Activity

Simple Obstacle Avoidance

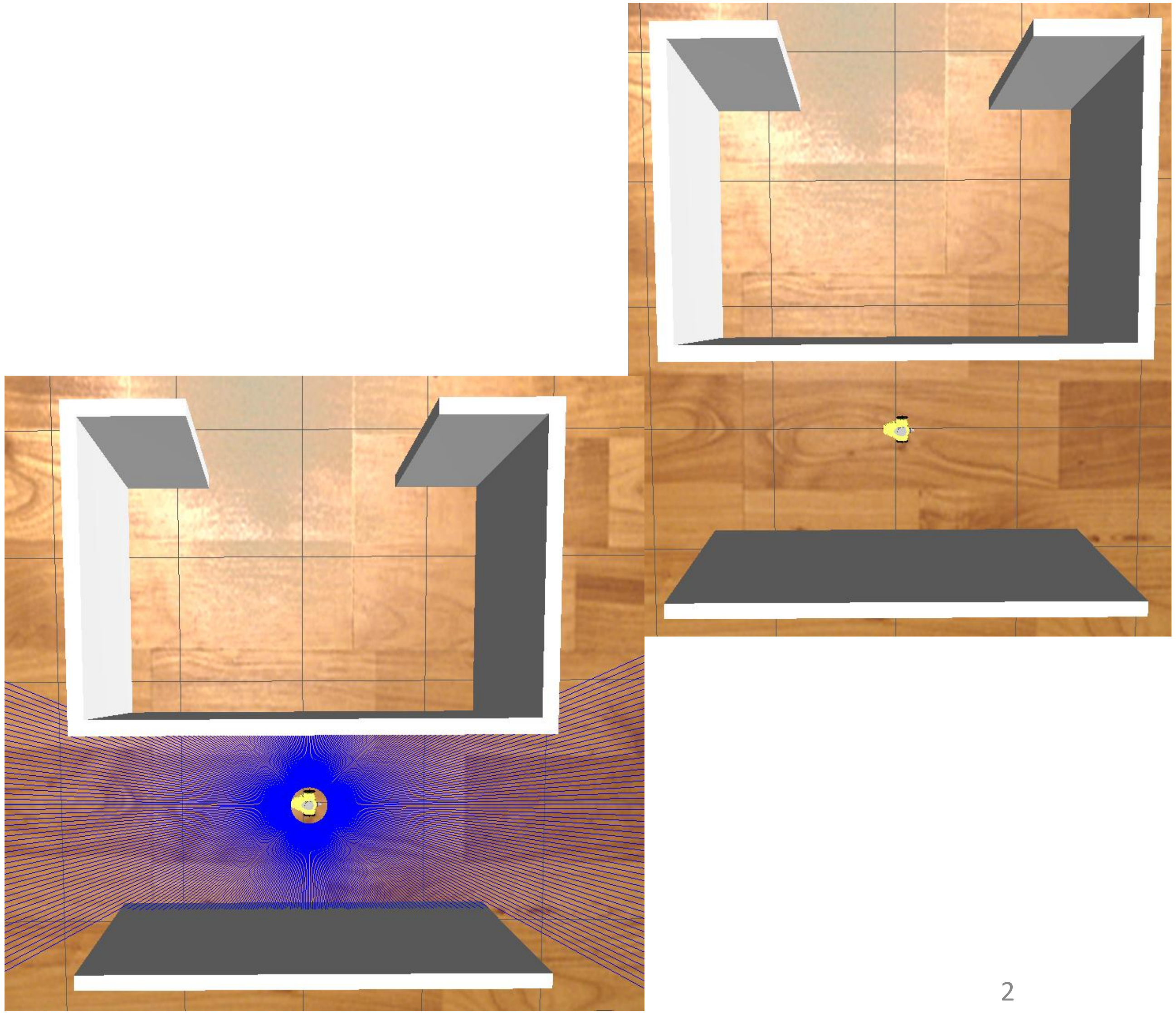
{Learn, Create, Innovate};

Manchester Robotics

Activity – Simple Obstacle Avoidance

Objective

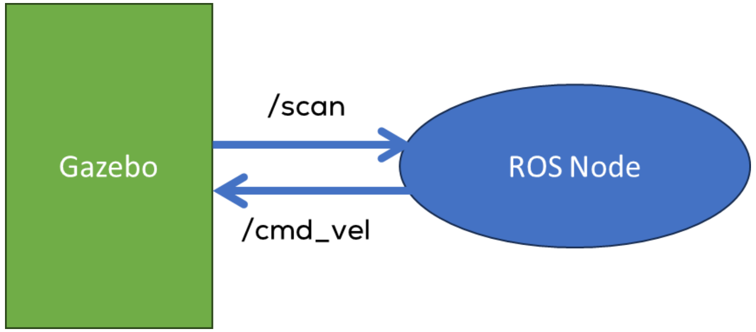
- The following image show Puzzlebot simulation in Gazebo along with some simple obstacles.
- The objective is to avoid the obstacles when the Puzzlebot is moving, using the LiDAR information.
- To this end a simple obstacle avoidance algorithm will be developed using the information from the LiDAR.

