

# Lab Aprendiendo Docker

Oscar Grande - Didier Posse.

October 2025

## 1 Desarrollo de Herramientas.

### 1.1 Herramienta ROS (Robot Operating System):

- **Que es:** Es un marco de software o Middleware (capa de intermediaria de conexión que facilita la comunicación entre diferentes sistemas, aplicaciones o base de datos.) diseñado para el desarrollo y control de robots, con un conjunto de herramientas y librerías que ayudan a que los diferentes componentes de un robot se puedan comunicar fácilmente entre sí.
- **Que hace:** Exactamente permite una programación de forma modular, es decir:
  - Un modulo (nodo) puede encargarse de un sensor.
  - Otro modulo puede calcular la trayectoria.
  - Otro controlar los motores.Todos estos nodos lo que harían sería una comunicación entre sí mediante **topics**(mensajes), servicios o acciones, sin que el programador tenga que manejarlos directamente los detalles de red o sincronización "en teoría una automatización".
- **Nodo:** Programa que cumple una función específica como lectura de datos en un robot.
- **Topic:** Canal de comunicación donde los nodos publican o suscriben mensajes.
- **Mensaje:** Estructura de datos que los nodos envían a través de los topics.
- **Servicio:** Comunicación puntual (pregunta-respuesta) entre dos nodos.
- **Master:** Nodo central que coordina el registro y la comunicación entre nodos.
- **Launch File:** Archivo que permite lanzar varios nodos al mismo tiempo.

- **Alcance en sector IoT:**

El IoT buscar conectar dispositivos fisicos a internet para el intercambio de datos, se controlen remotamente y tomen decisiones inteligentes. Ros lo que hace es compartir esa filosofia donde conectar multiples componentes inteligentes y permitir su comunicacion y cooperacion en la red, por eso esta ha evolucionado hasta ser una herramienta muy usada en IoT avanzado o en la industria 4.0.

Por ello cumplen unas funciones claves como la comunicacion distribuida con alta estandarizacion, escalabilidad, compatibilidad con hardware y simuladores y ademas integracion en la nube donde podran enviar y recibir datos en plataformas de nube para un analisis o control remoto.

## 1.2 Herramienta Movelt:

- **Que es:** Plataforma de software desarrollada con un paquetes de ROS, especializada en planificacion, control y manipulacion de robots, encargandose de que un robot se mueva de forma inteligente y segura sin chocar con objetos, calculando trayectorias necesarias.

- **Que hace:** Movelt combina varios campos avanzados como:

- **Cinematica directa e inversa:** Calculando como deben moverse las articulaciones para llegar a un punto.
- **Planificacion Trayectorias:** Determina el mejor camino que debe seguir el robot.
- **Evita Colisiones:** Detecta obstaculos en su entorno usando sensores o modelos 3D.
- **Control en tiempo real.**
- **Integracion con Hardware:** Como Gazebo o RViz.

- **Alcance en sector IoT.**

Movelt representara la capa de movimiento y autonomia a los sistemas roboticcos que forman parte de una red inteligente. Donde pueden recibir informacion desde la nube, y a partir de los sensores IoT envian la informacion y Movelt adaptar el movimiento segun los datos dados.

- Control remoto de robots
- Interaccion con sensores.
- Automatizacion inteligente.
- Integracion con sistemas MES/Scada (monitoreo y control industrial.)
- Optimizacion en tiempo real.

### 1.3 Herramienta Gazebo

- **Que es:** Gazebo es un simulador 3D de robots y entornos fisicos, diseñado para trabajar en conjunto con ROS. Permitiendo probar y visualizar como se moveria un robot real en un entorno virtual, con fisica realista y demás.
- **Que hace:** Gazebo crea el entorno virtual de simulacion donde se puede colocar distintos tipos de robots, objetos que sean de incluir en el entorno y sensores para este. Luego al ejecutar el robot con ROS y GAZEBO respondera igual que al mundo real con propiedades:
  - Gravedad, friccion, colisiones y dinamica del movimiento.
  - Prueba de algoritmos de control, vision o planificacion sin tener el robot fisico.
  - Ahorra tiempo y recursos en el desarrollo de prototipos.

