth, tales como cámaras adicionales y marcadores que ayuden a reconocer objetos y su ubicación y
- utilizar formato que sea fácil de utilizar en el futuro.

A partir del obsequio del robot educacional Create 3[1] de parte de la filial de iRobot[2] en Argentina a nuestro Departamento de Ciencias de la Computación decidimos que una buen trabajo final sería incorporarle al robot sensores y una computadora que estuvieran disponibles dejando todo funcionando para que sea posible directamente el uso del robot por próximos estudiantes de la materia para desarrollar sus proyectos, grabar datasets y demás.

2. Características del robot

2.1. Create 3

iRobot es una empresa que desarrolla y comercializa robots para la limpieza automática de los hogares. Tienen dos funciones principales: aspirar y fregar, no siempre ambas presentes en sus robots. Mediante la combinación de sensores incorporados y algoritmos estos robots son capaces de detectar si están en un "acantilado" (tal como una escalera), si chocaron contra algún obstáculo (como una pared) y demás y así mapear y localizarse automáticamente los entornos que son incorporados.

Paralelamente la empresa produce robots con fines educativos y dentro de estos se encuentra el Create3 que es físicamente similar a los robots de limpieza de hogar, particularmente la Roomba, pero no incorpora las funciones y el software de limpieza. Está diseñado para ser personalizable tanto a nive de software como de hardware. Por ejemplo la tapa superior tiene una grilla de orificios para poder anclarle cosas, posee un compartimiento de carga (un "cajoncito") interior y mapas 3D disponibles en la documentación online para imprimir accesorios que facilitan montar objetos.

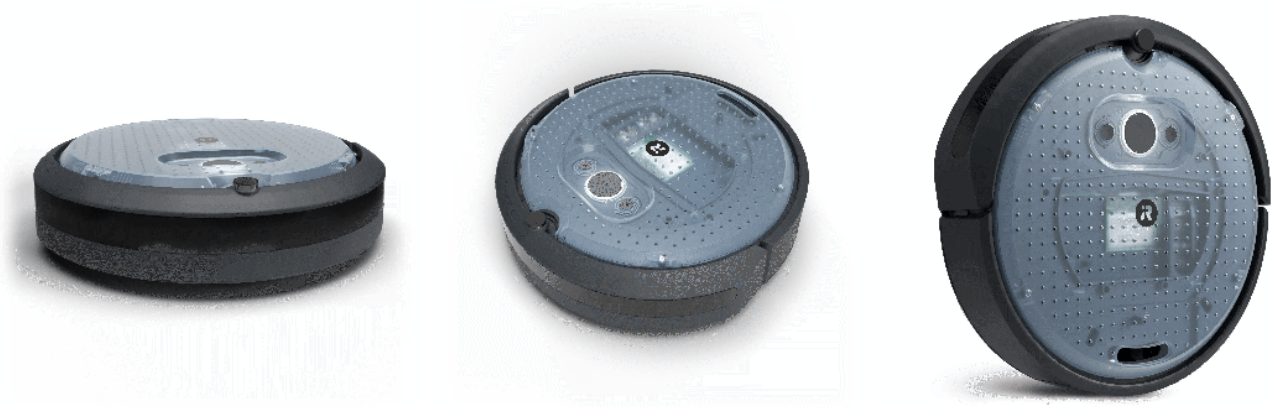


Figura 1: Imágenes oficiales del Create3

La parte frontal del robot tiene un bumper de gran amplitud y siete pares de sensores infrarrojos de proximidad para detectar obstaculos. La parte superior tiene tres botones: el central, el más grande, típicamente usado para el encendido y apagado, y dos botones a los costados del central que sus funciones pueden ser personalizadas. El botón central tiene un anillo de seis luces LED RGB que indican la cantidad batería, errores y también pueden ser personalizadas.

[1] https://edu.irobot.com/what-we-offer/create3
[2] https://www.irobot.com/

Removiendo la tapa superior se encuentra un adaptador que permite conectar el robot mediante USB-C o Bluetooth a computadoras externas.

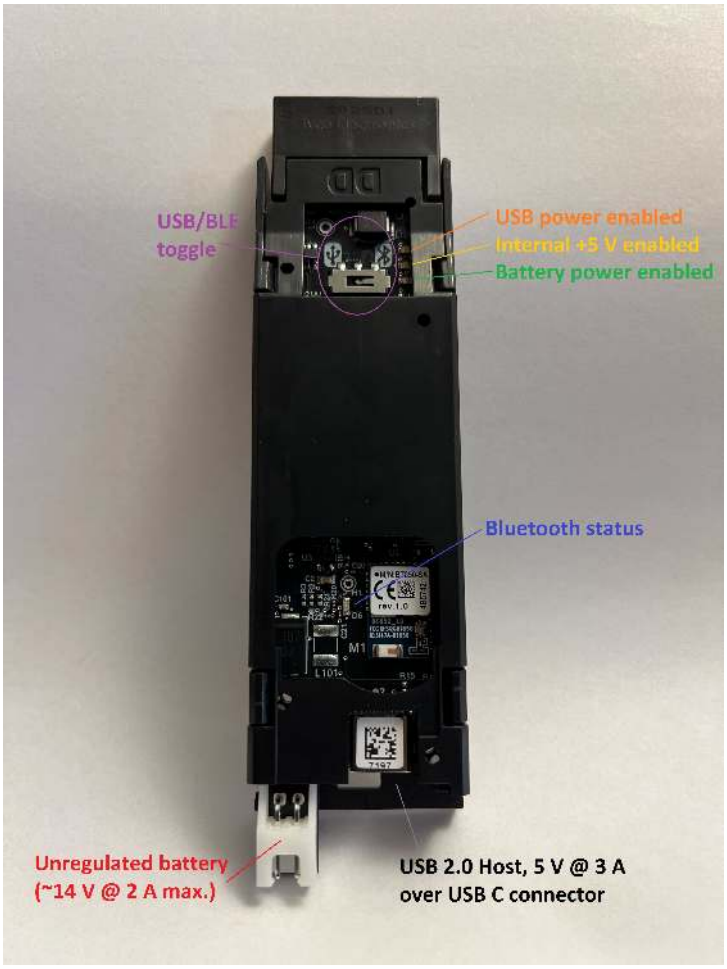


Figura 2: Adapter Board

La parte inferior tiene cuatro sensores cercanos al frente del robot para detectar acantilados, los pins de carga, dos ruedas motorizadas con encoders, un sensor óptico para odometria y una rueda giratoria libre. Además posee una IMU que no es visible (importante: los datos públicados de la IMU tienen cancelada la gravedad).

Posee un firmware propio que corre versiones de ROS2. Las versiones de firmware disponible están para las distribuciones de ROS2 Galactic y Humble. El robot trabaja sobre el ecosistema ROS2: sus sensores publican sus lecturas en mensajes ROS2 y la navegación al robot es indicada mediante ROS2. Actualmente está instalada la versión H.2.6[3] del firmware del Create3. Además posee un modo para conectarse mediante Bluetooth y manejarlo con una aplicación propia de iRobot pero no utilizamos esa opción.

Está disponible en internet la documentación oficial[4] que, si bien está en ingles, es super completa: con muchas explicaciones, imágenes y tutoriales. Este informe no está completo sin la consulta de la documentación oficial: se asume su lectura.

Este[5] tutorial, específicamente en la "Phase 3", detalla como conectar el Create3 a una red WiFi.
El Create 3 hostea un webserver[6] que sirve para modificar configuraciones del robot.

Algo importante y que puede dar problemas si no se tiene en cuenta es que todas las instancias de ROS2 que se comuniquen con el robot deben utilizar la misma implementación de RMW que el robot.

[3] https://iroboteducation.github.io/create3_docs/releases/h_2_6/
[4] https://iroboteducation.github.io/create3_docs/
[5] https://edu.irobot.com/create3-setup
[6] https://iroboteducation.github.io/create3_docs/webserver/overview/

Esta[7] entrada explica como asignar la implementación RMW que usa el robot y esta[8] entrada explica como asignar la implementación RMW a una instancia de ROS2.

Para encender el Create3 la única manera es posicionarlo sobre la estación de carga estando esta conectada a la corriente. Recomendamos encender la NUC rápidamente así cuando enciende el Create3 ya ve todas las interfaces de red (en particular la conexión cableadad con la NUC). Sino podría pasar que la instancia de ROS2 del Create3 no transmita sus mensajes por esa interface.

Hay dos problemas[9] que nos han pasado encendiendo el robot:

- El robot no enciende: simplemente no enciende al estar sobre la estación de carga. Una luz titila pero no enciende. Nos han propuesto que la razón podría ser que la batería este completamente descargada pero nos ha pasado con la batería con bastante carga también. La manera de solucionarlo tampoco es clara, hemos intentado, a veces con mas y a veces con menos suerte: esperar (a veces por un largo rato, como una hora), desenchufar y volver a enchufar la estación de carga, desconectar y volver a conectar la batería del Adapter Board (hacerlo con sumo cuidado ya que es bastante frágil)
- Al instante de encender o a los pocos minutos el anillo de luces queda en rojo. Esto siempre se soluciono solo con unos pocos minutos de espera.

Entre los varios comandos y acciones predefinidas del Create3 compartimos la manera de hacerlo salir de la estación de carga y de hacer que la encuentre y se posicione para cargarse automaticamente:

```
ros2 action send_goal /undock irobot_create_msgs/action/Undock "{}"

ros2 action send_goal /dock irobot_create_msgs/action/Dock "{}"
```

2.2. Computadora a bordo (NUC)

Se decidió instalarle una computadora a bordo al robot para que sea posible agregarle otros sensores que para ser leidos tienen que ser conectados a una computadora y para agregarle poder de procesamiento e independencia al robot.

Fue incorporada una mini computadora Intel NUC que estaba disponible. Esta electrizada mediante una salida eléctrica que ofrece el Create3 a partir de la batería que incorpora.

A la NUC se le instalo Ubuntu Server 22.04 como sistema operativo ya que es una versión LTS compatible con la versión de ROS2 con la que se iba a usar. La versión Server es sin interfaz gráfica ya que la NUC está pensada para ser controlada por SSH. También se le instalo el ROS2 Humble base, no la versión Desktop ni Desktop Full, ya que estas dos principalmente incorporan herramientas con interfaz gráfica. Adicionalmente, entre otras cosas, se instalaron paquetes para el funcionamiento de la cámara y el LiDAR incorporados que más adelante describimos.

La primera vez que se utilice la NUC probablemente sea necesario conectarle un teclado y conectarla a un monitor para poder utilizarla.

Usuario: robotica || Contraseña: robotica

[7] https://iroboteducation.github.io/create3_docs/setup/xml-config/
[8] https://iroboteducation.github.io/create3_docs/setup/provision/
[9] https://github.com/iRobotEducation/create3_docs/discussions/393

2.2.1. Redes y conexiones

Para que la NUC se conecte a la red WiFi deseada hay que modificar el siguiente archivo con permisos sudo:

`/etc/netplan/00-installer-config-wifi.yaml`

En ese archivo se pueden agregar y/o eliminar redes WiFi con su nombre (SSID) y contraseña asociadas a una determinada interface (placa de red integrada en NUC, adaptador WiFi USB, etc). Una vez modificado el archivo hay que ejecutar en la terminal los siguientes comandos en orden y reiniciar la NUC si fuera necesario:

```
sudo netplan generate
sudo netplan apply
```

Luego la NUC va a conectarse directamente a alguna de las redes configuradas y va a ser posible conectarse directamente a ella mediante SSH estando en la misma red.

Recomendamos agregarle una placa de red WiFi USB a la NUC ya que su placa de red WiFi interna aparentemente es demasiado débil captando señales y nos generó mucho problemas para detectar redes o mantener una conexión estable.

Recomendamos la herramienta nmcli (command-line tool for controlling NetworkManager), ya instalada en la NUC, para visualizar las redes WiFI detectadas, la intensidad de la señal y otras informaciones de red.