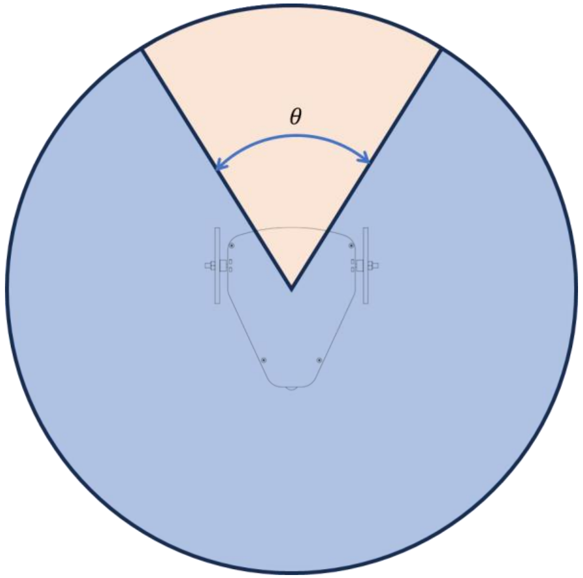


Activity – Simple Obstacle Avoidance

Instructions

- This activity consists of creating a node that subscribes to the LiDAR information given by Gazebo.
- Filter the required information to avoid the obstacle.
- Send the velocities to the robot simulation.

