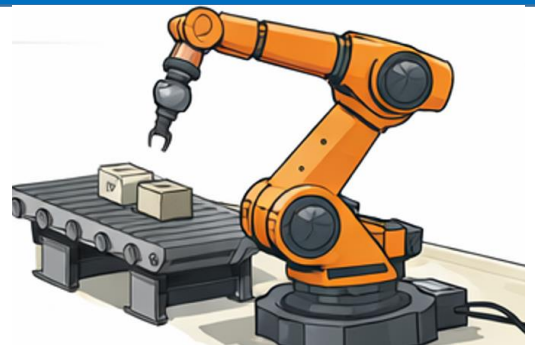
Robotique
et vision par ordinateur



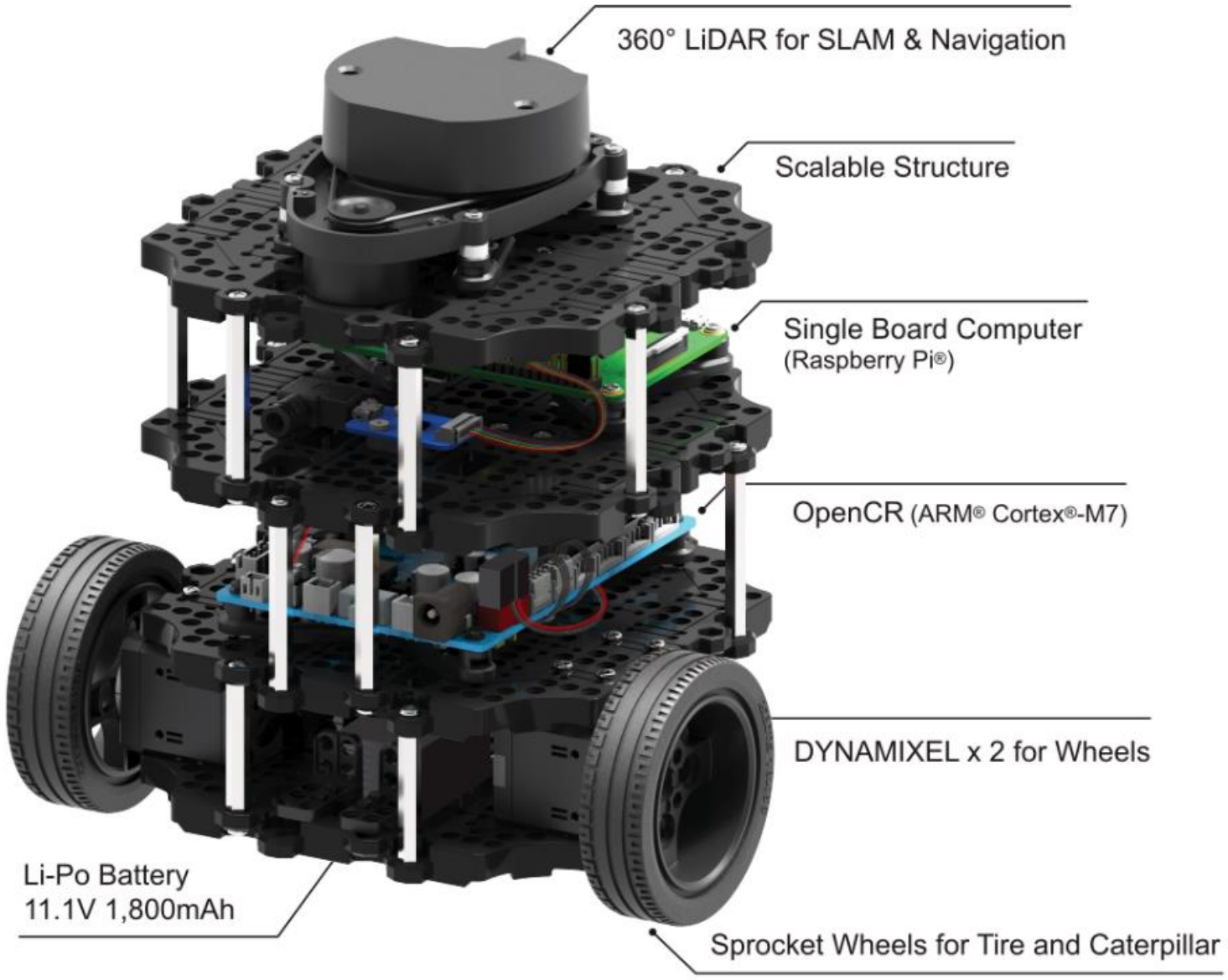
Master en ingénierie des systèmes embarqués
(MUS ISE)

Chapitre 3: Two Wheeled Robot Equipped with a Lidar

Année universitaire 2025/2026

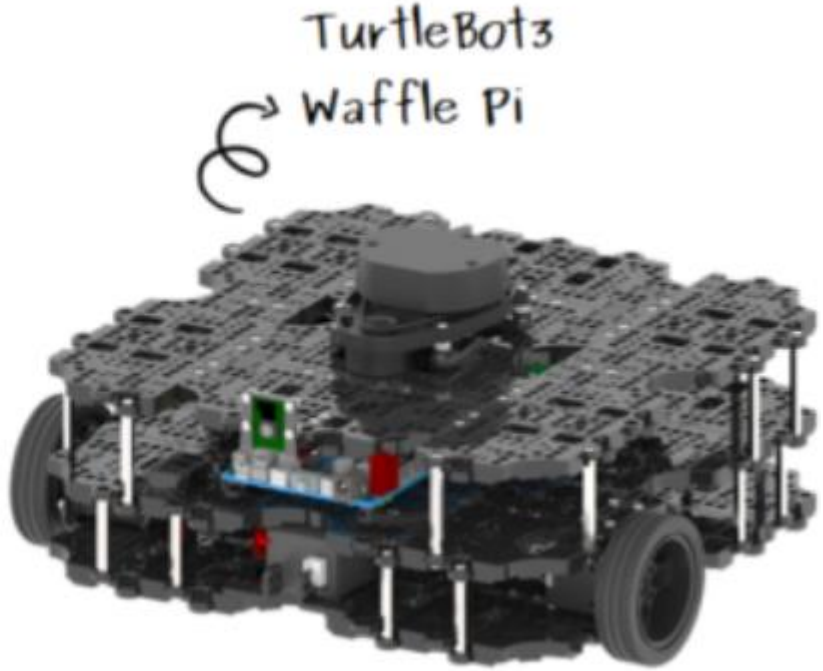
Dr. Ing. LASSIOUI Abdellah

TurtleBot is a standardized robotic platform developed for education and research.



