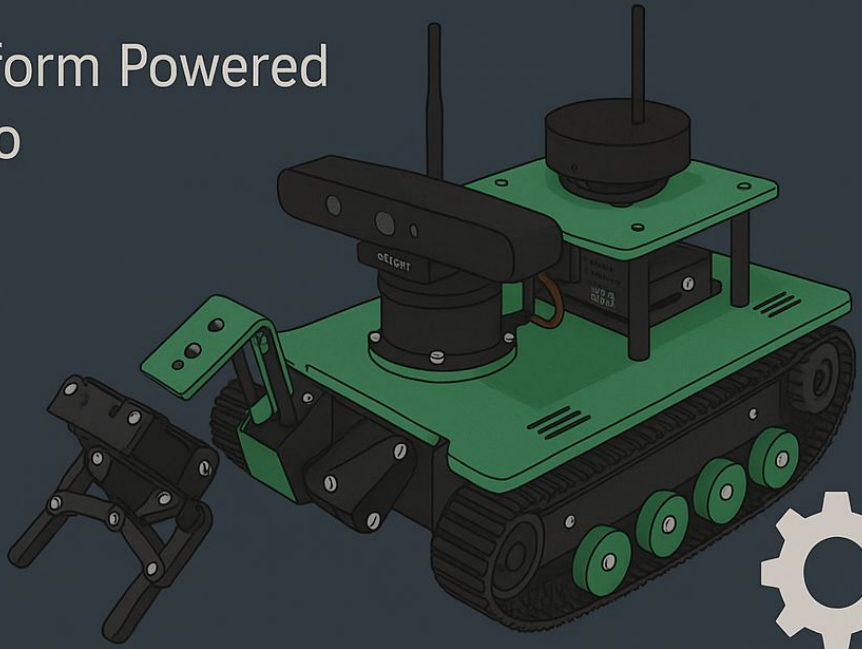
Transbot-JetsonNANO

A Smart Robotic Platform Powered by AI and Jetson Nano

Chris Kim
Rutgers University



About Me

- Third-year Student at Rutgers University
- Double Major in Computer Science and Data Science
- Interests: AI Prompt Engineering, Music

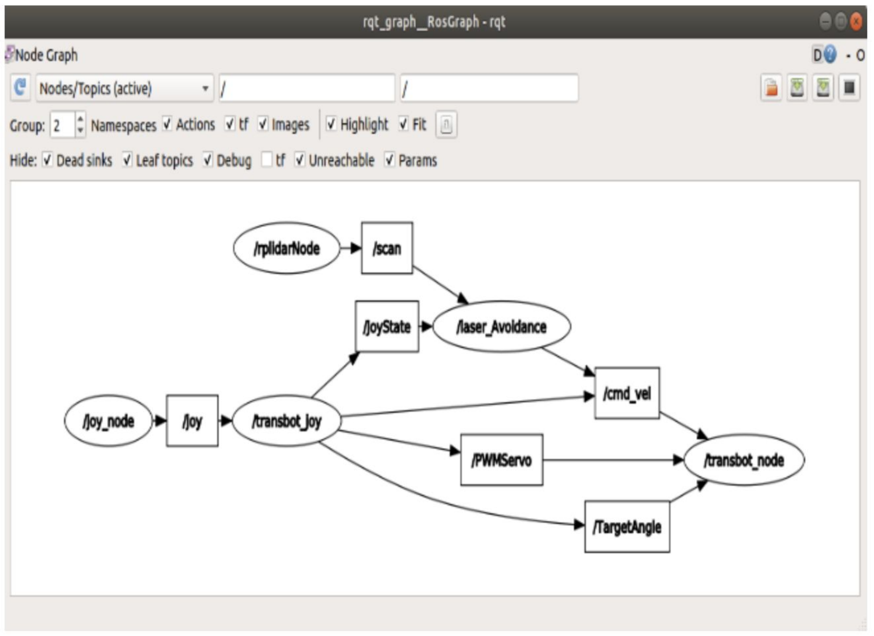
Agenda

- What is LIDAR and ROS?
- Key Components of the Robot
- Three LIDAR modes

  controlling with a Smartphone

ROS:

