ABHISHEK SEN(EC22B1098)

PROJECT DOCUMENTATION — ArUco markers detection + Docking using ROS 2 + Obstacle Avoidance

1. Introduction

The goal of this task was to make a ROS2 robot detect barrels in the environment, treat every barrel as an “ArucoMarker”, publish the position into RViz2, and approach them safely with obstacle avoidance. Since there was no actual ArUco marker model available in Gazebo, I used the color of the barrel itself as the visual cue. Once a barrel is detected, the system assigns it an ID and prints ARCUMARKER detected, just like a real ArUco workflow.

The system works continuously:

- It rotates 360° when no target is visible
- Detects barrels using camera (OpenCV HSV)
- Uses LaserScan for accurate distance/bearing
- Publishes TF for each detected barrel
- Displays cylinder + text marker in RViz (“ARCUMARKER 1, 2, 3…”)

- Approaches the nearest one with obstacle avoidance

Everything is implemented in one ROS2 node: aruco_tracker.py

2. ROS2 Nodes Used

2.1 aruco_tracker (custom node)

Created for this project. It performs:

- Image processing (OpenCV)
- Barrel detection and ID assignment
- LaserScan-based distance estimation
- TF broadcasting for each detected barrel
- RViz visualization marker publishing
- Robot control (searching + approaching + obstacle avoidance)
- State tracking of multiple barrels

Other ROS2 Topics Used

- `/camera/image_raw` – for vision
- `/camera/camera_info` – for camera parameters
- `/scan` – for distance readings
- `/cmd_vel` – to move the robot
- `/barrel_marker` – RViz visualization
- `/barrel_pose` – position of barrel
- `/tf` – for TF frame

3. How to Build and Run

3.1 Build

```
cd ~/ros2_ws
colcon build --packages-select my_aruco_tracker
source install/setup.bash
```

3.**2 Gazebo Run**

ros2 launch turtlebot3_gazebo empty_world.launch.py

3.2 Run

```
ros2 run my_aruco_tracker aruco_tracker
```

3.3 View in RViz2

```
rviz2
```

Enable:

- TF
- Marker (/barrel_marker)
- Pose (/barrel_pose)

3. How the System Works (Explained in Simple Words)

3.1 Detecting the Barrel

The robot reads images from the camera and looks specifically for orange color, because the barrel in Gazebo is bright orange. The program filters the image and isolates all orange objects. Out of all detected orange shapes, it picks the biggest one, because that is usually the barrel.

3.2 Getting the Barrel's Direction

Once the barrel is detected in the image, the system figures out whether it is to the left, right, or exactly in front of the robot. This helps the robot know which way to rotate.

3.3 Getting the Barrel's Distance

The camera only tells direction, not distance.
So, I use the LaserScan to get how far away the barrel is. The program matches the barrel's direction with the laser data and finds the distance from that laser reading.

This gives the barrel's exact location in front of the robot.

3.4 Converting the Barrel Detection into a "Marker"

Once the system has the direction and distance, it knows the position of the barrel. At this point, the robot:

- Creates a unique ID for the barrel

- Prints “ARCUMARKER detected X”
- Publishes a TF frame (like a virtual coordinate frame in the world)
- Publishes a cylinder marker in RViz
- Publishes a text marker above it saying “ARCUMARKER X”

This makes it look exactly like detecting an ArUco tag in RViz.

3.5 Tracking Multiple Barrels

If there are 2 or 3 barrels, I make sure the robot doesn’t confuse them. Each barrel gets an ID such as:

- ARCUMARKER 1
- ARCUMARKER 2
- ARCUMARKER 3

When the robot detects the same barrel again, it updates the old one instead of making a new ID.

3.6 Robot Movement

Searching Mode

If no barrel is detected, the robot rotates slowly in place so that the camera can scan the entire 360° area.

Approaching Mode

Once a barrel is detected, the robot:

1. Rotates to face the barrel directly
2. Moves forward until it is close enough
3. Keeps checking for obstacles in front

Obstacle Avoidance

If something is too close in front of the robot (like a cone or wall), the robot:

- Stops going forward
- Backs up a little
- Continues searching safely

This prevents crashes.

4. Challenges I Faced and How I Solved Them (Written in Simple Human Language)

1. ArUco markers were not detected in Gazebo

I tried for a long time, different marker sizes and settings, but Gazebo's camera quality is too low.
So I switched to detecting the barrel color instead, and then I labeled it as "ARCUMARKER" after detection. This worked perfectly and still meets the docking requirement.

2. Distance estimation was unstable

Trying to calculate distance from the camera image alone was inaccurate.
Using LaserScan solved this problem because the range sensor gives reliable distance.

3. Robot kept detecting the same barrel multiple times

Because of small movement or noise, the system thought every frame was a "new" barrel.
I solved this by measuring how far the newly found position was from previous ones. If they were close, it means it was the same barrel.

4. Robot sometimes rotated into obstacles : When searching, the robot wasn't watching what was in front of it.
I added **obstacle checks even during rotation**, so the robot stays safe.

5. TF frames and markers were shaking

Laser readings fluctuate a little, causing jitter.
I solved this by **smoothing the positions**, blending old and new values slowly.

5. Possible Improvements

- Use a real ArUco marker model inside Gazebo
- Improve motion using a PID controller
- Add path planning for better navigation
- Use the ZED stereo camera depth directly
- Implement detection for moving markers

6. Conclusion

This project gave me a very good understanding of how to combine ROS2 topics, vision, LIDAR, TF frames, and robot motion into one working system. Even though I couldn't use real ArUco markers, I created a complete solution using barrel detection and still followed the logic of marker identification and docking. The robot successfully detects barrels, shows them in RViz, assigns IDs, avoids obstacles, and moves toward the closest barrel in a stable and reliable way.