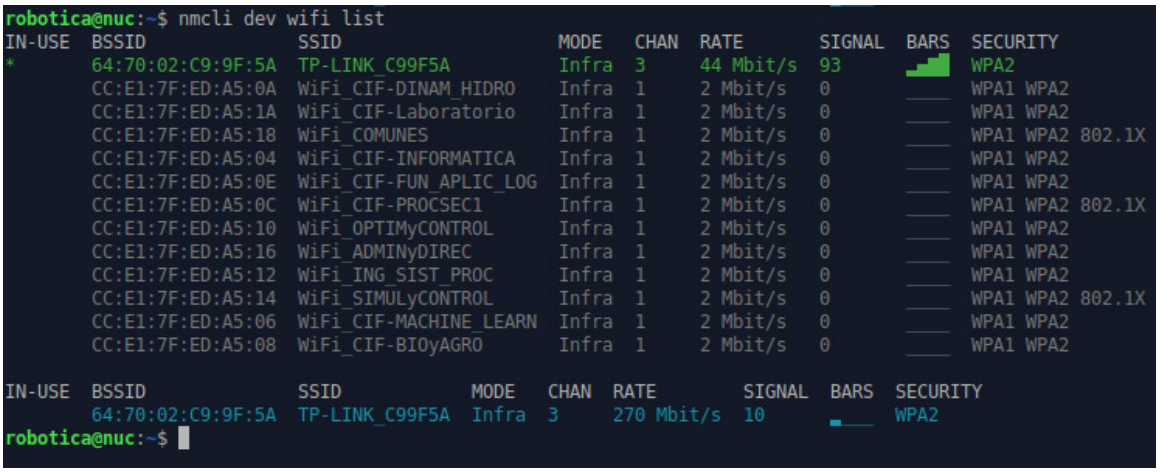


Figura 3: Salida del comando nmcli dev wifi list (para ver redes WiFi disponibles)

El Create3 puede conectarse a la red mediante WiFi, como se explico antes, y/o con un conector USB-C (además de Bluetooth pero que no usamos esa opción). El conector USB-C está en la Adapter Board[10] que posee el Create3.

[10] `https://iroboteducation.github.io/create3_docs/hw/adapter/`

La NUC está conectada al Create3 mediante Ethernet utilizando un conversor USB C-Ethernet.

Figura 4: adaptador USB C-Ethernet

Para la configuración de la red por Ethernet hay que modificar el siguiente archivo también con permisos sudo:

/etc/netplan/00-installer-config.yaml

Actualmente está configurado como indica en el paso 8 en esta[11] entrada.

Nuevamente, para aplicar la nueva configuración Ethernet hay que ejecutar los siguientes dos comandos:

```
sudo netplan generate
sudo netplan apply
```

en ese orden después reiniciar la NUC y el Create 3.

Hay que tener en cuenta que para que esto funcione debe estar correctamente configurada[12] la subred alámbrica del Create 3.

2.2.2. Network Time Protocol (NTP)

ROS2 publica sus mensajes con un timestamp que lo asigna con lo que marca el reloj del sistema operativo donde se está ejecutando.

Por eso es importante que todos los sistemas operativos que estén ejecutando una instancia de ROS2 tengan sus relojes lo más sincronizados posibles.

Sincronizar relojes es la función que cumple NTP y para eso seguimos el tutorial[13] de la documentación oficial. También puede ser de utilidad modificar[14] la configuración NTP del Create 3.

La topología que elegimos para el NTP es correr un servidor en la NUC que sirva sobre las redes Ethernet y WiFi y que el Create3 y la computadora externa que utilizamos (se podría agregar cualquier dispositivo más que se necesite tener su reloj sincronizado) se conecten como clientes.

[11] `https://iroboteducation.github.io/create3_docs/setup/pi4humble/`
[12] `https://iroboteducation.github.io/create3_docs/webserver/set-wired-subnet/`
[13] `https://iroboteducation.github.io/create3_docs/setup/compute-ntp/`
[14] `https://iroboteducation.github.io/create3_docs/webserver/edit-ntp-conf/`

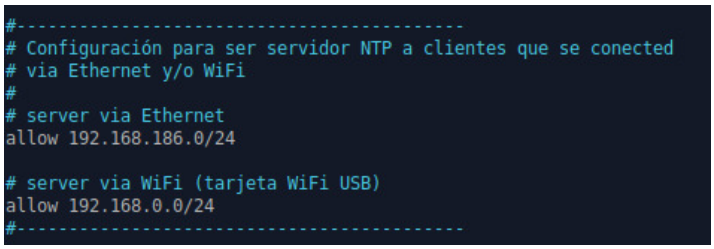


Figura 5: configuración de la NUC como servidor - archivo /etc/chrony/chrony.conf

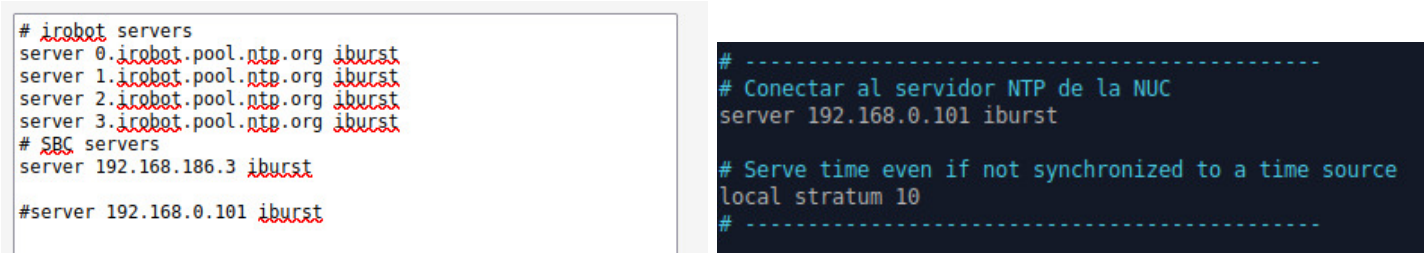


Figura 6: configuración del Create3 (izq - webserver) y PC externa (der - archivo) como clientes

Desde la NUC se puede visualizar los clientes que están conectados a ella ejecutando el siguiente comando (la columna NTP debe ser distinta de 0):

```
sudo chronyc clients
```

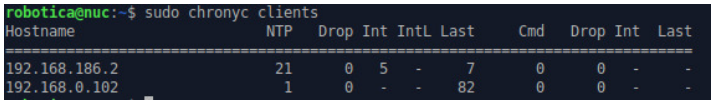


Figura 7: Clientes NTP del Create3 y PC externa conectados al servidor NTP en la NUC

2.3. Cámara estéreo (ZED)

Decidimos instalarle una cámara al robot para que sea posible estudiar métodos de computer vision. El requisito era que sea una cámara estéreo para poder calcular profundidades en las imágenes tomadas.

Se instalo la cámara ZED de StereoLabs. Es la primera ZED que existió, no es la ZED 2, ZED 2i ni la ZED X. como está fuera de producción no hay mucha documentación de este modelo en internet.

La cámara es estéreo, es decir, posee dos lentes, es a color y los sensores de la cámara son los mismos que los de la ZED 2: OV4689 CMOS sensors[15].

[15] https://community.stereolabs.com/t/original-zed-camera-specs/1843

Las posibles resoluciones de la cámara son:

VIDEO MODE	OUTPUT RESOLUTION (SIDE BY SIDE)	FRAME RATE (FPS)	FIELD OF VIEW
2.2K	4416x1242	15	Wide
1080p	3840x1080	30, 15	Wide
720p	2560x720	60, 30, 15	Extra Wide
WVGA	1344x376	100, 60, 30, 15	Extra Wide

Figura 8: Resoluciones cámara ZED

Utilizamos la menor resolución posible al menor frame rate posible: WVGA 1344x376 a 15 FPS para no sobrecargar la capacidad de procesamiento la NUC ni de la red.

La cámara está conectada vía USB-A a la NUC y utilizamos el paquete usb_cam[16], ya instalado en la NUC, para utilizarla y que sus imágenes se publiquen como mensajes en ROS2.

No usamos el SDK oficial[17] de la ZED porque requiere usar CUDA y para funcionar CUDA necesita una placa de video NVIDIA que la NUC no tiene. Parece haber formas de instalarlo sin CUDA pero no lo hemos intentado.

Para empezar a tomar imágenes con la ZED y publicarlas en ROS2 se debe ejecutar el siguiente comando en la NUC:

```
ros2 run usb_cam usb_cam_node_exe --ros-args -p image_width:=1344 -p
    image_height:=376 -p framerate:=15.0
```

Y así también seteamos la resolución y los FPS de la cámara.

Hay que notar que en ROS2 se publica en un mismo mensaje las imágenes de las cámaras izquierda y derecha concatenadas. Escribimos un paquete de ROS2 para dividir estas imágenes y ponerlas en dos mensajes separados. Más adelante lo mostraremos.

[16] https://github.com/ros-drivers/usb_cam/tree/ros2

[17] https://github.com/stereolabs/zed-sdk

2.3.1. Intento fallido (Tara)

En un primer intento al robot le instalamos la cámara Tara[18] que también es una cámara estéreo y esta, adicionalmente, dispone de una IMU incorporada. Pero por otro lado la cámara es en blanco y negro y como no tiene filtro infrarrojo cuando instalamos el LiDAR descubrimos que se veían los rayos infrarrojos de el a traves de esta cámara por lo que finalmente la reemplazamos por la ZED.

Figura 9: Tara

2.4. LiDAR (RPLIDAR A1)

Al robot le incorporamos el LiDAR RPLIDAR A1[19] de la marca SLAMTEC que escanea en 2 dimensiones, 360º y hasta 12 metros de distancia. El LiDAR está conectado a la NUC mediante USB-A.

f

Para controlar el LiDAR y que publique los datos en ROS2 utilizamos este[20] paquete que es un fork del paquete oficial[21]. Elegimos utilizar la versión del paquete del usuario allenh1 ya que con la versión oficial no nos funciono la opción para apagar el motor del LiDAR mientras que con la versión que utilizamos si funciona esa opción.

Se puede utilizar el siguiente comando en la terminal de la NUC para que el LiDAR comience a hacer mediciones y las publique como mensajes ROS2:

```
ros2 launch rplidar_ros rplidar.launch.py
```

```
robotica@nuc:~$ ros2 launch rplidar_ros rplidar.launch.py
[INFO] [launch]: All log files can be found below /home/robotica/.ros/log/2024-07-03-16-56-18-963747-nuc-2803
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [rplidar_composition-1]: process started with pid [2804]
[rplidar_composition-1] [INFO] [1720025779.224797498] [rplidar_composition]: RPLIDAR running on ROS 2 package rplidar_ros. SDK Version: '1.12.0'
[rplidar_composition-1] [INFO] [1720025781.733400762] [rplidar_composition]: RPLIDAR S/N: B3A0EDF9C7E29BCEA7E39EF223734304
[rplidar_composition-1] [INFO] [1720025781.733560718] [rplidar_composition]: Firmware Ver: 1.29
[rplidar_composition-1] [INFO] [1720025781.733619306] [rplidar_composition]: Hardware Rev: 7
[rplidar_composition-1] [INFO] [1720025781.735210232] [rplidar_composition]: RPLidar health status : '0'
[rplidar_composition-1] [INFO] [1720025781.735406883] [rplidar_composition]: Start
[rplidar_composition-1] [INFO] [1720025782.281366292] [rplidar_composition]: current scan mode: Sensitivity, max_distance: 12.0 m, Point number: 7.9K , angle_compensate: 2, flip_x_axis 0
```

Figura 13: LiDAR lanzado en la terminal

[19] https://www.slamtec.ai/product/slamtec-rplidar-a1/

[20] https://github.com/allenh1/rplidar_ros

[21] https://github.com/Slamtec/rplidar_ros/

Specifications

The below table describes the specification of Tara.

Table 2: Tara Specifications

Description	Specification
Size (L X W X H) in mm	100 x 30 x 35 (With Enclosure)
	95 x 17 x 27 (Without Enclosure)
Weight (Without USB Cable)	80.5 Grams (With Enclosure)
	28.5 Grams (Without Enclosure)
Output Format	8-bit/10-bit Monochrome
Supported Resolutions	WVGA (752 x 480p), VGA (640 x 480) and QVGA (320 x 240)
Supported OS	Windows7, 8.1 and 10, Ubuntu 12.04 and 16.04 (both 32bit and 64bit)
USB Version	3.0 and 2.0
USB Video Class Version	UVC Version 1.0
Product ID (PID)	0 x C114
Vendor ID (VID)	0 x 2560

Maximum Frame Rate Supported

The below table describes the maximum frame rate supported by Tara.

