AUTONOMOUS
INTELLIGENCE GROUP

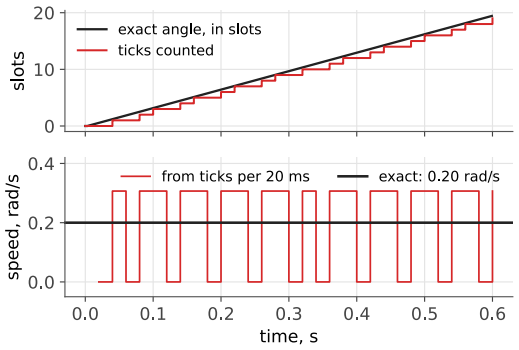
LECTURE 5 — SENSORS AND NOISE

WHAT A SENSOR MEASURES, HOW NOISY IT IS, AND WHERE THE LIDAR IS BLIND

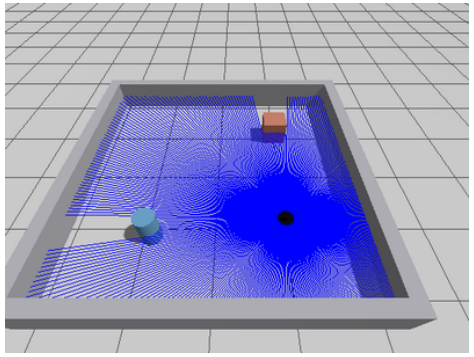
Topics: What Sensors Measure · The Lidar · Noise · Where the Lidar Is Blind

TODAY: SENSORS, THE LIDAR, NOISE, BLIND SPOTS

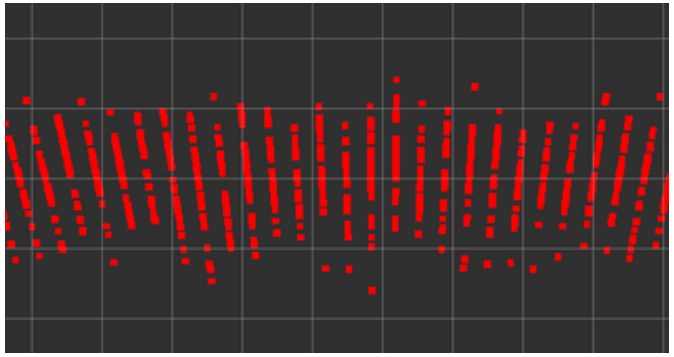
- **What does each sensor really measure?** — the wheel encoder, the IMU, the lidar
- **How does a lidar see?** — one ray, one scan, one message
- **How noisy is it?** — a noise model, the `<noise>` block in Gazebo, a measurement
- **Where is a 2D lidar blind?** — and what that costs later



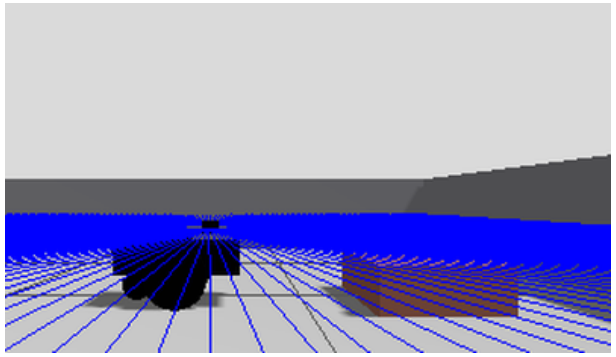
What a sensor measures



The lidar



Noise

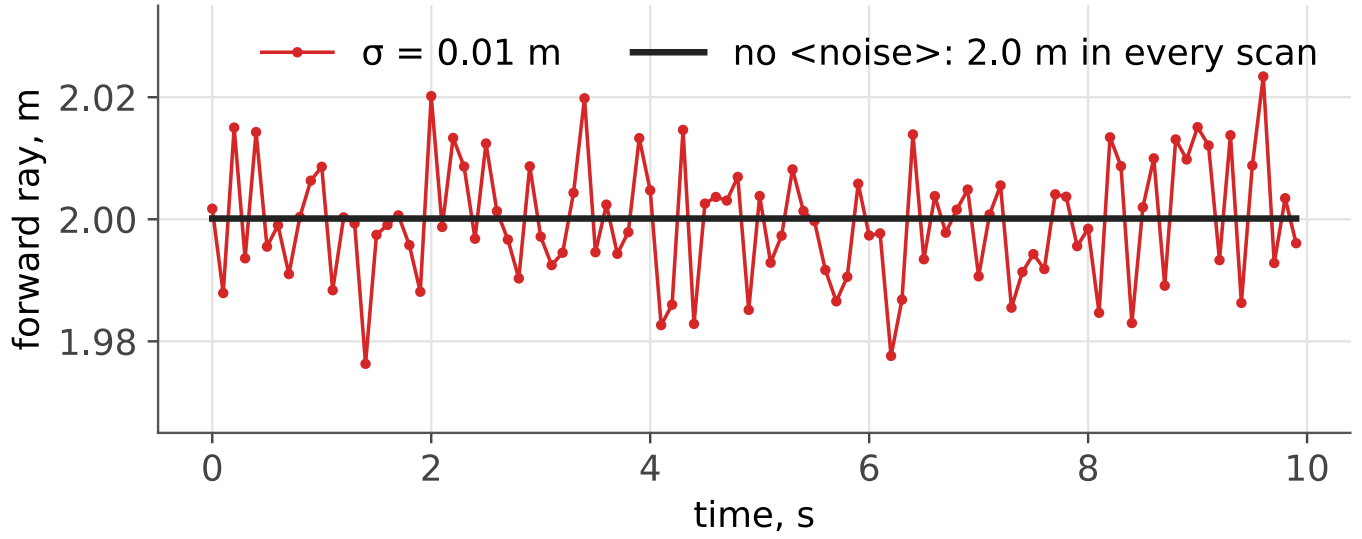


Where it is blind

A LIDAR 2 M FROM A WALL SAYS EXACTLY 2.0 M, EVERY TIME. IS IT A GOOD LIDAR?

ANSWER: TOO GOOD — A SIMULATED SENSOR IS PERFECT

- Gazebo returns the exact distance until you add noise: 2.0 m in every scan, spread 0
- A real lidar's reading jitters by about a centimetre
- Code tuned on a perfect sensor breaks on a real robot
- Later: a filter turns a jittery ray into a steady estimate



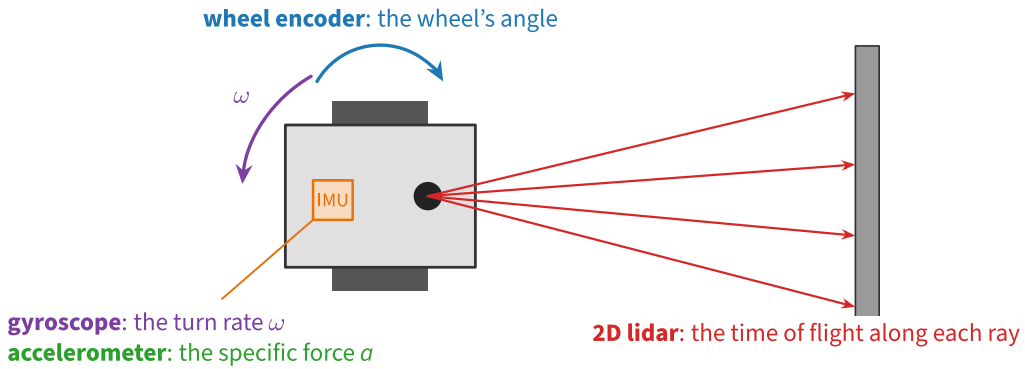
Measured in our setup: the forward ray, 100 scans in 10 s, without and with `<noise>`.

WHAT SENSORS MEASURE

SENSORS: WHAT THEY MEASURE, WHAT WE COMPUTE

SENSOR	MEASURES	WE WANT	HOW WE GET THERE	ABOUT
wheel encoder	the wheel's angle	where the robot is	integrate the wheel speeds — what DiffDrive does	itself
2D lidar	the time of flight along each ray	where the walls are	turn each ray into a point	the world
gyroscope	the turn rate	the yaw: where the robot faces	integrate	itself
accelerometer	the specific force	speed; roll and pitch	integrate twice; read gravity	itself

- Every step from *measures* to *we want* adds error — like odometry drift
- *About itself*: drifts, nothing outside to check against · *about the world*: fails where the world looks the same

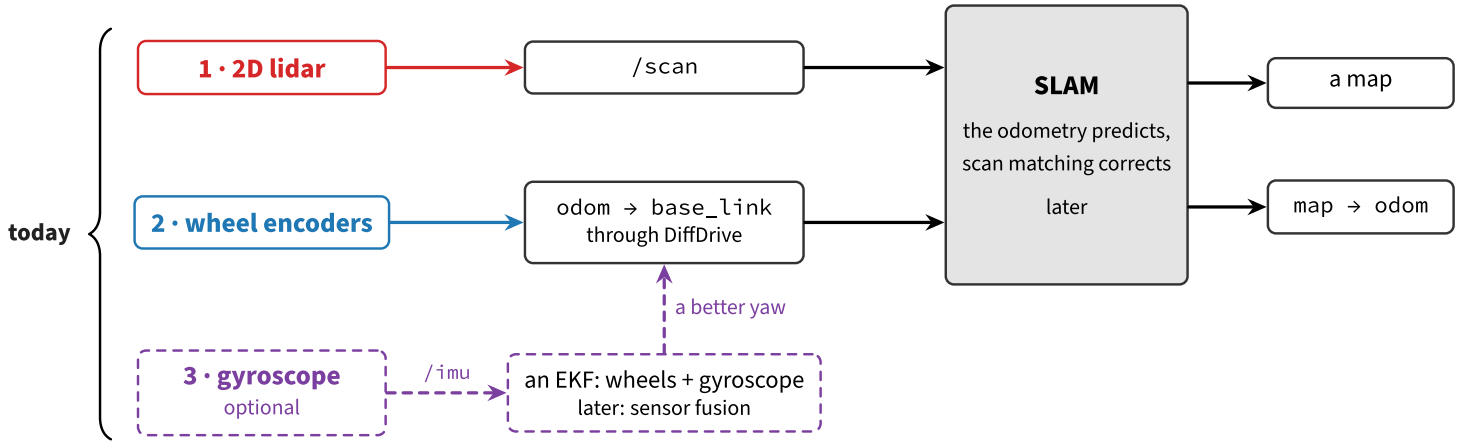


Our drawing: what each sensor on our robot measures.

📖 rpg.ifi.uzh.ch/docs/teaching/2025/Ch4_AMRobots.pdf

THE SIMPLEST SLAM: A LIDAR AND THE WHEELS

- **SLAM** (Simultaneous Localization and Mapping): build a map and find yourself on it, at the same time
- It takes two inputs: `/scan` from the lidar — what the walls look like — and `odom → base_link` from the wheel encoders — how far the robot moved between two scans
- A gyroscope does not go into SLAM: an EKF fuses it with the wheels into a better `odom → base_link` — later: sensor fusion
- Fusion is inside SLAM too: the odometry predicts, scan matching corrects — later: SLAM



Our drawing: the gyroscope reaches SLAM only through a better odometry.

🎯 github.com/SteveMacenski/slam_toolbox

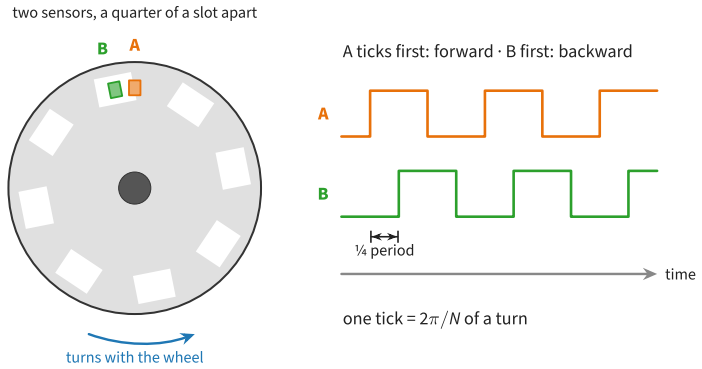
WHEEL ENCODER: HOW IT COUNTS

- A slotted disk turns with the wheel; a sensor counts the slots — **ticks**
- Two sensors a quarter-slot apart (*quadrature*): which ticks first gives the direction

$$\theta_{wheel} = 2\pi \frac{n}{N}$$

The wheel's angle from the ticks counted so far

- The angle moves in steps of one tick — *quantized*: $N = 1024$ → 0.35°



Our drawing: 8 slots, fewer than a real disk, so both sensors show; and the two signals.