#### • Alcance en IoT.

BAZEBO + IoT = Simulacion de ecosistemas conectados. Donde se usa para simular entornos completos donde multiples dispositivos o robots estan conectados y se comunican entre si. Suponga una simulacion de una gran suma de robots y todos estan interconectados y hacen envio de datos a un servidor virtual todo esto con pruebas sin hardware real, lo que es muy util en desarrollo o en investigacion. Con funciones claves de simulacion de sensores, pruebas de conectividad, analisis del desempeño, integracion con la nube y pruebas anteriores al despliegue real.

## 2 Desarrollo de sistemas.

### 2.1 LIDAR.

#### • Que es Lidar:

Lidar una tecnologia que utiliza **pulsos laser** para medir la distancia entre el sensor y los objetos del entorno. Donde el sensor emite un rayo laser y este rebota en un objeto, el sensor lo que hace es medir el tiempo de retorno del rayo y con esa informacion calcula la distancia precisa, todo esto repetido en millones de veces por segundo, esto permite construir un mapeo 3D muy detallado del entorno. Conformado por una fuente de luz laser, un receptor (fotodiodo o sensor optico), la electronica que mide el tiempo de vuelo y el software que genera una nube de puntos 3D que representa el entorno.

#### • Tipos.

- LIDAR 2D: Escanea en un solo plano, usado en robots moviles en interiores.
- LIDAR 3D: Escaneo en multiples planos, aplicado en robotica avanzada.

- LIDAR estado sólido: Usando modulares ópticos aplicado en drones ligeros.

LIDAR flash: Captura de imagen 3D completa con un solo pulso. "seguridad".

LIDAR multietílico: Detecta múltiples rebotes del mismo pulso.

- **LIDAR dentro de IoT.**

En un sistema IoT la aplicación de LIDAR son:

- Enviar datos de distancias y formas de objetos a la nube o servidor IoT.
- Permite que otros dispositivos tomen decisiones coordinadas.
- Se usa entornos compartidos, donde múltiples sensores LIDAR comunican información a través de redes IoT.

## 2.2 Tecnología SLAM

- **Qué es.**

(Localización de mapeo simultáneo) Es el proceso por el cual el robot construye un mapa de su entorno mientras calcula su propia posición dentro de él, sin depender de GPS u otra referencia externa. Donde los sensores capturan el entorno, este genera un mapa estimado del lugar, a medida que se mueve actualiza su posición dentro del mapa y si detecta nuevos obstáculos los incorpora en tiempo real.

- **Tipos.**

- LIDAR-SLAM: más preciso en distancias incluso sin luz.
- Visual SLAM: usa visión por computadora, detectando colores y formas.
- RGB-D SLAM: combina imagen y profundidad.
- Inercial SLAM: Estima movimiento y orientación.
- Multisensor SLAM: Mayor robustez y precisión.

- **SLAM dentro del IoT.**

Ella se convierte en una herramienta colaborativa y conectada:

- Los mapas generados por un robot pueden compartirse con otros dispositivos por red.
- Varios robots pueden cooperar para construir un mapa global compartido.
- La nube puede almacenar y procesar los datos SLAM para mejorar la navegación futura.