There are two variants of turtlebot robot:

- TurtleBot3 Burger
- TurtleBot3 Waffle Pi



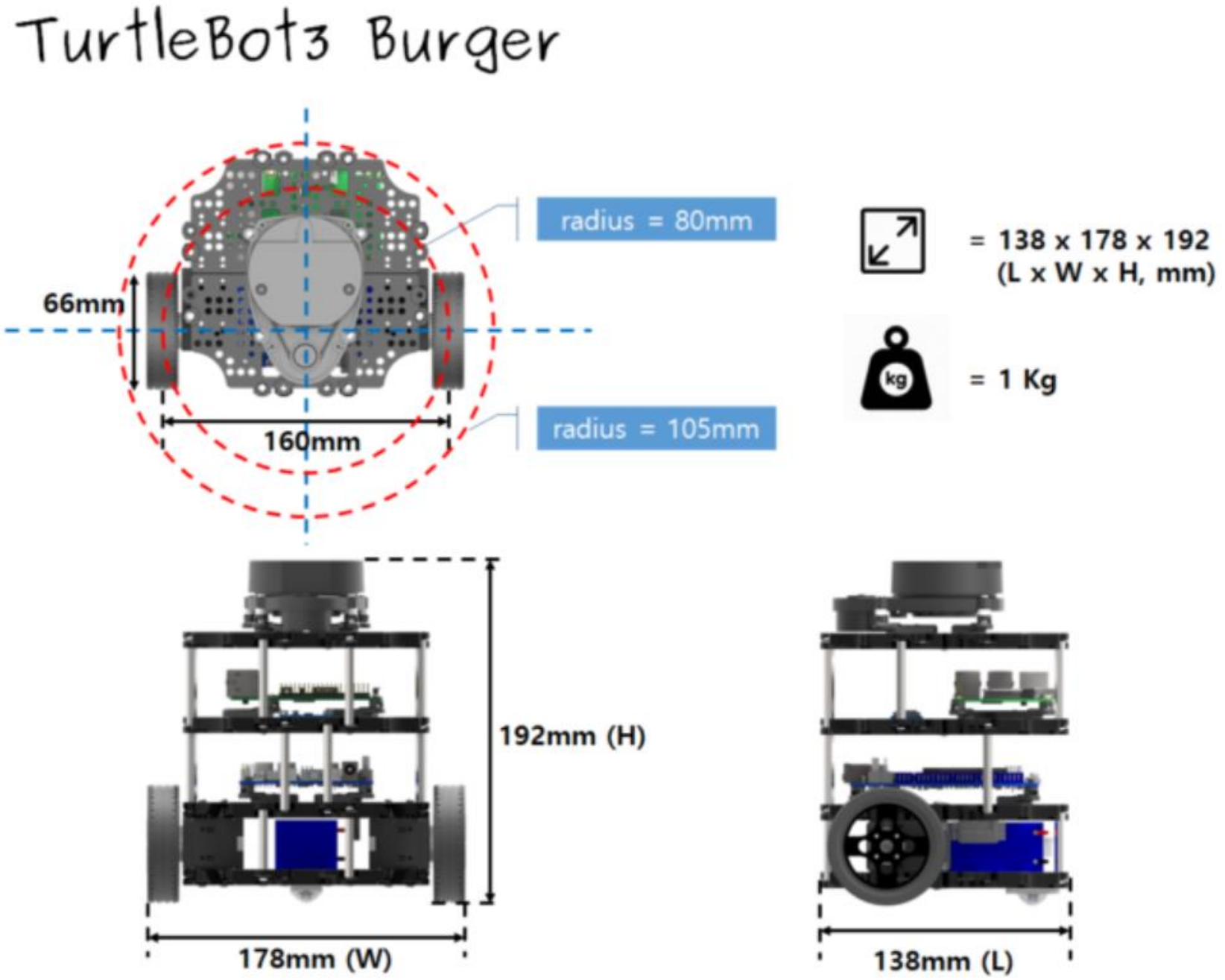
Specifications

Items	Burger	Waffle Pi
Maximum translational velocity	0.22 m/s	0.26 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)	1.82 rad/s (104.27 deg/s)
Maximum payload	15kg	30kg
Size (L x W x H)	138mm x 178mm x 192mm	281mm x 306mm x 141mm
Weight (+ SBC + Battery + Sensors)	1kg	1.8kg
Climbing Threshold	10 mm or lower	10 mm or lower
Expected operating time	2h 30m	2h
Expected charging time	2h 30m	2h 30m
SBC (Single Board Computer)	Raspberry Pi 4	Raspberry Pi 4
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	-	RC-100B + BT-410 Set (Bluetooth 4, BLE)
Actuator	XL430-W250	XM430-W210
LDS (Laser Distance Sensor)	360 Laser Distance Sensor LDS-02	360 Laser Distance Sensor LDS-02
Camera	-	Raspberry Pi Camera Module v2.1

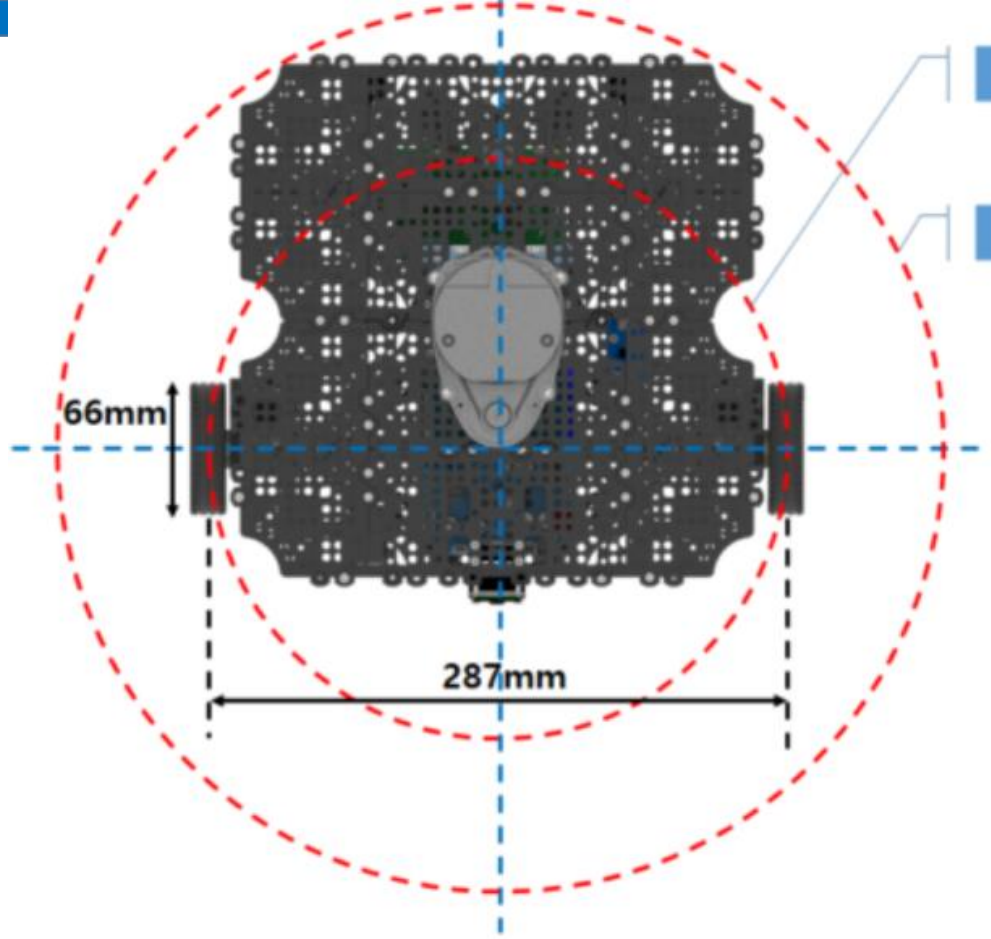
Specifications

Items	Burger	Waffle Pi
IMU	Gyroscope 3 Axis Accelerometer 3 Axis	Gyroscope 3 Axis Accelerometer 3 Axis
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A	3.3V / 800mA 5V / 4A 12V / 1A
Expansion pins	GPIO 18 pins Arduino 32 pin	GPIO 18 pins Arduino 32 pin
Peripheral Connections	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
DYNAMIXEL ports	RS485 x 3, TTL x 3	RS485 x 3, TTL x 3
Audio	Several programmable beep sequences	Several programmable beep sequences
Programmable LEDs	User LED x 4	User LED x 4
Status LEDs	Board status LED x 1 Arduino LED x 1 Power LED x 1	Board status LED x 1 Arduino LED x 1 Power LED x 1
Buttons and Switches	Push buttons x 2, Reset button x 1, Dip switch x 2	Push buttons x 2, Reset button x 1, Dip switch x 2
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
PC Connection	USB	USB
Firmware Upgrade	via USB / via JTAG	via USB / via JTAG
Power Adapter (SMPS)	Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A	Input : 100-240V, AC 50/60Hz, 1.5A @max Output : 12V DC, 5A