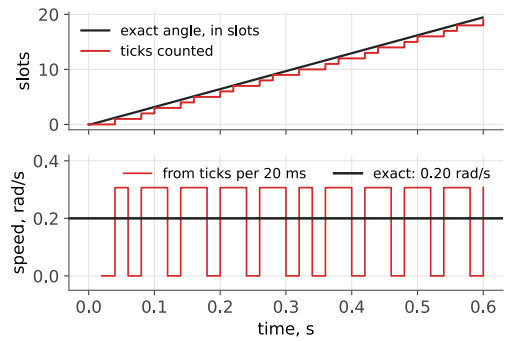
LEGEND

n — ticks counted
N — ticks per turn
θ_{wheel} — the wheel's angle, rad
2π rad — one full turn, 360°
$2\pi/N$ — one tick: the resolution

📖 rpg.ifi.uzh.ch/docs/teaching/2025/Ch4_AMRobots.pdf

`/joint_states`: WHAT THE ENCODER GIVES ROS 2

```
$ ros2 interface show sensor_msgs/msg/JointState
# … 19 lines of comments: rad or m, rad/s or m/s,
#   Nm or N; arrays optional, one size …
std_msgs/Header header
        builtin_interfaces/Time stamp
        string frame_id
string[] name
float64[] position
float64[] velocity
float64[] effort
```



Measured: `/joint_states` at 0.02 m/s, counted as a 1024-slot encoder every 20 ms.

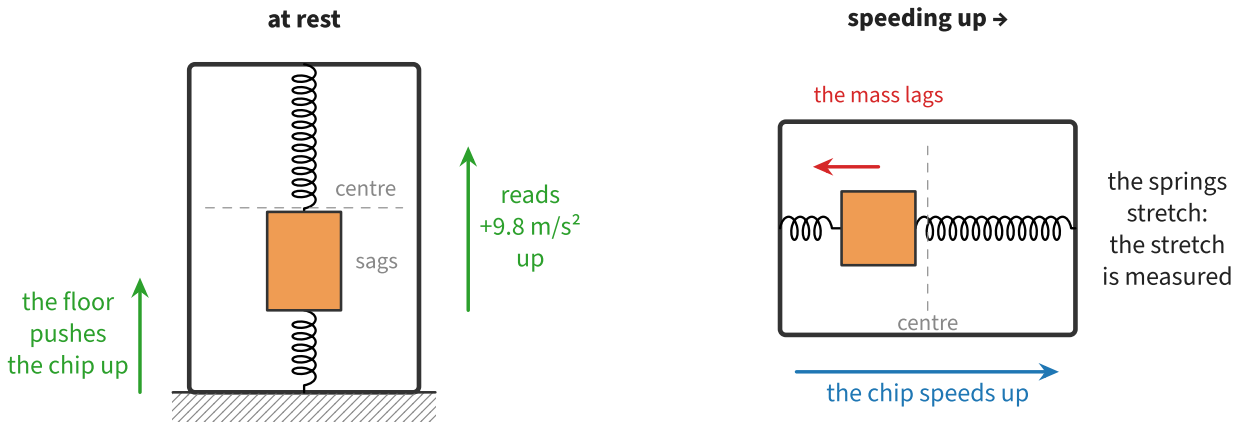
📖 github.com/ros2/common_interfaces/blob/jazzy/sensor_msgs/msg/JointState.msg

```
$ ros2 topic echo /joint_states --once     # shortened
name: [left_wheel_joint, right_wheel_joint]
position: [1.1814, 1.1814]
velocity: [0.2, 0.2]
effort: [0.0, 0.0]
```

- For SLAM: the two driven wheels — DiffDrive turns their angles into `odom → base_link`
- One entry per joint, in the order of `name`; an empty array — «not reported»
- `effort`: the joint's torque, N·m (N for a sliding joint) — a real motor gives it from its current · ours 0.0: DiffDrive sets the wheels' speed, not their torque
- Left: at low speed, ticks per Δt jump 0, 1, 0 · Gazebo's angle is exact, a real one also slips

ACCELEROMETER: A MASS ON SPRINGS

- **MEMS** (micro-electro-mechanical system): a tiny mass on springs in a chip
- The chip speeds up → the mass lags → the springs stretch → the stretch is measured
- It measures **specific force**, m/s²: every push on the robot — not gravity's pull
- At rest the floor pushes up: **+9.8 m/s² up**, not 0



Our drawing: the proof mass at rest and while the chip speeds up.

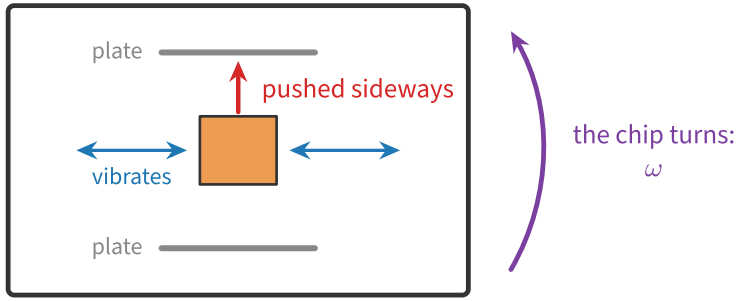
howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/
deeper — errors and drift: www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf

GYROSCOPE: A VIBRATING MASS AND THE CORIOLIS EFFECT

	THE CLASSIC GYROSCOPE — THE NAME COMES FROM IT	THE MEMS GYROSCOPE — IN OUR IMU CHIP
inside	a wheel spinning fast, in rings that turn freely	a tiny mass that vibrates all the time
when it turns	the wheel keeps its axis — the rings turn around it	the Coriolis effect pushes the mass sideways, harder the faster it turns
it gives	the angle between the frame and the axis	the push → **angular velocity**, rad/s — **not** the angle



«Gyroscope operation», Lucas Vieira — public domain, Wikimedia Commons.



the faster the turn, the harder the push — the push is measured

Our drawing: the vibrating mass and the sideways push. The angle is the turn rate added up — two slides on.

howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/
deeper — errors and drift: www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf

`/imu`: WHAT THE IMU GIVES ROS 2

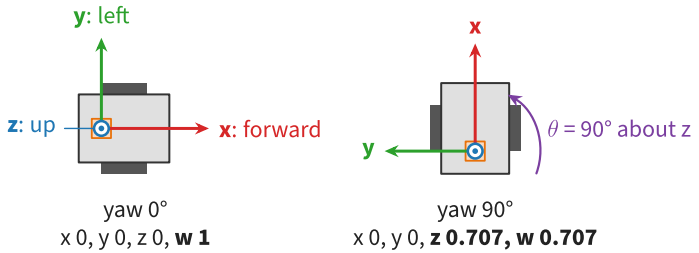
- **IMU** (Inertial Measurement Unit): an accelerometer and a gyroscope in one chip

```
$ ros2 interface show sensor_msgs/msg/Imu
# Accelerations should be in m/s^2 (not in g's),
# and rotational velocity should be in rad/sec
# … covariance: all zeros = unknown;
#   element 0 = -1: no estimate, ignore it …
std_msgs/Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance
```

- Frame `imu_link`, named by `<gz_frame_id>` · `/imu` only with the world's `Imu` system (measured)

📖 github.com/ros2/common_interfaces/blob/jazzy/sensor_msgs/msg/Imu.msg

- `angular_velocity` in rad/s, `linear_acceleration` in m/s² — not in g
- `orientation`: a quaternion, as `tf2_echo` prints it — no jump at ±180°, no gimbal lock at pitch ±90°, turns combine by multiplying · from the IMU's own filter; a bare chip: −1; Gazebo: the true one (measured)
- Covariance: −1 «not provided», zeros «unknown» · ours: σ² of each `<noise>` on the diagonal (measured) — later

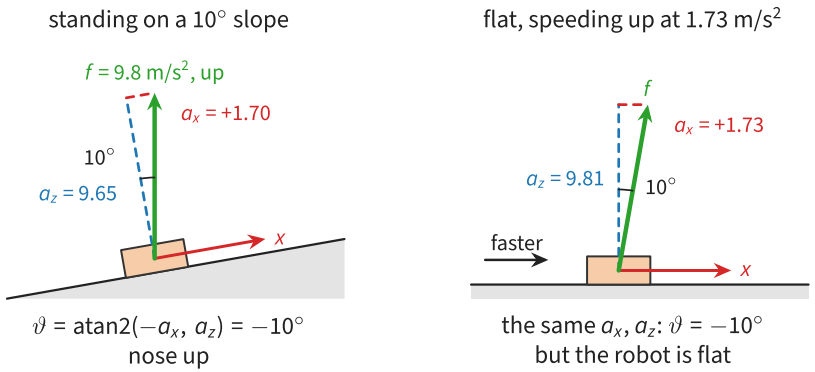


Our drawing, numbers from our Gazebo; axes coloured as in RViz.

USING AN IMU: ROLL AND PITCH FROM GRAVITY, YAW FROM TURN RATE

$$\phi = \operatorname{atan2}(a_y,\ a_z) \qquad \vartheta = \operatorname{atan2}\left(-a_x,\ \sqrt{a_y^2 + a_z^2}\right)$$

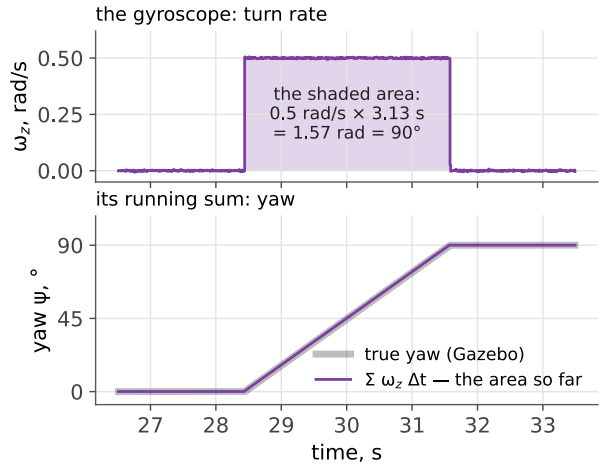
Roll and pitch: which way gravity points in the chip — is the robot on a slope?



Our drawing: a slope and speeding up — the same reading.

$$\psi_k = \psi_{k-1} + \omega_z\,\Delta t$$

Yaw — where the robot faces: add up the turn rate, step by step



Measured in our setup: a 90° turn at 0.5 rad/s; IMU at 100 Hz.

LEGEND

a_x, a_y, a_z — accelerometer: the push it feels, m/s²

f — that push as one arrow; at rest the floor's, 9.8 m/s² up

ϕ — roll, ϑ — pitch, rad

ω_z — turn rate about z, rad/s

ψ — yaw, rad

Δt — time between readings, s

$\operatorname{atan2}(y, x)$ — the angle of the vector (x, y), from -180° to 180°

adding up rate × time — integration: the area under the turn rate

- Moving, the robot's own acceleration adds to gravity — one IMU cannot tell a slope from speeding up · later: sensor fusion

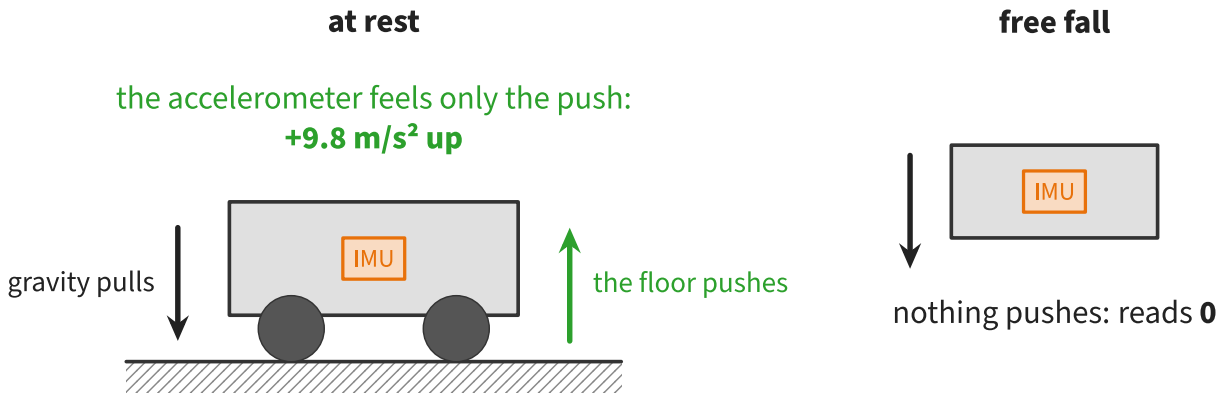
github.com/MOGI-ROS/Week-5-6-Gazebo-sensors

THE ROBOT STANDS STILL. THE ACCELEROMETER READS +9.8 M/S² ON Z. IS IT ACCELERATING?