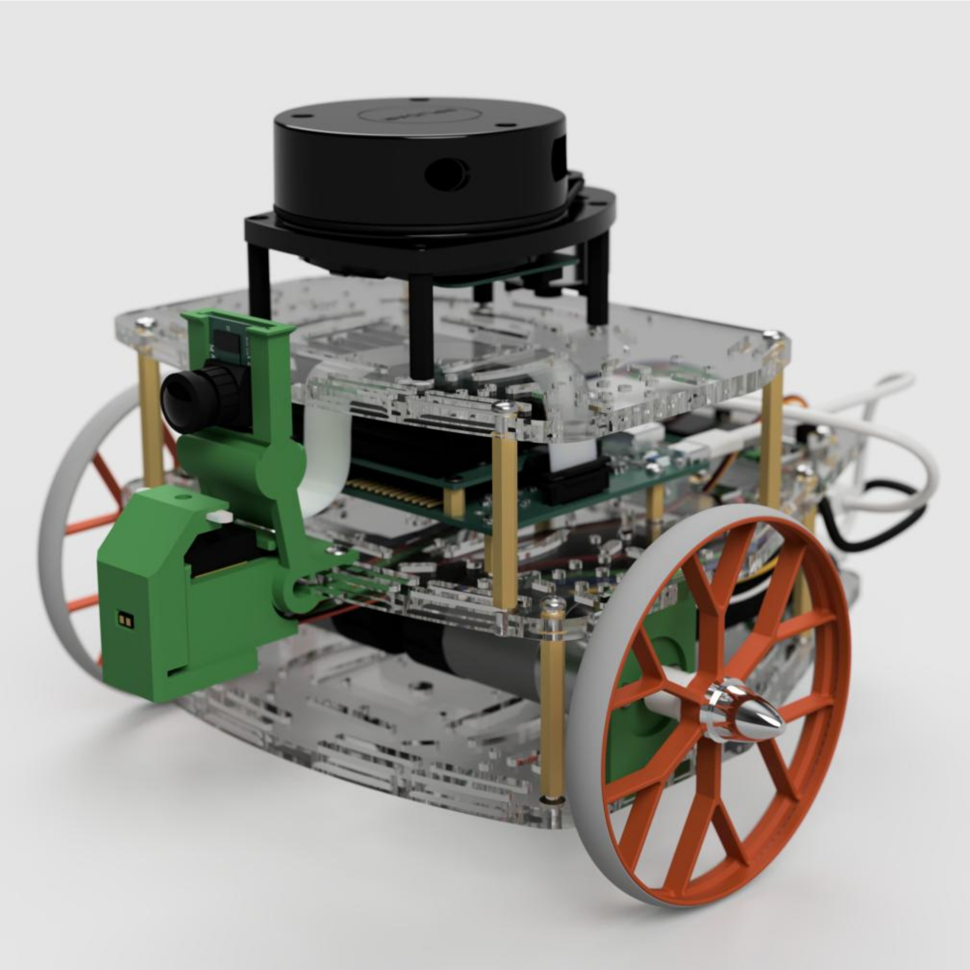




Activity – Simple Obstacle Avoidance

LiDAR

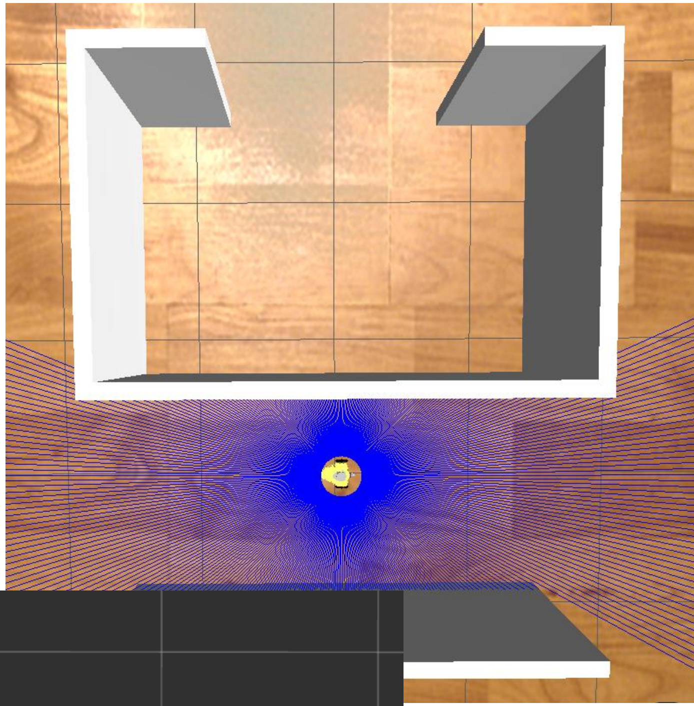
- LiDAR (Light Detection And Ranging) is a sensor used by robots to detect and measure distances to objects around them.
- It works by:
  - Emitting laser pulses in multiple directions.
  - Measuring the time, it takes for the laser to bounce off an object and return.
  - Calculating distances from the measured time.

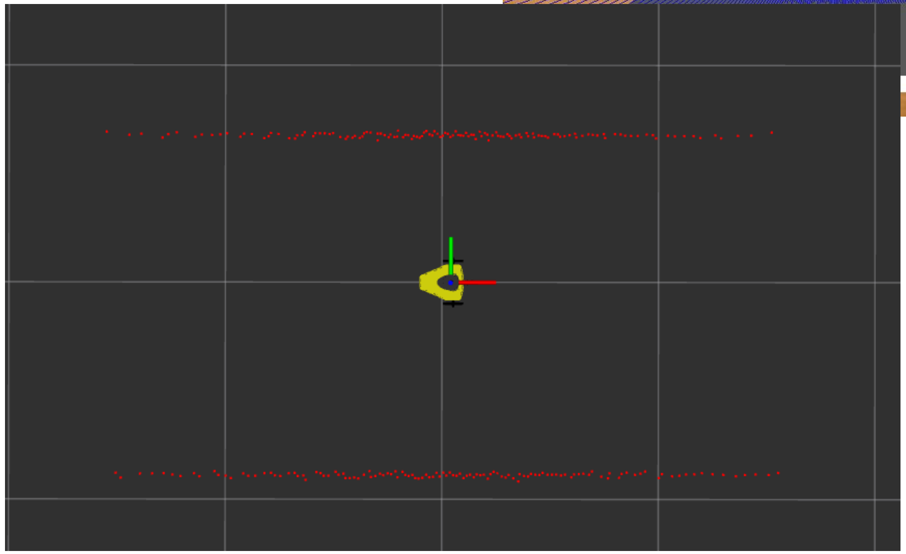


Activity – Simple Obstacle Avoidance

LiDAR in Gazebo

- In Gazebo, the LiDAR sensor is simulated to allow robots to perceive their virtual environment realistically.
- Gazebo simulates LiDAR by generating virtual laser rays around the robot.
- Each ray measures the distance to the nearest object it encounters in the simulation.
- The sensor publishes this data as a LaserScan message in ROS2.