### 2.3 Aplicacion de un cuadrupedo con PyBullet.

Para la simulación del cuadrúpedo se sigue el siguiente proceso:

- crear un directorio donde en el creamos e incluiremos archivos con el código (.py), un txt quien va a contener las librerías usadas y el Dockerfile quien hará la Dockerización para el entorno virtual. los archivos son los siguientes:

```
21 de ene 2020  
diferenciado y posee visión por IMU. Canguro laptop-19 Python3 [Cangurosped]  
diferenciado y posee visión por IMU. Canguro laptop-19 Python3 [Cangurosped]  
diferenciado y posee visión por IMU. Canguro laptop-19 Python3 [Cangurosped]  
  
[ 0] Archivo 8.3  
[ 1] Ejecutar python3  
  
Simulación simplificada del robot cuadrípedo Miataur usando PyBullet  
No requiere el repositorio completo de motion_untation  
  
[ 0] Archivo 8.3  
[ 1] Ejecutar python3  
  
Import pybullet as p  
Import pybullet_data  
Import math  
Import math  
Import numpy as np  
  
# Importación de los componentes del Miataur  
NUM_LEGS = 4  
NUM_MOTORS = 8 # 2 motores por pierna  
MOTOR_TYPE = "Position"  
# Motor types:  
# "motor.Front_leftJoint",  
# "motor.Front_rightJoint",  
# "motor.Back_leftJoint",  
# "motor.Back_rightJoint",  
# "motor.Front_leftJoint",  
# "motor.Front_rightJoint",  
# "motor.Back_leftJoint",  
# "motor.Back_rightJoint",  
# "motor.Front_leftJoint",  
# "motor.Front_rightJoint",  
# "motor.Back_leftJoint",  
# "motor.Back_rightJoint",  
  
class MiataurSimple:  
    def __init__(self, render=True):  
        """ Inicialización del Miataur """  
        # Conectar a Pybullet  
        if render:  
            self.client = p.connect(p.GUI)  
        else:  
            self.client = p.connect(p.DIRECT)  
        p.setPhysicsEngineParameter(  
            physicsEngine=pysim.PyBulletData.getSetupPath())  
        p.setGravity(0, 0, -10)  
  
    # Configurar cámara
```

Figure 1: Archivo minotaur.py

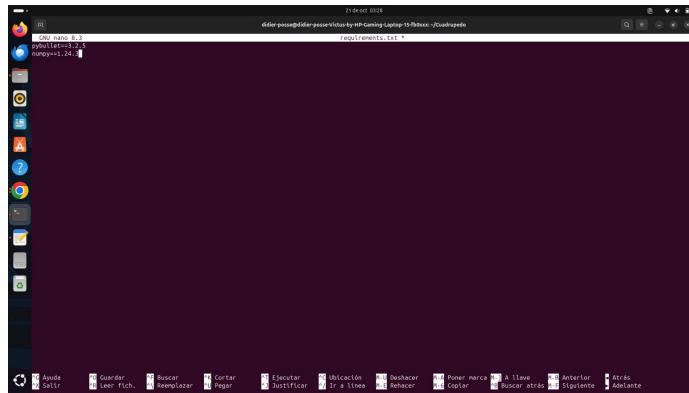


Figure 2: Archivo requirements.txt

```
21 oct 00:48
dider-pesque@dider-pesque-VirtualBox:~/D-Gaming-Laptop/15-Python/7-Cuadropedos
```

CODI Name: 8-3

LABEL description="Simulación de robot cuadropedos Mintaur con PyBullet"

LABEL version="1.0"

# Variables de entorno

ENV DEBIAN\_FRONTEND=noninteractive

ENV DISPLAY=:0

# Instalar dependencias del sistema para Pybullet

RUN apt-get update && apt-get install -y --no-install-recommends \

gcc \

g++ \

libglib2.0-0 \

libgl1-mesa \

libglu1-mesa \

libxrandr1 \

libxtst6 \

mesa-utils \

python3 \

libsm6 \

&& rm -rf /var/lib/apt/lists/\*

WORKDIR /app

# Copiar requirements e instalar dependencias Python

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

pip install --no-cache-dir -r requirements.txt

# Copiar el código

COPY mintaur.py .

# Comando por defecto: ejecutar marcha automática

CMD [ "python", "mintaur.py", "-i" ]

Figure 3: Archivo Dockerfile

- Hecho esto proceda a contruir la Dockerizacion de la siguiente manera:

Figure 4: Procedimiento archivos y Dockerizacion.