ANSWER: IT MEASURES SPECIFIC FORCE, NOT ACCELERATION

```
$ ros2 topic echo /imu --field linear_acceleration --once
x: 0.01918240520398151
y: 0.005574458062445393
z: 9.780245543463543
```

- Not exactly 0 and 9.8: we gave our IMU noise
- The floor pushes the robot up — the accelerometer feels that push; in free fall it reads 0
- Before integrating for speed, subtract gravity — later, sensor fusion



Our drawing: what the accelerometer feels.

THE LIDAR

LIDAR: TIME OF FLIGHT (TOF)

- **Lidar** (light detection and ranging): send a light pulse, time its echo — the **time of flight**

$$d = \frac{c\,\Delta t}{2}$$

The distance from the echo's delay — there and back, so half

- Our wall 2.0 m away: the echo is back after 13.3 ns; 1 cm farther — 67 ps later
- Every timing error is a distance error — the lidar's noise

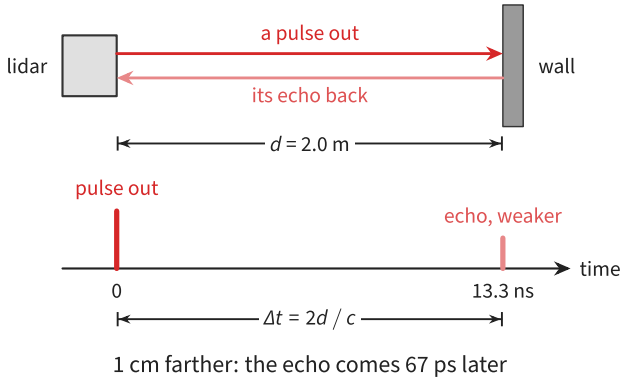
LEGEND

d — distance, m

c — speed of light, $3 \cdot 10^8$ m/s

Δt — echo delay, s

1 ns = 10^{-9} s · 1 ps = 10^{-12} s



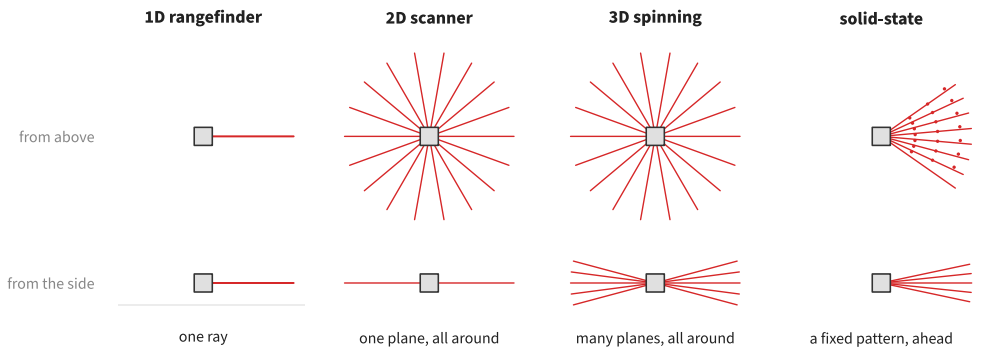
Our drawing: our wall in space and in time.

youtu.be/eJZXRncGaGM

KINDS OF LIDAR: 1D, 2D, 3D, SOLID-STATE

KIND	BEAMS	COVERS	USED FOR	IN GAZEBO
1D rangefinder	one	one direction	distance to the ground, a bumper	`gpu_lidar`, one sample at 0 rad (measured)
2D scanner	one plane of rays	a circle or a sector	indoor SLAM and navigation	**ours: 360 rays, one plane**
3D spinning	16–128 planes	all around, a band up and down	cars, outdoor mapping	`<vertical>` samples above 1
solid-state	a fixed pattern, nothing spins	a window in front	cars, drones	no pattern; a narrow `gpu_lidar` comes close

- The specs: **range** · **angular resolution** · **rate** · **field of view (FOV)** — ours: 0.12–8 m · 1° · 10 Hz · 360°
- Gazebo has one lidar sensor for all four kinds, `gpu_lidar`: its samples and angles set the kind · no sonar, no radar

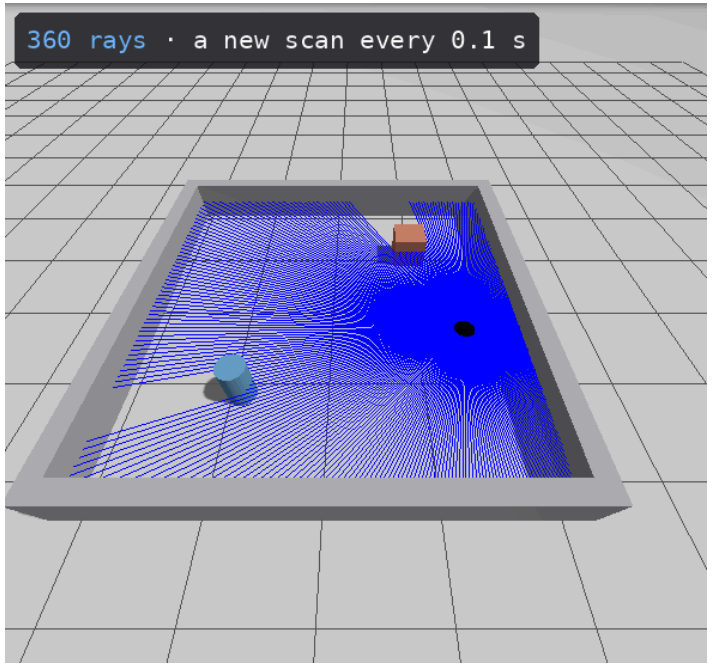


Our drawing: what each kind covers.

📖 emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

ONE SCAN: A FAN OF RAYS

- A spinning mirror sends one ray per angle step — one turn is one scan
- Ours: 360 rays, about 10 scans per second (measured)
- Gazebo can draw the rays: add the **Visualize Lidar** plugin to its window, press ↻, pick `/scan` and «Rays»

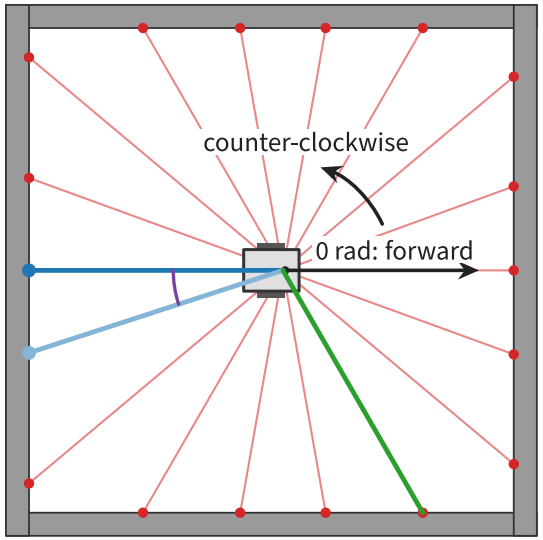


Measured in our setup: Visualize Lidar, one loop at 0.3 m/s.

gazebosim.org/docs/harmonic/sensors

`/scan`: WHAT IS STORED

```
$ ros2 topic info /scan
Type: sensor_msgs/msg/LaserScan
Publisher count: 1
$ ros2 interface show sensor_msgs/msg/LaserScan   # comments shortened
# Single scan from a planar laser range-finder
std_msgs/Header header # acquisition time of the first ray
float32 angle_min       # start angle of the scan [rad]
float32 angle_max       # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]
float32 time_increment  # time between measurements [seconds]
float32 scan_time       # time between scans [seconds]
float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]
float32[] ranges        # range data [m]
                        # (values < range_min or > range_max: discard)
float32[] intensities   # intensity data [device-specific units]
```



Our drawing: the fields over our room; rays every 20°, really every 1°.

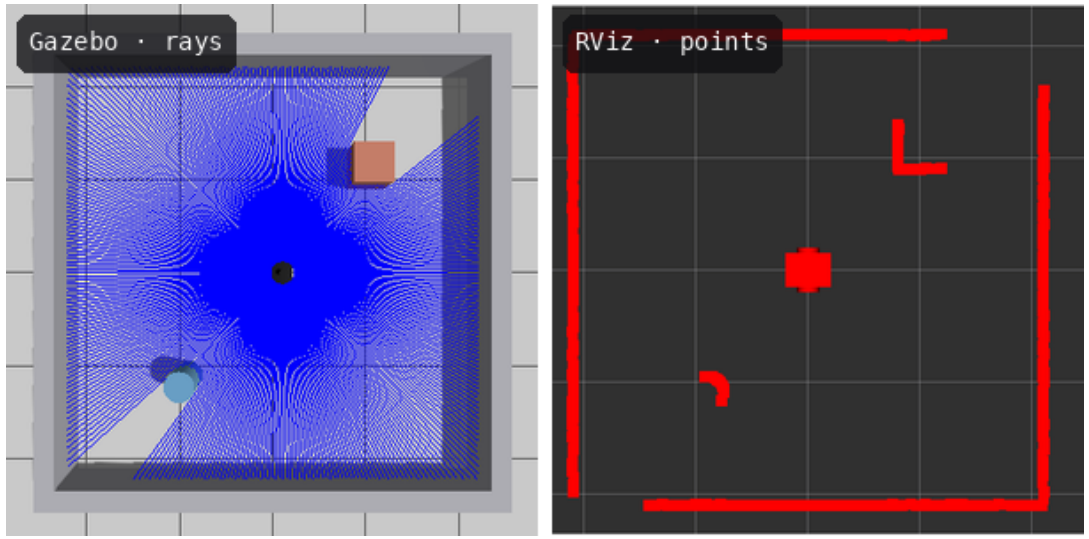
- —— `ranges[0]`: the first ray, at `angle_min` = −π — behind
- —— `ranges[1]`: the next ray, one `angle_increment` on (drawn wider)
- —— `ranges[i]`: how far ray i went, in metres
- Outside `range_min` … `range_max`: discard; nothing hit → `inf` · `intensities`: 360 zeros in our Gazebo

📖 github.com/ros2/common_interfaces/blob/jazzy/sensor_msgs/msg/LaserScan.msg

`ros2 topic echo /scan`: ONE REAL SCAN

```
$ ros2 topic echo /scan --once --truncate-length 4
header:
  stamp: {sec: 7, nanosec: 400000000}
  frame_id: lidar_link
angle_min: -3.141590118408203
angle_max: 3.141590118408203
angle_increment: 0.017501894384622574
time_increment: 0.0
scan_time: 0.0
range_min: 0.11999999731779099
range_max: 8.0
ranges: [2.2001, 2.2007, 2.2017, 2.2034, '...']
intensities: [0.0, 0.0, 0.0, 0.0, '...']
```

- `time_increment` and `scan_time` are 0: Gazebo does not pass them, so the bridge writes zeros
- Shortened: the header and the lists on one line, numbers to 4 decimals
- `ranges` starts behind the robot: the back wall, 2.2 m from the lidar, then round to the front



Measured in our setup, no noise. Left, Gazebo: the rays · right, RViz: where they stop (LaserScan Size 0.1).