Specifications

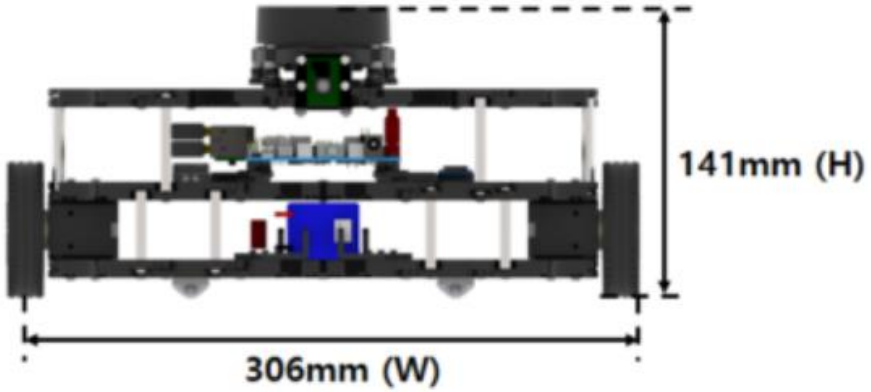


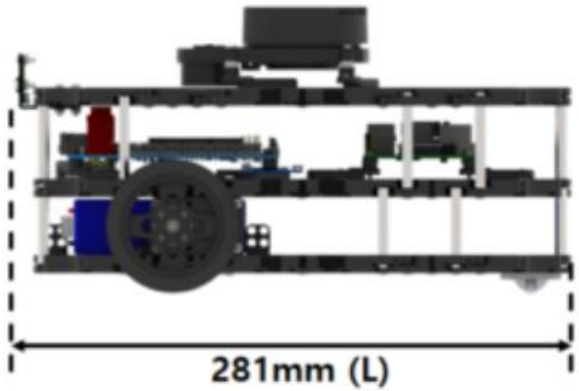
Specifications



= 281 x 306 x 141
(L x W x H, mm)