- An open-source, flexible framework for writing robot software
- Provides tools, libraries, and conventions to build complex, reliable robot behavior across many platforms
- Widely used in research and industry for autonomous vehicles, robotic arms, drones, and service robots



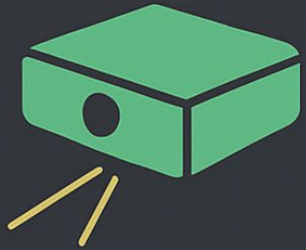
LIDAR

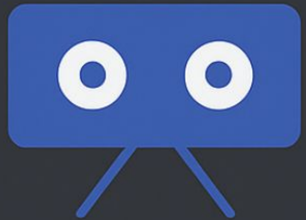
- Single-line LIDAR emits one laser beam from the source
    - Two main types: Triangular Ranging and Time of Flight (ToF)

- High-speed scanning with excellent resolution
    - Offers better angular frequency and sensitivity compared to multi-line LiDAR

- Ideal for precise distance measurement and obstacle detection

How LiDAR and Depth Camera Work Together





LiDAR

provides precise distance measurements and 2D scanning

Depth camera

adds 3D perception and object recognition image and depth data

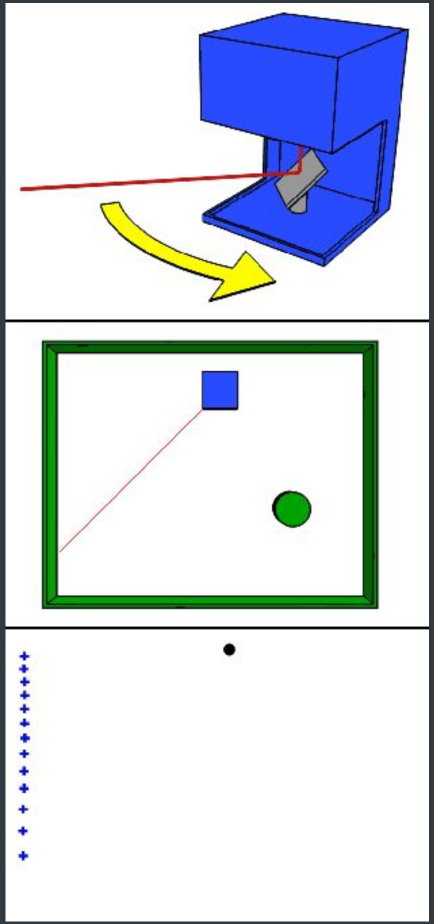
- Together, they create a detailed map of the robot's surroundings
- Enables accurate navigation, obstacle avoidance, and object interaction

How LIDAR Perceives the Environment

Top: LiDAR emits a laser signal from the sensor

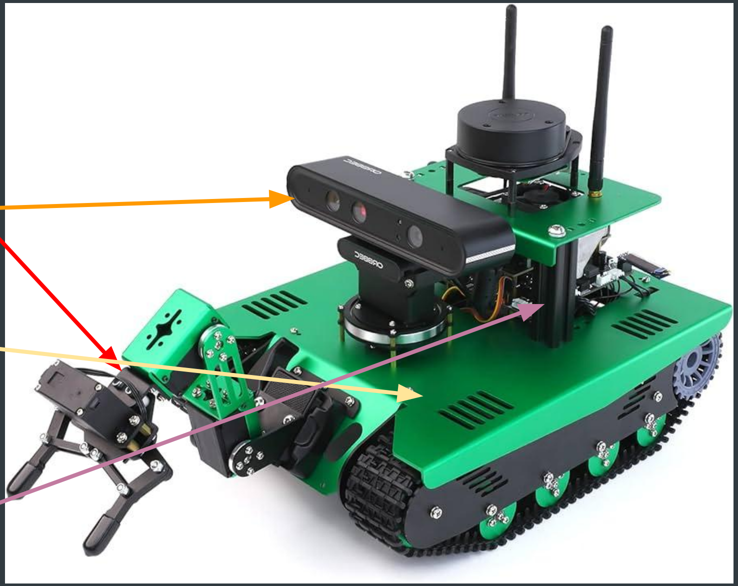
Middle: Obstacles and distances are detected and mapped