Table 3: Frame Rate Supported

Format	Resolution	Frame Rate (FPS)	
		USB 3.0	USB 2.0
Y16 (8 bit RAW)	320 x 240	60	60
	640 x 480	30 and 60	30
	752 x 480	30 and 60	30
RGB24 (10 bit RAW)	320 x 240	60	60
	640 x 480	30 and 60	30
	752 x 480	30 and 60	30

CMOS Image Sensor Specification

The below table describes the specifications of CMOS Image sensor used in this Tara camera board. For more information about the MT9V024 sensor or for datasheet, please contact On Semiconductor.

Table 4: CMOS Image Sensor Specification

Sensor Specification	
Type / Optical Size	1/3" Optical format CMOS Image sensor
Resolution	WVGA
Sensor Type	10-bit Monochrome
Pixel Size	6.0 µm x 6.0 µm
Sensor Active Area	752 x 480
Responsivity	4.8 V/lux-sec (550 nm)
SNR	NA
Dynamic Range	>55dB in linear and >100dB in HDR mode

Figura 10: Especificaciones Tara

Figura 11: RPLIDAR A1

Se pueden ver las lecturas gráficamente con RViz2 usando la visualización LaserScan en el topic /scan.

Hay que configurar el “Fixed Frame” en las “Global Options” como “laser” si no esta la transformación necesaria en el tf para laser.

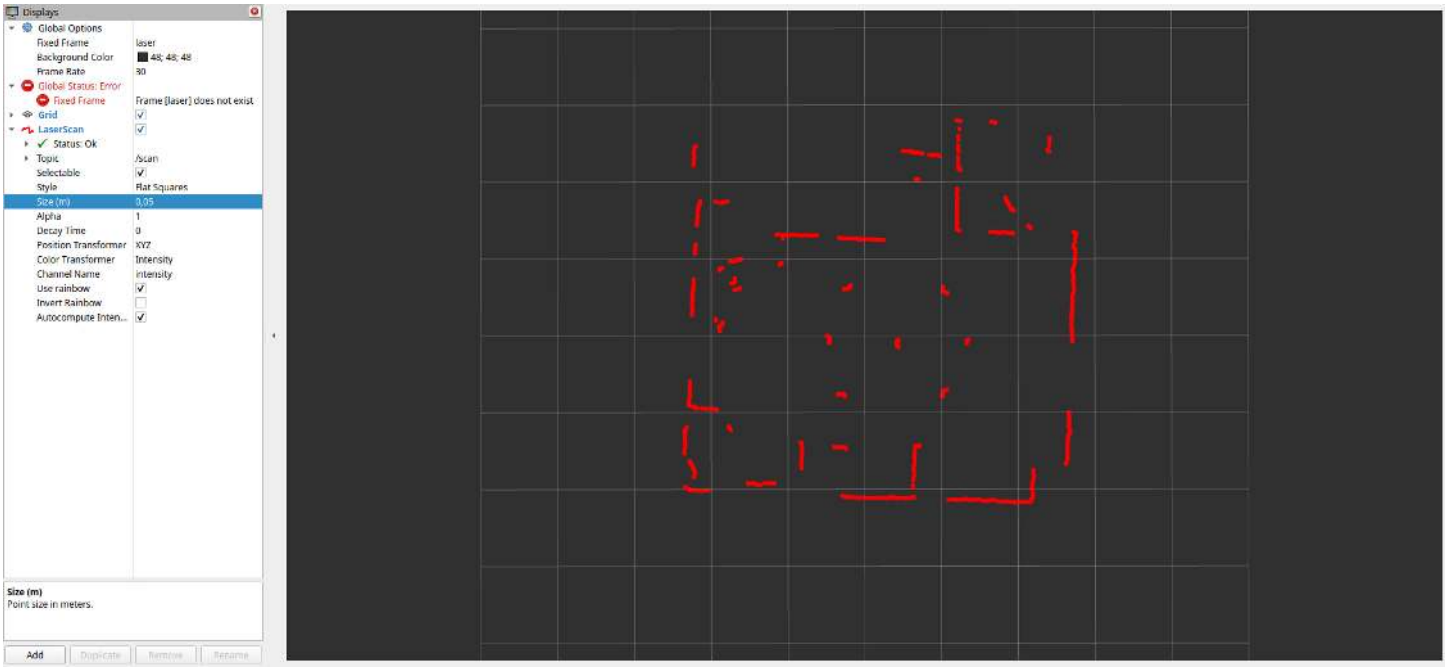


Figura 14: LiDAR visualizado en RViz2

Para apagar y prender el motor del LiDAR se pueden utilizar los siguientes comandos:

```
ros2 service call /stop_motor std_srvs/Empty
ros2 service call /start_motor std_srvs/Empty
```