Una vez confirmada su construccion, proceda a correr su simulacion Dock-erizada.

Figure 5: Docker run

Una vez ejecutado se desplegará la venta con su cuadrupedo:

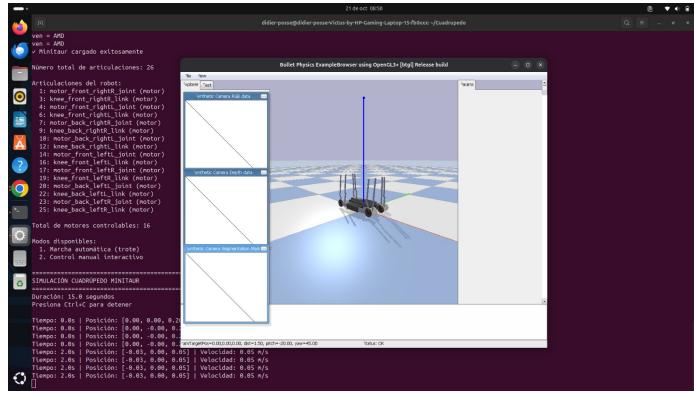


Figure 6: Simulacion completada

## 2.4 Aplicación de un TurtleBot3 con LIDAR y SLAM.

- como es común y casi que una rutina es ingresar a el directorio donde tenemos el entorno virtual activado (venv) una vez alli creamos otro directorio llamado Turtlebot, alli se creara un archivo Turtlebot.py, y otro para el Dockerfile con el fin de Dockerizarlo.

```

root@oscar-grande-Nitro-ANV15-41:~$ cd SimulacionesPyBullet
oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet$ ls
bipedo brazo_control brazo_robotico carro Entornos pista_carro
oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet$ mkdir Turtlebot
oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet$ ls
bipedo brazo_robotico Entornos Turtlebot
brazo_control carro pista_carro
oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet$ cd Turtlebot
oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet/Turtlebot$ nano Turtlebot.py
oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet/Turtlebot$ nano Dockerfile
oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet/Turtlebot$ docker build -t turtlebot3_sim .
[+] Building 53.6s (5/6) docker:default
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 3/2B 0.0s
=> WARN: ConsistentInstructionCasing: Command 'Run' should match the cas 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/3] FROM docker.io/osrf/ros:noetic-desktop-full@sha256:7dbfb9576d8 47.0s
=> => resolve docker.io/osrf/ros:noetic-desktop-full@sha256:7dbfb9576d8e 0.0s
=> => sha256:13bf9576d08e66226c31e06129482aaab0702695f38 3.06kB / 3.06kB 0.0s
=> => sha256:13bf9576d08e66226c31e06129482aaab0702695f38 3.06kB / 3.06kB 0.0s
=> => sha256:69a38b2c0995ef0578571ebe8d07a70d48a40bbfb1 7.02kB / 7.02kB 0.0s
=> => sha256:d20358ca2eb5aafe8d52f0c03ee909a919235 9.98MB / 9.98MB 0.55
=> => sha256:60ca050/171d76e79443cb48e8e4038c3555f8ca12ae 279B / 279B 0.75
=> => sha256:59cc10980325859d6423bbc515001cb577760d 370.50MB / 370.50MB 25.55
=> => sha256:6932d16c264574201be5a55ba79dd248dab8a2aa32b8ed 196B / 196B 0.95
=> => sha256:342ecf093a9124f383e5acc3a78d21be94f20515 49.75MB / 49.75MB 3.0s
=> => extracting sha256:13bf9576d08e66226c31e06129482aaab0702695f38 3.0s
=> => sha256:6c240714a1b7806b25c5f9ddeeab272ceaa47a 348.01kB / 348.01kB 1.35
=> => sha256:ff045193b17856844e9c92165120d313434cb 975.63kB / 975.63kB 1.65
=> => sha256:3d5c979477dedcc14742e4345a6bbcccd5ee6689 15.87MB / 15.87MB 3.0s
=> => extracting sha256:0da4792f77416db6e600bc26bb845d00bd1cef16f3e10a4c 0.2s
=> => extracting sha256:d20358ca2eb5aafe8d52607c52f0c03ee909a919235f1c104 0.1s
=> => extracting sha256:5dce350d6ea44cf7fa3c78bdda0998aca3aeb72fffbfc141 0.0s
=> => extracting sha256:60ca0507171d76e79443cb48e8e4038c3555f8ca12ae 0.0s
=> => sha256:4f15d50362861b86ad8c598ad482a437789014 335.52MB / 335.52MB 18.2s
=> => sha256:10468568b3ae84608caf9b006b4ad9406dsaa 351.44MB / 351.44MB 24.0s
=> => extracting sha256:59cc10980325859d6423bbc515001cb577760d73999bb2e 8.0s