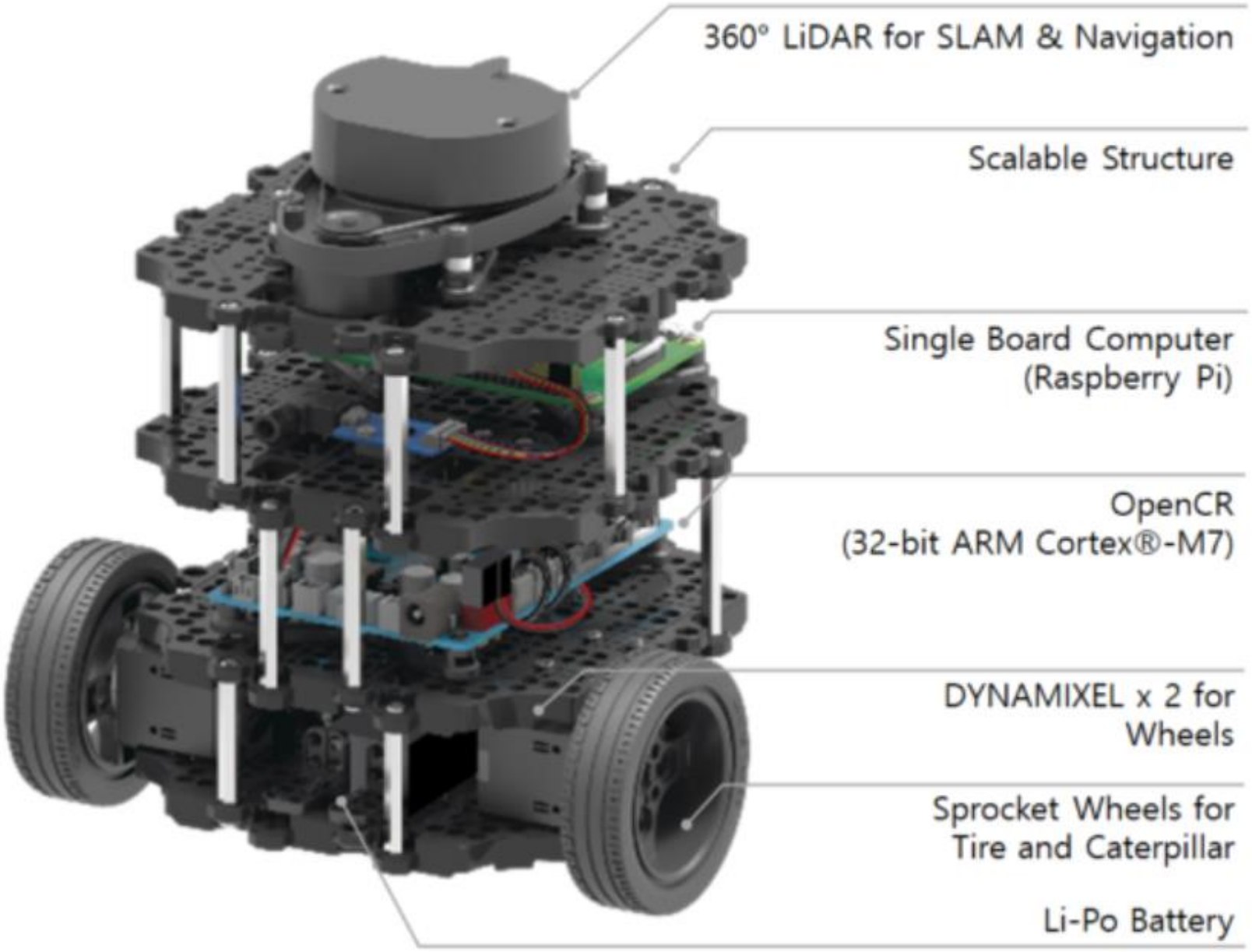
= 1.8 Kg



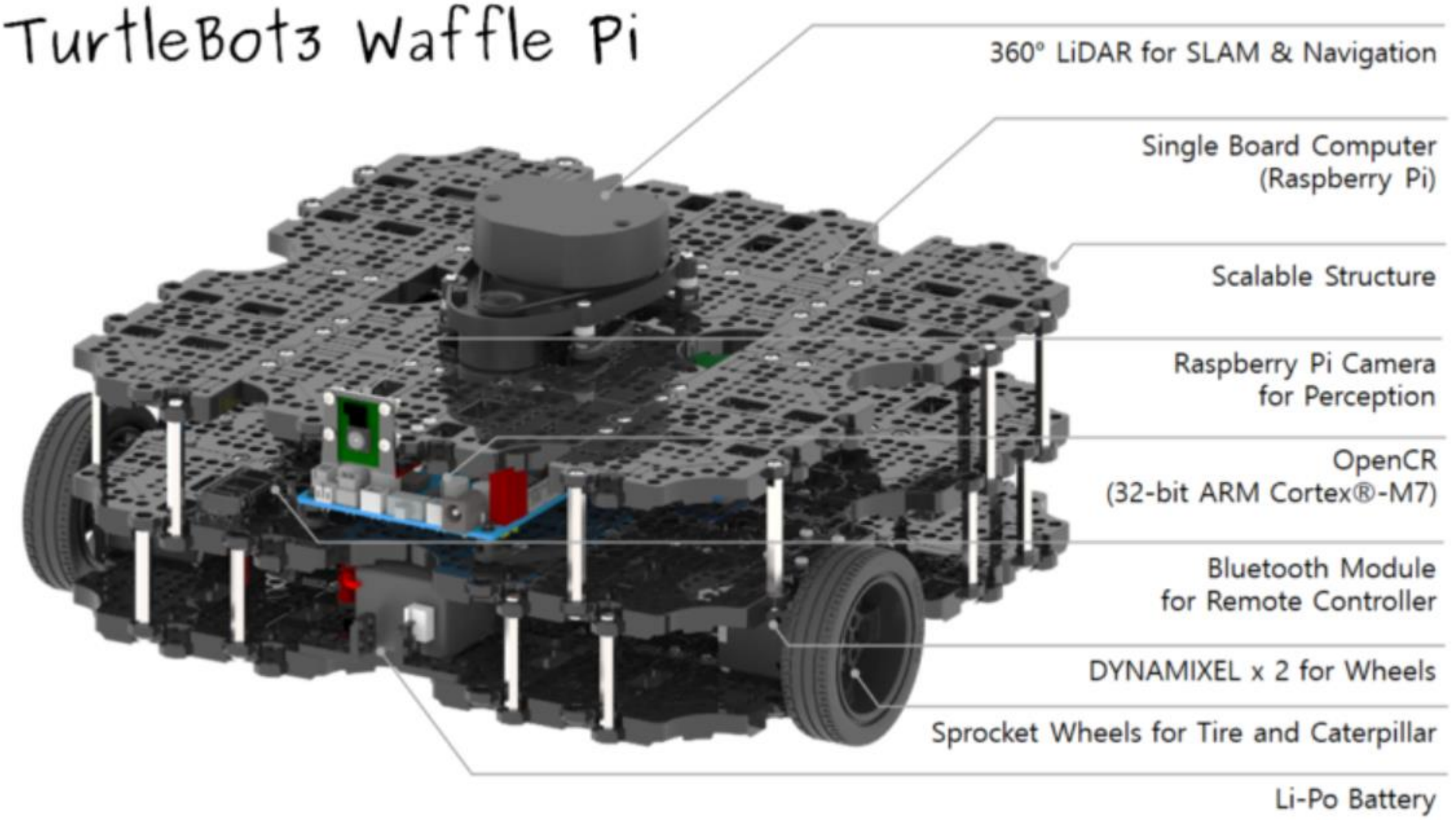


Specifications

SLAM (Simultaneous Localization and Mapping) is a technique that allows a robot to **build a map of an unknown environment while simultaneously determining its own position** within that map.

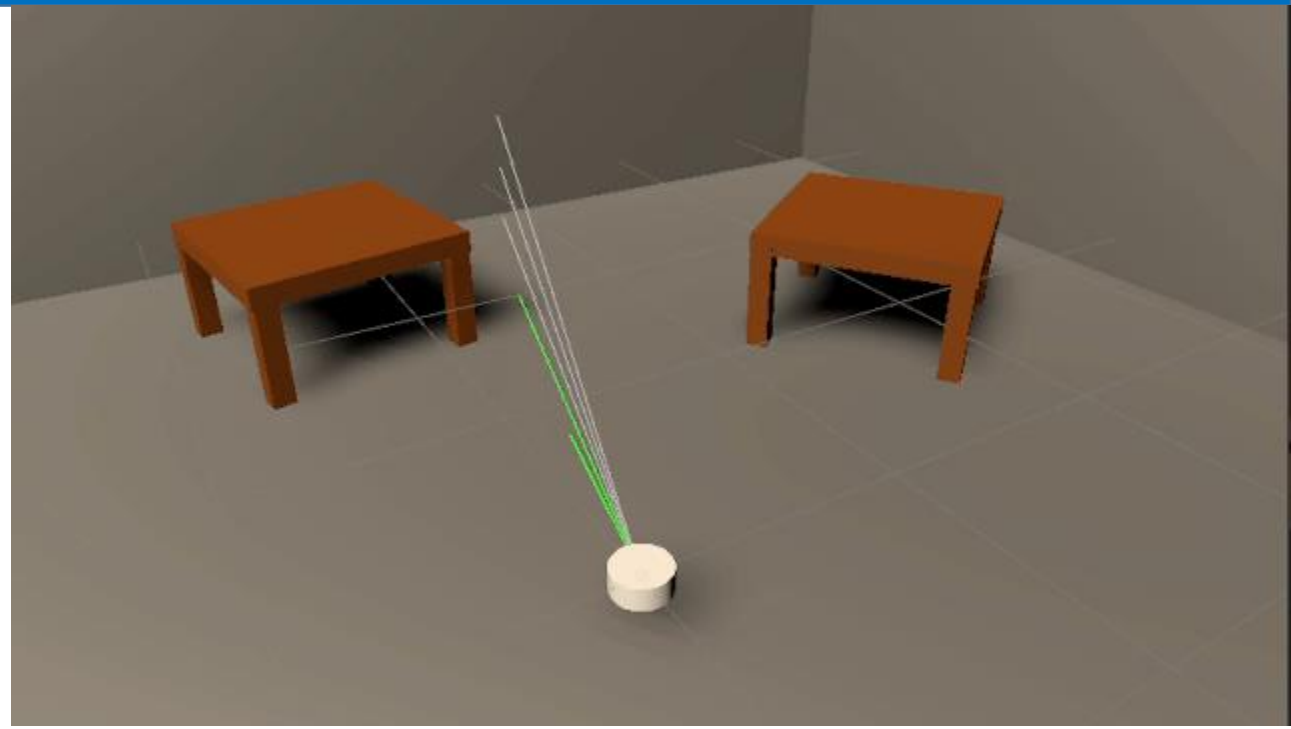


Specifications



LiDAR principle

- LiDAR (Light Detection and Ranging) is a remote sensing technology that measures distances by using laser light.
- It is widely used in autonomous vehicles, robotics, topographic mapping, and environmental monitoring.