2.5. Navegación mediante teclado

Para navegar el robot con el teclado utilizamos el paquete teleop_twist_keyboard[22] de ROS2 que se puede ejecutar desde cualquier instancia de ROS2 que esté conectada al Create3. Se ejecuta con el siguiente comando:

```
ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

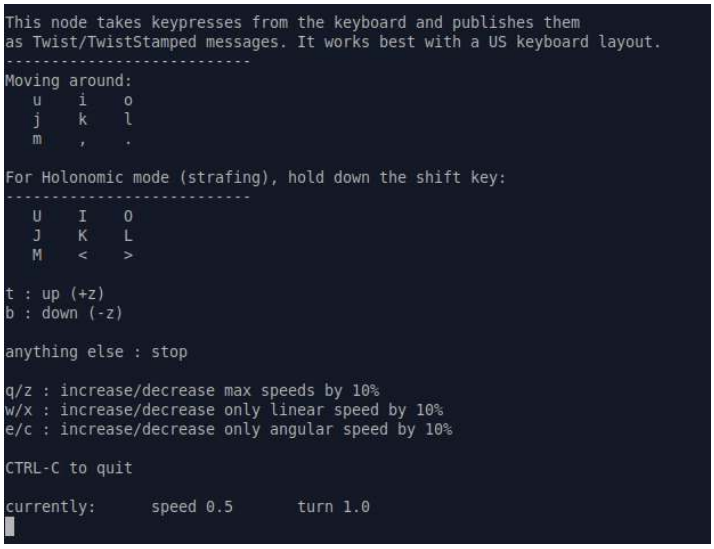


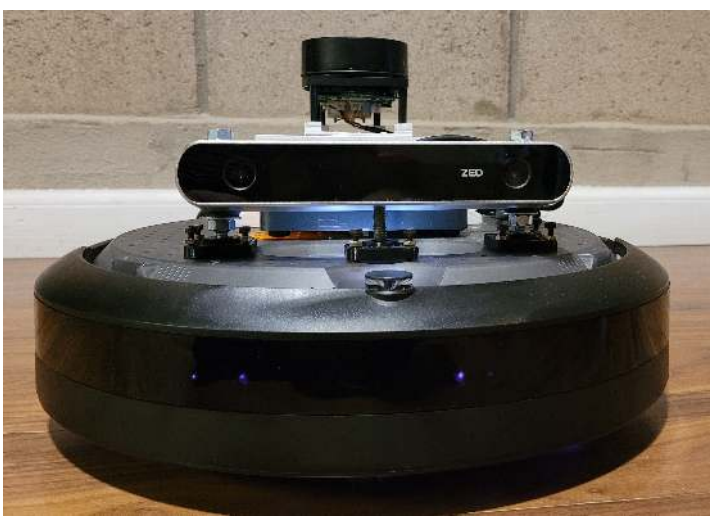
Figura 15: teleop_twist_keyboard lanzado en la terminal

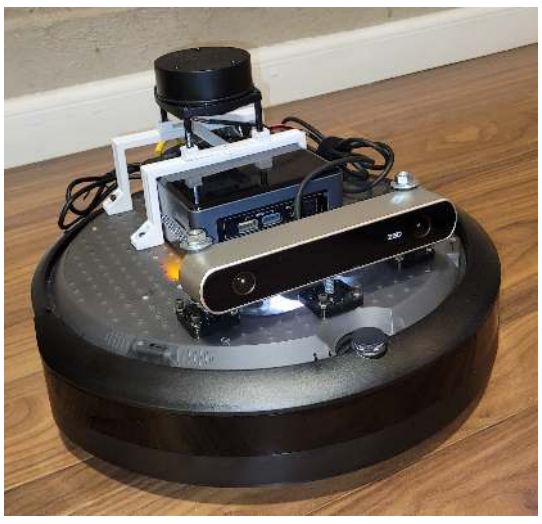
2.6. Armado del robot

El armado del robot y la unión de las partes fue un trabajo bastante artesanal y manual, y para el cual recibimos varias ayudas.

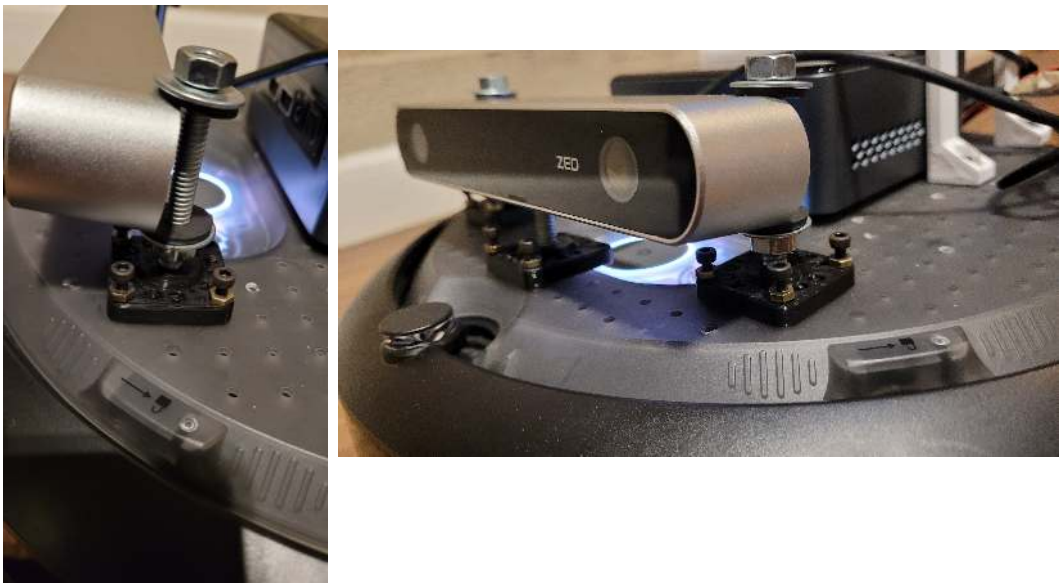
Agradecemos a todo el Laboratorio de Robótica del CIFASIS que todos y todas nos ayudaron con distintas cosas. Con su conocimiento nos fueron ayudando a hacer funcionar muchas cosas, a instalar algunos de los perifericos y resolver varias cosas con las que nos trabábamos. Además Del Laboratorio son la NUC, la ZED, la Tara, el RPLIDAR A1, el patrón de calibración, el adaptador WiFi USB que usamos para grabar el dataset y probablemente mas cosas que se nos olvidan mencionar.

Agradecemos también al Espacio Maker[23] que nos imprimieron unos soportes 3D para instalar la ZED.





[23] https://www.fceia.unr.edu.ar/espaciomaker/



2.6.1. Conexiones

El robot esta conectado a la NUC a través de una conexión Ethernet, la cual necesitó de un adaptador USB C-Ethernet ya que el Create3 no cuenta con puerto Ethernet (si con USB-C).

La conexión entre el LiDAR y la NUC es mediante, primero, un adaptador de la salida UART del LiDAR a Mini USB hembra y luego con un cable Mini USB-USB A hacia la NUC.

La conexion de la ZED a la NUC es a través de USB A.

El control de la NUC es mayormente por SSH por medio de una conexión WiFi con la computadora del usuario.

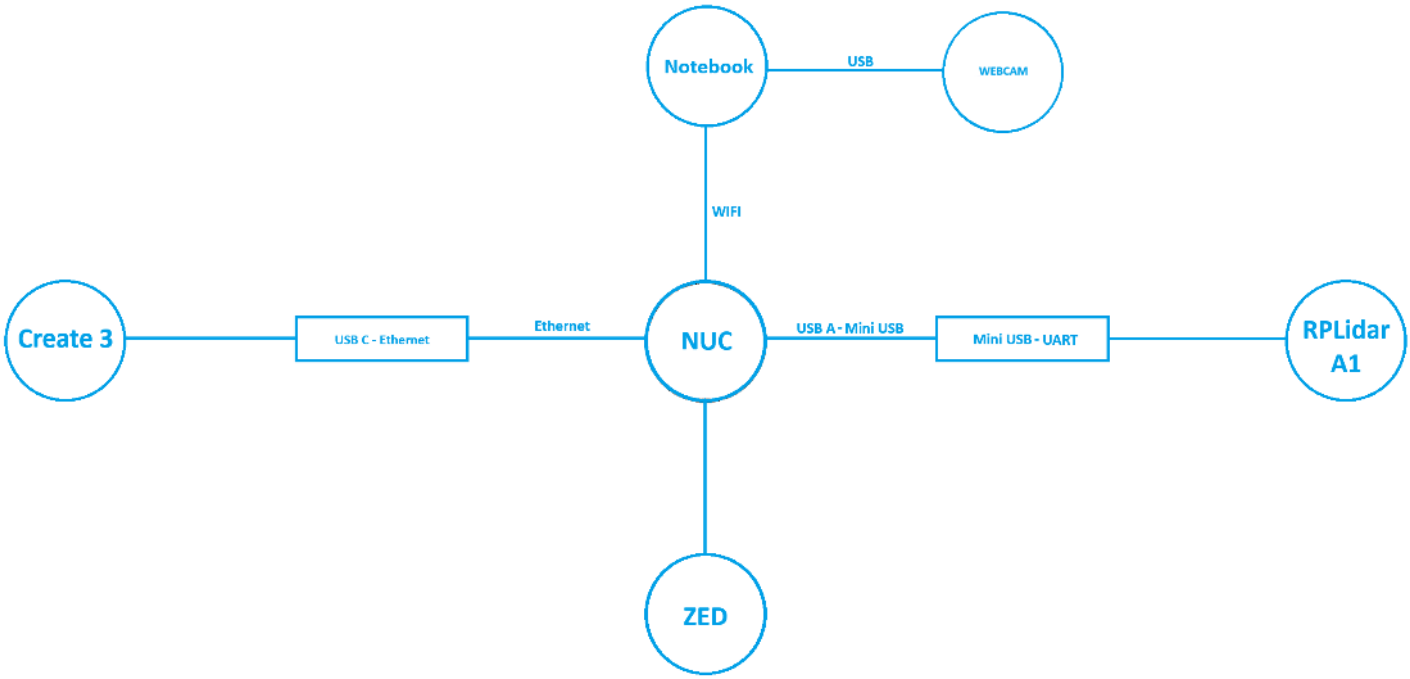


Figura 16: Diagrama de conexiones

Para las grabaciones del dataset se le agrego un dungle WiFi TP-LINK TL-WN723N a la NUC para mejorar el alcance y la calidad de conexión. Esto facilitó mucho la usabilidad por lo que lo recomendamos.

2.6.2. Ejes de coordenadas

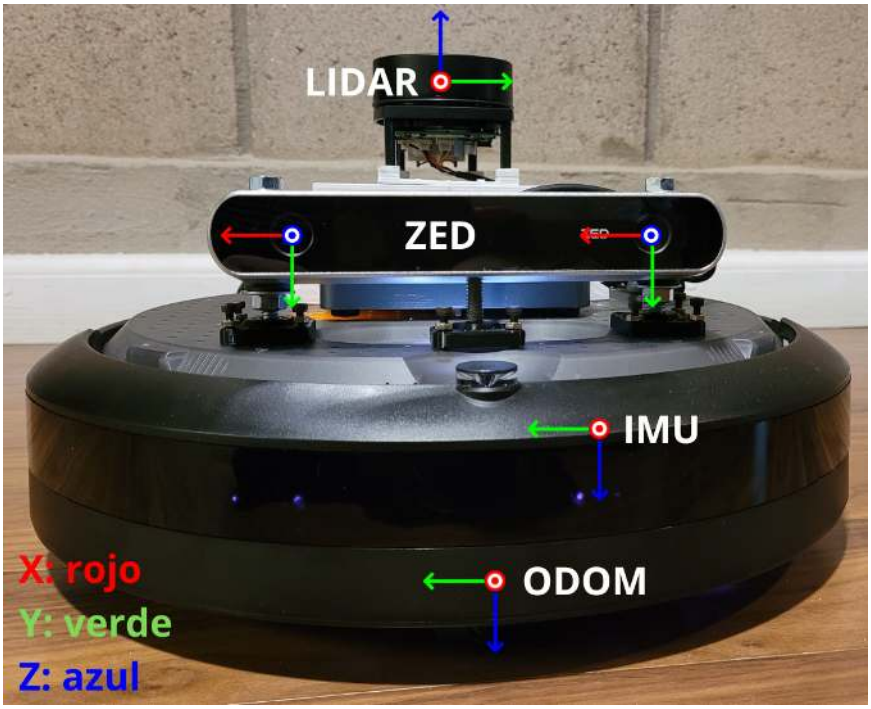


Figura 17: Ejes de coordenadas

Posición del centro de cada eje de coordenadas:

- ODOM (base_link): es el centro de rotación del robot con la altura intersecando con el suelo.
- IMU del robot: se posiciona unos centímetros más adelante y a la izquierda del centro de odometría, mirando el robot de frente estaría en la esquina más alejada a la derecha del espacio que ocupan los botones, sobre la parte del robot quitándole la tapa.

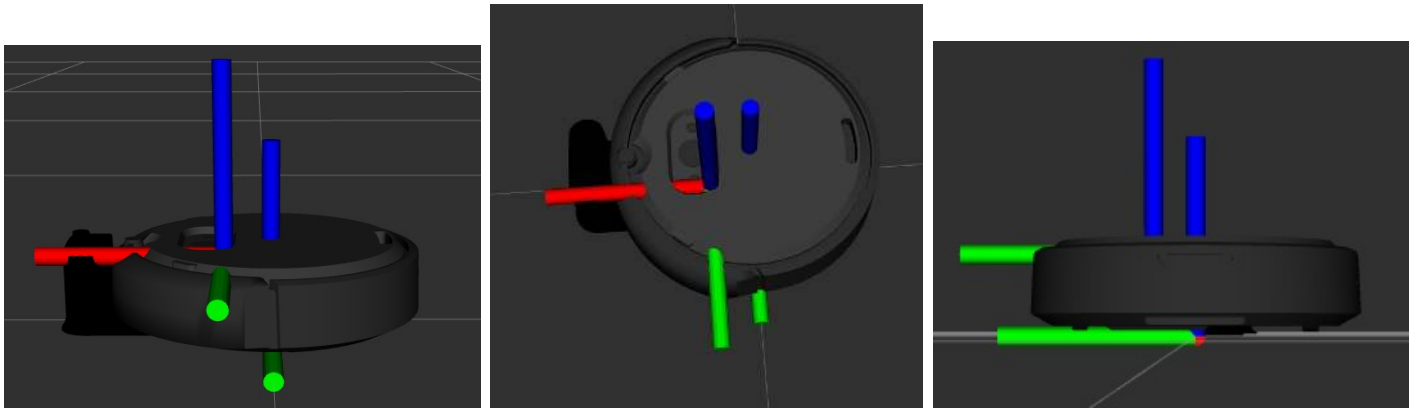


Figura 18: Ejes IMU y odometría

- LiDAR: esta a la altura del laser y centrado sobre la pieza giratoria.
- ZED: El centro real esta sobre en la lente izquierda mirando de frente a la cámara, el otro centro se obtiene a partir de la calibración

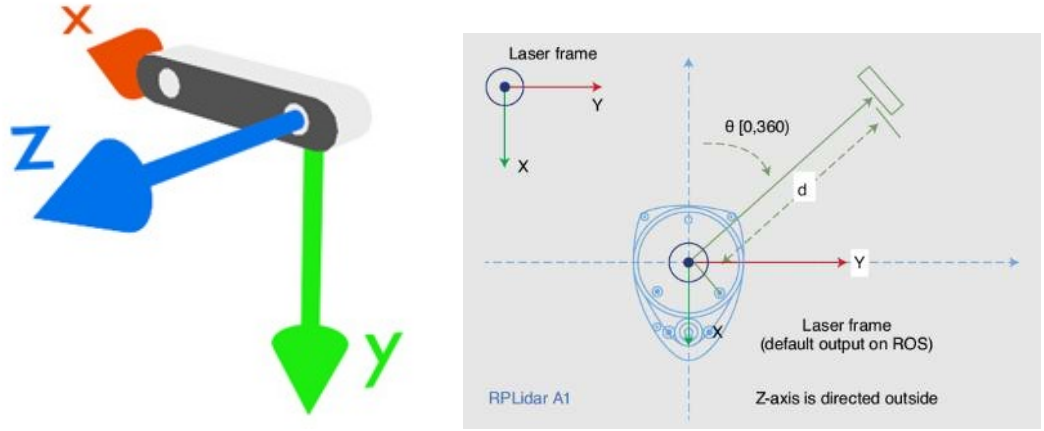


Figura 19: Ejes ZED y LiDAR

3. Calibraciones y puesta a punto

3.1. Intrínseca IMU (Allan Variance)

La primera calibración que realizamos fue la calibración de ruido[24] de la IMU que es para medir cuanto "ruido" tiene ya que es necesaria para futuras calibraciones. Utilizamos el paquete allan_variance_ros[25] de ROS1.

Para realizar la calibración con este paquete solamente se necesita un rosbag de muchas horas (al menos 3) de lecturas de la IMU con la IMU lo más quieta posible. Grabamos dos rosbags: uno de casi 8 horas y el otro de casi 5 horas y medias. Los grabamos con el robot sobre una frazada para aislarlo lo máximo posible de vibraciones.

Como el paquete para calibrar es de ROS1 lo que hicimos fue instalarlo dentro del contenedor de Docker que creamos para correr Kalibr (mas adelante está el link al tutorial) ya que trae una versión de ROS1 instalada.

Como los rosbags los grabamos con ROS2 utilizamos la libreria de python rosbags[26] que se instala con el siguiente comando:

```
pip install rosbags
```

Y con el siguiente comando se convierte un rosbag2 a rosbag1:

```
rosbags-convert ROS2_BAG_INPUT_PATH --dst ROS1_BAG_OUTPUT_.bag
```

Realizamos las calibraciones siguiendo siguiendo el tutorial de allan_variance_ros y obtuvimos los siguientes resultados:

```
#Accelerometer
accelerometer_noise_density: 0.0007300963544454572
accelerometer_random_walk: 4.0126283707379367e-07

#Gyroscope
gyroscope_noise_density: 8.010093278460418e-05
gyroscope_random_walk: 3.4433298154030514e-07

rostopic: '/imu' #Make sure this is correct
update_rate: 100.0 #Make sure this is correct
```

Listing 1: imu.yaml - dataset 8hs

[24] https://github.com/ethz-asl/kalibr/wiki/IMU-Noise-Model
[25] https://github.com/ori-drs/allan_variance_ros
[26] https://ternaris.gitlab.io/rosbags/index.html

```
#Accelerometer
accelerometer_noise_density: 0.000726797601186177
accelerometer_random_walk: 4.099199651049313e-07

#Gyroscope
gyroscope_noise_density: 8.048908640784946e-05
gyroscope_random_walk: 1.3789632962320302e-07

rostopic: '/imu' #Make sure this is correct
update_rate: 100.0 #Make sure this is correct
```

Listing 2: imu.yaml - dataset 5hs

Es probable que el algoritmo de calibración de allan_variance_ros asuma que la gravedad esta presente en los datos de la IMU pero la IMU del Create3 la cancela. Por esto las calibraciones podrían ser incorrectas

3.2. Intrínseca y extrínseca cámara ZED (Kalibr)

Para esta calibración usamos la librería Kalibr[27] que está implementada en ROS1.

El objetivo es realizar la calibración intrínseca de cada uno de los dos lentes que tiene la cámara ZED y la calibración extrínseca entre estos dos lentes, es decir, obtener la matriz de rotación y traslación entre las lentes izquierda y derecha.

La Wiki en el repositorio GitHub de la librería tiene documentados varios tutoriales útiles.

Siguiendo el tutorial para la calibración intrínseca de la cámara el primer paso a realizar es grabar un rosbag dejando la cámara fija y moviendo por delante un patrón de calibración (o target).

Utilizamos un Aprilgrid como patrón de calibración que nos prestó el Laboratorio de Robótica. Está impreso y pegado sobre un tablón de madera. Es el más pesado de los disponibles en el Laboratorio.

