AUTONOMOUS
INTELLIGENCE GROUP

LECTURE 4 — BUILDING THE ROBOT

A BODY ROS 2 CAN SEE, FRAMES IT CAN TRUST, AND HOW FAR ODOMETRY DRIFTS

Topics: Frames and TF · A robot in URDF · One clock · Measuring drift

TODAY: FRAMES, A ROBOT DESCRIPTION, TIME, DRIFT

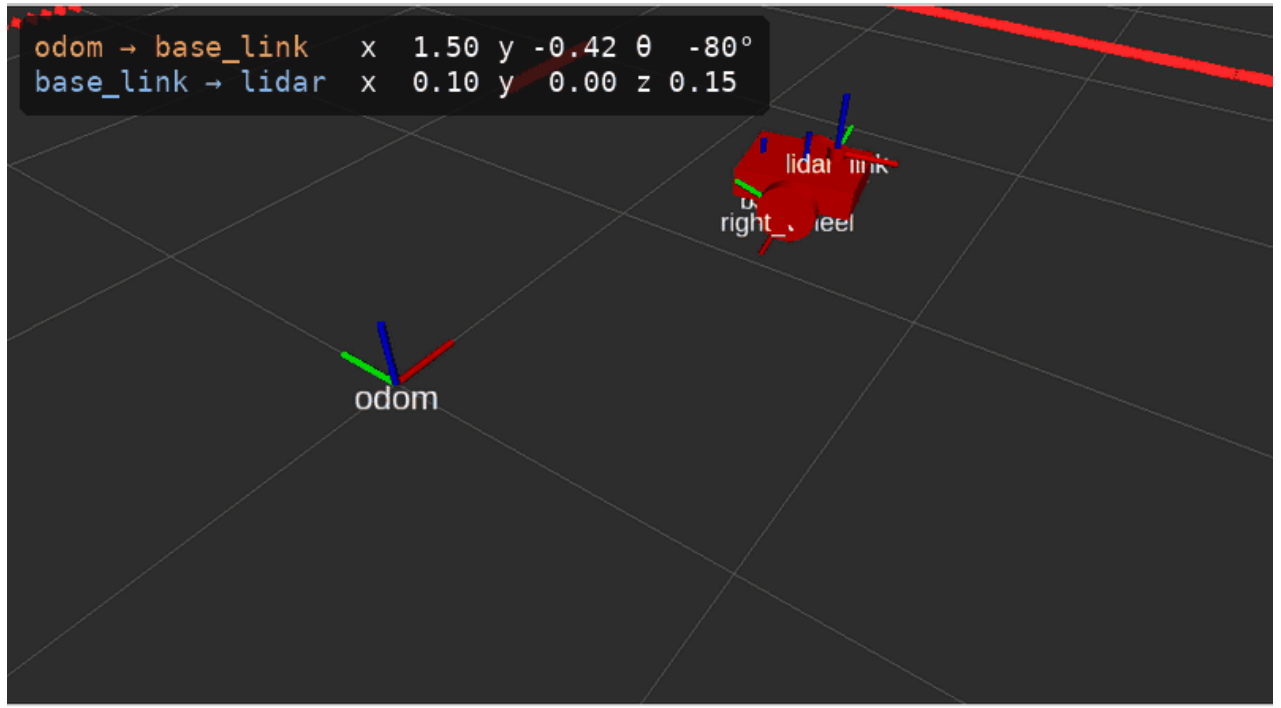
- **Where is each part of the robot?** — frames and TF
- **What does the robot look like to ROS 2?** — URDF
- **Whose clock?** — one for every node
- **How far off is odometry?** — measuring drift

FRAMES: WHERE EVERY MEASUREMENT LIVES

EVERY MEASUREMENT IS MADE SOMEWHERE. TF SAYS WHERE.

TF: WHERE THE SENSOR IS + WHERE THE ROBOT IS

- The lidar says *"a wall 2 m ahead of me"* — the robot needs *"where is it relative to me, and in the room?"*
- **Static:** where the lidar is mounted — never changes
- **Dynamic:** where the robot is in the room — changes as it drives
- **TF** keeps both kinds of transforms, with their times, and combines them on request
- One frame needed no TF: RViz drew the path in `odom` and only warned *No tf data*



One loop, recorded in our setup. The numbers are TF at that moment: `odom → base_link` changes — dynamic; `base_link → lidar_link` does not — static.

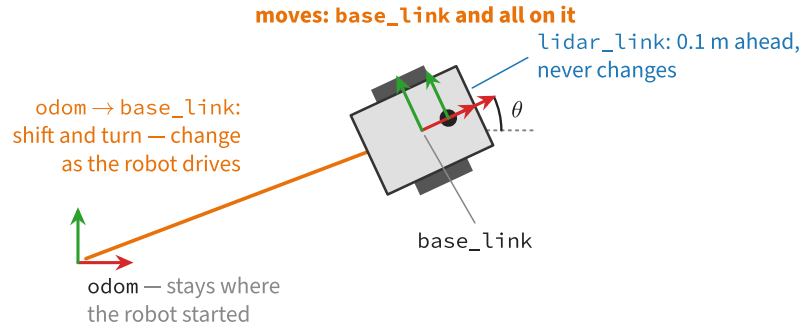
articulatedrobotics.xyz/tutorials/ready-for-ros/tf/

FRAME AND TRANSFORM: A PLACE, AND THE WAY BETWEEN TWO

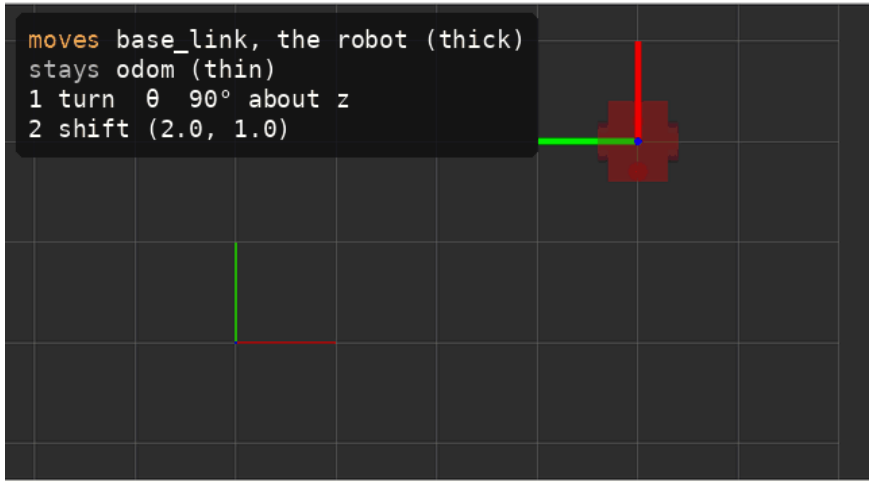
- **Frame:** a named place with three axes, attached to something — a sensor, the body, the floor
- **Transform** from B to A: turn by R, then shift by t. Two on our robot:

TRANSFORM	WHAT MOVES	TURN	SHIFT
`base_link → lidar_link`	nothing — the lidar is bolted to the body	none	0.1 m ahead, 0.15 m up — never changes
`odom → base_link`	the robot, in the room	its heading θ — changes as it turns	its x, y — change as it drives

- **Shift:** nearly every part — nothing sits exactly at `base_link`'s origin
- **Turn:** a part that does not look forward — and the robot itself, as it turns



Our drawing: `odom` stays; `base_link` moves, and the lidar with it.



In our setup: the robot (thick axes) turns, then shifts; `odom` (thin) stays.

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/transformation/setup_transforms/

TRANSFORM FORMULAS: FROM THE ROBOT TO THE ROOM, BY HAND

The robot stands at (2, 1), facing +y (θ = 90°). A point 1 m ahead of it: $p_B = (1, 0, 0)$ in `base_link` (B). Where is the point in the room — in `odom` (A)?

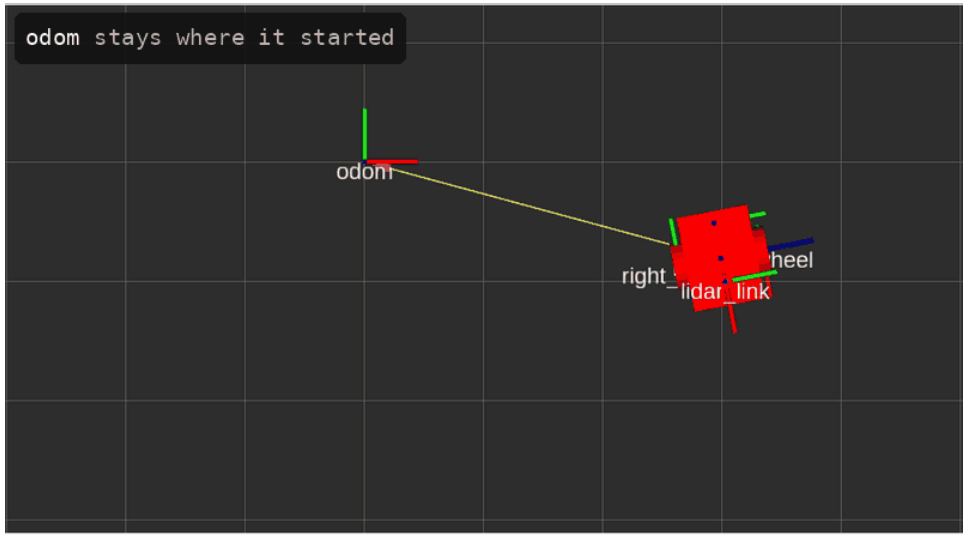
	FORMULA	WHAT IT IS FOR	EXAMPLE: THE ROBOT IN THE ROOM
①	$p_A = R_{AB}\, p_B + t_{AB}$	a point known in B, written in A: turn it, then shift it	$R\,p_B + t = (0,1,0) + (2,1,0) = (2,2,0)$ — the point is at (2, 2) in the room
②	$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	the turn by θ about z — the only kind a robot on the floor needs	$R_z(90^\circ) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ turns the robot's "ahead" $(1,0,0)$ into the room's +y $(0,1,0)$
③	$T_{AB} = \begin{bmatrix} R_{AB} & t_{AB} \\ 0 & 1 \end{bmatrix}$	the turn and the shift packed into one 4×4 matrix	$T_{AB} = \begin{bmatrix} 0 & -1 & 0 & 2 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
④	$\begin{bmatrix} p_A \\ 1 \end{bmatrix} = T_{AB} \begin{bmatrix} p_B \\ 1 \end{bmatrix}$	① as one multiplication — so a chain of transforms is a product of matrices	$T_{AB}\,(1,0,0,1)^\top = (2,2,0,1)$ — the same as ①

LEGEND p_A, p_B — one point in A and in B, m · R_{AB} — the turn from B's axes to A's · t_{AB} — B's origin in A, m · T_{AB} — R and t in one 4×4 · θ — the turn angle, rad · orange: the robot's pose, `odom → base_link`

youtu.be/NGPn9nvLPmg

`odom` AND `base_link`: WHERE THE ROBOT STARTED, WHERE IT IS

FRAME	ATTACHED TO	MOVES?	ITS PARENT — PLACED BY
`odom`	the floor, where odometry started	no — but slowly drifts off the true start	none: the root of our tree
`base_link`	the robot's body, on the wheel axle	with the robot	`odom` — the drive, every step
`lidar_link`	the lidar, bolted to the body	with the robot, fixed on it	`base_link` — the URDF, once
`map`	the room, as a map knows it	no	later in the course



In our setup: `odom` stays where the robot started; `base_link` and `lidar_link` drive away together. Yellow: `odom → base_link`, drawn by RViz's TF display.