🎯 github.com/gazebosim/ros_gz/blob/jazzy/ros_gz_bridge/README.md

WHICH RAY POINTS FORWARD?

$$i = \frac{\theta - \theta_{min}}{\Delta\theta}$$

Count the steps from the first ray to the angle you want

- **What for:** the ray that looks at the wall ahead — later, a filter stops the robot with it
- 360 rays over ±π: no ray at exactly 0° — rays 179 and 180 sit at -0.50° and +0.50°; round i to the nearest

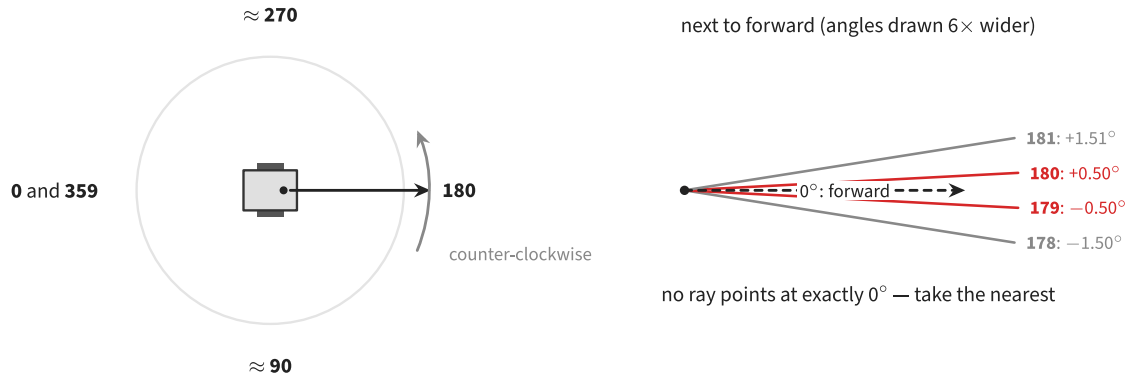
LEGEND

i — index in `ranges[]`

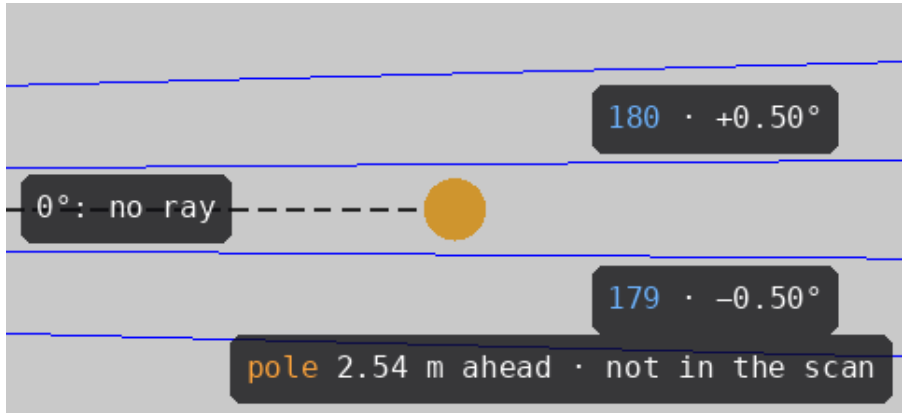
θ — the angle you want, rad: 0 forward, counter-clockwise positive

θ_{min} — `angle_min`, the angle of `ranges[0]`

$\Delta\theta$ — `angle_increment`, rad



Our drawing, angles from our `/scan`.

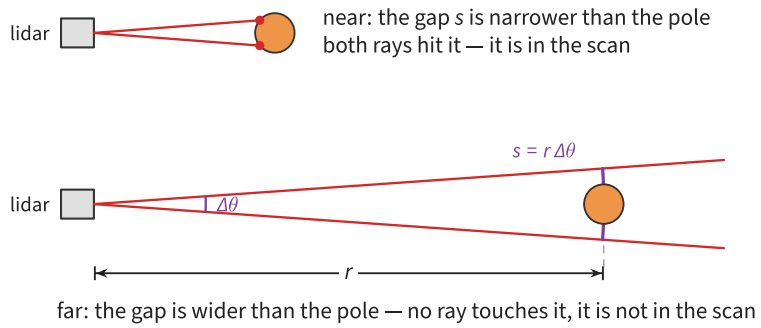


Gazebo, measured: our 3 cm pole straight ahead, between rays 179 and 180.

GAPS BETWEEN RAYS: R · ΔΘ

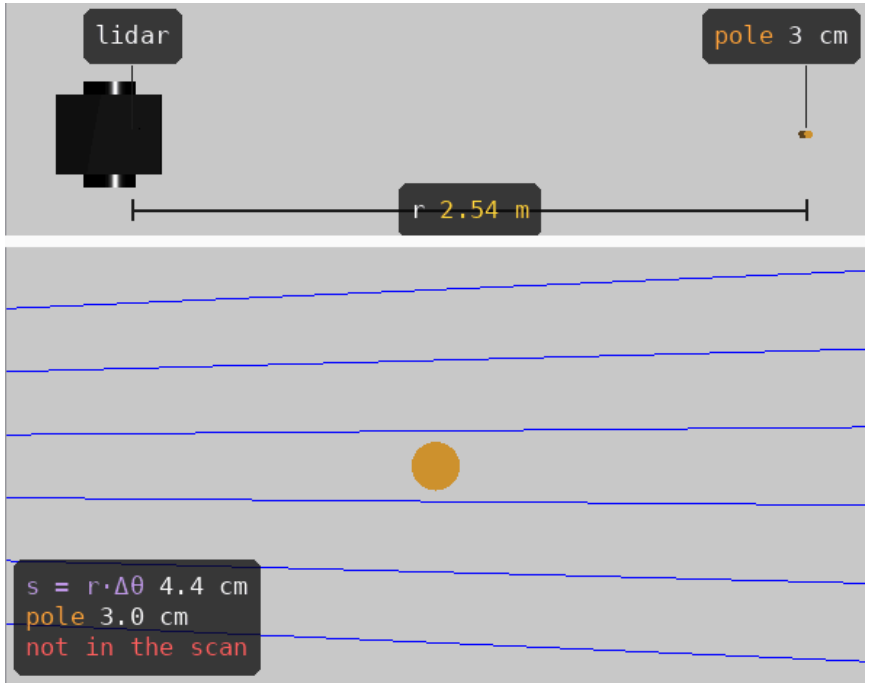
$$s = r\,\Delta\theta$$

How far apart two neighbouring rays land at distance r — an object thinner than s can fall between them



Our drawing: the same pole near and far; Δθ drawn as 8°, ours is 1°.

- 1° at 3 m → s = 5 cm: a chair leg slips through — later, the costmap misses it too
- Our 3 cm pole: in the scan up to 1.91 m, gone from 1.94 m



Gazebo, measured. Top: only the rays that hit · bottom: over the pole, every ray.

LEGEND

s — the gap between two neighbouring hits, m

r — distance from the lidar, m

$\Delta\theta$ — `angle_increment`: the angle between two rays, rad

an arc = radius × angle in radians

1° = π/180 ≈ 0.0175 rad

THE LIDAR IN URDF: THE `gpu_lidar` BLOCK

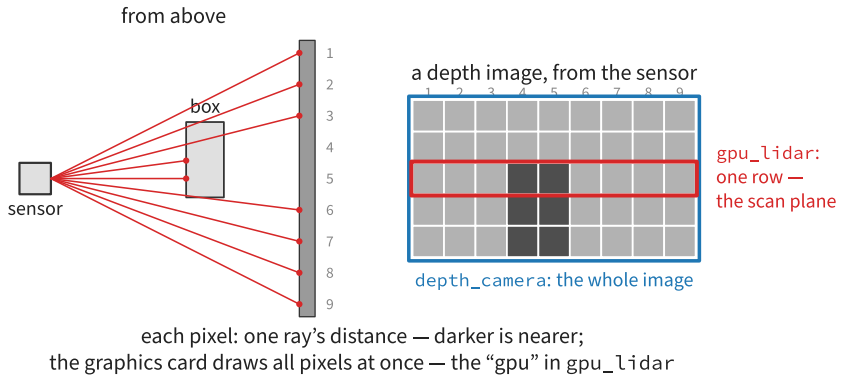
URDF · inside `<gazebo reference="lidar_link">`

```
<sensor name="lidar" type="gpu_lidar">
  <topic>…</topic>
  <gz_frame_id>lidar_link</gz_frame_id>
  <update_rate>…</update_rate>
  <lidar>
    <scan><horizontal>
      <samples>…</samples>
      <min_angle>…</min_angle>
      <max_angle>…</max_angle>
    </horizontal></scan>
    <range><min>…</min><max>…</max></range>
    <noise>…</noise>
  </lidar>
</sensor>
```

- `<gz_frame_id>`: the scan's frame
- `type="gpu_lidar"`: why «gpu» — next
- The fields, one by one: the tutorial below

gazebosim.org/docs/harmonic/sensors

WHY «GPU»: A SCAN IS ONE ROW OF A DEPTH IMAGE



Our drawing: one picture of distances — the lidar keeps a row, the depth camera all of it.

`type` IN URDF	WHAT IT DOES	IN ROS 2	OUR HARMONIC (MEASURED)	WHERE IT WORKS
`gpu_lidar` (old name `gpu_ray`)	keeps one row of the picture — the scan plane	`LaserScan`	works — ours	every new Gazebo; Classic as `gpu_ray`
`depth_camera`	keeps the whole picture	`Image`	works	every Gazebo
`lidar`, `ray`	casts rays one by one on the processor	—	«not supported yet»	Classic; new Gazebo only from Jetty
`sonar`	an ultrasonic cone	—	«not supported yet»	Classic only

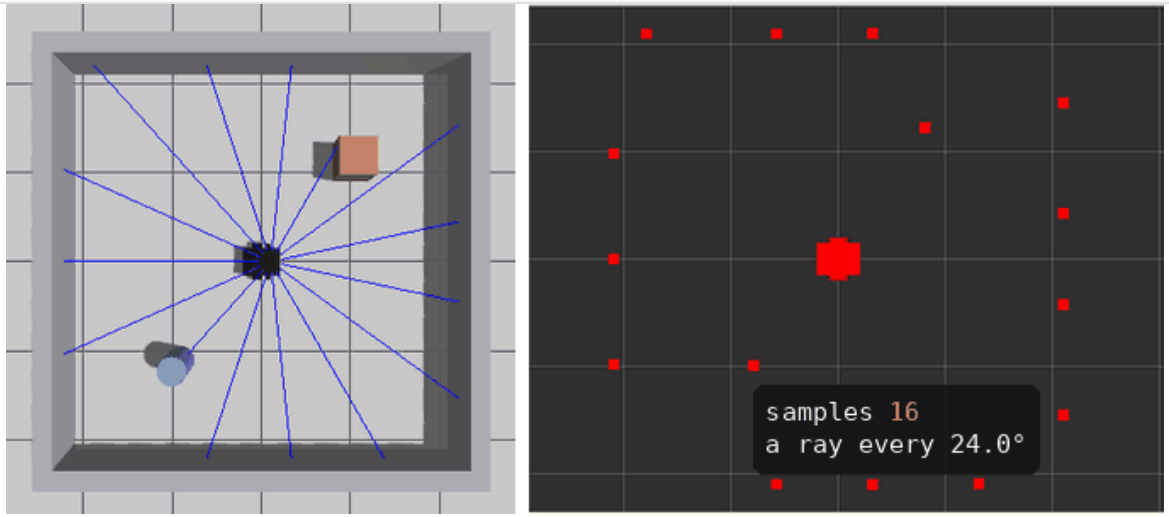
- **Why «gpu»:** a graphics card draws this picture at once; ours has none — the processor draws it, slower, same numbers
- **Why not in Harmonic:** the new Gazebo builds every range sensor on this picture; the processor lidar is back in Jetty

📖 gazebosim.org/docs/all/comparison/ · gazebosim.org/api/sensors/10/classgz_1_1sensors_1_1CpuLidarSensor.html

LIDAR PARAMETERS: WHAT EACH ONE CHANGES

PARAMETER	WHAT IT CHANGES	WHAT YOU SEE
`samples`	rays per scan	more rays, denser points on the walls
`min_angle` , `max_angle`	the field of view	a sector instead of a circle
`range` `min` , `max`	too close, too far	past `max` : `inf` , no point
`update_rate`	scans per second	more CPU; in a heavy world the real-time factor (RTF) drops below 1

- 16 samples over the full circle: the first and the last ray both point backwards — 15 steps of 24°
- Later: SLAM caps its own range at the scan's `range_max` — 20 m in its settings, 8 m from ours, with a warning



Measured in our setup: `samples` 16, 90, 360 — left Gazebo's Visualize Lidar, right RViz (Size 0.1).