Bottom: The system processes this data to build a 2D representation of the surroundings

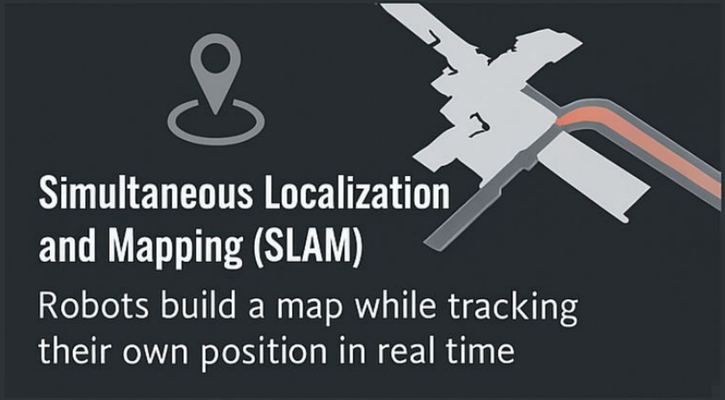


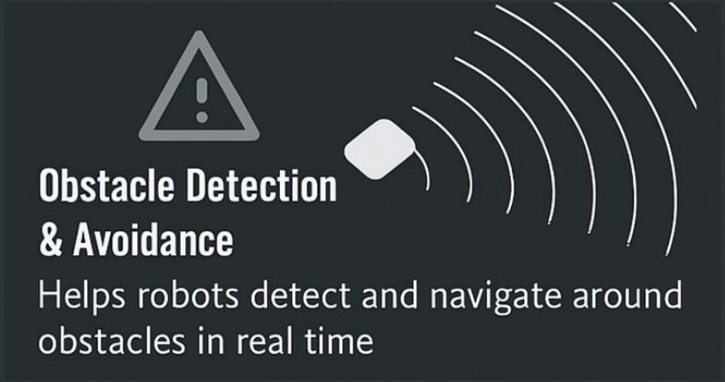
Important Parts of the Robot

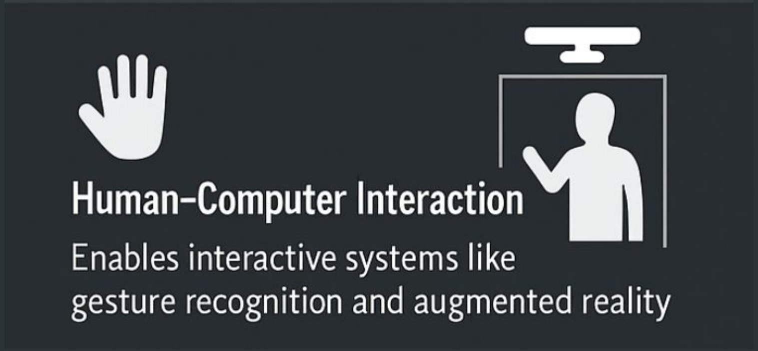
- Robotic Arm – Picks up small objects and moves them to different positions
- Depth Camera – Enables navigation, obstacle detection, and object recognition
- Robot Frame – The physical structure that holds all components together
- Expansion Board – Integrates additional sensors and modules to enhance functionality