```
# config target aprilgrid pesado
target_type: 'aprilgrid' #gridtype
tagCols: 6 #number of apriltags
tagRows: 6 #number of apriltags
tagSize: 0.088 #size of apriltag, edge to edge [m]
tagSpacing: 0.3 #ratio of space between tags to tagSize
#example: tagSize=2m, spacing=0.5m --> tagSpacing=0.25[-]
```

Listing 3: aprilgrid_cifasis_pesado.yaml

[27] https://github.com/ethz-asl/kalibr

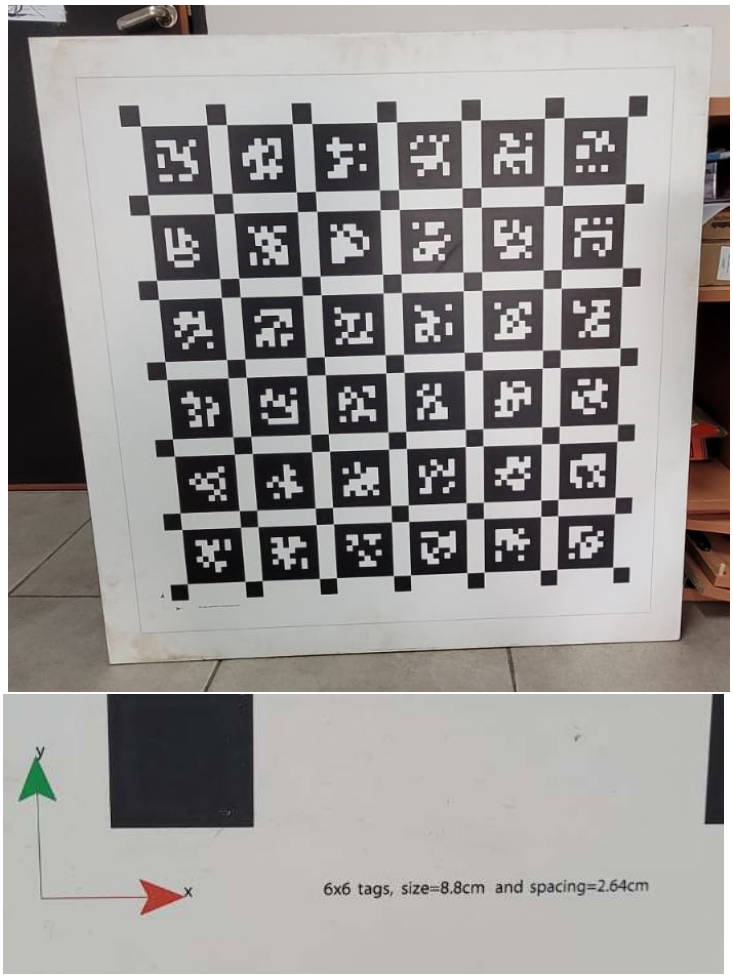


Figura 20: Aprilgrid utilizado

Una vez grabado el rosbag quedan hacer dos pasos para poder seguir con el tutorial:

- Separar las imágenes de la cámara en dos tópicos. Por defecto las imágenes de las dos lentes de la cámara vienen en un sólo tópico concatenadas pero Kalibr las consume en dos tópicos separados. Para eso implementamos un nodo de ROS2 que toma un rosbag2 con las imágenes de la cámara y devuelve otro rosbag2 en el los imágenes de los dos lentes están en dos tópicos distintos. Eso está descripto en la subsección Divisor imágenes cámara.
- Como Kalibr es de ROS1 hay que convertir el rosbag2 a rosbag1 como se explico antes.

Luego siguiendo el tutorial el segundo paso es ejecutar la calibración en sí que va a requerir el rosbag1, la descripción del patrón (aprilgrid_cifasis_pesado.yaml), el modelo de los lentes de la cámara (pinhole-radtan) y el nombre del tópico donde están las imágenes de cada lente.

Para ejecutar Kalibr utilizamos un contender de Docker como contamos en la subsección Docker.

Obtuvimos el siguiente resultado:

```
cam0:
cam_overlaps: [1]
camera_model: pinhole
distortion_coeffs: [-0.16357988928982473, 0.023969035181607864, -0.0008239877650054601,
    -0.0006292969329467902]
distortion_model: radtan
intrinsics: [339.7439962983956, 338.47728312074616, 346.4408167998671, 189.75350916807716]
resolution: [672, 376]
rostopic: /cam_left_image
cam1:
T_cn_cnm1:

- [0.9998724603785629, 0.0015019643650704183, 0.015899908162197378,
    -0.11956037475529557]
- [-0.0014606315672830194, 0.9999955249841186, -0.002610855638075898,
    0.00012208158659800502]
- [-0.01590375842198656, 0.002587298742757583, 0.9998701797499874,
    -4.283602436550253e-05]
- [0.0, 0.0, 0.0, 1.0]
cam_overlaps: [0]
camera_model: pinhole
distortion_coeffs: [-0.1629524598957878, 0.02358689050261179, -0.0007484460900404424,
    -0.0008277151059167675]
distortion_model: radtan
intrinsics: [340.6336303221715, 339.4183581178426, 341.63087263010334, 206.81078004191215]
resolution: [672, 376]
rostopic: /cam_right_image
```

Listing 4: calibracion_zed_kalibr.yaml

3.3. Extrínseca Cámara-IMU (Kalibr)

Para obtener la matriz de rotación y traslación entre la cámara y la IMU intentamos realizar la calibración que ofrece Kalibr.

Tienen un tutorial que incluye un video de como grabar el rosbag para la calibración. En este caso también se necesita un patrón de calibración pero, a diferencia de la calibración de la cámara, el patrón queda fijo y se mueve el robot.

Una vez grabamos el rosbags, dividimos las imágenes de la cámara, lo convertimos a ROS1 y corrimos la calibración notamos que los valores obtenidos no eran los esperados por lo que analizamos los datos y ahí fue que notamos que había un problema con los datos RAW que entrega la IMU del Create3 ya que se cancela la gravedad. Pero Kalibr asume que la gravedad si está presente en los datos por lo que la calibración da valores incorrectos.

Javier Cremona estudiante de doctorado en el Laboratorio de Robótica del CIFASIS modificó kalibr para que no utilice la gravedad en la optimización y realizó una calibración con esta modificación. El siguiente diff muestra los cambios que realizó:

```
diff --git
    a/aslam_offline_calibration/kalibr/python/kalibr_imu_camera_calibration/IccSensors.py
    b/aslam_offline_calibration/kalibr/python/kalibr_imu_camera_calibration/IccSensors.py
index a7238ff..3a705aa 100644
--- a/aslam_offline_calibration/kalibr/python/kalibr_imu_camera_calibration/IccSensors.py
+++ b/aslam_offline_calibration/kalibr/python/kalibr_imu_camera_calibration/IccSensors.py
```

```
@@ -695,7 +695,7 @@ class IccImu(object):
diff --git
    a/aslam_offline_calibration/kalibr/python/kalibr_imu_camera_calibration/IccSensors.py
    b/aslam_offline_calibration/kalibr/python/kalibr_
imu_camera_calibration/IccSensors.py
index a7238ff..3a705aa 100644
--- a/aslam_offline_calibration/kalibr/python/kalibr_imu_camera_calibration/IccSensors.py
+++ b/aslam_offline_calibration/kalibr/python/kalibr_imu_camera_calibration/IccSensors.py
@@ -695,7 +695,7 @@ class IccImu(object):
                w_dot_b = poseSplineDv.angularAccelerationBodyFrame(tk)
                C_i_b = self.q_i_b_Dv.toExpression()
                r_b = self.r_b_Dv.toExpression()
- a = C_i_b * (C_b_w * (a_w - g_w) + \
+ a = C_i_b * (C_b_w * (a_w) + \
                            w_dot_b.cross(r_b) + w_b.cross(w_b.cross(r_b)))
                aerr = ket.EuclideanError(im.alpha, im.alphaInvR * weight, a + b_i)
                aerr.setMEstimatorPolicy(mest)
@@ -996,7 +996,7 @@ class IccScaledMisalignedImu(IccImu):
                w_dot_b = poseSplineDv.angularAccelerationBodyFrame(tk)
                C_i_b = self.q_i_b_Dv.toExpression()
                r_b = self.r_b_Dv.toExpression()
- a = M * (C_i_b * (C_b_w * (a_w - g_w) + \
+ a = M * (C_i_b * (C_b_w * (a_w) + \
                                  w_dot_b.cross(r_b) + w_b.cross(w_b.cross(r_b))))

                aerr = ket.EuclideanError(im.alpha, im.alphaInvR * weight, a + b_i)
@@ -1038,7 +1038,7 @@ class IccScaledMisalignedImu(IccImu):
                C_b_w = poseSplineDv.orientation(tk).inverse()
                a_w = poseSplineDv.linearAcceleration(tk)
                r_b = self.r_b_Dv.toExpression()
- a_b = C_b_w * (a_w - g_w) + w_dot_b.cross(r_b) + w_b.cross(w_b.cross(r_b))
+ a_b = C_b_w * (a_w) + w_dot_b.cross(r_b) + w_b.cross(w_b.cross(r_b))

                C_i_b = self.q_i_b_Dv.toExpression()
                C_gyro_i = self.q_gyro_i_Dv.toExpression()
@@ -1153,7 +1153,7 @@ class IccScaledMisalignedSizeEffectImu(IccScaledMisalignedImu):
                Iy = self.Iy_Dv.toExpression()
                Iz = self.Iz_Dv.toExpression()

- a = M * (C_i_b * (C_b_w * (a_w - g_w)) + \
+ a = M * (C_i_b * (C_b_w * (a_w)) + \
                        Ix * (C_i_b * (w_dot_b.cross(rx_b) + w_b.cross(w_b.cross(rx_b)))) + \
                        Iy * (C_i_b * (w_dot_b.cross(ry_b) + w_b.cross(w_b.cross(ry_b)))) + \
                        Iz * (C_i_b * (w_dot_b.cross(rz_b) + w_b.cross(w_b.cross(rz_b)))) )
```

Resultado de la calibración:

```
cam0:
  T_cam_imu:
  - [-0.013690326215057835, -0.99982495512059, -0.012752806994401067,
     0.008346163851424004]
  - [0.12852147670901287, 0.010888687182533996, -0.9916469465066569,
     0.039733681435786004]
  - [0.9916122251125491, -0.015214979774947623, 0.12834991000303808,
     -0.06014174531141015]
  - [0.0, 0.0, 0.0, 1.0]
  cam_overlaps: [1]
```