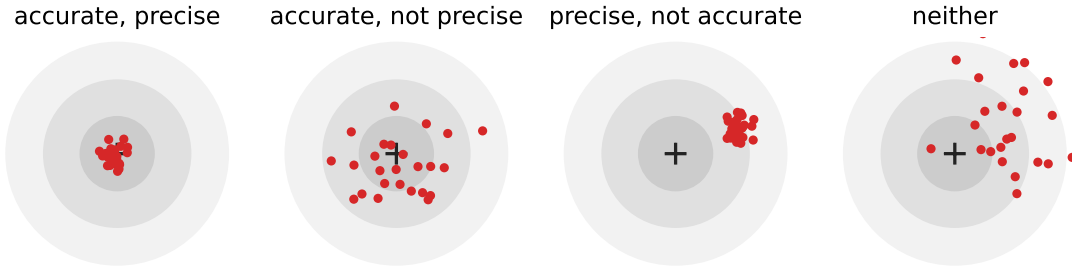
youtu.be/gEAdcgM7cCk

NOISE

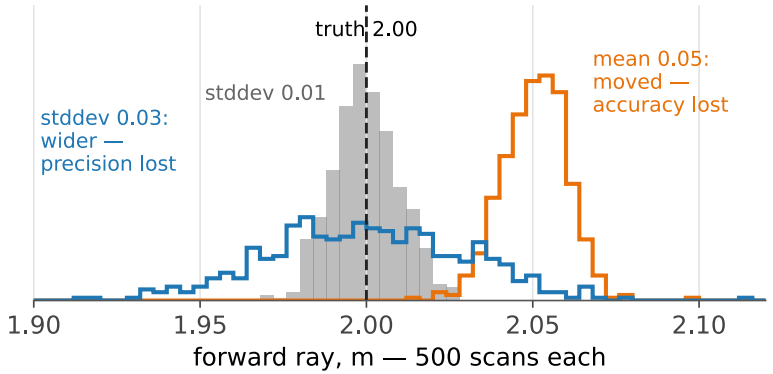
ACCURACY VS PRECISION: BIAS AND SPREAD

	ACCURACY	PRECISION
asks	how close is the *average* to the truth?	how close are the readings to *each other*?
spoiled by	a bias: every reading off by the same	noise: every reading off differently
fixed by	calibration	averaging, filtering (later)
LDS-01 datasheet, 0.5–3.5 m	±5.0 %	±3.5 %

- Odometry drift had both: a wrong wheel base — a bias; slip — noise



Our plot, 25 seeded shots per target.



Measured in our setup: the forward ray, 500 scans per run.

📖 emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

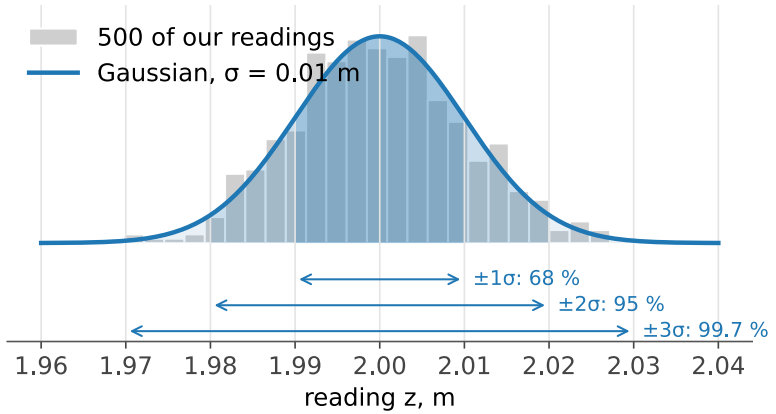
NOISE MODEL: THE GAUSSIAN

$$z = d + v, \qquad v \sim \mathcal{N}(\mu,\ \sigma^2)$$

A reading is the truth plus an error — our wall with σ = 0.01: d = 2.0 m, z = 2.002, 1.988, 2.015

$$p(v) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(v-\mu)^2}{2\sigma^2}}$$

How likely each error is — σ is the one number Gazebo asks for, and the one a Kalman filter needs later



Measured in our setup: ray 180, 500 scans, `<stddev>` 0.01.

LEGEND

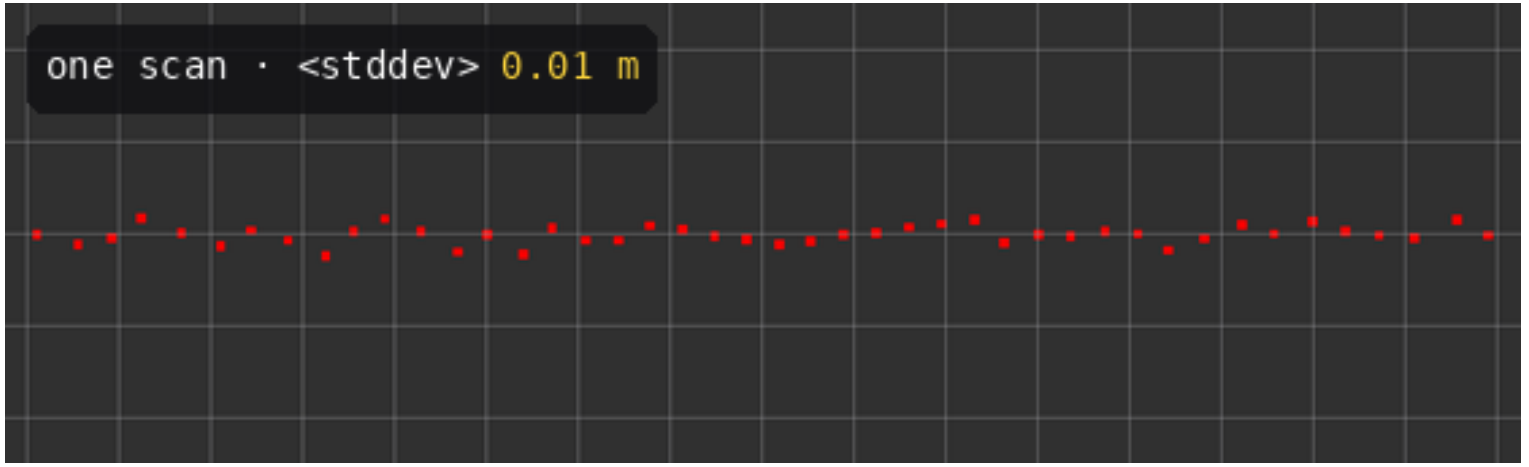
z — a reading, m
d — the true distance, m
v — the error, m
μ — its mean: the bias, m
σ — its spread, m: the square root of the variance σ^2, the average squared distance from the mean
$\mathcal{N}(\mu, \sigma^2)$ — the Gaussian, or normal, distribution
$p(v)$ — the Gaussian's density: how likely an error v is — the bell on the plot
±σ, ±2σ, ±3σ hold 68, 95, 99.7 % of readings

TUNING NOISE IN GAZEBO: THE `<noise>` BLOCK

URDF · inside the lidar's `<lidar>`

```
<noise>
  <type>gaussian</type>
  <mean>0.0</mean>
  <stddev>0.01</stddev>
</noise>
```

1. Pick σ from the datasheet
2. Write it into `<noise>`
3. Relaunch: the robot description is read once, at spawn
4. Measure σ with the robot standing still
5. Compare with what you wrote; adjust

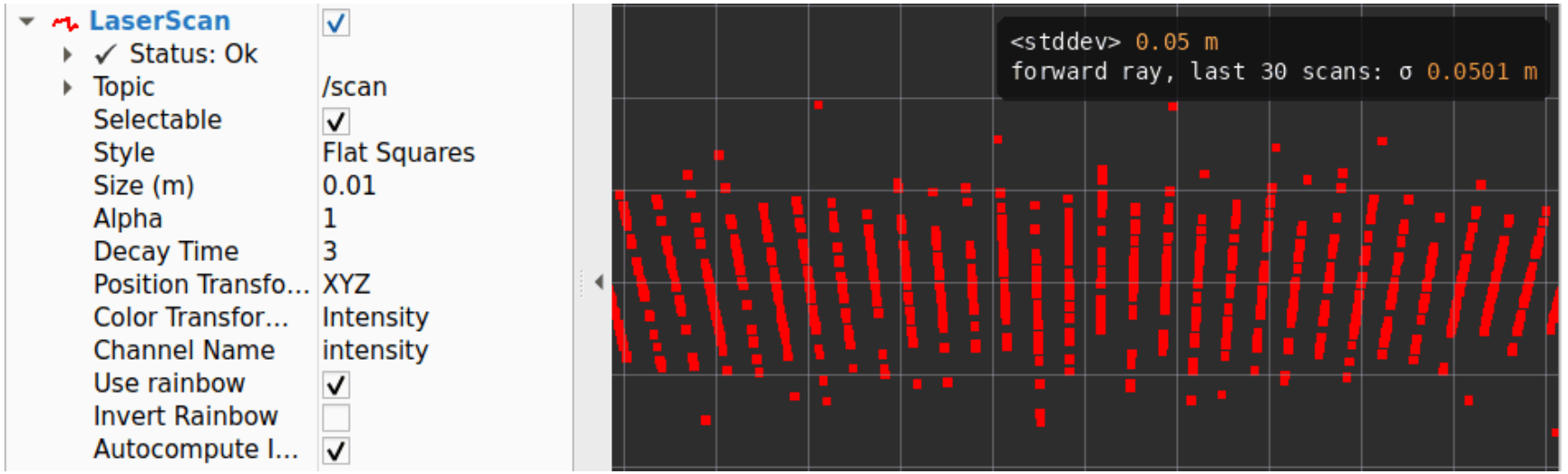


RViz, measured in our setup: one scan; Grid 0.1 m in `lidar_link` — its middle line is the wall's face, 2.0 m away.

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/sensors/setup_sensors_gz/

SEEING NOISE: RVIZ DECAY TIME

- One scan hardly shows noise: every ray gets a new random error, every scan
- **Decay Time** keeps old scans on the screen: 3 s means 30 scans on top of each other
- **How it is shown:** RViz looks down at the wall ahead — the middle grid line is the wall's face; a ray's 30 readings scatter along the ray, so each ray draws a streak across the wall
- σ = 0: a thin line · σ = 0.01 m: a band ≈ 4 cm, 95 % within ±2σ · σ = 0.05 m: ≈ 20 cm



RViz, measured in our setup: 30 scans on top of each other; Grid 0.1 m in `lidar_link` — its middle line is the wall's face.

📖 docs.ros.org/en/jazzy/Tutorials/Intermediate/RViz/RViz-User-Guide/RViz-User-Guide.html

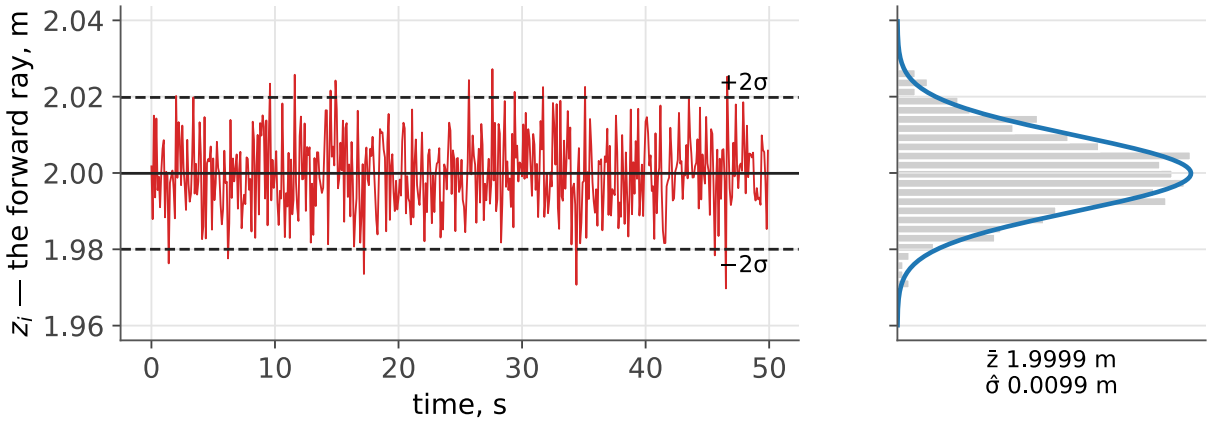
SEEING NOISE: ONE RAY IN NUMBERS

- Keep the forward ray from 500 scans (50 s) → its mean and σ

$$\hat{\sigma} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(z_i - \bar{z})^2}$$

The spread you measured — compare it with the `<stddev>` *you set*

- Our runs, `<stddev>` 0.005–0.05 m: the measured σ within 5 %, the mean 2.0 m every time — no bias



Measured in our setup: the forward ray, 500 scans, σ 0.01.