Basic Principle

LiDAR operates on the **time-of-flight** principle.
A laser pulse is emitted toward a target, reflected by the object, and then received back by a sensor. By measuring the time taken for the pulse to return, the distance to the object is calculated.

$$\text{Distance} = \frac{c \times \Delta t}{2}$$

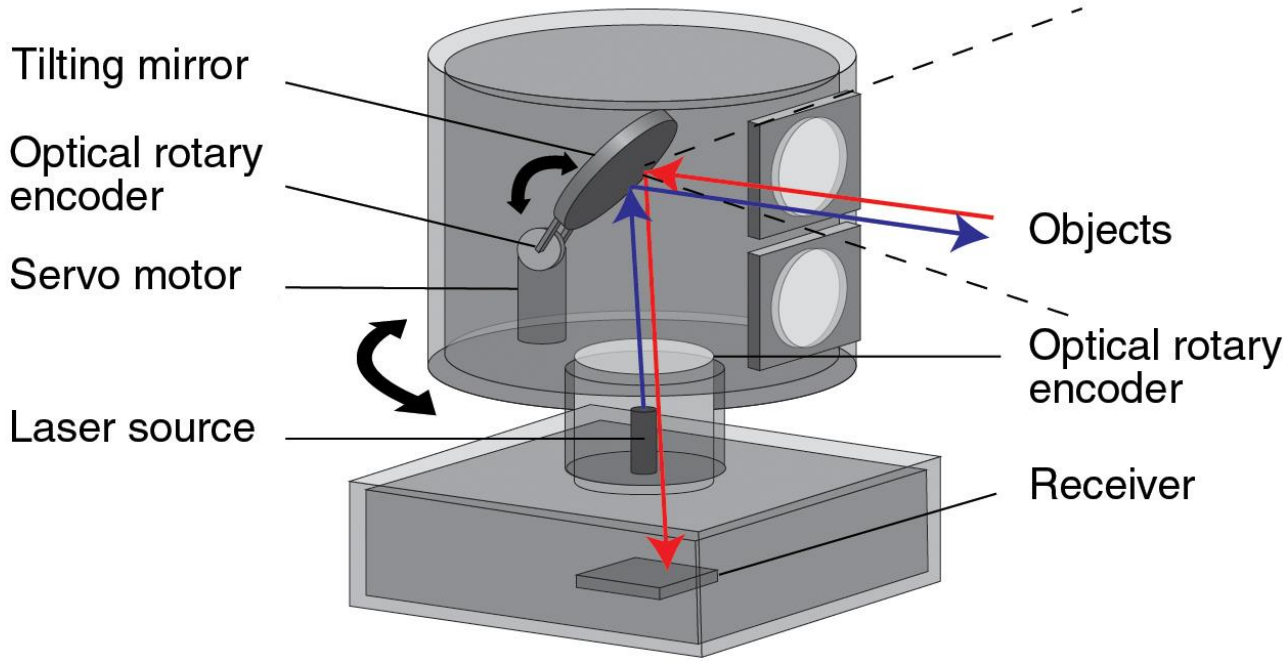
LiDAR uses laser (light) instead of radio waves because lasers provide much higher resolution and accuracy due to their short wavelength, enabling precise distance measurement and detailed 3D mapping.

LiDAR principle

Main Components

A typical LiDAR system includes:

- **Laser source**: emits short, high-frequency light pulses (usually infrared).
- **Scanner / rotating mirror**: directs the laser beam across the environment.
- **Photodetector**: detects the reflected light.
- **Timing and processing unit**: computes distances and builds a 3D model.
- Optical rotary encoder: measures the **exact angular position and rotation speed of the scanning mechanism**, allowing each laser pulse to be assigned a **precise direction** for accurate 2D/3D mapping.



LiDAR uses **laser light**, so its frequency is in the **optical (terahertz) range**

$$f = \frac{c}{\lambda}$$

905 nm → $f \approx 3.31 \times 10^{14}$ Hz (331 THz)

1550 nm → $f \approx 1.93 \times 10^{14}$ Hz (193 THz)

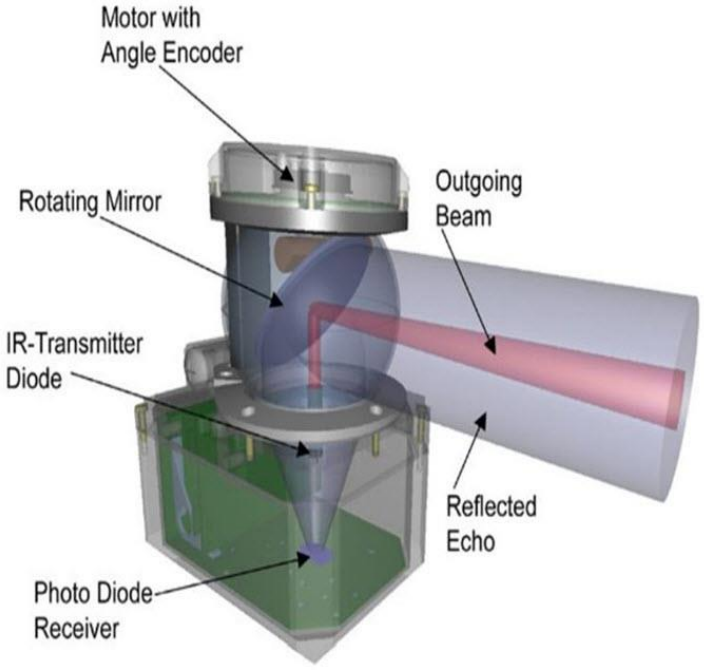
LiDAR principle

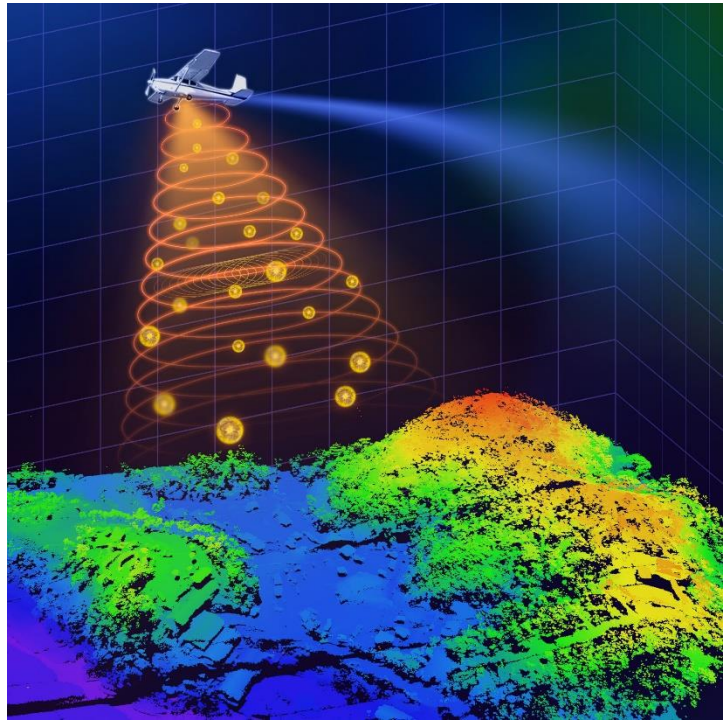
Scanning and Point Cloud Generation

As the laser scans across a scene, millions of distance measurements are taken per second. Each measurement corresponds to a point in space. The collection of these points forms a **3D point cloud**, representing the shape and position of surrounding objects.

Each point typically contains:

- ❑ x,y,z coordinates,
- ❑ Reflection intensity (useful for material classification).