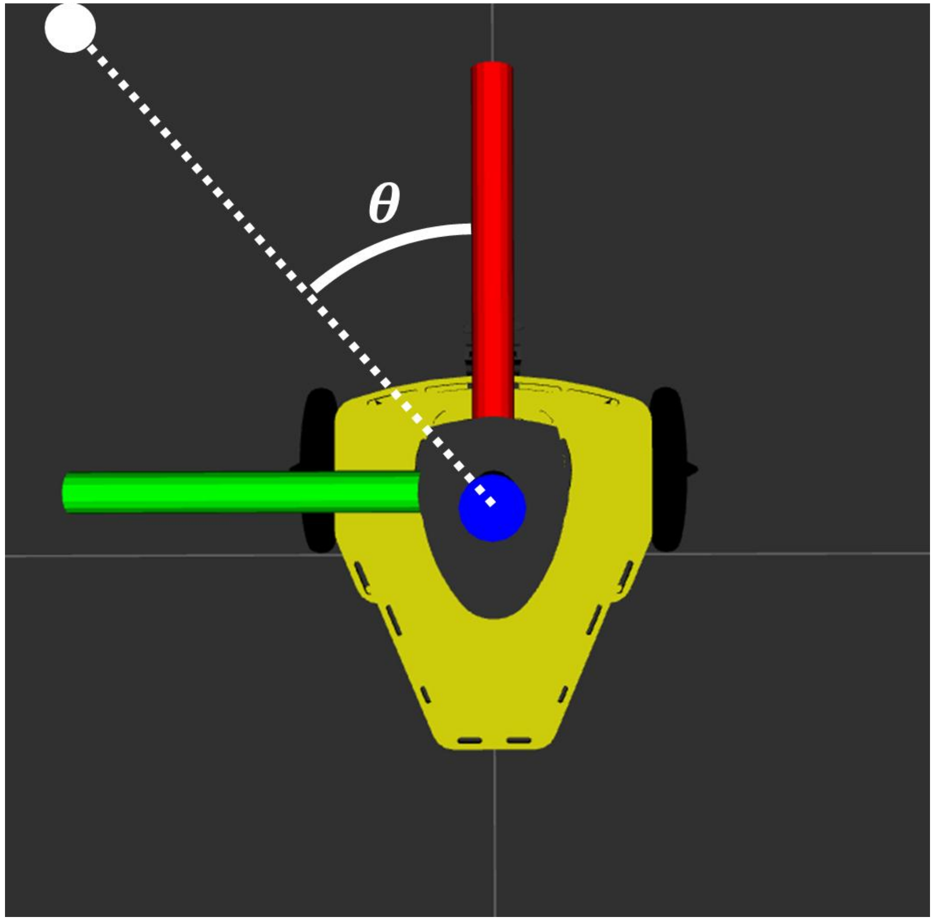


Activity – Simple Obstacle Avoidance

- The `sensor_msgs/LaserScan` Message contains the information regarding the LiDAR.
- Header: Includes the timestamp of the first ray in the scan. The `frame_id` defines the measurement angles.
  - *They are measured around the positive Z axis counterclockwise. Zero angle being forward along the x axis.*
- angle_min/angle_max: Start and End angles of the scan.
- angle_increment: Angular distance between measurements.
- time_increment: time between measurements.
- scan_time: time between scans
- range_min/range_max: minimum/maximum range value
- ranges[ ]: range data [m] (Note: values ‹ range_min or › range_max should be discarded)
- intensities[ ]: Intensities data

```
header:
  stamp:
    sec: 614
    nanosec: 900000000
  frame_id: robot1/laser_frame
angle_min: 0.0
angle_max: 6.283185005187988
angle_increment: 0.01750190742313862
time_increment: 0.0
scan_time: 0.0
range_min: 0.15000000596046448
range_max: 12.0
ranges:
```

Activity – Simple Obstacle Avoidance



- As stated previously, The angles are measured around the positive Z axis counterclockwise. Zero angle being forward along the x axis.
- Angle range: Usually covers 360° around the robot.
- Data: Provided as distances measured at specific angle increments.