LEGEND

z_i — the i-th reading, m

$\bar{z}$ — their mean, $(z_1 + \cdots + z_N) / N$, m

N — readings

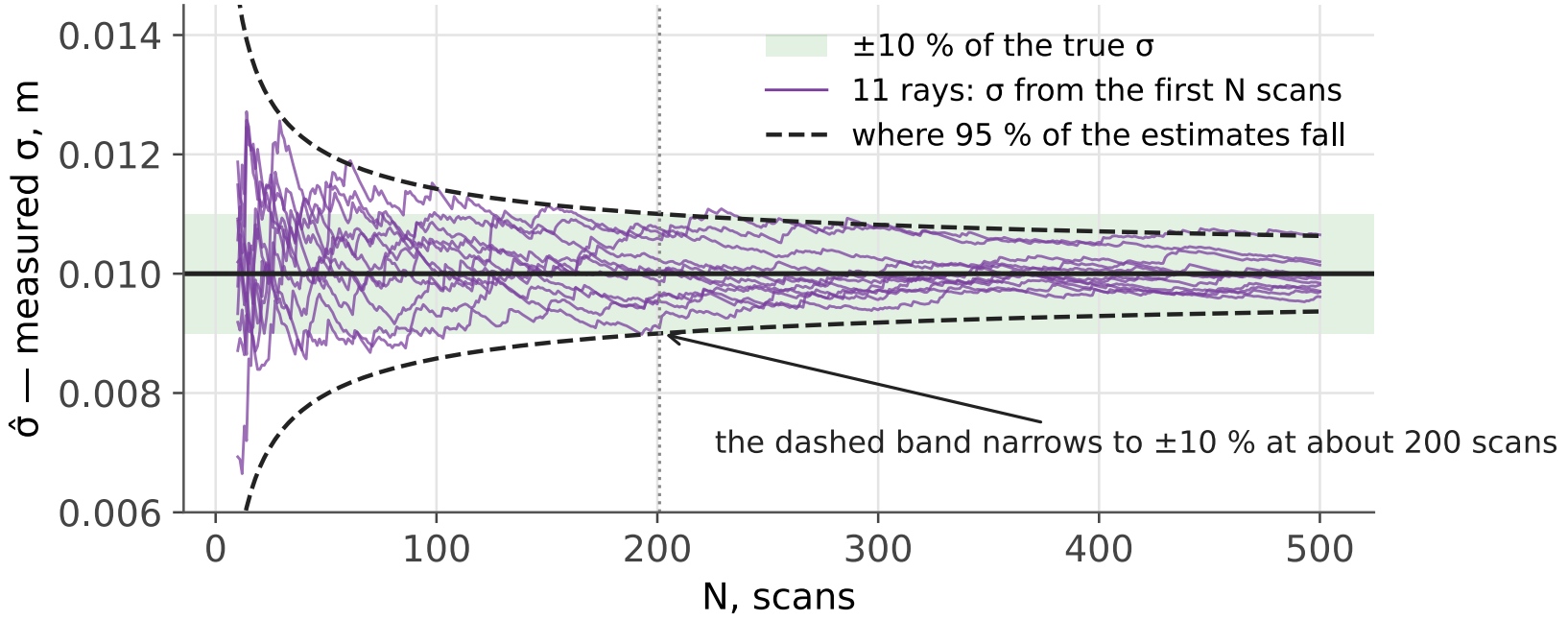
$\hat{\sigma}$ — the measured spread, m

$N - 1$, not N: $\bar{z}$ came from the same readings

a histogram — how many readings fall into each small interval

HOW MANY SCANS ARE ENOUGH?

- The measured σ wanders for a few scans and settles as N grows — like any average: the more readings, the steadier
- The dashed band shows how far it can still be off; it narrows to ±10 % at about 200 scans — 20 s at 10 Hz
- ±5 % takes about four times as many: the error shrinks only with the square root of N



Measured in our setup: 11 rays × 500 scans, `<stddev>` 0.01; dashed — where 95 % of the estimates fall.

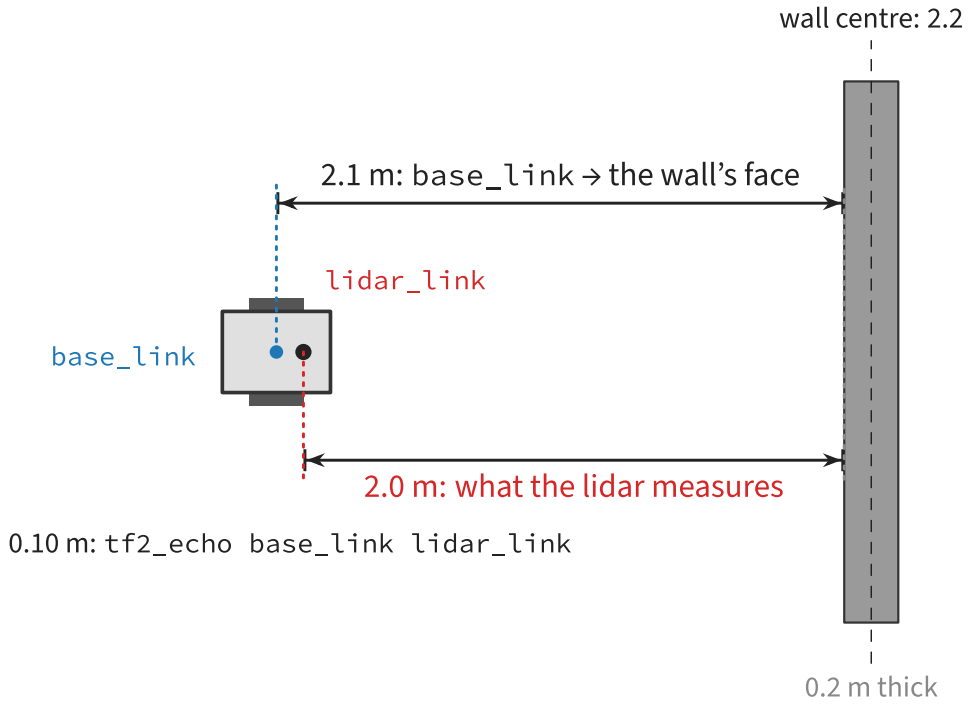
THE MEAN IS 2.0 M. IN THE WORLD THE WALL'S FACE IS AT 2.1 M. IS THE LIDAR BIASED?

ANSWER: MEASURE FROM THE LIDAR, NOT FROM THE ROBOT

- In the world file the wall's centre is at x = 2.2 m and it is 0.2 m thick → its face is at 2.1 m
- The lidar sits 0.1 m ahead of `base_link`:

```
$ ros2 run tf2_ros tf2_echo base_link lidar_link
- Translation: [0.100, 0.000, 0.150]
```

- Expected: 2.1 − 0.1 = **2.0 m** — no bias



Our drawing, numbers from the world file and the URDF.

BIAS: `<mean>` MOVES THE WALL

- `<mean>` ≠ 0 adds the same error to every reading — the scan leaves the wall
- More scans will not remove it: a bias does not average out
- Calibration removes it; a filter can only if it estimates the bias too — later, the gyroscope's bias in sensor fusion

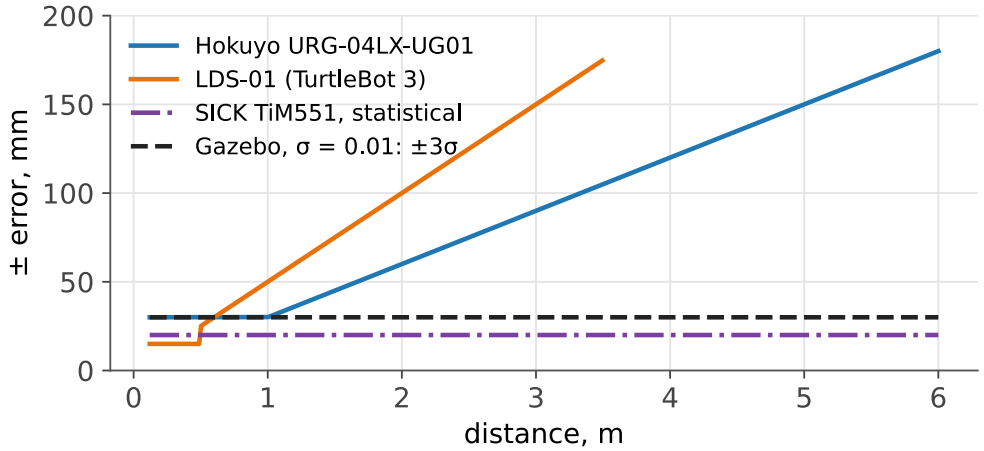


RViz, measured in our setup: `<stddev>` 0.01, Decay Time 1 s; Grid 0.1 m in `lidar_link` — the face on its middle line, 0.05 m is half a cell.

CHOOSING Σ: READ THE DATASHEET

LIDAR	ACCURACY AS PRINTED	HOW TO READ IT	Σ TO SET
Hokuyo URG-04LX-UG01	±30 mm (0.06–1 m), ±3 % beyond	Gazebo's own example: ±30 mm = 3σ	0.01 m
LDS-01 (TurtleBot 3)	accuracy ±15 mm, precision ±10 mm (0.12–0.5 m)	precision as 3σ, by the rule below	≈ 0.003 m
SICK TiM551	systematic ±60 mm, statistical 20 mm	systematic → a bias; statistical → a spread, but σ or 3σ? The sheet does not say	measure it

- «±30 mm» with no note: read it as 3σ — 99.7 % of readings within ±30 mm, so σ = 0.01 m, as Gazebo's own example does
- An error in % of the distance: σ at your working distance — Gazebo has one σ for every distance

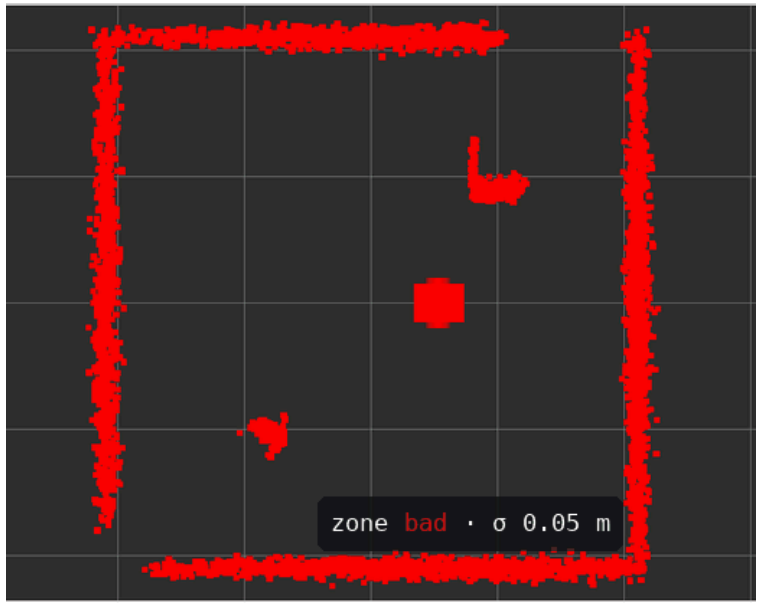


Our plot of the three datasheets' printed accuracy.