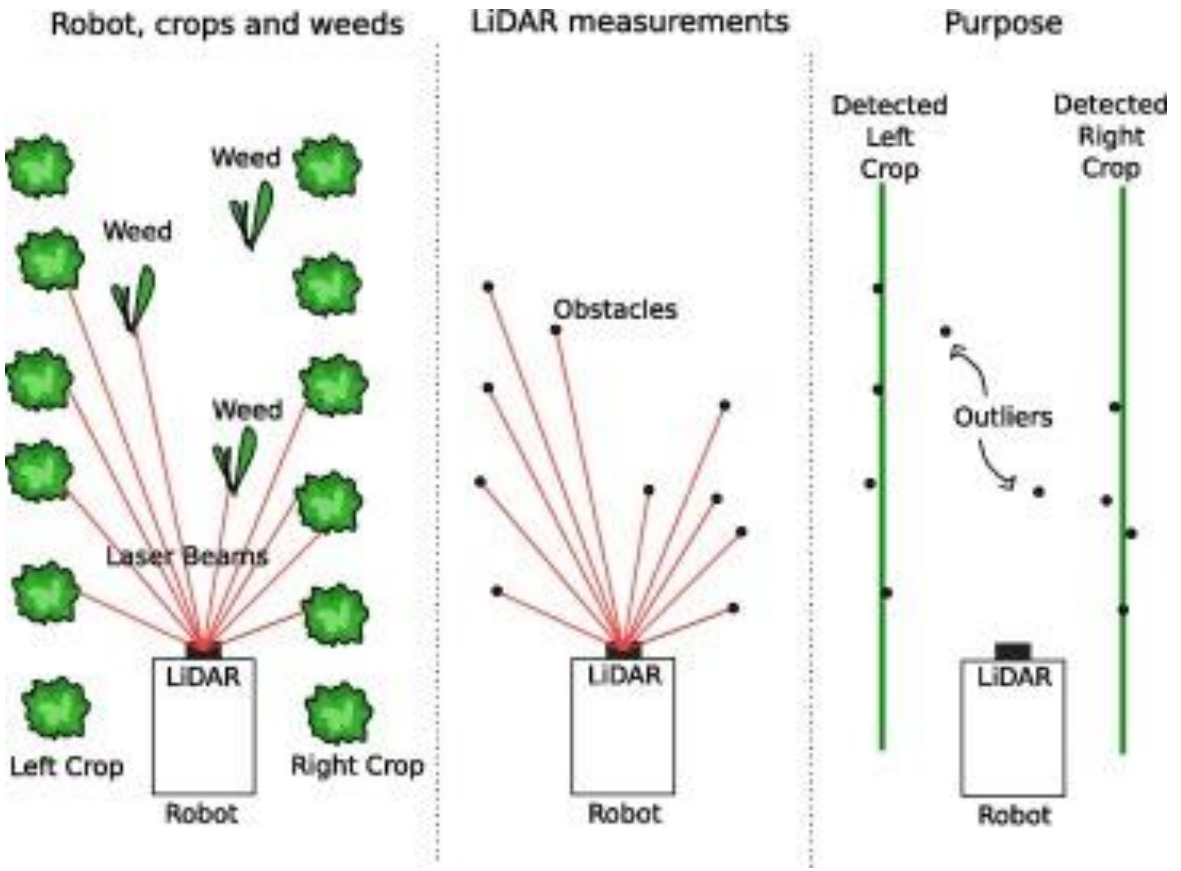


Use of LiDAR in robotics

LiDAR is a core perception sensor in robotics because it provides accurate, real-time distance measurements and 3D environmental structure. Its principal uses include the following.

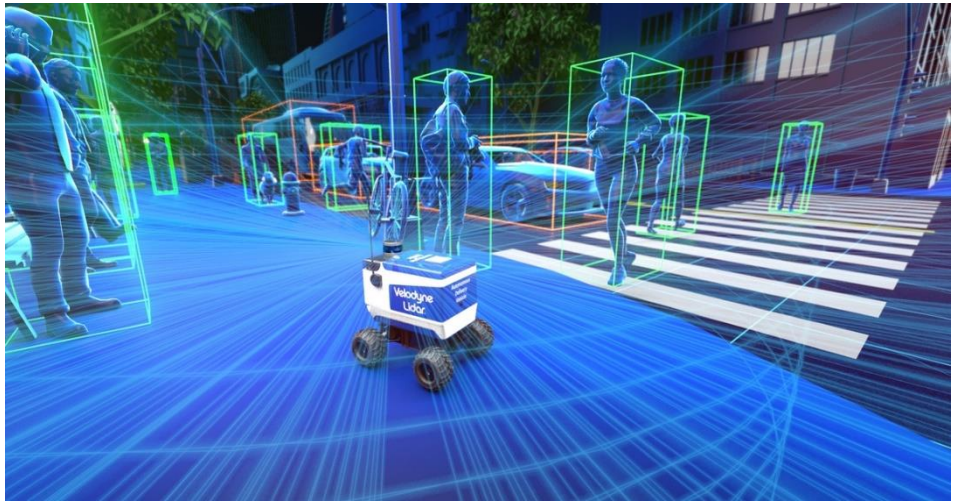
Environment Mapping (SLAM)

- LiDAR is central to **Simultaneous Localization and Mapping (SLAM)**. By continuously scanning the surroundings, a robot builds a geometric map (2D or 3D) while estimating its own position within that map.
- This capability is essential for mobile robots operating in **unknown or changing environments.**



Use of LiDAR in robotics

LiDAR is a core perception sensor in robotics because it provides accurate, real-time distance measurements and 3D environmental structure. Its principal uses include the following.



Obstacle Detection and Avoidance

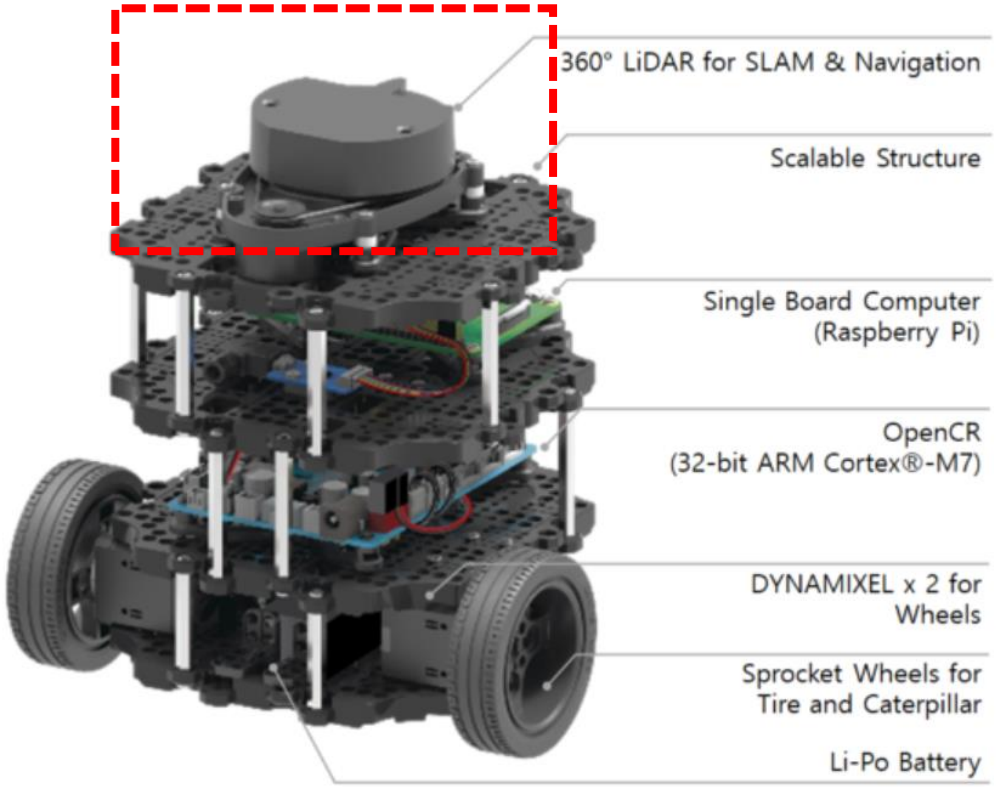
LiDAR provides precise distance and shape information for nearby objects. Robots use this data to:

- ❑ Detect static and dynamic obstacles,
- ❑ Estimate free space,
- ❑ Execute real-time collision avoidance maneuvers.