Range (m)	4.0	4.2	3.2		inf	inf
Angle (deg)	0	1	2	...	358	359

Activity – Simple Obstacle Avoidance

Instructions

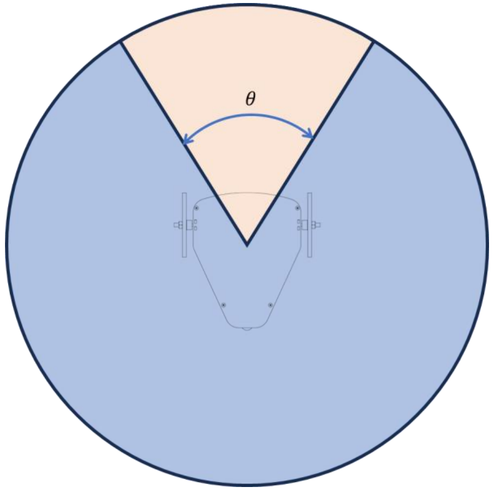
- Download the activity template package "puzzlebot_control" from GitHub or create a package with the following characteristics.

```
ros2 pkg create --build-type ament_python
puzzlebot_control --node-name
obstacle_avoidance_simple --dependencies rclpy
ros2launch python3-numpy std_msgs geometry_msgs
nav_msgs sensor_msgs --license Apache-2.0 --
maintainer-name 'Mario Martinez' --maintainer-email
'mario.mtz@manchester-robotics.com'
```

- In the package "puzzlebot_control" open the file "obstacle_avoidance_simple.py" on a text editor.

```
$ cd ~/ros2_ws/src/puzzlebot_control
$ code .     (for vscode)
```

- For this code, the LiDAR readings inside an angle of +-15 deg from the front of the robot will be read to avoid obstacles in front of the robot.



```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import numpy as np
import signal

class ObstacleAvoidance(Node):
    def __init__(self):
        super().__init__('obstacle_avoidance_node')

        self.publisher_ = self.create_publisher(Twist, 'robot1/cmd_vel', 10)
        self.subscription = self.create_subscription(
            LaserScan, 'robot1/scan', self.scan_callback, 10)

        self.obstacle_threshold = 0.5  # obstacle distance (m)
        self.forward_angle_width = 30  # degrees to check directly ahead

        self.get_logger().info("Obstacle Avoidance Node Started.")

    def scan_callback(self, msg: LaserScan):
        ranges = np.array(msg.ranges)

        # Angles to check directly ahead (±15 degrees)
        half_angle = np.radians(self.forward_angle_width / 2)
        idx_center = 0  # Angle 0 is directly ahead
        angle_increment = msg.angle_increment
        num_points = len(ranges)

        # Points within ±15 degrees ahead
        idx_offset = int(half_angle / angle_increment)

        indices = list(range(-idx_offset, idx_offset + 1))
        front_indices = [(idx_center + idx) % num_points for idx in indices]
        front_ranges = ranges[front_indices]

        # Check if obstacle detected in front
        if np.any(front_ranges < self.obstacle_threshold):
            self.get_logger().info('Obstacle detected! Turning...')
            self.turn()
        else:
            self.move_forward()

    def move_forward(self):
        twist = Twist()
        twist.linear.x = 0.2
        twist.angular.z = 0.3
        self.publisher_.publish(twist)

    def turn(self):
        twist = Twist()
        twist.linear.x = 0.0
        twist.angular.z = 0.5
        self.publisher_.publish(twist)

    def wait_for_ros_time(self):
        self.get_logger().info('Waiting for ROS time to become active...')
        while rclpy.ok():
            now = self.get_clock().now()
            if now.nanoseconds > 0:
                break
            rclpy.spin_once(self, timeout_sec=0.1)
        self.get_logger().info(f'ROS time is active! Start time: {now.nanoseconds *
1e-9:.2f}s')

    def stop_handler(self,signum, frame):
        """Handles Ctrl+C (SIGINT)."""
        self.get_logger().info("Interrupt received! Stopping node...")
        raise SystemExit
```

```python
def main(args=None):

    rclpy.init(args=args)

    node = ObstacleAvoidance()

    signal.signal(signal.SIGINT, node.stop_handler)

    try:
        rclpy.spin(node)
    except SystemExit:
        node.get_logger().info('SystemExit triggered. Shutting down 
cleanly.')
    finally:
        # Make sure the robot stops
        twist = Twist()
        twist.linear.x = 0.0
        twist.angular.z = 0.0
        node.publisher_.publish(twist)

        node.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Instructions

- Save and compile the file

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

- Start the Gazebo Simulation with the following settings
  - robot:

```
'name': 'robot1',
'type': 'puzzlebot_jetson_lidar_ed',
'x': 0.0, 'y': 0.0, 'yaw': 0.0,
'lidar_frame': 'laser_frame',
'camera_frame': 'camera_link_optical',
'tof_frame': 'tof_link'
```

  - World: “obstacle_avoidance_4.world”

- Run the node as follows

```
$ ros2 run puzzlebot_control obstacle_avoidance_simple
--ros-args -p use_sim_time:=true
```

Results