📖 gazebosim.org/docs/harmonic/spawn_urdf

NOISE THAT VARIES BY PLACE AND TIME

- Gazebo's `<noise>` has one σ: the same everywhere, all the time
- A bad zone or a sudden failure: put a small node between Gazebo and your code — it reads `/scan`, adds noise by the robot's position or by the time, and publishes the same `LaserScan`
- The same message type in and out: your code cannot tell — like the keyboard driving any robot in the first lab
- Built into Gazebo: a slippery floor patch (`mu`) for the wheels; a bias that drifts with time for the IMU
- Later: SLAM and localization must survive exactly this



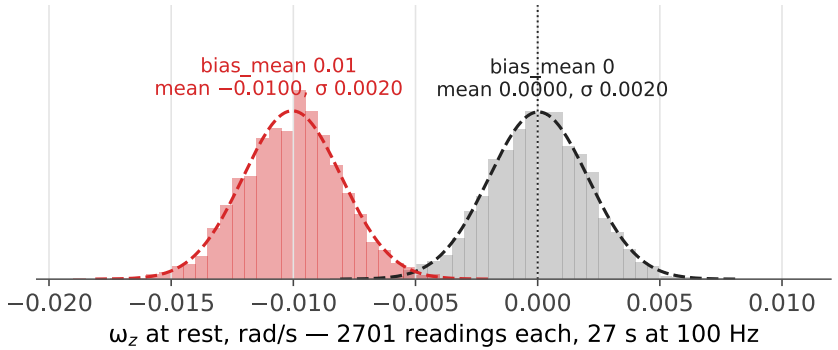
RViz, measured in our setup: `<stddev>` 0.01; right of the middle grid line our node adds noise up to σ 0.05; LaserScan Size 0.05.

IMU NOISE: BIAS, DRIFT, ROUNDING

ELEMENT	MODELS
`stddev`	jitter, new in every reading
`bias_mean`, `bias_stddev`	an offset, drawn once per run — its sign at random: +0.01 in one run, -0.01 in the next (measured)
`dynamic_bias_stddev`, `dynamic_bias_correlation_time`	an offset that wanders slowly — a typical time is 3600 s
`precision`, type `gaussian_quantized`	rounding to steps

URDF · inside the IMU's `<sensor>`

```
<imu>
  <angular_velocity>
    <z>
      <noise type="gaussian">
        <stddev>…</stddev>
        <bias_mean>…</bias_mean>
        …
      </noise>
    </z>
  </angular_velocity>
</imu>
```



Measured in our setup: stddev 0.002 rad/s in both runs; dashed — the Gaussian with each run's mean and σ.

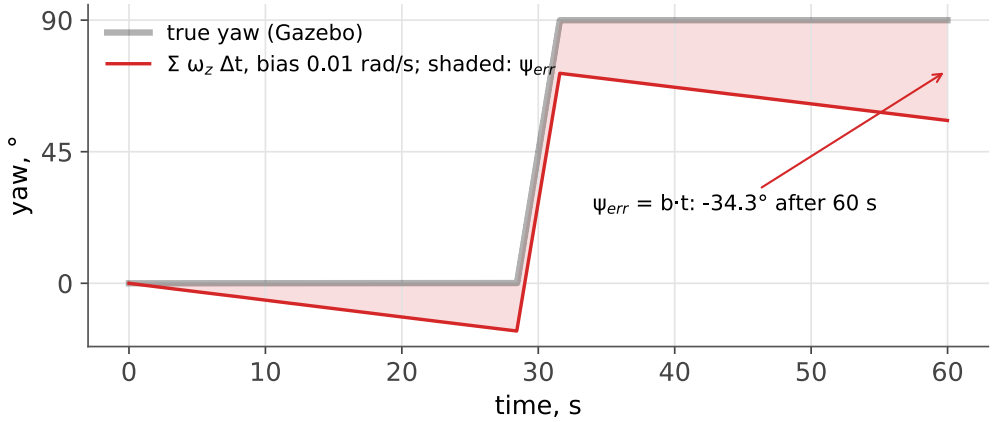
🎯 github.com/gazebosim/ros_gz/blob/jazzy/ros_gz_interfaces/msg/SensorNoise.msg

A BIASED GYRO: YAW CREEPS AWAY

$$\psi_{err}(t) = b\,t$$

How far the integrated yaw is off after t seconds — why a gyroscope alone drifts

- b = 0.01 rad/s → 0.6 rad ≈ 34° in a minute — our run: 34° after 60 s
- An accelerometer integrated twice: the position error grows as t² — seconds, not minutes
- Later: sensor fusion — the wheels and the gyroscope check each other



Measured: the same one-minute IMU run, with bias_mean 0.01 rad/s.

LEGEND

ψ_{err} — yaw error: the integrated yaw minus the true one, rad

b — the gyroscope's bias, rad/s

t — time since the start, s

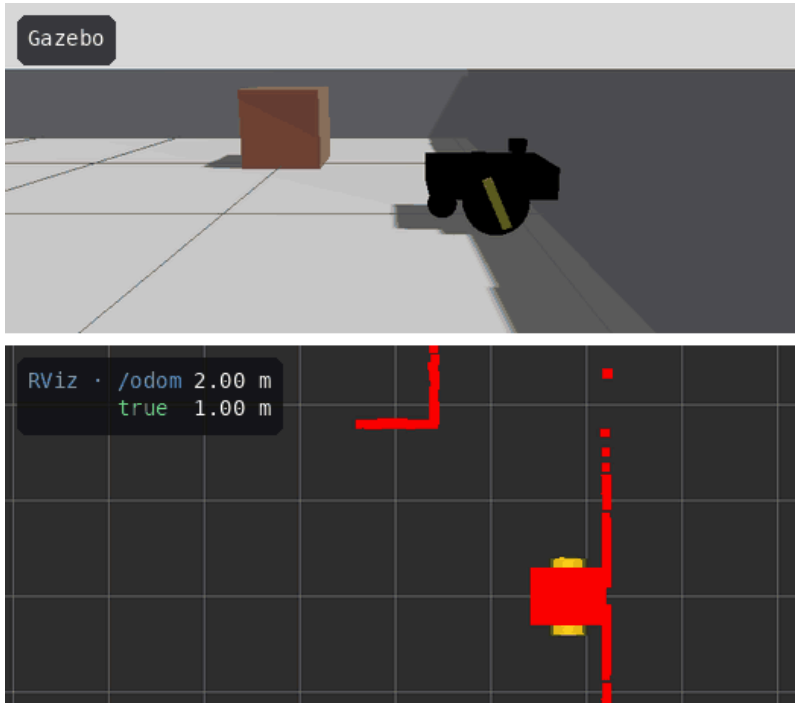
$b\,t$ — a constant integrated once: a straight line

$\frac{1}{2}b\,t^2$ — integrated twice: a parabola

📖 www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf

UNCERTAINTY BEYOND THE SENSOR

MECHANISM	IN GAZEBO	EFFECT
friction	`mu` , `mu2` of a surface	the wheel turns, the robot does not
controlled slip	the `WheelSlip` system	slip as a parameter
moving objects	`<actor>`	later: ghosts in a map
wind	the `WindEffects` system	drones drift
timing	`update_rate`, RTF below 1	fewer, later messages
a noisy pose	`OdometryPublisher` , `<gaussian_noise>`	a noisy «ground truth»
the random seed	`gz sim --seed 7`	the same noise in every run (measured)



Measured in our setup: 0.2 m/s for 10 s — `/odom` 2.0 m, true 1.0 m; RViz draws the robot, and its scan, where `/odom` says.

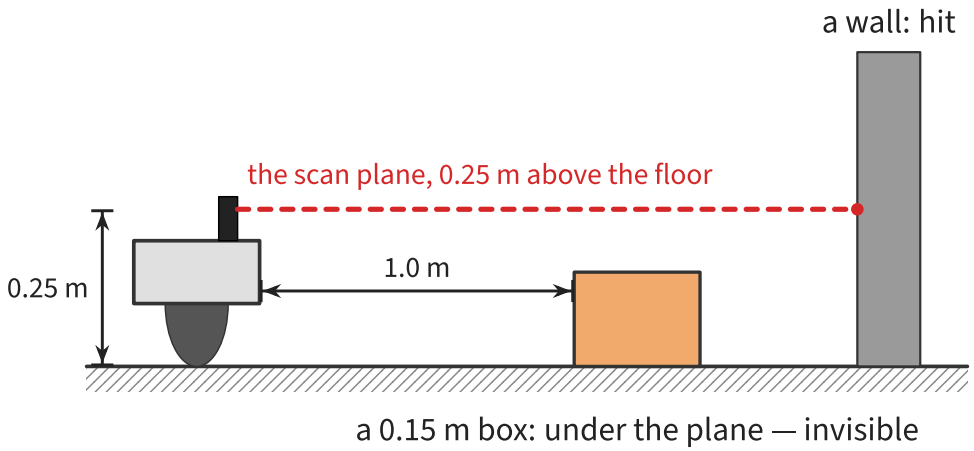
📖 gazebosim.org/docs/harmonic/actors

WHERE THE LIDAR IS BLIND

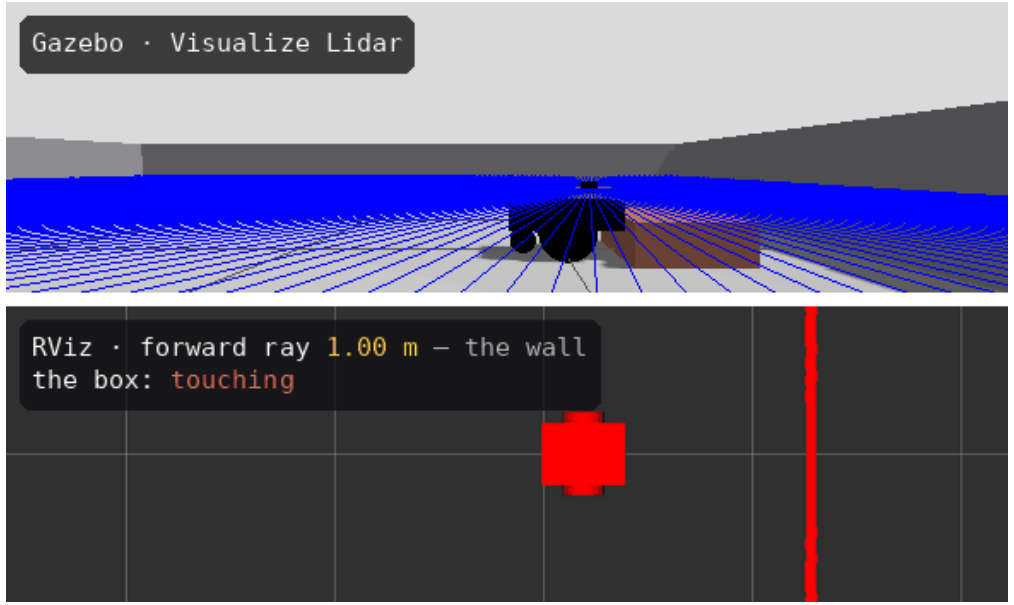
A BOX STANDS IN FRONT OF THE ROBOT. THE SCAN SHOWS NOTHING. THE ROBOT DRIVES INTO IT. WHY?

ANSWER: A 2D LIDAR SEES ONE PLANE

- Our scan plane is 0.25 m above the floor — where `lidar_joint` puts the lidar in the URDF
- Anything lower or higher is invisible: a step, a low box, a table top on legs · later: the navigation costmap misses it too
- What helps, later: a depth camera or a 3D lidar sees below and above the plane; a bumper stops the robot on contact



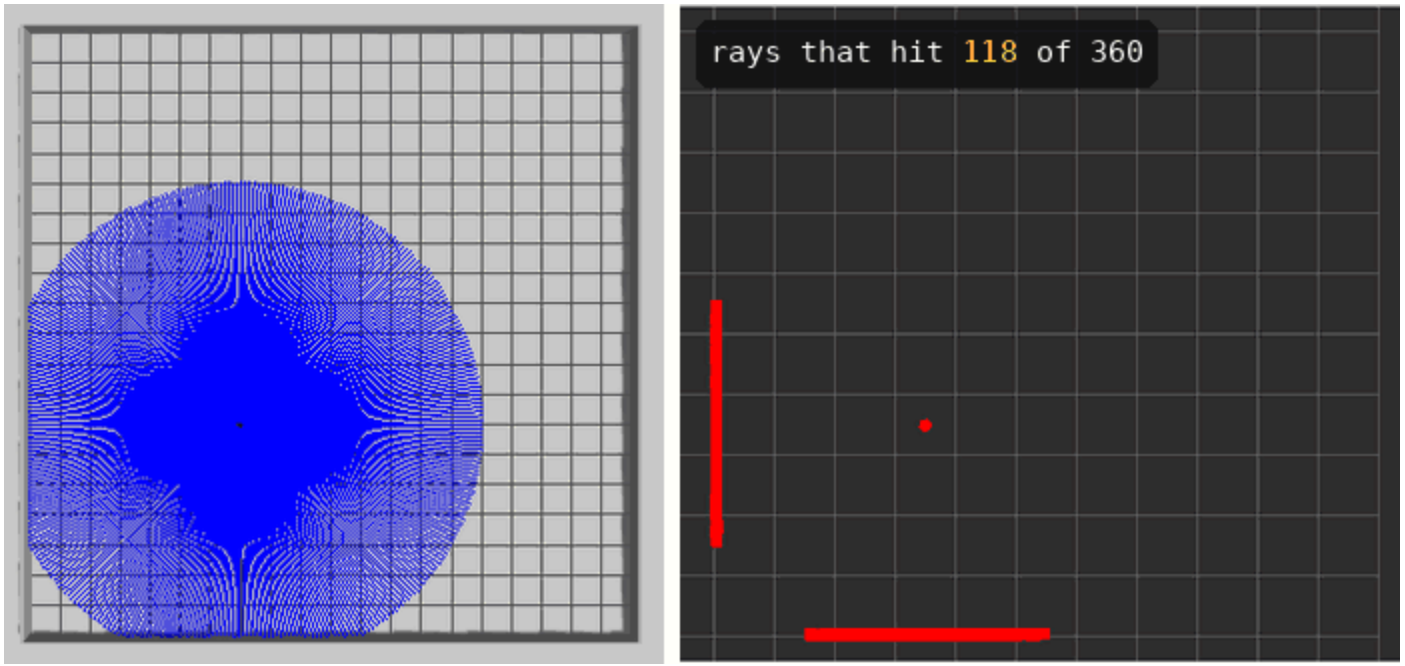
Our drawing: the scan plane from the side.