  This is critical for safety in human-robot shared spaces.

Specifications of the LDS-02 Lidar used in TurtleBot Robot

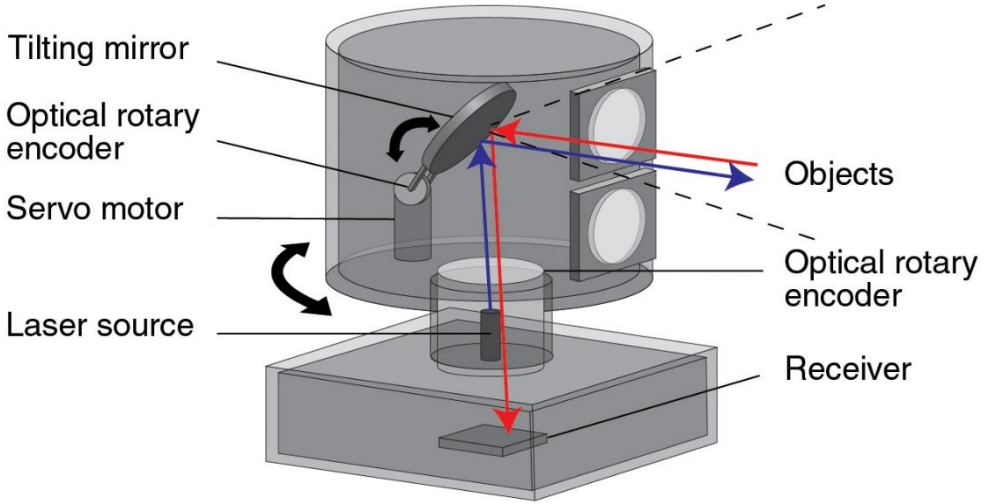
- 360 Laser Distance Sensor LDS-02 is a 2D laser scanner capable of sensing 360 degrees that collects a set of data around the robot to use for SLAM (Simultaneous Localization and Mapping) and Navigation.
- The LDS-02 is used for TurtleBot3 Burger and Waffle Pi models.
- Only Tx UART interface is available for the LDS-02 sensor.



General Specifications

Items	Specifications
Operating supply voltage	5V DC ± 10%
PWM Frequency	10 ~ 30 KHz (Square wave, High : 3.3V, Low : 0V)
PWM Duty Cycle	0 ~ 100%
LASWER Wave Length	Low powered Infrared Laser (λ=793 nm)
LASER safety	Class I, 21 CFR 1040.10 and 1040.11
Current consumption	240 mA (Start up current 400 mA)
Detection distance	160 ~ 8,000 mm
Interface	3.3V USART (115200 bps, 8 data bits, no parity, 1 stop bit), Tx Only
Ambient Light Resistance	25,000 lux
Life Time	1,000 hrs
Sampling Rate	2.3kHz (Fixed)
Operating Temperature	-10 ~ 40 °C
Storage Temperature	-30 ~ 70 °C
Dimensions	70(W) X 90(D) X 42(H)mm
Mass	131 g

- **PWM frequency (10–30 kHz):**
  The switching frequency of the PWM signal used to **control the LiDAR motor speed**
- **PWM duty cycle (0–100%):**
  The percentage of ON time of the PWM signal, which **sets the motor speed**
- **Sampling rate (2.3 kHz):**
  How often the LiDAR **measures and outputs distance data** (≈ 2300 measurements per second).



Specifications

Measurement Performance Specifications

Items	Specifications
Distance Range	160 ~ 8,000mm
Distance Accuracy (160 ~ 300 mm)	±10mm
Distance Accuracy(300 ~ 6,000 mm)	±3.0%
Distance Precision(6,000 ~ 8,000 mm)	±5.0%
(1) Scan Frequency	5Hz or above
Angular Range	360 °
(2) Angular Resolution	1 °

NOTE :

(1) Scan Frequency may vary by each product.

(2) Due to the fixed sampling rate, the Angular Resolution may vary by the Scan Frequency.

- **Accuracy:** How close a measurement is to the **true or actual value**
- **Precision:** How **consistent or repeatable** measurements are, regardless of the true value

- The **scan frequency** of a LiDAR is the **rate at which the sensor completes a full 360° (or specified angle) rotation**, usually measured in **Hz**.
  - Example: **5 Hz** → LiDAR completes 5 full rotations per second.
  - It determines how often the environment is **fully scanned**

Specifications

How the LDS-02 Communicates Data

The **LDS-02 LiDAR sensor** communicates measurement data to the robot controller through a **serial digital interface**, designed for continuous real-time transmission of distance measurements.

Physical Interface

The LDS-02 typically uses a **UART (TTL-level serial communication)** interface.

Communication Protocol

The communication is based on a **custom binary serial protocol**.

General features:

- ❑ Asynchronous serial communication (UART)
- ❑ Fixed **baud rate** (commonly around 115200 bps, depending on the version)
- ❑ Data sent as **frames (packets)**
- ❑ Each frame contains multiple distance samples corresponding to a small angular sector of the 360° scan.

Data Frame Structure (Conceptual)

Although exact byte definitions depend on the firmware version, a typical LDS-02 data frame includes:

- **Frame header**: synchronization bytes (used to detect start of frame)
- **Rotation or angle information**: starting angle of the scan segment
- **Distance measurements**: multiple distance values (in millimeters)
- **Intensity or quality values** (optional)
- **Checksum**: for data integrity verification

The controller parses the incoming stream to reconstruct a full 360° scan.

Timing and Data Flow

- The LDS rotates continuously.
- For each partial rotation, the sensor sends a packet with:
  - A group of distance points
  - Corresponding angular positions
- A full revolution is reconstructed by accumulating successive packets.

This allows **real-time mapping and obstacle detection**

The **LDS-02** is a **2D LiDAR sensor** — it scans in a single horizontal plane to create a **2D map** of the environment.

Application: Control program

```
# library import
from controller import Robot
import math
```

- Robot: Webots API class that represents the robot and provides access to sensors, actuators, and simulation control.
- math: Imported for mathematical operations (not used in this specific code, but often required for angle or coordinate computations).

```
#Robot initialization
robot = Robot()
```

- Creates an instance of the TurtleBot3 Burger robot.
- This object is the main interface between the controller and the simulated robot.

Application: Control program

```
#time step definition
timestep = int(robot.getBasicTimeStep())
```

- The time step defines how often the controller is executed.

- It must match the simulation step defined in Webots (typically in milliseconds).

- All sensors must be updated using this same time step.

```
#Motor configuration
left_motor = robot.getDevice("left wheel motor")
right_motor = robot.getDevice("right wheel motor")
```

- Retrieves the left and right wheel motors by name (as defined in the robot model).

```
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))
```

- Sets the motors to velocity control mode.

- float('inf') disables position control, allowing continuous rotation at a given speed.