```
  camera_model: pinhole
  distortion_coeffs: [-0.16357988928982473, 0.023969035181607864, -0.0008239877650054601,
    -0.0006292969329467902]
  distortion_model: radtan
  intrinsics: [339.7439962983956, 338.47728312074616, 346.4408167998671, 189.75350916807716]
  resolution: [672, 376]
  rostopic: /cam_left_image
  timeshift_cam_imu: 0.453929023622255
cam1:
  T_cam_imu:
  - [0.002270997833929811, -0.9999230001852906, -0.012199847100933914,
    -0.11211184502400572]
  - [0.12595194172727942, 0.012388738462420945, -0.9919589848045323,
    0.04000041595797144]
  - [0.9920337448615177, 0.0007161422746997381, 0.12597037824931734,
    -0.060206706188982516]
  - [0.0, 0.0, 0.0, 1.0]
  T_cn_cnm1:
  - [0.9998724603785623, 0.0015019643650704185, 0.01589990816219738,
    -0.11956037475529557]
  - [-0.0014606315672830196, 0.9999955249841179, -0.0026108556380758986,
    0.00012208158659800502]
  - [-0.01590375842198656 4, 0.0025872987427575833, 0.9998701797499867,
    -4.283602436550253e-05]
  - [0.0, 0.0, 0.0, 1.0]
  cam_overlaps: [0]
  camera_model: pinhole
  distortion_coeffs: [-0.1629524598957878, 0.02358689050261179, -0.0007484460900404424,
    -0.0008277151059167675]
  distortion_model: radtan
  intrinsics: [340.6336303221715, 339.4183581178426, 341.63087263010334, 206.81078004191215]
  resolution: [672, 376]
  rostopic: /cam_right_image
  timeshift_cam_imu: 0.45525879500855976
```

Listing 5: imu_cam_kalibr_gravedad_ignorada.yaml

3.4. Extrínseca LiDAR-odometría

Esta calibración es para obtener la matriz de rotación y traslación entre el Create3 y el LiDAR. Utilizamos el paquete OdomLaserCalibraTool[28] que está implementado en ROS1.

Primero grabamos el rosbag que debe tener las velocidades de las ruedas (el tópico /wheel_vels) y las lecturas del LiDAR (/scan). En el paper[29] sobre el que está basada la librería hay algunos tips de como grabar el rosbag, específicamente en el capítulo VI. EXPERIMENTS en el punto A 5) Trajectories.

El paquete para calibrar lee los datos de la odometría con un tipo de mensaje distinto al que utiliza el Create3. Para resolver esto implementamos un nodo de ROS2 que convierte esos mensajes y los adapta para que se puedan utilizar con el calibrador. Está explicado en la subsección Adecuador para OdomLaserCalibraTool.

[28] https://github.com/MegviiRobot/OdomLaserCalibraTool
[29] https://censi.science/pub/research/2012-joint_calibration.pdf

Una vez adecuado el rosbag y convertido a ROS1 lo que hicimos fue instalar la librería para calibrar en el contenedor de Docker que creamos como se detalla en la subsección Docker y ejecutarlo desde ahí.

Realizamos cuatro de estas calibraciones y este fue el resultado para una de ellas:

```
-------Calibration Results-------
Axle between wheels: 0.25343
LiDAR-odom x: -0.0667046
LiDAR-odom y: 0.00687901
LiDAR-odom yaw: 3.1302
Left wheel radius: 0.0356572
Right wheel radius: 0.0357236

-------Errors (std dev)-------
LiDAR-odom x: 2.54953 mm
LiDAR-odom y: 0.713538 mm
LiDAR-odom yaw: 0.819389 deg

-------Uncertainty-------
Uncertainty Left wheel radius : 0.260329 mm
Uncertainty Right wheel radius : 0.270187 mm
Uncertainty Axle between wheels : 4.18199 mm
Uncertainty LiDAR-odom-x : 1.27925 mm
Uncertainty LiDAR-odom-y : 2.81666 mm
Uncertainty LiDAR-odom-yaw : 0.148252 deg
```

Listing 6: resultado3.txt (calibración LiDAR-odom)

Hay que notar que este método da solamente la rotación yaw y la traslación sólo la da de las componentes x e y por lo que falta una dimensión que debe ser calculada manualmente esta es la altura desde el suelo hasta el LiDAR, para la componente z.

Juntando toda esta información se puede agregar al tf de ROS2 la rototraslación entre el LiDAR y el base_link del Create3. Este tutorial[30] detalla como realizar un nodo de ROS2 para realizarlo y también un comando para hacerlo directamente que lo dejamos acá con datos al azar de la rototraslación:

```
ros2 run tf2_ros static_transform_publisher --x 0 --y 0.1 --z 0.3 --yaw 0 --pitch 0 --roll 0
    --frame-id base_link --child-frame-id laser
```

3.5. Extrínseca LiDAR-cámara

Hay algunas librerías para hacer esta calibración que requieren que la cámara vea el laser del LiDAR pero como la cámara actual (ZED) no lo detecta porque tiene filtro infrarrojo no lo probamos.

Que queda como una posibilidad para una prueba futura.

3.6. Extrínseca cámara-odometría

Se podría probar con el paquete [31]CAM_ODO_CALIB que es del mismo grupo que hizo el paquete que usamos en la calibración LiDAR-odometría.

3.7. Docker

Como algunas librerías que utilizamos en las calibraciones están implementadas en ROS1 y no encontramos análogos para ROS2 decidimos utilizarlas igual.

[30] https://docs.ros.org/en/humble/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Static-Broadcaster-Py.html

[31] https://github.com/MegviiRobot/CamOdomCalibraTool

Para que no se generen problemas de tener instalada una versión de ROS2 y ROS1 simultáneamente decidimos que lo mejor era instalar una versión de ROS1 en un contenedor de Docker para que quede aislada.

Para nuestra suerte Kalibr tiene disponible una imagen de Docker que trae instalada una versión de ROS1 y Kalibr sobre Ubuntu 16.04, 18.04 o 20.04.

Para hacer la instalación del contenedor seguimos el tutorial[32] y elegimos la versión de Ubuntu 20.04.

Una vez creado el contenedor sobre este mismo fuimos agregando e instalando los demás paquetes de ROS1 que utilizamos.

Un inconveniente que tuvimos siguiendo el tutorial para armar la imágen de Docker es que al ejecutar el comando:

```
docker build -t kalibr -f Dockerfile_ros1_20_04 .
```

Se nos colgaba el sistema operativo. Para solucionarlo modificamos en el archivo Dockerfile_ros1_20_04 la línea:

```
catkin build -j$(nproc)
```

por:

```
catkin build -j1
```

Así utiliza un solo núcleo del procesador en vez de todos los disponibles.

4. Implementaciones propias

4.1. Repositorio

Creamos un repositorio[33] en el GitLab del DCC donde subimos el código que escribimos, el resultado de algunas calibraciones y demás cosas relacionadas al proyecto.

En el repositorio hay un paquete de ROS2 en Python que escribimos y se llama bag_nodes_py que tiene tres nodos que cumplen distintas funciones:

- bag_stereo_cam_splitter
- create3_bag_adequation_for_OdomLaserCalibraTool
- rosbag_recorder

Como todos los paquetes de ROS2 para compilarlo y poder utilizarlo hay que poner la carpeta bag_nodes_py adentro de la carpeta src de algún workspace de ROS2, por ejemplo en: ∼/ros2_ws/src/bag_nodes_py

Y desde el directorio ∼/ros2_ws ejecutar:

```
colcon build --symlink-install
source install/local_setup.bash
```

4.2. Divisor imágenes cámara

El primero de los nodos que implementamos es el bag_stereo_cam_splitter que sirve para transformar un rosbag2 que tiene las imágenes de las dos lentes de la cámara estéreo en un solo tópico en otro rosbag2 que tiene las imágenes de las dos lentes por separado en distintos tópicos.

[32] https://github.com/ethz-asl/kalibr/wiki/installation

[33] https://repositorio.cifasis-conicet.gov.ar/alerivo/trayectoria-con-create3

Esto fue necesario ya que la mayoría de las veces las imágenes de las lentes se necesitan por separado, en distintos tópicos.

Para ejecutarlo se debe utilizar:

```
ros2 run bag_nodes_py bag_stereo_cam_splitter --ros-args -p
    input_bag_path:=/path/al/rosbag/de/entrada
```

Se pueden especificar mas argumentos agregándolos precedidos por -p a cada uno. En el archivo donde esta implementado el nodo bag_stereo_cam_splitter.py se pueden ver los distintos argumentos aunque estos son opcionales porque que ya tienen un valor por defecto asignado.

4.3. Adecuador para OdomLaserCalibraTool

El paquete OdomLaserCalibraTool es el que utilizamos para hacer la calibración externa entre la odometría y el LiDAR. El paquete recibe de entrada las velocidades de la rueda izquierda y derecha con el tipo de mensaje geometry_msgs::Vector3Stamped pero el Create3 publica esa información con el tipo de mensaje irobot_create_msgs/msg/WheelVels.

Por esto implementamos un nodo de ROS2 que tome un rosbag2 y convierte esos mensajes para poder utilizar el calibrador. Los mensajes del LiDAR si ya vienen con el tipo sensor_msgs/LaserScan que es el que usa el calibrador así que a esos los deja igual.

Para ejecutarlo se debe utilizar:

```
ros2 run bag_nodes_py create3_bag_adequation_for_OdomLaserCalibraTool --ros-args -p
    input_bag_path:=/path/al/rosbag/de/entrada
```

Se pueden especificar mas argumentos agregándolos precedidos por -p a cada uno. En el archivo donde esta implementado el nodo create3_bag_adequation_for_OdomLaserCalibraTool.py se pueden ver los distintos argumentos aunque estos son opcionales porque que ya tienen un valor por defecto asignado.

4.4. Grabación rosbag2

El nodo que implementamos en el archivo rosbag_recorder.py fue pensado para grabar rosbags2 con MCAP como storage plugin con el objetivo de que se puedan grabar varios al mismo tiempo y en diferentes máquinas. Pero no funciona como era esperado.

Al ejecutar el paquete hay que asignarle los siguientes parámetros:

- rosbag_path: una string con el path donde guardar el rosbag2
- rosbag_topics: una string con los topics a grabar separados por espacio

Ejemplo para ejecutarlo:

```
ros2 run bag_nodes_py rosbag_recorder --ros-args -p rosbag_path:=/path/rosbag/de/salida -p
    rosbag_topics:='/wheel_vels /wheel_ticks'
```

Una vez ejecutado se queda escuchando el tópico /rosbag_recorder y al recibir el mensaje “start” comienza a grabar hasta recibir el mensaje “stop” que termina la grabación.

Comandos en la terminal para empezar y dejar de grabar:

```
ros2 topic pub -1 /rosbag_recorder std_msgs/String "{data: 'start'}"
ros2 topic pub -1 /rosbag_recorder std_msgs/String "{data: 'stop'}"
```

El problema es que cuando hay varias instancias de ROS2 en la misma red tanto el mensaje de start como stop a veces llega a una sola de las instancias y no a todas.

El funcionamiento esperado es que cuando se envíe un mensaje de start o stop llegue a todas las instancias de ROS2 así todas las grabaciones inician o terminan.

Tal vez hay una manera de configurar para que el mensaje sea enviado a todas las instancias.

Finalmente no utilizamos este nodo en la grabación de los datasets.

5. Grabación dataset

5.1. Objetivo

Grabar un conjunto de rosbags2 que contengan todos los datos que brinda el robot: tanto los disponibles nativamente por el Create3 como por los sensores adicionales incorporados. Adicionalmente grabarlo en un entorno que ayude y sea útil para desarrollar aplicaciones de robótica: que tenga objetos que luego puedan ser reconocidos, agregar marcadores que ayuden a tener referencias de ubicación y agregar una cámara externa para poder tener una referencia adicional.

La grabación de estos rosbags está pensada para que los futuros estudiantes de la materia Optativa Robótica Móvil desarrollen y prueben los algoritmos estudiados en la materia y en los trabajos prácticos que se realicen.

5.2. Marcadores ArUco

ArUco[34] es una librería que sirve para hacer estimaciones de la ubicación de unos marcadores específicos en relación a una cámara. Su implementación depende de OpenCV y eigen3 y hay paquetes disponibles de ROS2 para utilizarla. Reconoce marcadores ya existentes tal como el AprilTag y además tiene sus propios marcadores también llamados ArUco.

Estos marcadores, que son de dos dimensiones, al tener una forma y tamaño conocido permiten conocer la posición a la que están de manera relativa a la cámara.

Gráficamente parecen similares a los códigos QR y siempre pertenecen a un diccionario de los cuales ya hay varios predefinidos.

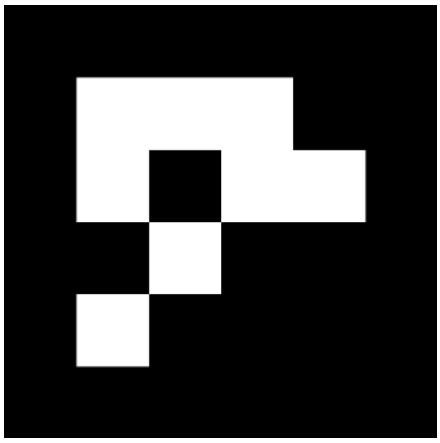


Figura 21: Marcador ArUco ID 0 del diccionario DICT_4X4_100

Es posible generar los marcadores ArUco online[35].

[34] `https://www.uco.es/investiga/grupos/ava/portfolio/aruco/`
[35] `https://chev.me/arucogen/`

Para detectar los marcadores ArUco y publicar las detecciones directamente en ROS2 como mensajes recomendamos el paquete de ROS2 ros2_aruco[36] aunque también se puede utilizar directamente la librería nativa de ArUco[37] en algún paquete que se implemente de ROS2.

Para poder utilizar el paquete tuvimos que realizar los siguientes pasos extras además de clonar el repositorio y compilarlo:

- Instalar la librería transforms3d que marca como dependencia,
- instalar el paquete de ROS2 tf_transformations[38] (aunque podría ya estar instalado) e
- instalar específicamente la versión 4.6.0.66 de OpenCV para solucionar el error:
  AttributeError: module 'cv2.aruco' has no attribute 'Dictionary_get'. Did you mean: 'Dictionary'?