Measured in our setup: a 0.15 m box 1.0 m ahead, 0.2 m/s; the forward ray reads the wall: 2.0 → 1.0 m.

TOO FAR: A HALL BIGGER THAN `range_max`

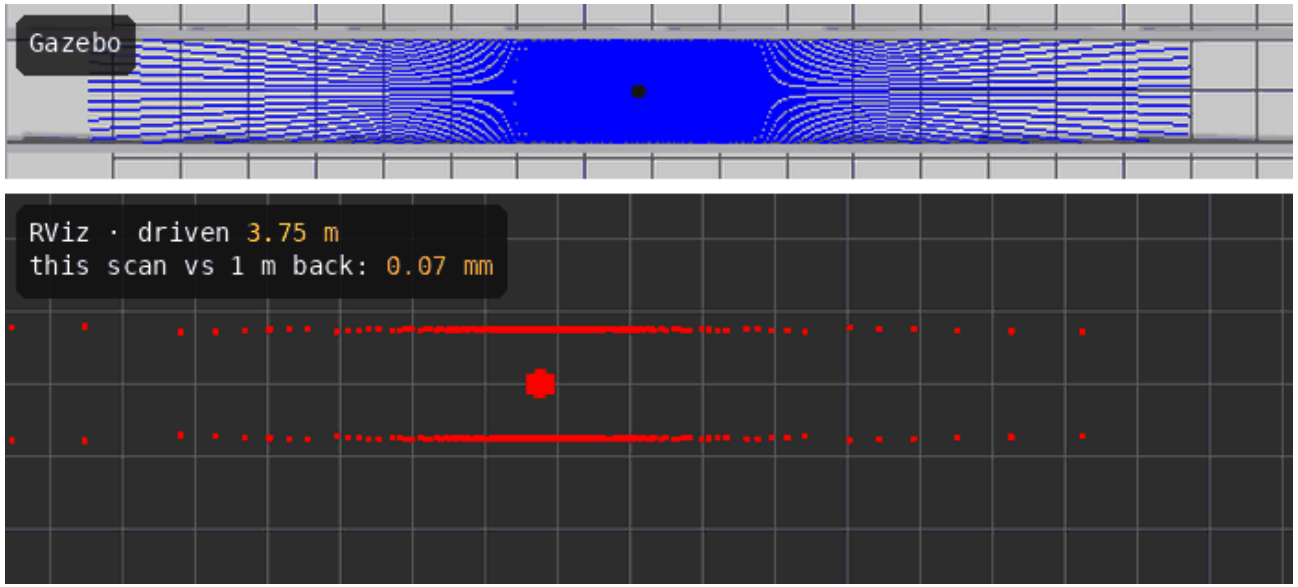
- A ray that hits nothing within 8 m comes back as `inf` — in an empty world all 360 rays, with noise too
- In a 20 × 20 m hall: 226 of 360 rays hit near a corner, none in the middle (measured) · later: SLAM needs walls in range to match one scan to the next
- What helps, later: a lidar with a longer range; odometry and the IMU carry the robot across the empty middle



Measured in our setup: left Gazebo's Visualize Lidar — rays that hit nothing are drawn to the full 8 m; right RViz, grid cells 2 m, LaserScan Size 0.4.

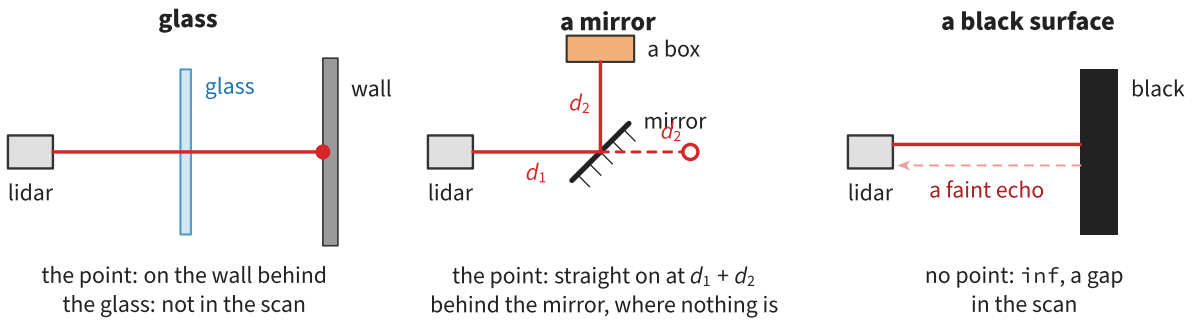
SAME SCAN, DIFFERENT PLACE: THE CORRIDOR

- A long smooth corridor looks the same every metre: two scans 1 m apart differ by at most 0.12 mm (measured)
- Later: SLAM slides along such a corridor; in two rooms that look alike it may close a loop in the wrong place
- What helps, later: odometry and the IMU keep counting the metres the scan cannot see; anything that differs along the way — a door, a camera's view



Measured in our setup: 8 m at 0.5 m/s. Top, Gazebo: a fixed camera · bottom, RViz: the view follows the robot, the grid stays in `world`, lines 1 m apart; LaserScan Size 0.08.

OTHER FAILURES — GLASS, MIRRORS, AND THE WIRING



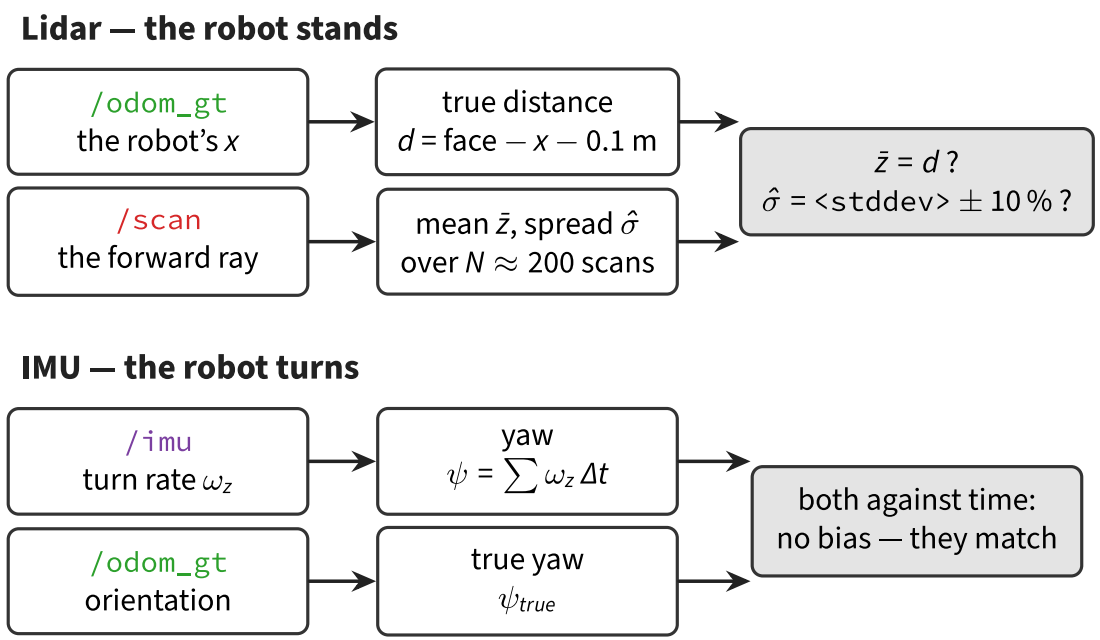
Our drawing, from above: the lidar times the echo and puts the point straight along the ray.

FAILURE	WHAT THE SCAN SHOWS	IN GAZEBO	WHAT HELPS — LATER
glass	the wall behind it — the glass is missing	stops on it, like on a wall (measured)	a sensor that sees glass — ultrasonic, a camera; glass on the map
a mirror	a point straight on, at the whole path's length — where nothing is	not modelled	drop the points that break the map; mirrors on the map
black	nothing: `inf`, a gap	stops on it (measured)	a camera beside the lidar; a stronger laser

- Gazebo draws what is solid: these fail only on a real robot — test there, or add the failure yourself, like the noise zone
- Not the sensor but the wiring: no `use_sim_time`, a wrong `frame_id`, RTF below 1 — a silent topic or «Showing [0] points»; check `ros2 topic hz`, `tf2_echo`, `view_frames`

LAB: YOUR OWN ROBOT — THE TASK

1. **Build your own robot** in URDF: a body and 2 to 4 wheels — the first tutorial below, step by step
2. **Give it a lidar and an IMU**, each with `<noise>`; the lidar's σ from a real datasheet
3. **Turn on ground truth** — the true pose from Gazebo on `/odom_gt`
4. **Play with the noise:** σ = 0, your σ, a gyroscope bias
5. **Check against ground truth** — how: right



Our drawing: what is compared with what.

▶ the robot: github.com/MOGI-ROS/Week-3-4-Gazebo-basics
▶ the lidar and the IMU: github.com/MOGI-ROS/Week-5-6-Gazebo-sensors

LAB: REQUIREMENTS AND BONUS

REQUIREMENTS

- **Build the robot** — a URDF: a body, 2–4 wheels, a lidar and an IMU, each with inertia
- **Add the Gazebo blocks** — DiffDrive (`odom` → `base_link`), friction; the lidar and the IMU with `<gz_frame_id>`
- **Turn on ground truth** — `OdometryPublisher` on `/odom_gt`
- **Create a world** — walls, and the `Imu` system
- **Bridge and launch** — clock, cmd_vel, odom, tf, scan, imu, odom_gt; one launch file starts everything
- **Drive it** — from the command line and the keyboard, as in the first lab
- **Show it in RViz** — the robot, TF, the scan on the walls, the odometry and ground-truth paths
- **Give the lidar noise** — σ from a real datasheet in `<noise>`; the wall with Decay Time, σ = 0 and yours
- **Measure the noise** — the forward ray's mean and σ, the robot standing, enough scans for ±10 %; a histogram
- **Give the gyroscope a bias** — in `<noise>`; the integrated yaw against ground truth
- **Verify** — the mean = the true distance from ground truth; σ within 10 % of `<stddev>`

BONUS

- **Meshes** — the body from a mesh instead of boxes
- **Field of view** — `samples` or the angles: show the effect on the scan
- **σ against distance** — σ measured at a few distances; Gazebo's one σ against your datasheet's %
- **The gyroscope's bias from rest** — estimate it from the first seconds at rest, subtract it: does the yaw follow ground truth?
- **A noise zone** — a node that adds noise by position
- **σ from a bag** — `ros2 bag`, without the simulator

WHAT'S NEXT

Lecture 6 — State Estimation

A noisy sensor and a model of motion · combined into an estimate better than either · the Kalman filter, step by step · the noise you measure in this lab is what it needs

📖 github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python