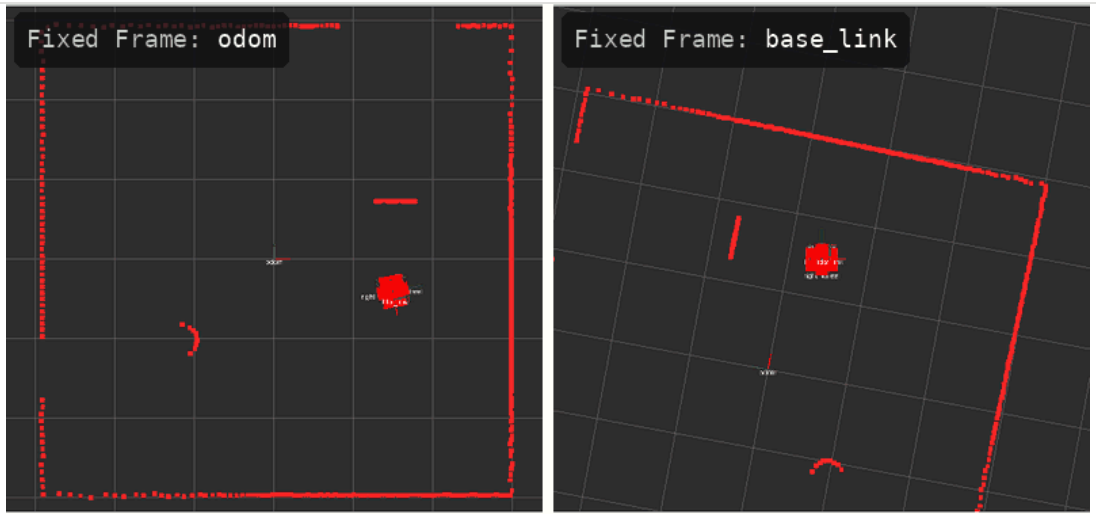
- `odom → base_link` **is** odometry: where the robot is, counted from where it started
- One name, one meaning — RViz, Nav2 and SLAM find these frames by name
- The yellow line **is** that transform, drawn by RViz's TF display: child → parent, pink at the parent

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/transformation/setup_transforms/

FIXED FRAME: THE FRAME RVIZ DRAWS IN

- The scan comes in `lidar_link`, odometry in `odom`, the model in `base_link` — one screen needs **one** frame to stand on: the **Fixed Frame**
- RViz converts every message from its `frame_id` into it, with TF
- Like a camera: on a tripod in the room (`odom`), or strapped to the robot (`base_link`)

FIXED FRAME	STANDS STILL	MOVES
`odom`	the walls, the path	the robot
`base_link`	the robot	the walls, the path
`map` — RViz's default	—	nothing is drawn: *Frame [map] does not exist* → choose `odom`



One drive, two Fixed Frames: in `odom` the room stands still; in `base_link` the robot does.

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

CHAINING TRANSFORMS: A LIDAR POINT IN THE ROOM

- Lidar 0.1 m ahead of and 0.15 m above `base_link`; robot at (2, 1), facing +y (90°); the lidar sees a wall 1 m ahead

STEP	WHAT HAPPENS	THE POINT
in `lidar_link`	as measured	(1, 0, 0)
→ `base_link`	+ the mount (0.1, 0, 0.15)	(1.1, 0, 0.15)
→ `odom`	turn 90°, then + (2, 1, 0)	**(2.0, 2.1, 0.15)**

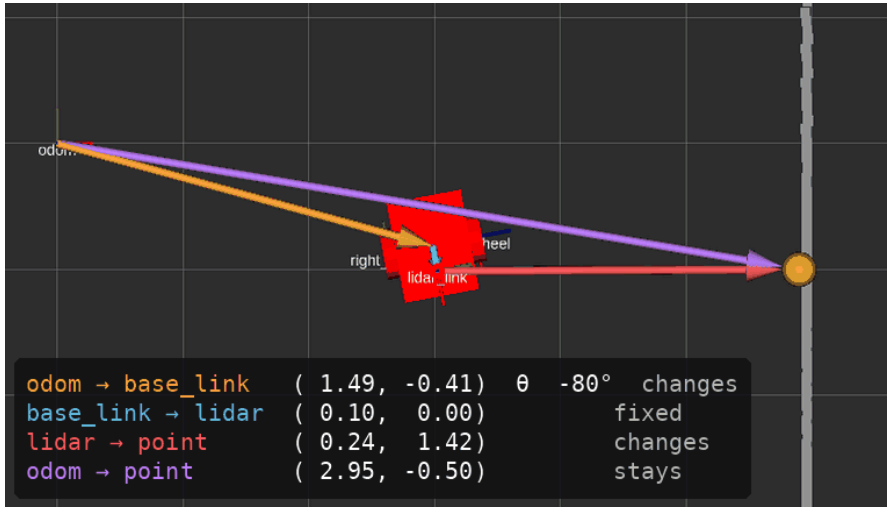
$$p_{odom} = T_{odom,base}\, T_{base,lidar}\, p_{lidar}$$

Where the lidar's point is in the room: the mount first, then the robot's pose — read right to left

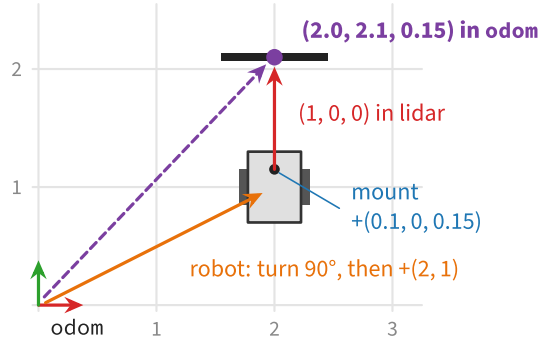
LEGEND

p_{lidar} — the point as the lidar sees it, m · $T_{base,lidar}$ — the mount, static · $T_{odom,base}$ — the robot now, dynamic · p_{odom} — the point in the room, m

foxglove.dev/blog/understanding-ros-transforms



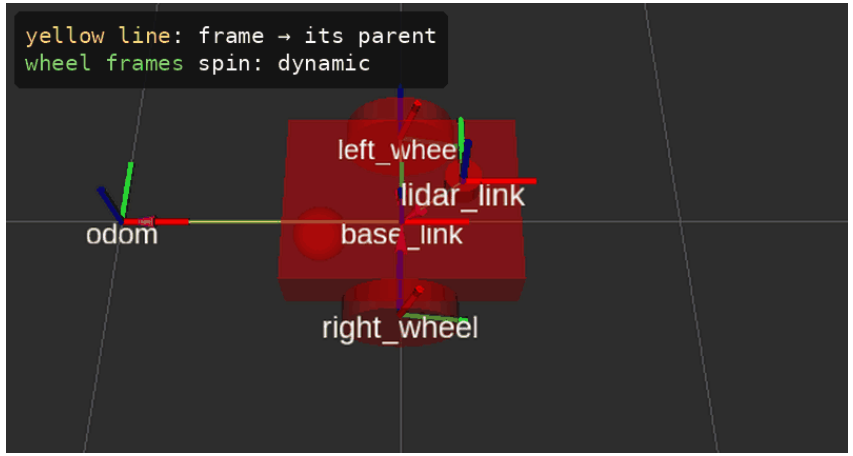
Our setup, coloured as the formula: orange the robot, blue the mount, red the point for the lidar, purple the same point in the room.