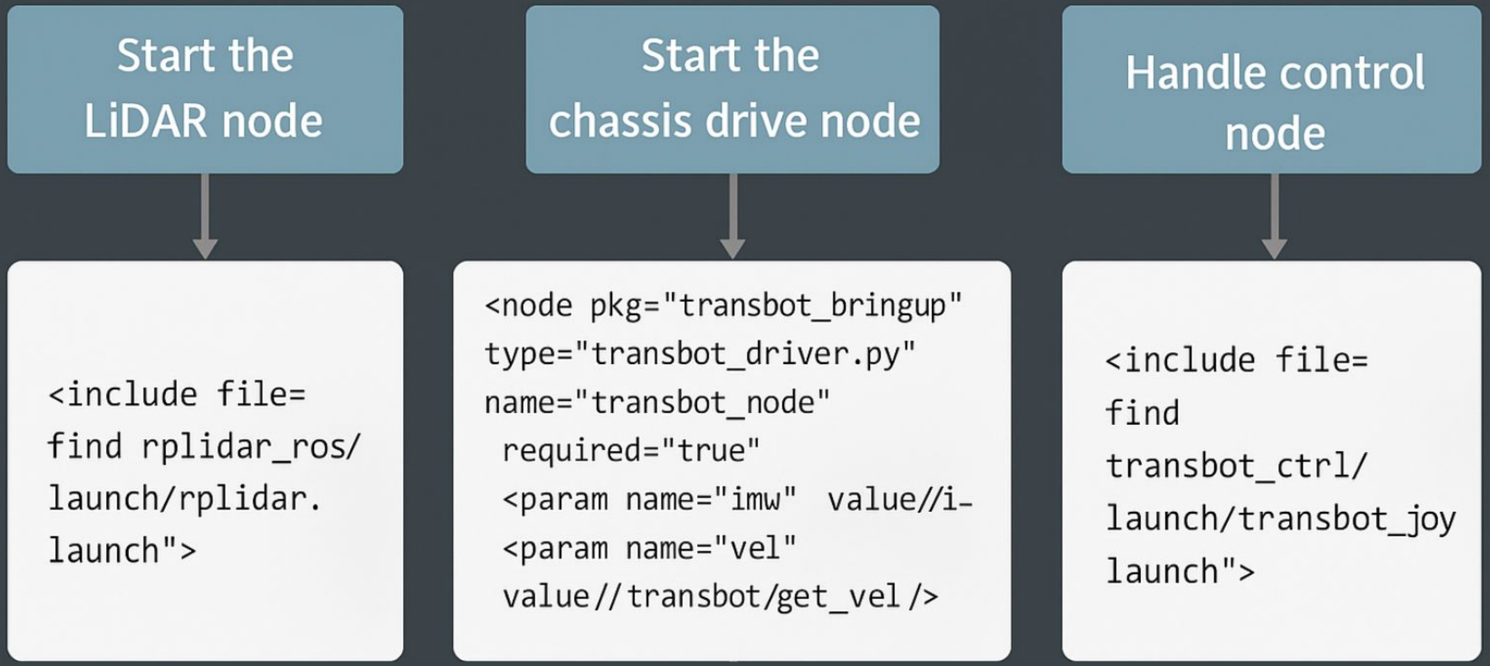
Real-World Applications of LIDAR and Depth Sensing







