Using the Buzzer on the RRC Lite Board - Advanced Challenges

This notebook will help you explore **advanced buzzer functionalities** using the **RRC Lite Board** in **ROS2**.

Learning Objectives

- Use the buzzer for **interactive alerts**.
- Implement **patterned sounds** for robotic responses.
- Create **new auditory signals** for different robot actions.
- Enhance your **coding skills** through **custom challenges**.

In [ ]:
```
import sys
import os


import rclpy
from controllers.omni_robot_controller import OmniWheelControlNode  # Ensure
# Initialize ROS2 node
rclpy.init()
node = OmniWheelControlNode()
```

1. Quick Review

Before starting the challenges, review the following:

- **How does a buzzer generate sound?**
- **What is the difference between an active and a passive buzzer?**
- **How can we change the frequency of the buzzer in ROS2?**

Challenge 1: Buzzer Alert Based on Distance

Objective

Write a loop that continuously reads the robot's front distance sensor and adjusts the buzzer frequency based on the detected distance.

Instructions

1⃣ Read Sensor Data

- Retrieve the **distance** measurement from the LiDAR sensor.
- Use the appropriate **distance variable** from the robot's `OmniWheelControlNode` class.

2⃣ Calculate Buzzer Frequency

- Create a function that **maps the distance to a buzzer frequency**.
- Consider that **closer objects** should have a **higher pitch**, and farther objects a **lower pitch**.
- Think about using a **linear or exponential mapping function** to adjust the frequency.

3⃣ Play the Buzzer Based on Distance

- Use the **robot's buzzer function** to generate sound.
- Ensure that the buzzer plays **with intervals** and **stops when needed**.

4⃣ Use a Loop to Continuously Check Distance

- Continuously update the distance and adjust the buzzer.
- Stop the buzzer when no object is detected or when the robot moves beyond a threshold.

Available Functions & How to Use Them

Here are the key functions from the `OmniWheelControlNode` class that you can use:

** Read Distance**

- **Use one of the following variables to get distance:**
  - `node.front_distance`
  - `node.back_distance`
  - `node.left_distance`

    - `node.right_distance`
- These variables update automatically with sensor readings.

** Play a Buzzer Sound**

```
node.play_buzzer(frequency, on_time, off_time, repeat)
```

In [ ]:
```
# Your code here:
while True:
    distance = node.front_distance  # Replace with appropriate direction var
    frequency = calculate_frequency(distance)
    node.play_buzzer(frequency, 0.5, 0.5, 3)
```

Challenge 2: Countdown Timer

- Program the buzzer to **beep once per second** for **10 seconds**.
- After 10 seconds, play a **final long beep** to indicate time is up.
- Use a loop with **time.sleep()** to create the effect.

Objective

Create a countdown timer using the buzzer. The buzzer should beep once per second for 10 seconds. After the countdown is complete, it should play a final long beep to indicate time is up.

Instructions

1 Use a Loop to Control Timing

- Use a `for` loop to repeat the buzzer sound 10 times (once per second).
- Use `time.sleep(1)` to create the delay between each beep.

2 Play a Short Beep

- Call the buzzer function inside the loop.
- Use an appropriate frequency for a short beep.
- Adjust the `on_time` and `off_time` to ensure the beep is clear.

3 Play a Final Long Beep

- After the loop finishes, play one final beep that is longer than the previous beeps.
- Choose a different frequency to make it distinguishable.

In [ ]:
```python
# Your code here:
for i in range(10):
    node.play_buzzer(1000, 0.5, 0.5, 2)
    time.sleep(1)
node.play_buzzer(500, 0.5, 0.5, 1)  # Final long beep
```

Challenge 3: Morse Code Beeper

Objective

Write a function that converts **text into Morse code** beeps and plays it using the buzzer. Implement a buzzer sequence for **SOS** (**... --- ...**).

Instructions

1 Create a Morse Code Dictionary

- Define a **dictionary** that maps letters to Morse code symbols (`.` for dots and `-` for dashes).
- Example:
  `morse_code = {'S': '...', 'O': '---'}`

2 Loop Through the Letters

- Iterate through each letter in the word **"SOS"**.
- Convert the letter into **Morse code symbols** using the dictionary.

3 Convert Symbols to Beeps

- **Dots (`.`) should be short beeps**.
- **Dashes (`-`) should be long beeps**.
- Use `node.play_buzzer()` to play the corresponding beep.
- Add a small delay (`time.sleep(0.2)`) between each beep for clarity.

4 Add Pauses Between Letters

- Introduce a slightly **longer pause** between letters.
- This helps distinguish separate characters in Morse code.

Available Functions & How to Use Them

** Play a Buzzer Sound**

```
node.play_buzzer(frequency, on_time, off_time, repeat)
```

- **Parameters:**
  - `frequency` : The sound frequency in Hz.
  - `on_time` : Duration the buzzer stays ON.
  - `off_time` : Duration the buzzer stays OFF.
  - `repeat` : Number of times the buzzer plays.