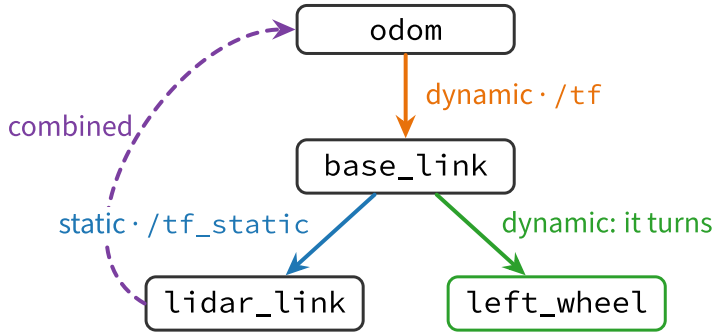
Our drawing of the example, in the same colours.

THE TF TREE: ONE PARENT, ONE ANSWER

- TF stores each frame only relative to **one parent**
- Ours: `lidar_link` and the two wheels → `base_link` → `odom`, the root
- A wheel's frame turns with the wheel, so `base_link → left_wheel` is **dynamic**, like the robot's own; the lidar's mount is static
- Any pair is computed along the tree: `lidar_link → base_link → odom` is two transforms
- One parent each → one path → one answer



In our setup: every frame points to its parent; the wheel frames spin as the robot drives.



Our drawing: the drive's edge and a wheel's change, the lidar's does not; TF multiplies two of them into the purple one.

youtu.be/QyvHhY4Y_Y8

STATIC AND DYNAMIC: SENT ONCE VS STREAMED

Static

- never changes
- `/tf_static` — sent once; late subscribers still get it
- `fixed` joints in the URDF — the ball, the lidar — published by `robot_state_publisher`

Dynamic

- changes all the time
- `/tf` — a stamped stream
- moving joints — the wheels, from their angles · `odom → base_link`, from the drive

```
$ ros2 run tf2_ros static_transform_publisher --x 0.1 --z 0.15 --frame-id base_link --child-frame-id lidar_link
[INFO] [static_transform_publisher_xOFKK8Khgao0R9Dm]: Spinning until stopped - publishing transform
translation: ('0.100000', '0.000000', '0.150000')
rotation: ('0.000000', '0.000000', '0.000000', '1.000000')
from 'base_link' to 'lidar_link'
```

One static transform by hand — enough for a test, not for a robot

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/transformation/setup_transforms/

`TransformStamped`: A TRANSFORM AS A MESSAGE

```
$ ros2 topic echo /tf_static --once
transforms:
- header: …                    # base_link → ball, cut
- header:
    stamp:
      sec: 0
      nanosec: 0
    frame_id: base_link
  child_frame_id: lidar_link
  transform:
    translation:
      x: 0.1
      y: 0.0
      z: 0.15
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
```

FIELD	MEANING	HERE
`header.frame_id`	parent frame	`base_link`
`child_frame_id`	child frame	`lidar_link`
`translation`	shift, m	0.1, 0, 0.15
`rotation`	turn	none here

- A frame itself is only a name — the `frame_id` string
- Gazebo carries a transform as `gz.msgs.Pose_V`; ROS 2 `/tf` as `tf2_msgs/msg/TFMessage`

📖 docs.ros.org/en/jazzy/Concepts/Intermediate/About-Tf2.html

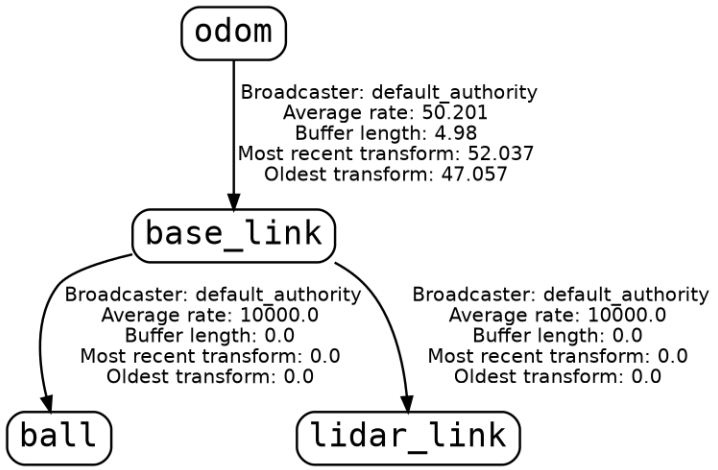
`tf2_echo` AND `view_frames`: ONE TRANSFORM, THE WHOLE TREE

```
$ ros2 run tf2_ros tf2_echo base_link lidar_link
At time 0.0
- Translation: [0.100, 0.000, 0.150]
- Rotation: in Quaternion (xyzw) [0.000, 0.000, 0.000, 1.000]
- Rotation: in RPY (radian) [0.000, -0.000, 0.000]
- Rotation: in RPY (degree) [0.000, -0.000, 0.000]
- Matrix:
  1.000  0.000  0.000  0.100
  0.000  1.000  0.000  0.000
  0.000  0.000  1.000  0.150
  0.000  0.000  0.000  1.000
```

One transform, live — once a second; a static one shows time 0

No wheels? A wheel's transform needs its angle, and nothing sends the angles in this run — later: who does

```
$ ros2 run tf2_tools view_frames   # → frames_<date>.pdf
```