base.launch – Startup Configuration for Transbot

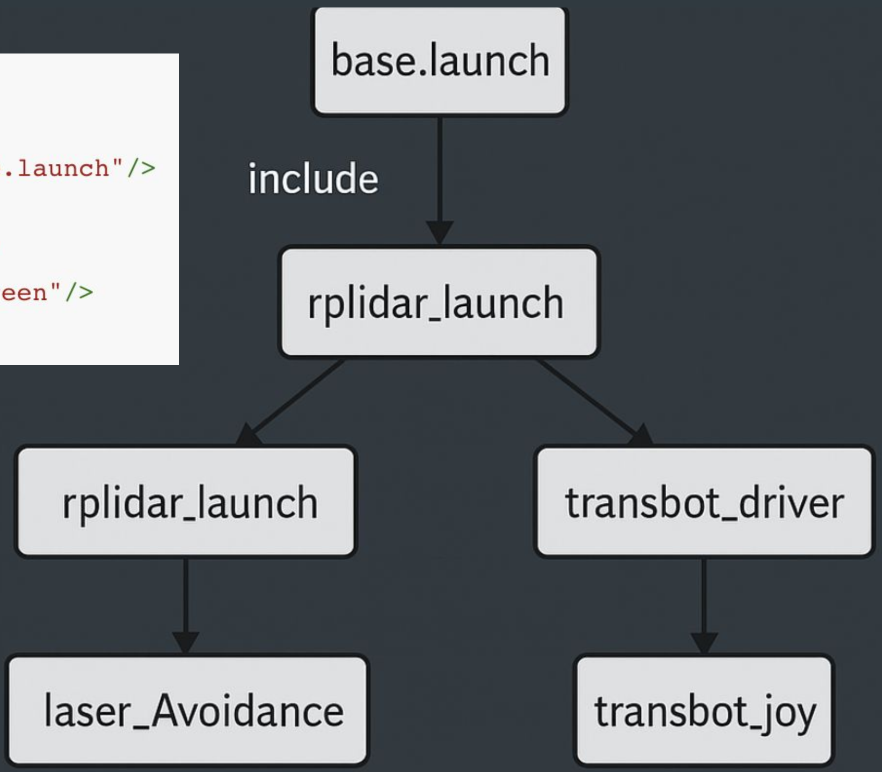


Lidar Obstacle Avoidance

1. Set Detection Angle & Range: Define the detection angle (e.g., 180° or 360°) and the range within which obstacles are detected.
2. Initial Movement: The robot moves forward in a straight line unless an obstacle is detected.
3. Obstacle Detection: LiDAR scans for obstacles and determines their position relative to the robot:
    a. Front Left - Obstacle detected on the front-left side.
    b. Front Right - Obstacle detected on the front-right side.
    c. Straight Ahead - Obstacle directly in the front.
4. Robot Response: Based on the detected position, the robot adjusts its path:
    a. Front Left: Turn slightly right.
    b. Front Right: Turn slightly left.
    c. Straight Ahead: If close: Turn sharply (big circle) either left or right depending on the clearance.
        i. If distant: Make a smaller turn.

laser_Avoidance.launch – LiDAR-Based Obstacle Avoidance

```
<launch>
    <!-- Start base.launch file-->
    <include file="$(find transbot_laser)/launch/base.launch"/>
    <!-- Start the lidar obstacle avoidance node -->
    <node name='laser_Avoidance' pkg="transbot_laser"
type="laser_Avoidance.py" required="true" output="screen"/>
</launch>
```



Lidar Guard

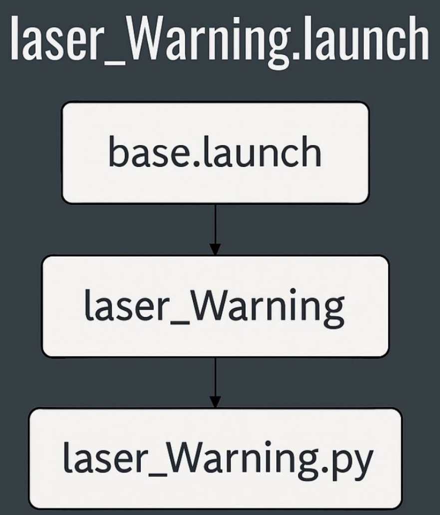
- Once the detection angle and response distance are set, the robot locks onto the closest target as soon as it powers on.

- If the target enters the predefined response range, a buzzer is triggered and continues until the target exits that range.

- Additionally, you can fine-tune the robot's rotational response using PID control to ensure smooth and efficient tracking.

laser_Warning.launch - Activating LiDAR Guard Node

```
<launch>
    <!-- Start base.launch file -->
    <include file="$(find transbot_laser)/launch/base.launch"/>
    <!-- Start the lidar guard node  -->
    <node name='laser_Warning' pkg="transbot_laser"
type="laser_Warning.py" required="true" output="screen"/>
</launch>
```