```

Figure 7: Proceso Dockerizado

Una vez construida la Dockerizacion se vera de la siguiente manera:

```

oscar-grande@oscar-grande-Nitro-ANV15-41:~/SimulacionesPyBullet/Turtlebot$ sudo docker build . -t turtlebot3_sim
[+] Building 0.9s (8/8) FINISHED docker:default
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 409B 0.0s
=> [internal] load metadata for docker.io/osrf/ros:noetic-desktop-full 0.0s
=> [auth] osrf/ros pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/3] FROM docker.io/osrf/ros:noetic-desktop-full@sha256:7dbfb9576d8 0.0s
=> => CACHED [2/3] RUN apt-get update && apt-get install -y ros-noetic-turtlebot 0.0s
=> => CACHED [3/3] RUN echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc 0.0s
=> => exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:52974e31fc... 0.0s
=> => naming to docker.io/library/turtlebot3-sim 0.0s

```

Figure 8: Dockerizado

Los archivos con la informacion en su interior son los siguientes:

```
oscar-grande@oscar-grande-Nitro-ANV15-41: ~/SimulacionesPyBullet/Turtlebot
```

```
/opt/ros/n... x root@osc... x /opt/ros/n... x oscar-gran... x oscar-gran... x oscargran... x
```

```
GNU nano 8.3 Dockerfile
```

```
FROM osrf/ros:noetic-desktop-full
```

```
RUN apt-get update && apt-get install -y \
ros-noetic-turtlebot3 \
ros-noetic-turtlebot3-simulations \
ros-noetic-slam-gmapping \
ros-noetic-turtlebot3-navigation
```

```
RUN echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc
```

```
CMD ["bash", "-c", "source /opt/ros/noetic/setup.bash && \
rosrun turtlebot3_gazebo turtlebot3_world.launch"]
```

Figure 9: Dockerfile

Figure 10: `codigo.py`

Figure 11: Código.py

Hecho esto y que todo este bien y funcional haremos un docker run con el fin de ejecutar el turtlebot3.