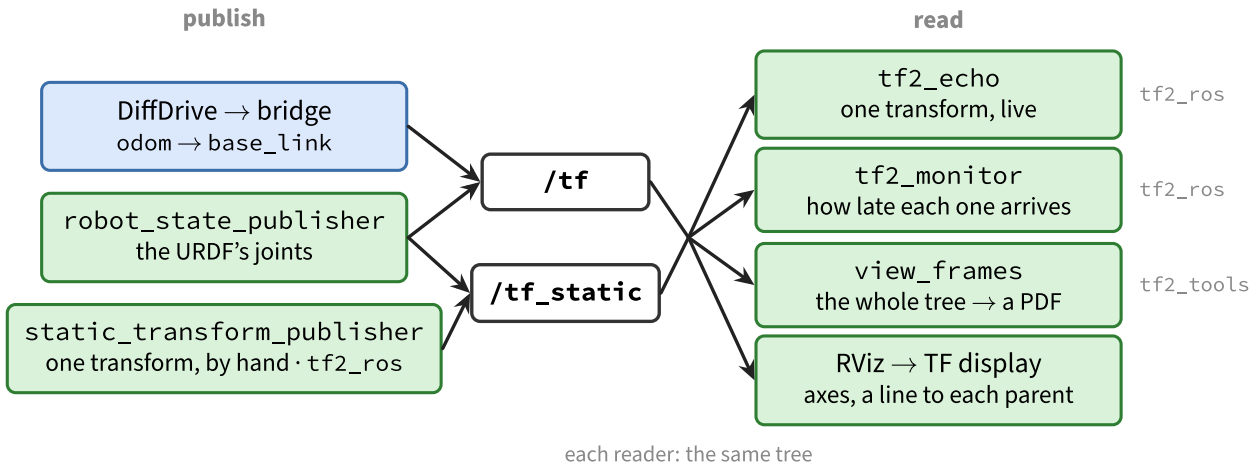
`view_frames` on our robot: `odom → base_link` from the drive at 50.2 Hz; `ball`, `lidar_link` static.

The whole tree — like `rqt_graph`, but for frames instead of nodes

docs.ros.org/en/jazzy/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html

TF TOOLS: WHAT EACH ONE SHOWS

TOOL	PACKAGE	SHOWS	USE IT TO
`tf2_echo A B`	`tf2_ros`	one transform, once a second	check one pair: is it there, are the numbers right
`view_frames`	`tf2_tools`	the whole tree, as a PDF	one tree? every frame in it?
RViz → Add → **TF**	`rviz2`	each frame's axes, a line to its parent	see the tree on the robot
`tf2_monitor`	`tf2_ros`	how late each transform arrives	find a stale publisher or a wrong clock
`static_transform_publisher`	`tf2_ros`	nothing — it publishes one static transform	test without a URDF



docs.ros.org/en/jazzy/Tutorials/Intermediate/Tf2/Debugging-Tf2-Problems.html

ROBOT DESCRIPTION: ONE FILE FOR ROS 2 AND GAZEBO

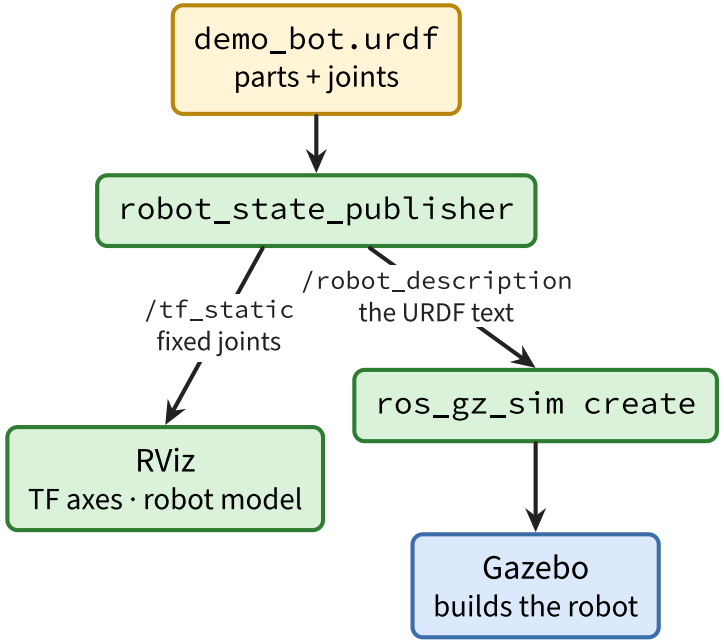
DESCRIBE THE ROBOT ONCE; LET THE SIMULATOR AND ROS 2 BOTH READ IT.

ROBOT DESCRIPTION: EVERY PART AND JOINT IN ONE FILE

- By hand, a sensor's place lives twice — in Gazebo's SDF model and in your launch command. Change one, forget the other: data in the wrong place, no error
- A turning wheel cannot be a static transform; RViz still has no body to draw
- One file lists the parts and joints; each joint's `<origin>` **is** a transform; a program publishes them; Gazebo reads the same file

URDF · demo_bot.urdf

```
<joint name="ball_joint" type="fixed">
  <parent link="base_link"/>
  <child link="ball"/>
  <origin xyz="-0.15 0 -0.05"/>  <!-- = the transform base_link → ball -->
</joint>
```



One file, two readers: ROS through `robot_state_publisher`, Gazebo through `create`.

articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/

URDF: THE ROBOT FORMAT ROS 2 TOOLS READ

- ROS tools read URDF natively: `robot_state_publisher`, RViz's robot model, MoveIt, the motor-control framework
- The same URDF goes to the real robot, where there is no simulator
- Gazebo reads URDF too — it converts it on spawn; the world stays SDF
- Robot makers ship URDF: TurtleBot 3, Clearpath Husky and Jackal, Universal Robots arms

READS…	URDF	SDF
Gazebo	✅ converts on spawn	✅ native
`robot_state_publisher`, RViz	✅ native	via `sdformat_urdf`, limited
motor-control framework	✅	❌
the real robot	✅	❌ no simulator there

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/urdf/setup_urdf/

SDF TO URDF: ELEMENT BY ELEMENT

WHAT	SDF (YOU KNOW)	URDF (NEW)
placing a part	`<pose>` on the part	`<origin>` on the joint
parent, child	`<parent>chassis</parent>`	`<parent link="base_link"/>`
axis	`<axis><xyz>0 1 0</xyz></axis>`	`<axis xyz="0 0 1"/>`
shapes	`<box><size>0.4 0.3 0.1</size></box>`	`<box size="0.4 0.3 0.1"/>`
sensor	`<sensor>` inside `<link>`	`<gazebo reference="…"><sensor>`
plugin	`<plugin>` inside `<model>`	`<gazebo><plugin>`

SDF · model.sdf

```
<!-- SDF: the part carries its own pose -->
<link name="left_wheel">
  <pose>0 0.175 0 -1.5707 0 0</pose> …
</link>
<joint name="left_wheel_joint" type="revolute">
  <parent>chassis</parent><child>left_wheel</child>
  <axis><xyz expressed_in="__model__">0 1 0</xyz></axis>
</joint>
```