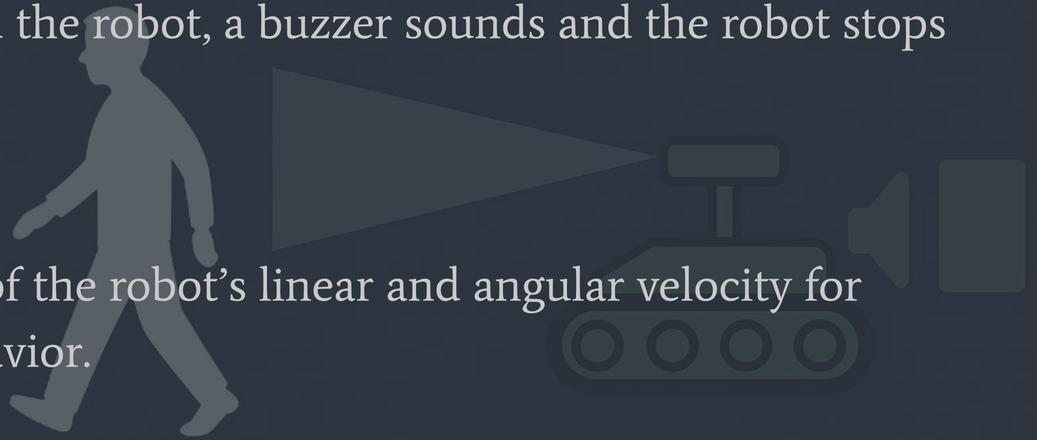


Lidar Follow Mode

- After setting the detection angle and distance, the robot automatically follows the closest target while maintaining a safe buffer.

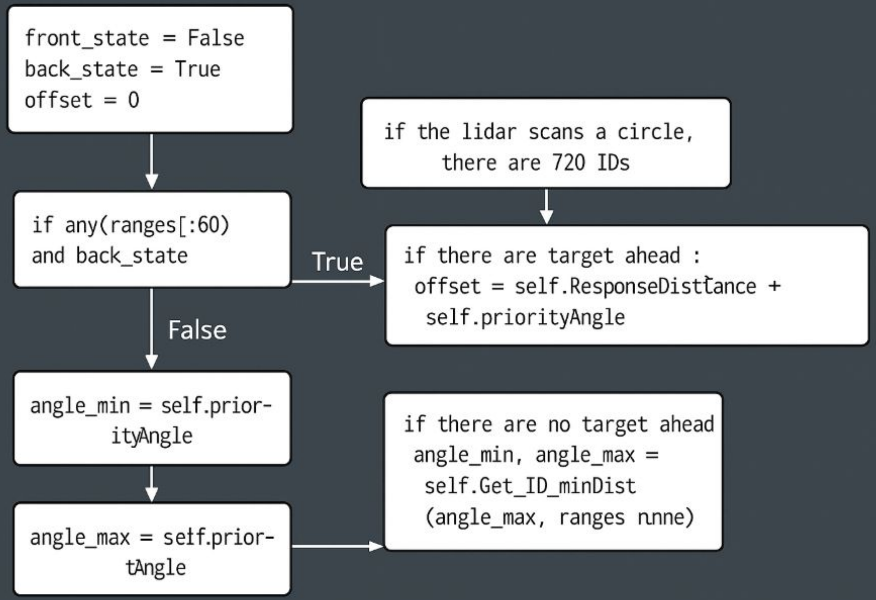
- If an obstacle is detected behind the robot, a buzzer sounds and the robot stops until the path is clear.

- PID control allows fine-tuning of the robot’s linear and angular velocity for smooth, efficient following behavior.



laser_Tracker.py - Target Detection Logic

```
front_state = False
back_state = True
offset = 0.5
... ...
# If the lidar scans a circle, there are 720 IDs
if len(np.array(scan_data.ranges)) == 720:
    for i in range(270, 450):
        # Check whether there are target behind
        if ranges[i] < 0.5: back_state = False
    for i in range(0, self.priorityAngle * 2):
        # Check whether there are target front left
        if ranges[i] < (self.ResponseDist + offset): front_state = True
    for i in range(720 - self.priorityAngle * 2, 720):
        # Check whether there are target front right
        if ranges[i] < (self.ResponseDist + offset): front_state = True
    if front_state == True:
        # When there are target ahead
        angle_min = self.priorityAngle * 2
        angle_max = 720 - self.priorityAngle * 2
        # Get target ID and minimum distance
        minDistID, self.minDist = self.Get_ID_minDist(angle_min,
angle_max, ranges)
    else:
        # When there are no target ahead
        angle_min = self.laserAngle * 2
        angle_max = 720 - self.laserAngle * 2
        # Get obstacles ID and minimum distance
        minDistID, self.minDist = self.Get_ID_minDist(angle_min,
angle_max, ranges)
```



DEMO

Thank You!