Figure 12: Docker run

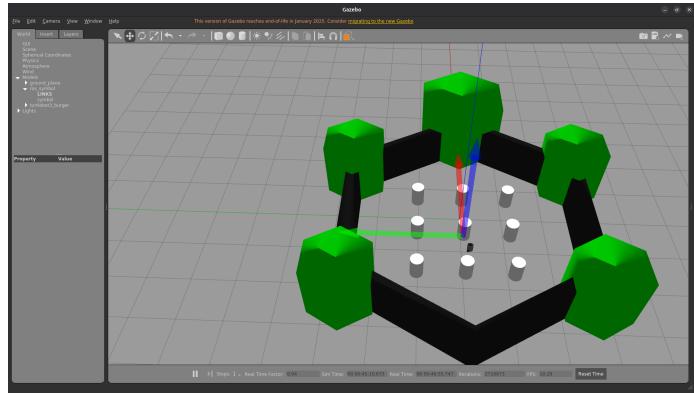


Figure 13: Gazebo Turtlebot

es importante que en el momento de Dockerizar usted sea consciente de los nombres que le impone a cada archivo, a cada aplicativo Dockerizado, comprenda que es lo que esta haciendo y siga la logica que usted este usando. Para el Docker run se ejecuto el siguiente codigo:

```
- sudo docker run -it -rm -env="DISPLAY" -env="TURTLEBOT3_MODEL = burger" --network=host -v /tmp/.X11-unix:/tmp/.X11-unix -name=turtlebot3 -sim=turtlebot3 -sim
```

Por consiguiente a esto, se realiza el siguiente código con el fin de realizar el mapeo de su entorno mientras calcula su propia posición, esto se hará en otra terminal aparte pero sin abandonar en la ubicación en donde está todo el proceso:

```
[R] osangrande@osangrande-NB0-AMT5...          oscar.grande@osangrande-NB0-AMT5-41...@simulationsbybullet/Turtlebot
osangrande@osangrande-NB0-AMT5...          oscar.grande@osangrande-NB0-AMT5-41...@simulationsbybullet/Turtlebot
User Pose: 1.05576 -0.897999 0.57686
Average Scan Matching Score:319.283
average: 92.1633
Registering Scans(Scan)
average: 92.1633
Update: 1.05576 -0.897999 0.57686
update_id:0.0773248 adsl:0.0041
average: 92.1633
average: 1.05568 -0.8935533 -0.0288453
count: 2562
Average Scan Matching Score:338.918
average: 92.1633
Registering Scans(Scan)
average: 92.1633
Update: 1.05576 -0.897999 0.57686
update_id:0.116625 adsl:0.58551
average: 1.1155 -0.15346 1.55746
count: 2562
Average Scan Matching Scores:306.656
average: 92.1633
Registering Scans(Scan)
average: 92.1633
(turtlebot3_sim_gmapping-2) killing on exit
(robot_state_publisher-1) killing on exit
(tf2_ros-1) killing on exit
update_id:0.155813 adsl:0.63662
average: 0.99326 -0.120584 2.0891
count: 2564
Average Scan Matching Scores:392.417
average: 72.9327
Registering Scans(Scan)
average: 72.9327
Update: 0.9426558 adsl:5.22089
average: 0.956943 -0.301982 -1.56109
count: 2565
Average Scan Matching Scores:348.285
average: 72.9327
Registering Scans(Scan)
average: 72.9327
(gridfileprocessor-1) killing on exit
(gridfileprocessor-2) killing on exit
virtual_OMP_NUM_THREADS=1(gridfileprocessor) Start
virtual_OMP_NUM_THREADS=1(gridfileprocessor) Deleting tree
gridfileprocessor shutting down processing monitor complete
done
osangrande@osangrande-NB0-AMT5-41...@simulationsbybullet/Turtlebot$ sudo docker exec -it turtlebot3-sim bash -c 'source /opt/ros/noetic/setup.bash && rosrun turtlebot3_sim_turtlebot3_sim.launch sim_methods:=gmapping'
```

Figure 14: código mapeo SLAM

De donde emergera la siguiente ventana:

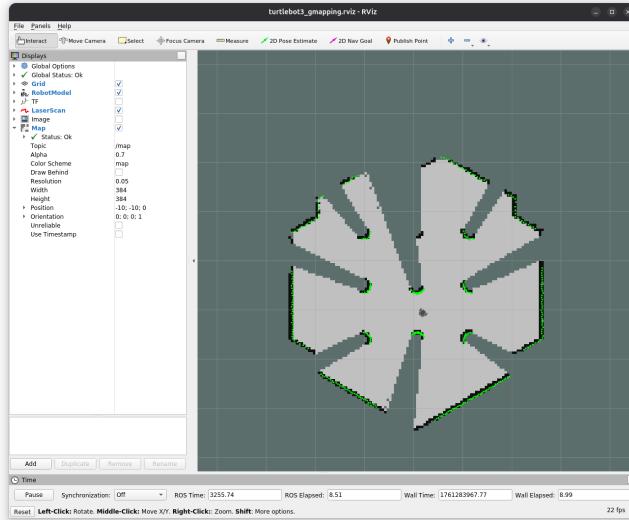


Figure 15: Mapeo SLAM

Luego de ello, usted podra visualizar el entorno, visualizar el mapeo con detección de obstaculos, pero vera que no puede tener control del Turtlebot, por lo tanto en otra ventana sin abandonar la ubicacion que se encuentra, se ejecuta el siguiente codigo con el fin de poder manejar el robot:

Figure 16: Control de movimientos

Aqui confirma que sus movimientos a la vez que oprima una tecla, realiza inmediatamente la toma de datos que va obteniendo segun sus movimientos. Por ultimo una visualizacion de mapeo con .rviz donde se ve de la siguiente manera:

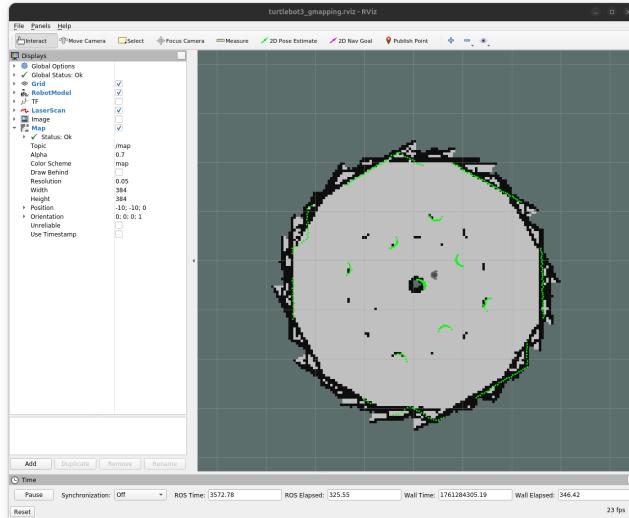


Figure 17: Mapeo con rviz

ambos mapeos son iguales solo que tienen una diferencia, cuando se realiza el primer gmapping lo que sucede es establecer la Herramienta SLAM con un entorno mas "general incluyendo Darware real con un proposito de una base con laser". Con Virz lo que emplea es un entorno en Gazebo con el proposito de una version mas adaptada con el entorno simulado y con parametros similares pero que dan un poco de mejora a este, con ambos funciona y cualquiera esta bien, dependeria mas de la empleabilidad que se le quiera dar.

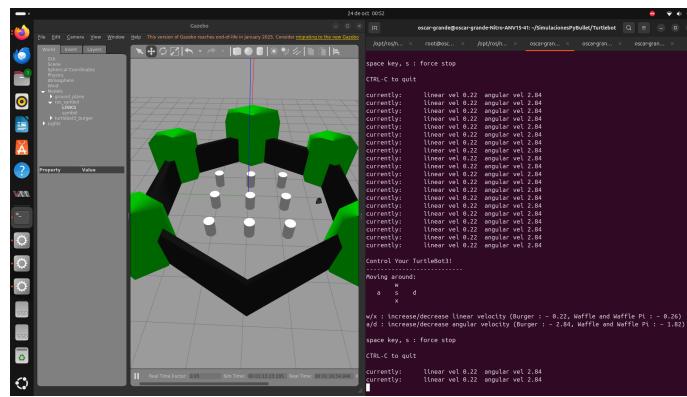


Figure 18: Resultado final

### **3 Referencias**

- “NightOwl: Robotic Platform for Wheeled Service Robot”.
- ”MoveIt Tutorials” — documentación oficial de MoveIt con ejemplos de integración con robots.
- Research on SLAM and Path Planning Method of Inspection Robot.
- Isaac ROS Visual SLAM.