URDF · demo_bot.urdf

```
<!-- URDF: the joint places the part -->
<joint name="left_wheel_joint" type="continuous">
  <parent link="base_link"/><child link="left_wheel"/>
  <origin xyz="0 0.175 0" rpy="-1.5708 0 0"/>
  <axis xyz="0 0 1"/>
</joint>
<link name="left_wheel"> … </link>
```

📖 sdformat.org/tutorials/specification/spec_model_kinematics/

`sdformat_urdf`: A SIMPLE SDF ROBOT IN ROS 2

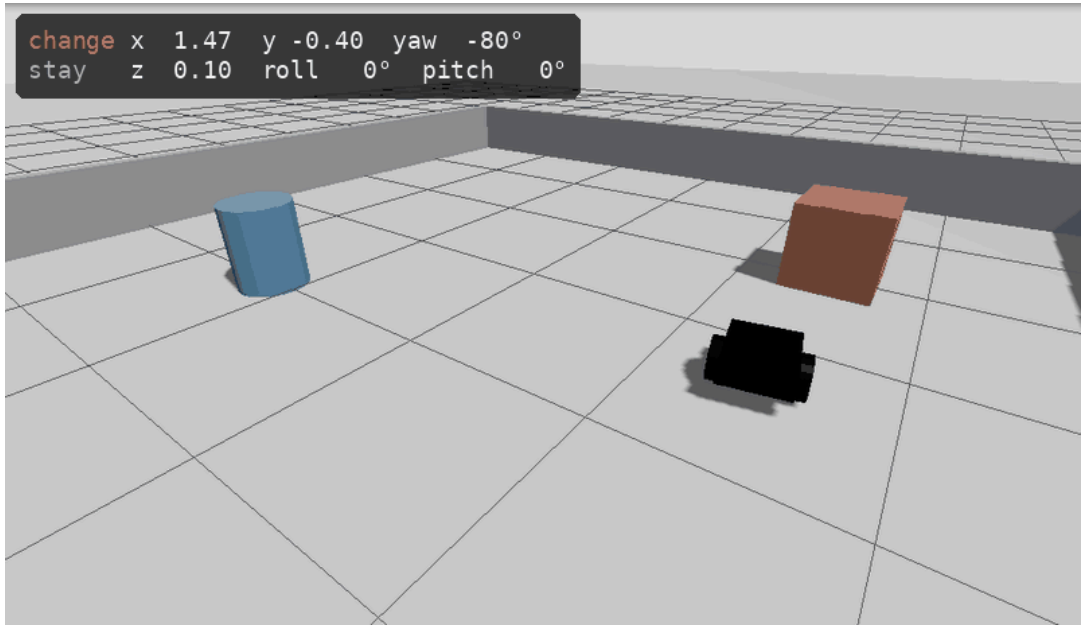
- `sdformat_urdf` lets `robot_state_publisher` read a simple SDF model — Nav2's own Gazebo guide does exactly this
- Limits: one model, no world, no nested models; simple shapes; no ball joint
- It works on a real robot too, but the motor-control framework and MoveIt need URDF
- **Rule:** a robot that will run on hardware → URDF; a robot that lives only in the simulator → SDF is fine

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/sdf/setup_sdf/

DEGREES OF FREEDOM: WHAT A JOINT LEAVES FREE

	HOW MANY	WHICH NUMBERS
a free body in space	6	shifts x, y, z · turns roll, pitch, yaw
a robot on the floor	3	x, y, yaw — the floor holds z, roll, pitch
a gear on its axle	1	its angle
`fixed` joint	0	— welded
`revolute`, `continuous` joint	1	the angle about `<axis>`
`prismatic` joint	1	the shift along `<axis>`
`planar` joint	3	x, y and the turn, in its plane
`floating` joint	6	all six

- A degree of freedom = one number you need to say where the body is



Gazebo's own pose of our robot, all six numbers: x, y, yaw change; z, roll, pitch stay — the floor holds them.

modernrobotics.northwestern.edu/nu-gm-book-resource/2-1-degrees-of-freedom-of-a-rigid-body/

WHEEL CONSTRAINT: ROLL, NEVER SLIDE SIDEWAYS

$$-\dot{x}\,\sin\theta + \dot{y}\,\cos\theta = 0$$

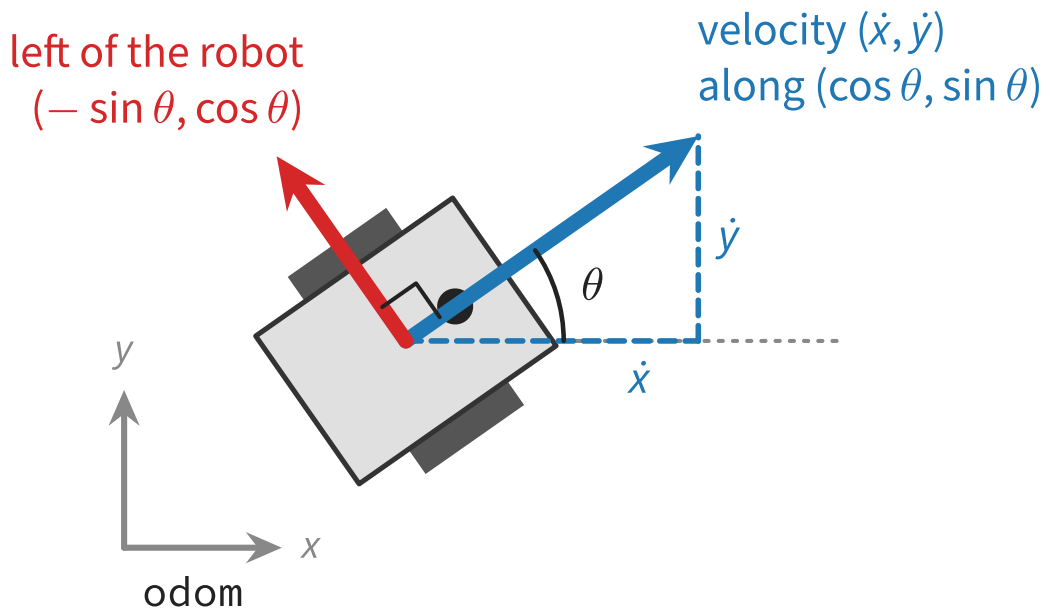
How fast the robot moves to its own left — always zero