```
pip install opencv-contrib-python transforms3d
sudo apt-get install ros-humble-tf-transformations
pip install opencv-contrib-python==4.6.0.66
```

Listing 7: comandos utilizados para que funcione ros2_aruco

Para lanzar el detector utilizamos el siguiente comando:

```
ros2 run ros2_aruco aruco_node --ros-args --params-file arucos_cubos_IDs_0-49.yaml
```

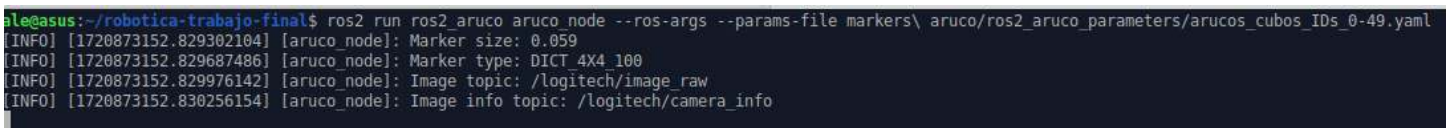


Figura 22: ros2_aruco ejecutado en la terminal

Notar que los parámetros del nodo de ROS2 lo pasamos mediante un archivo YAML.
Utilizamos los marcadores ArUco para ponerlos en lugares estratégicos que puedan ser útiles para los posibles usos del dataset tal como en las paredes, en los cilindros y en el mismo robot.

Figura 23: marcadores ArUco visualizados en RViz2

[36] https://github.com/JMU-ROBOTICS-VIVA/ros2_aruco
[37] https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html
[38] https://index.ros.org/p/tf_transformations/

5.3. Puntos de referencia

- Paredes: son censadas por el LiDAR y este puede detectar a que distancia se encuentran.
- Cilindros: también son censados por el LiDAR y, además de la distancia, podemos calcular el diámetro. El dato del diámetro es conocido y se podría utilizar en algún experimento o trabajo práctico para comparar que los resultados sean correctos.

  La primera opción fue utilizar botellas pintadas o rodeadas de papel (sino el laser del LiDAR no rebota o se desvía y no las lee bien) para generar las columnas pero a la complejidad de conseguir la cantidad necesaria y del mismo diámetro y del transporte decidimos cortar en varias secciones un caño de PVC. El tamaño de estos es 22.8cm de alto y 10.4cm de diámetro (es un caño de 4").
- Marcadores ArUco: Para poner los marcadores ArUco en los cilindros generamos varios de estos marcadores y creamos cubos, sin base, donde cada cara del cubo es un marcador distinto para poner encima de cada cilindro y que estos sean visibles desde todos los ángulos, inclusive desde arriba. Generamos 3 "tipos" distintos de marcadores ArUco y cada uno tiene un tamaño distinto. Por eso es necesario lanzar 3 instancias de ros2_aruco ya que uno de los parámetros es el tamaño del marcador.

1. 10 cubos en los que cada una de las 5 caras visibles tiene un marcador cuyas IDs son consecutivas. Situamos a los cubos sobre los cilindros. La cara superior de este cubo es detectada por la cámara externa y las laterales por el robot cuando que están en su rango de visión (en este caso conviene hacer un filtro para que sólo se detecten los marcadores correspondientes a la cara superior, pues notamos que la cámara externa a veces ve los ArUcos laterales).

```
/aruco_node:
    ros__parameters:
    marker_size: 0.059
    aruco_dictionary_id: DICT_4X4_100
    image_topic: /logitech/image_raw
    camera_info_topic: /logitech/camera_info
    # camera_frame: map
```

Listing 8: arucos_cubos_IDs_0-49.yaml

2. Un marcador colocado sobre el robot, específicamente sobre la NUC, para que sea visible desde la cámara externa.

```
/aruco_node:
    ros__parameters:
    marker_size: 0.074
    aruco_dictionary_id: DICT_4X4_100
    image_topic: /logitech/image_raw
    camera_info_topic: /logitech/camera_info
    # camera_frame: map
```

Listing 9: arucos_robot_ID_50.yaml

3. Algunos son colocados en las paredes para que los visualice la cámara ZED montada en el robot.

```
/aruco_node:
    ros__parameters:
    marker_size: 0.148
    aruco_dictionary_id: DICT_4X4_100
    image_topic: /logitech/image_raw
    camera_info_topic: /logitech/camera_info
    # camera_frame: map
```

Listing 10: arucos_A4_IDs_50-65.yaml

5.4. Cámara externa

Agregamos una cámara externa para poder visualizar y tener una referencia del escenario completo del experimento ubicada a 2.67m del suelo (tomado desde la lente).

Utilizamos la cámara web Logitech C930e[39] que fue comprada por el CIFASIS y que ofrece buena calidad de imagen.

Figura 24: Logitech C930e

Para lanzarla usamos:

```
ros2 run usb_cam usb_cam_node_exe --ros-args -p image_width:=1280 -p image_height:=720 -p
    framerate:=15.0 -p video_device:=/dev/video1 -r __ns:=/external_cam
```

Hay que chequear el parámetro video_device porque la cámara se puede montar en distintas rutas e ir cambiando cada vez. Para ver en que ruta está se puede usar la herramienta V4L2 instalándola así:

```
sudo apt-get update
sudo apt-get install v4l-utils
```

Y usándola así:

```
v4l2-ctl --list-devices
```

Para calibrar la cámara usamos el paquete camera_calibration[40] de ROS2 ejecutandolo con el siguiente comando:

```
ros2 run camera_calibration cameracalibrator --size 9x7 --square 0.02 --ros-args --remap
    image:=/external_cam/image_raw
```

[39] https://support.logi.com/hc/en-us/articles/360024321133-Getting-started-Webcam-C930e
[40] https://docs.ros.org/en/rolling/p/camera_calibration/tutorial_mono.html

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	TBD	0.15-12	TBD	White objects
Angular Range	Degree	n/a	0-360	n/a	
Scan Field Flatness	Degree	-1.5		1.5	
Distance Resolution	mm	n/a	<0.5 <1% of the distance	n/a	<1.5 meters All distance range*
Angular Resolution	Degree	n/a	≤1	n/a	5.5Hz scan rate
Sample Duration	Millisecond(ms)	n/a	0.125	n/a	
Sample Frequency	Hz	n/a	≥8000	8010	
Scan Rate	Hz	1	5.5	10	Typical value is measured when RPLIDAR A1 takes 360 samples per scan

Figura 12: Especificaciones RPLIDAR A1

Connection Type	USB
USB VID_PID	0843
USB Protocol	USB 2.0
USB Speed	High-speed
UVC Support	Yes (UVC 1.5)
Microphone	Yes
Microphone Type	Dual Stereo
Lens and Sensor Type	Glass
Focus Type	Auto
Optical Resolution	True: 3MP Software Enhanced: No
Diagonal Field of View (FOV)	90°
Horizontal Field of View (FOV)	82.1°
Vertical Field of View (FOV)	52.2°
Focal Length	Not applicable
Image Capture (4:3 SD)	Not applicable
Image Capture (16:9 W)	Not applicable
Video Capture (4:3 SD)	Not applicable
Video Capture (16:9 W)	Not applicable
Frame Rate (max)	1080P @ 30fps
Right Light	Right Light 2
Video Effects (VFX)	Not applicable
Buttons	Not applicable
Indicator Lights (LED)	Yes
Privacy Shade	Yes (Hardware)
Tripod Mounting Option	Yes
Universal Clip Adjustability (range)	Not applicable
Cable Length	6 feet or 1.83 meters

Product Dimensions				
Product component	**Width**	**Depth/Length**	**Height**	**Weight**
Webcam	94 mm (3.70 inch)	24 mm (0.94 inch)	29 mm (1.14 inch)	162 g (5.71 ounce)

Figura 25: Especificaciones Logitech C930e

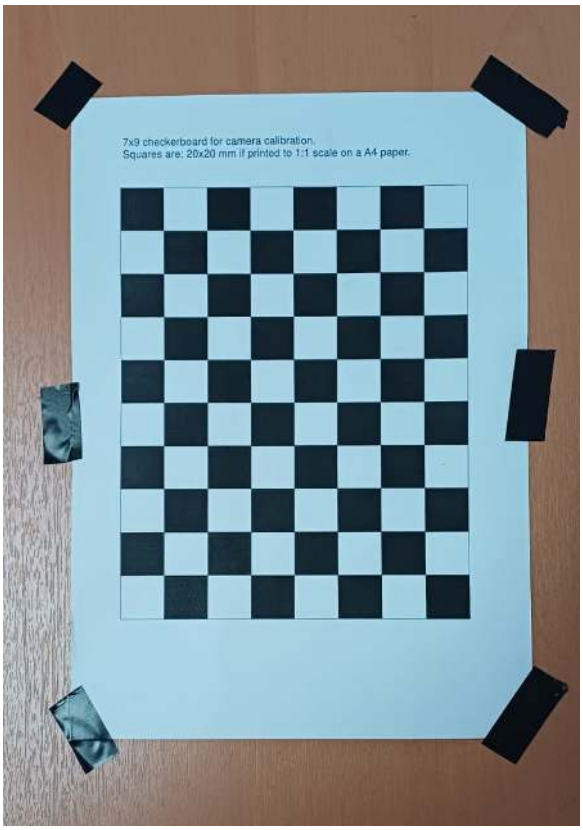


Figura 26: Checkerboard usado para calibrar cámara Logitech

Figura 27: Cámara Logitech en el entorno de grabación

```
image_width: 1280
image_height: 720
camera_name: narrow_stereo
camera_matrix:
  rows: 3
  cols: 3
  data: [793.57507, 0. , 625.92331,
         0. , 794.40769, 368.52218,
         0. , 0. , 1. ]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
```

```
  cols: 5
  data: [0.078405, −0.146406, 0.000249, 0.000565, 0.000000]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1., 0., 0.,
         0., 1., 0.,
         0., 0., 1.]
projection_matrix:
  rows: 3
  cols: 4
  data: [782.49353, 0. , 623.84826, 0. ,
         0. , 801.75928, 368.11914, 0. ,
         0. , 0. , 1. , 0. ]
```

Listing 11: logitech_camera_info.yaml

Como primera opción habíamos usado la cámara gran angular de un celular conectada a través de Iriun Camera[41]. El software funcionó correctamente e incluso pudimos ejecutar librerías de detección de ArUco en tiempo real pero al enviar la información de vídeo inalámbricamente se saturaba la red Wi-Fi, que también se utiliza para la comunicación entre el usuario y la NUC.

5.5. Trayectorias predefinidas

Para hacer las trayectorias predefinidas utilizamos como ayuda el paquete create3_examples[42]. En particular el nodo *dance* dentro del directorio *create3_examples_py*.

Ese nodo establece una trayectoria predefinida que se define con movimientos de una duración determinada en segundos, la velocidad hacia adelante o hacia atrás en m/s y la velocidad de rotación en grados/s.

Definimos nodos nuevos para las trayectorias que deseamos. Para utilizarlas primero hay que instalar el paquete como se indica en su repositorio y luego agregarle los archivos del directorio *create3_examples* de nuestro repositorio (uno es reemplazar uno que ya existe *setup.py*) y luego se ejecuta así:

```
ros2 run create3_examples_py create3_dance create3_circulo.py
```

Actualmente las trayectorias están "hardcodeadas" por lo que modificarlas requiere de un tedioso trabajo manual. Como trabajo futuro sería bueno escribirlas para que, dando los valores requeridos (velocidad, radio del circulo, lado del rectángulo, cantidad de vueltas, etc) como argumentos, realice las trayectorias.

5.6. Entorno de grabación

Sugerencias a tener en cuenta al momento de preparar un entorno de grabación:

- Usar una habitación o ambiente donde los limites (paredes) sean regulares,
- que no haya objetos que obstruyan la detección del LiDAR o la visión de la cámara,
- colocar los marcadores ArUco de la pared de manera equidistante,
- que los códigos ArUco de las paredes estén a la misma altura (en este caso esta aproximadamente a 40 cm del suelo pero al estar colocados en distintas superficies irregulares puede variar unos 3cm),
- colocar las columnas de una manera regular o en caso contrario tomar al menos las medidas a las paredes y columnas aledañas,

[41] https://iriun.com/

[42] https://github.com/iRobotEducation/create3_examples

- tomar las medidas (en este caso era complejo, pues no todos los ArUcos estaban en las paredes, y había escritorios, sillas y estanterías que generaban límites muy irregulares,
- tener buena iluminación y
- que la batería del Create3 esté bien cargada previo a empezar la grabación.

5.6.1. Medidas del entorno de grabación

Hicimos un esquema del espacio que armamos con algunas medidas. Las distancias (número en el centro de las flechas) están en centímetros, los números en los rectángulos representan los números correspondientes a los códigos ArUco de las paredes (que están a aprox. 40cm de suelo) y los que están cerca de las columnas representan los números de la cara superior y la cara que ve a la pared donde está el pizarrón (en el esquema sería a la derecha).

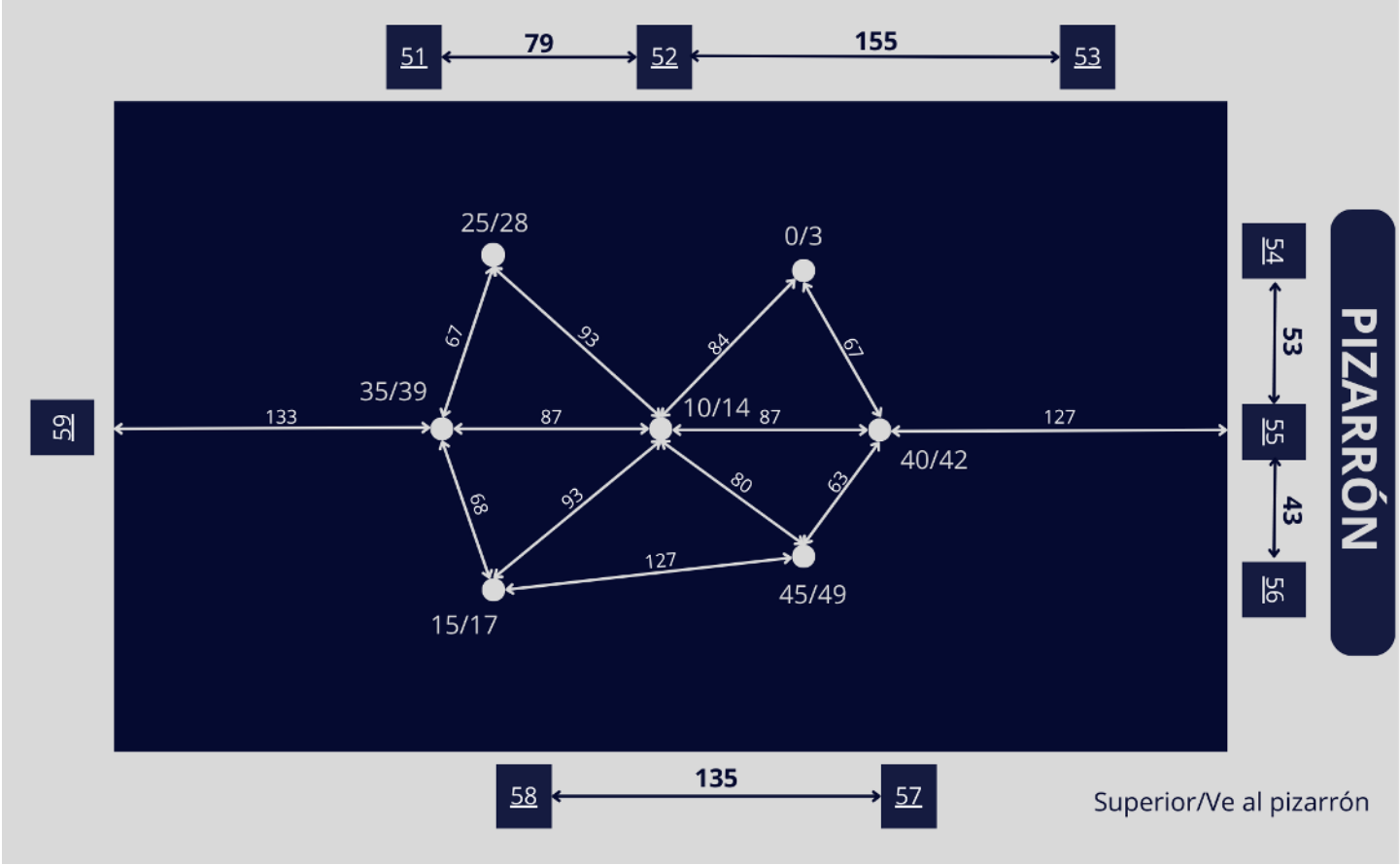


Figura 28: Esquema del entorno de grabación

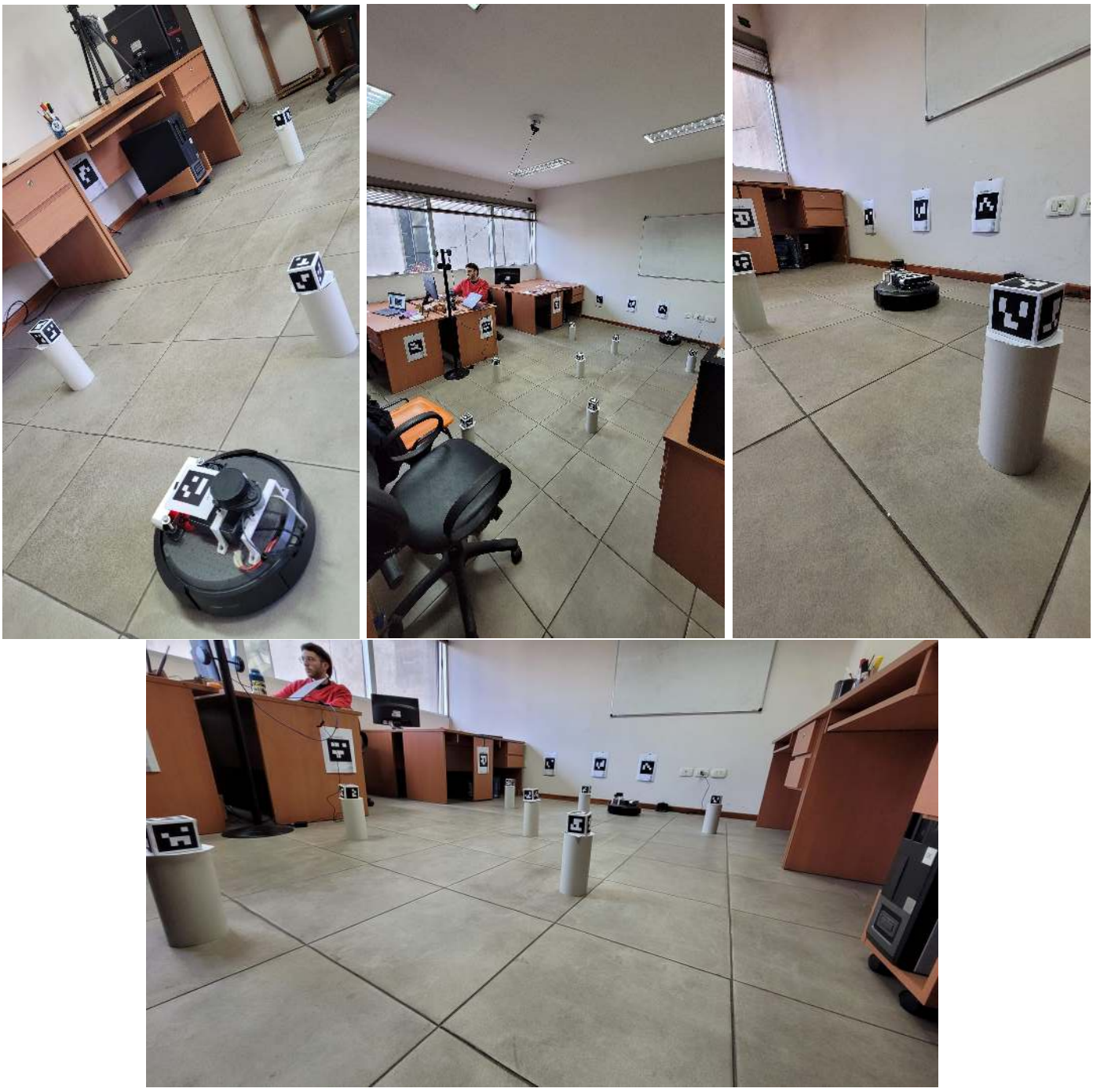


Figura 29: Entorno de grabación

5.7. Grabación rosbags

Los rosbags que grabamos usan MCAP[43] como storage plugin que tiene una performance mejor que sqlite3 y es la opción por defecto desde la versión Iron Irwini (iron) de ROS2.

[43] `https://mcap.dev/`

Para cada trayectoria grabada hay dos rosbags grabados:

- Uno desde la NUC que incluye todos los datos de los sensores tanto internos del Create3 como los sensores adicionales que le agregamos al robot.

```
ros2 bag record -s mcap -o NOMBRE_DATASET /battery_state /camera_info /cliff_intensity
    /cmd_audio /cmd_lightring /cmd_vel /dock_status /hazard_detection /image_raw /imu
    /interface_buttons /ir_intensity /ir_opcode /kidnap_status /mouse /odom /scan
    /slip_status /stop_status /tf /tf_static /wheel_status /wheel_ticks /wheel_vels
```

Listing 12: comando grabacion rosbag NUC

- y el otro desde una computadora externa que incluye los datos de la cámara externa:

```
ros2 bag record -s mcap -o NOMBRE_DATASET /logitech/camera_info /logitech/image_raw
```

Listing 13: comando grabacion rosbag computadora externa

5.8. Sincronización temporal con luz

Para incorporar un método adicional de sincronización entre los dos rosbags de cada dataset, además del NTP, hicimos señas de luces que son tomadas por ambas cámaras por si fuera necesario.

5.9. Datasets

Grabamos los siguientes datasets:

- Trayectoria 8: dos datasets. El robot realiza la trayectoria con la forma de un 8 (o infinito) dos veces en cada dataset. Cada círculo tiene un radio de 0.5m.
- Trayectoria círculo: dos datasets. El robot realiza la trayectoria con la forma de un círculo de radio 1m a 0.2m/s hacia la izquierda dos veces en cada dataset.
- Trayectoria libre: un dataset. Realizamo una trayectoria libre que comandamos mediante el teleop.
- Trayectoria rectángulo: hicimos algunos datasets intentando hacer la trayectoria de un rectángulo pero los girós de 90 grados que hacía no eran perfectos, tenían error, y se terminaba chocando con los objetos. Para que esto no pase fuimos moviendo el robot manualmente en el medio de la grabación por lo que aconsejamos no usar estos datasets o, al menos, hacerlo con el cuidado de saber que el robot fue manipulado manualmente durante la grabación.

Las calibraciones de las cámaras no están incluidas en los datasets grabados. Es decir, los tópicos /camera_info tienen valores por defecto. Tampoco están cargadas las calibraciones extrinsecas de los sensores en el tf. Queda cómo trabajo futuro agregarlas.

Se pueden encontrar en el siguiente link: `http://fs01.cifasis-conicet.gov.ar:90/~pire/datasets/create3%20datasets/`