** Time Delay Function**

```
time.sleep(seconds)
```

- **Pauses execution** for the specified number of seconds.
- Use this to create spacing between Morse code signals.

Hints

Use a **dictionary** to store Morse code mappings.
Loop through each **letter** and then each **symbol** in Morse code.
Assign **shorter beeps** for dots (`.`) and **longer beeps** for dashes (`-`).
Add **pauses** between letters to make the message readable.

** Example (Not a Solution)**

> "Convert the word 'SOS' into Morse code using beeps, with short beeps for dots and long beeps for dashes. Ensure there are brief pauses between each beep and a longer pause between letters."

By following this structured approach, you will **understand how to convert text into Morse code using the buzzer** without seeing a direct solution.

In [ ]:
```
# Your code here:
morse_code = {'S': '...', 'O': '---'}
for letter in 'SOS':
    for symbol in morse_code[letter]:
        if symbol == '.':
            node.play_buzzer(1000, 0.3, 0.5, 1)  # Short beep
        else:
            node.play_buzzer(1000, 1.0, 0.5, 1)  # Long beep
        time.sleep(0.2)
```

Challenge 4: Sudden Stop Buzzer Alert

Objective

Detect when the robot **suddenly stops** due to an obstacle and trigger a **buzzer alert**.

Instructions

1 Detect a Sudden Stop

- Use the **LiDAR sensor** to monitor the front distance.
- Check if the **front distance** is **less than the critical distance** (`node.critical_distance`).
- If the robot is **too close to an obstacle**, trigger the alert.

2 Trigger the Buzzer Alert

- If the condition is met, the buzzer should **beep 5 times** to signal the sudden stop.
- Use the **buzzer function** to play a short beep.
- Add a **brief pause** between beeps for clarity.

Available Functions & How to Use Them

** Read Distance from the LiDAR Sensor**

Use the `front_distance` variable to check the distance ahead:

```
node.front_distance
```

The critical threshold is:

```
node.critical_distance
```

** Play a Buzzer Sound**

```
node.play_buzzer(frequency, on_time, off_time, repeat)
```

Parameters:

- `frequency` : The beep frequency in Hz (e.g., `1000`).
- `on_time` : How long the buzzer stays ON (in seconds).
- `off_time` : How long the buzzer stays OFF before repeating.
- `repeat` : Number of times the buzzer plays.

** Pause Between Beeps**

`time.sleep(0.2)`
Adds a short delay between buzzer sounds.

Hints

Use an `if` condition to check `front_distance < critical_distance`.
Beep **5 times** if an obstacle is too close.
Use `time.sleep(0.2)` to add a pause between alerts.
Test different values to find an appropriate alert duration.

** Example (Not a Solution)**

"If the robot detects an obstacle closer than 10 cm, it should beep rapidly 5 times to warn the user."

In [ ]:
```
if node.front_distance < node.critical_distance:  # Check if an obstacle is
    for _ in range(5):
        node.play_buzzer(1000, 0.1, 0.1, 1)  # Adjust frequency and duration
        time.sleep(0.2)
```

Challenge 5: Melodic Robot Tune

Objective

Program the buzzer to **play a simple melody** using a sequence of frequencies.

Instructions

1 Create a Melody

- Define a **list of frequencies** that represent musical notes.
- Use different values to create a **simple melody**.

2 Play Each Note

- Use a **loop** to play each frequency in the melody.
- Each frequency should play for **0.3 seconds**.

3 Add Pauses Between Notes

- Insert **0.1 seconds of silence** between each note to create rhythm.

Available Functions & How to Use Them

** Define a Melody**

Store the sequence of frequencies in a list:

```
melody = [500, 700, 900, 700, 500]
```

** Play a Note**

```
node.play_buzzer(frequency, on_time, off_time, repeat)
```
Parameters:

- `frequency` : The note frequency in Hz.
- `on_time` : Duration the note plays (`0.3` seconds).
- `off_time` : Delay before the next note (`0.1` seconds).
- `repeat` : Number of times the note plays.

** Add a Pause Between Notes**

```
time.sleep(0.1)
```
Creates a short silence between each note.

Hints

Store the melody frequencies in a list.
Use a `for` loop to play each note.
Set `on_time = 0.3s` and `off_time = 0.1s` to maintain rhythm.
Experiment with different frequencies to create unique tunes.

** Example (Not a Solution)**

"The robot should play a sequence of five tones, each lasting 0.3 seconds with a 0.1-second break in between."

In [ ]:
```
# Your code here:
melody = [500, 700, 900, 700, 500]
for freq in melody:
    node.play_buzzer(freq, 1.0, 1.0, 2)
    time.sleep(0.1)
```

3. Debugging & Troubleshooting

Fill in the missing solutions:

Issue	Solution
Buzzer not playing?	
No sound?	
Syntax errors?	

4. Reflection

- What was the most **challenging** part of these activities?
- How can you **improve** your buzzer functions for **real-world applications**?
- What **new ideas** do you have for buzzer-based interactions in robotics?

In [ ]:
```
node.destroy_node()
rclpy.shutdown()
```