- Blue: the velocity; red: the robot's left, $(-\sin\theta,\ \cos\theta)$. The left side is the part of blue along red — the wheels grip, so it is 0

THE ROBOT FACES	Θ	THE FORMULA BECOMES	SO IT CANNOT MOVE
along x	0	$\dot{y} = 0$	along y
along y	90°	$-\dot{x} = 0$	along x
diagonally	45°	$\dot{y} = \dot{x}$	off the diagonal

- So it moves only where it faces: $(\dot{x},\ \dot{y}) = v\,(\cos\theta,\ \sin\theta)$ — parking takes several moves, like a car

> LEGEND $\dot{x}, \dot{y}$ — the robot's velocity in `odom`, m/s · θ — heading, rad · v — forward speed, m/s



part of the velocity along the left:

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0$$

modernrobotics.northwestern.edu/nu-gm-book-resource/13-1-wheeled-mobile-robots

OVER-CONSTRAINT: FOUR WHEELS MUST SLIDE IN A TURN

- A wheel rolls without sliding only while the robot turns about a point on that wheel's axle line
- Two wheels: one axle line through the centre, $a = 0$ · four wheels: two lines, a ahead and behind — **over-constrained**: in a turn each wheel is dragged sideways

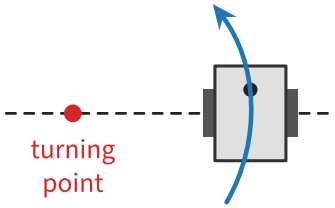
$$v_{side} = \omega\, a$$

How fast a wheel slides sideways while the robot turns — zero only on the centre's axle line

- The Husky: a = 0.256 m → at ω = 0.5 rad/s each wheel slides 0.128 m/s · ours: a = 0

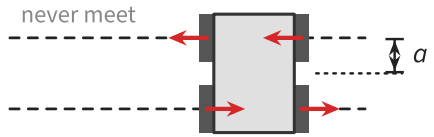
> LEGEND v_{side} — a wheel's sideways speed, m/s · ω — the robot's turn rate, rad/s · a — the axle's distance ahead of or behind the centre, m

two wheels: one axle line



wheels on the line: a = 0
v_{side} = 0, no slip

four wheels: two parallel lines



turning at ω: $v_{side} = \omega\, a$
front one way, rear the other

Our drawing: one axle line has a turning point; two parallel ones do not.

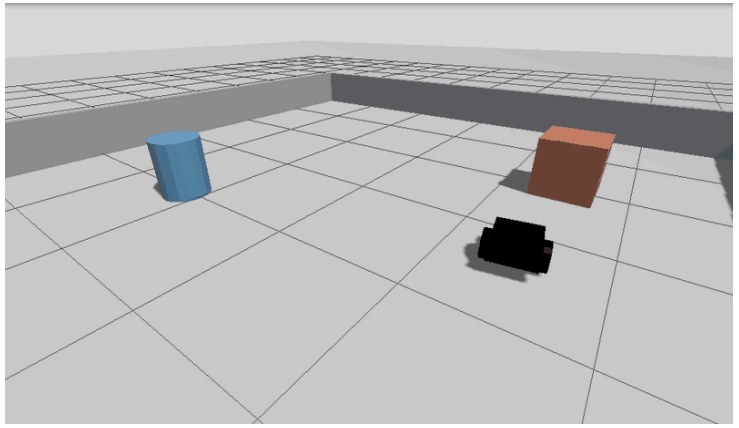


One circle, same command — white: truth, blue: odometry, blind to the slide.

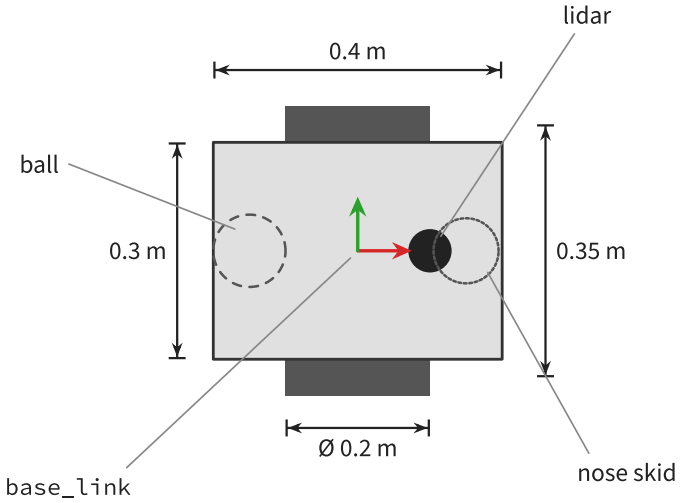
modernrobotics.northwestern.edu/nu-gm-book-resource/13-1-wheeled-mobile-robots

OUR ROBOT: TWO WHEELS, A BALL, A LIDAR

- A box body on two driven wheels
- A support ball at the back: with the two wheels, three supports — all three always touch the floor
- A frictionless skid under the nose, 5 mm up — the body balances just over the axle and may tip forward
- A lidar on top of the body
- Frames: `base_link` (on the axle), `left_wheel`, `right_wheel`, `ball`, `lidar_link`



The reference robot in our world, one loop.