Application: Control program

```
#LiDAR initialization
lidar = robot.getDevice("LDS-01")
lidar.enable(timestep)
lidar.enablePointCloud()
```

- Retrieves the LDS-01 LiDAR sensor.
- enable(timestep): Activates the LiDAR and updates it every simulation step.
- enablePointCloud(): Enables 2D point cloud generation (useful for visualization or advanced processing, though not strictly required here).

```
#Control parameters
MAX_SPEED = 6.67
SAFE_DISTANCE = 0.5  # meters
```

- MAX_SPEED: Maximum angular speed of the TurtleBot3 wheels (rad/s).
- SAFE_DISTANCE: Minimum allowed distance to an obstacle in front of the robot.

Application: Control program

```
#Main control loop
while robot.step(timestep) != -1:
```

- Main loop of the controller.

- Executes every timestep milliseconds until the simulation ends.

```
#reading LiDAR data
ranges = lidar.getRangeImage()
num_points = len(ranges)
```

- getRangeImage() returns a list of distance measurements (in meters).

- Each value corresponds to a specific LiDAR beam angle.

- num_points is the the total number of measurments in one scan

Application: Control program

```
#selecting the front sector
front_indices = range(
    int(num_points * 0.45),
    int(num_points * 0.55)
)
```

- The LiDAR provides a **360° scan**.
- This code selects the **central 10% of the scan**, corresponding approximately to **±15° in front of the robot**.
- This region is critical for detecting obstacles directly ahead.

```
front_distance = min([ranges[i] for i in front_indices])
```

- Computes the **minimum distance** among the front-facing LiDAR beams.
- This represents the distance to the closest obstacle in front of the robot.

Application: Control program

```
#obstacles avoidance logic
if front_distance < SAFE_DISTANCE:
```

- Checks whether an obstacle is closer than the safety threshold

```
# case 1 : obstacle detected
left_motor.setVelocity(-0.3 * MAX_SPEED)
right_motor.setVelocity(0.3 * MAX_SPEED)
```

- Left wheel rotates backward, right wheel forward
- The robot performs an **in-place left rotation** to avoid the obstacle

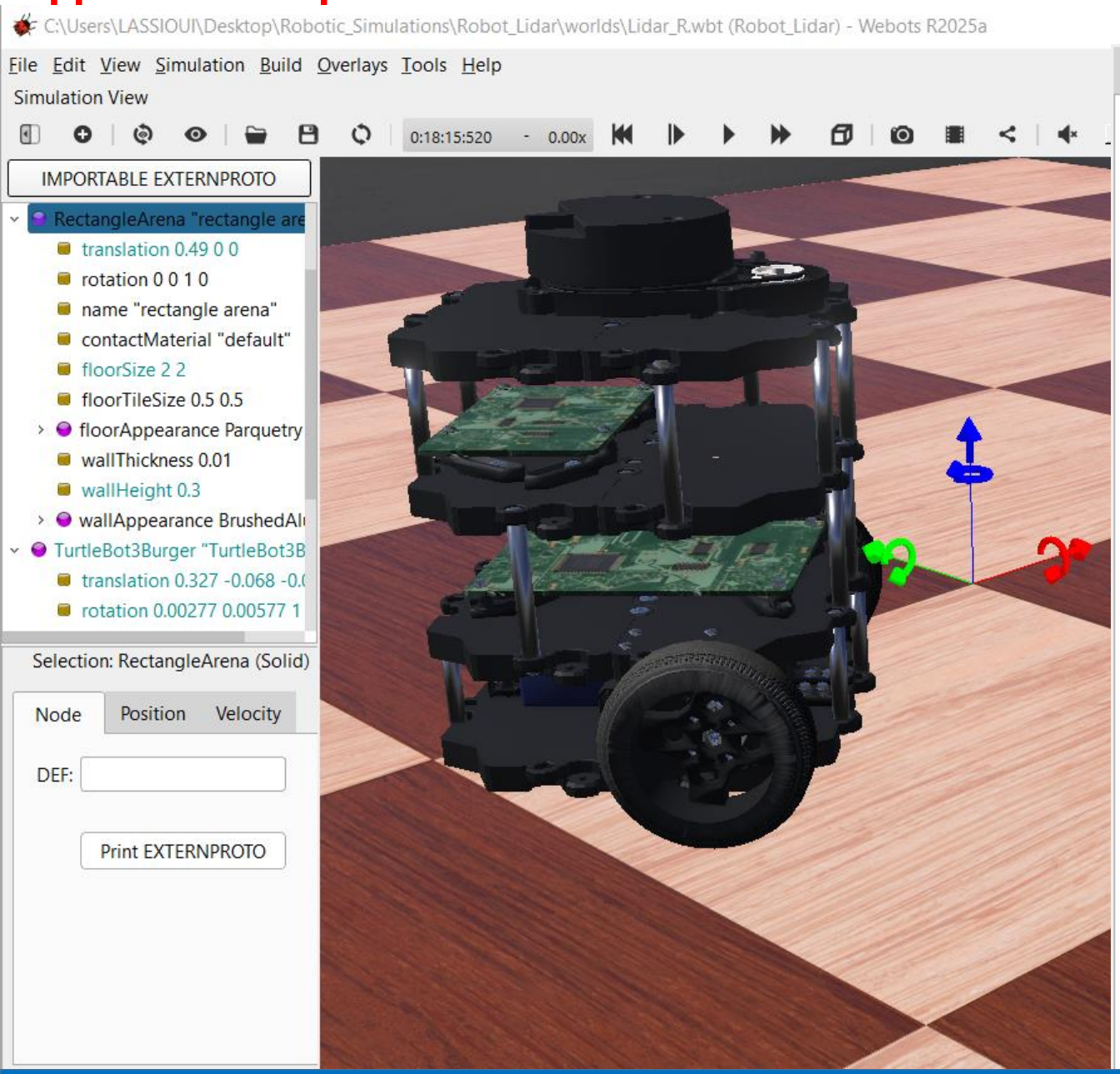
```
# Case 2: free path
left_motor.setVelocity(0.5 * MAX_SPEED)
right_motor.setVelocity(0.5 * MAX_SPEED)
```

Overall behavior

This controller implements a **reactive obstacle avoidance strategy**:

1. Continuously scan the environment using LiDAR.
2. Monitor obstacles directly in front of the robot.
3. Move forward if the path is clear.
4. Rotate left if an obstacle is detected within a safe distance.

Application: Implementation and simulation of the robot



turtlebot3_ostacle_avoidance.c | controller_1.py

```python
# LiDAR
lidar = robot.getDevice("LDS-01")
lidar.enable(timestep)
lidar.enablePointCloud()

# Parameters
MAX_SPEED = 6.67
SAFE_DISTANCE = 0.5  # meters

# Main loop
while robot.step(timestep) != -1:

    # Get lidar data
    ranges = lidar.getRangeImage()
    num_points = len(ranges)

    # Check front area (±15 degrees)
    front_indices = range(
        int(num_points * 0.45),
        int(num_points * 0.55)
    )

    front_distance = min([ranges[i] for i in front_indices])

    # Decision logic
    if front_distance < SAFE_DISTANCE:
        # Obstacle detected → turn left
        left_motor.setVelocity(-0.3 * MAX_SPEED)
        right_motor.setVelocity(0.3 * MAX_SPEED)
    else:
        # Free path → move forward
        left_motor.setVelocity(0.5 * MAX_SPEED)
        right_motor.setVelocity(0.5 * MAX_SPEED)

```