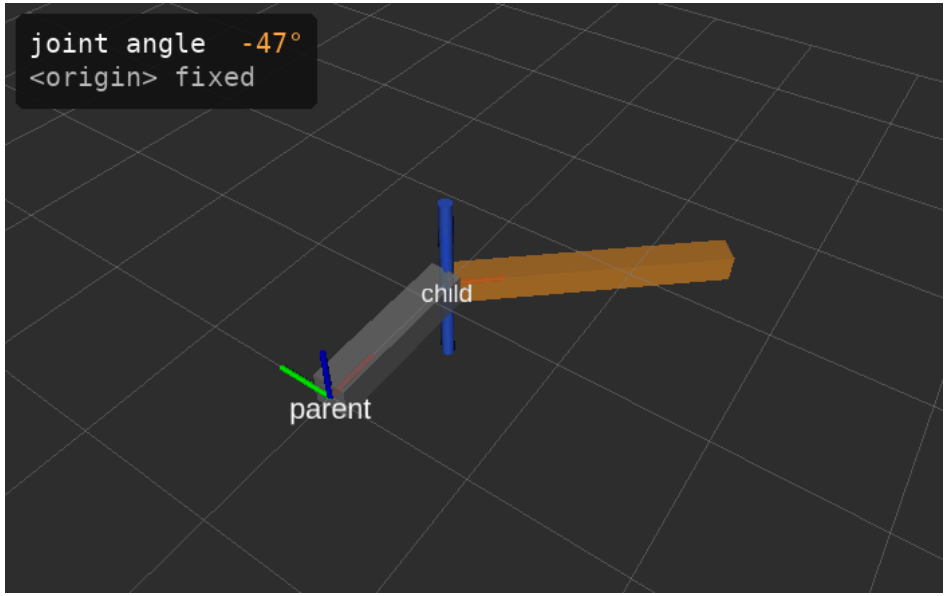


Our robot from above, real sizes.

github.com/MOGI-ROS/Week-3-4-Gazebo-basics

LINK AND JOINT: PARTS AND HOW THEY CONNECT

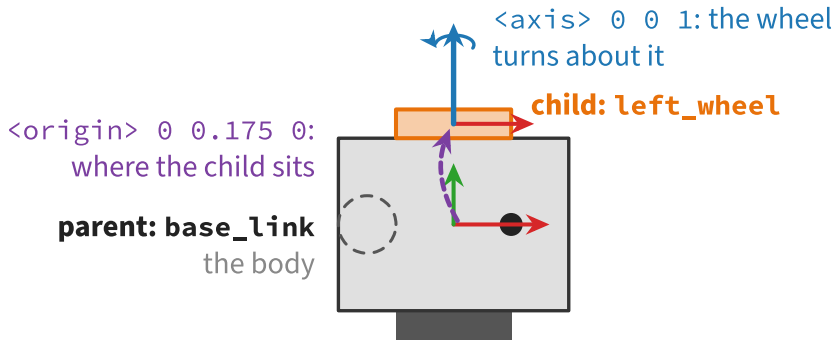
	LINK	JOINT
what it is	a rigid part	a connection between two parts
inside	`<visual>` — what you see · `<collision>` — what touches, usually simpler · `<inertial>` — mass	`<parent>`, `<child>` · `<origin>` — where the child sits · `<axis>` — in the child's frame · `<limit>`
in TF	a frame, with the link's name	the transform parent → child — its `<origin>`
kinds	—	`fixed` — welded · `continuous` — spins forever · `revolute` — turns within limits · `prismatic` — slides
ours	`base_link`, `left_wheel`, `ball`, `lidar_link`	`left_wheel_joint` — continuous · `ball_joint`, `lidar_joint` — fixed



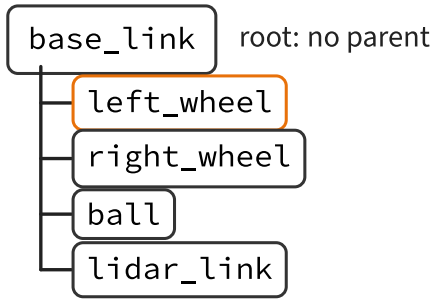
A revolute joint: `<origin>` fixes where the child sits; the angle turns it about `<axis>`.

articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/

PARENT AND CHILD: WHO MOVES WITH WHOM



the body drives → the wheel goes with it;
the joint only turns it



each link: one parent;
a parent: any number of children

URDF · demo_bot.urdf

```
<joint name="left_wheel_joint" type="continuous">
  <parent link="base_link"/>    <!-- hangs on -->
  <child link="left_wheel"/>    <!-- moves -->
  <!-- where the child's frame sits, in the parent's -->
  <origin xyz="0 0.175 0" rpy="-1.5708 0 0"/>
  <axis xyz="0 0 1"/>           <!-- the child turns about it -->
</joint>
```

	PARENT	CHILD
in the joint	the part it hangs on	the part that moves
its frame	its own	at `<origin>`, measured in the parent's frame
the parent moves	—	moves with it
a moving joint	—	turns about `<axis>`
in TF	`frame_id`	`child_frame_id`

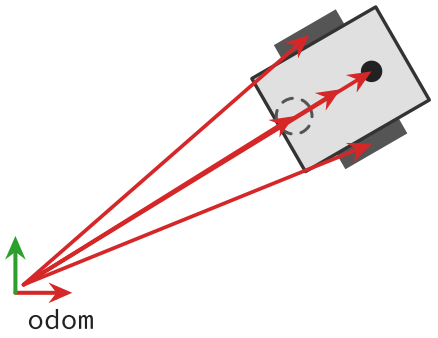
docs.ros.org/en/jazzy/Tutorials/Intermediate/URDF/Building-a-Visual-Robot-Model-with-URDF-from-Scratch.html

WHY PARENT → CHILD: MOVE THE PARENT, THE CHILDREN FOLLOW

	EVERY PART PLACED IN THE ROOM	EVERY PART PLACED ON ITS PARENT
the robot drives	all five poses change	one transform changes: `odom → base_link`
a wheel turns	its pose in the room	one angle, on its joint
the lidar moves 1 cm on the body	its pose, recomputed all the time	one `<origin>` in the file
where the lidar is from a wheel	worked out by hand	TF, along the tree

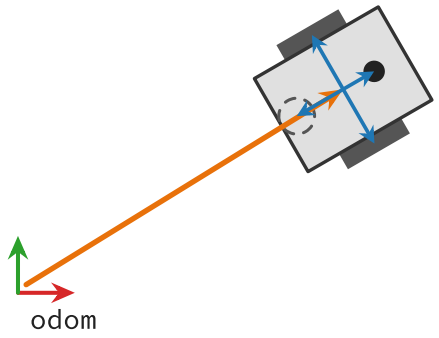
- The tree is how the robot is built: what is mounted on what
- Swap parent and child, and the body would turn with the wheel

every part placed in the room



the robot drives:
all five poses change

every part placed on its parent



the robot drives: one transform changes,
the parts on it stay

articulatedrobotics.xyz/tutorials/ready-for-ros/tf/

URDF 1/4: THE BODY

URDF · demo_bot.urdf

```
<robot name="demo_bot">
  <!-- base_link sits on the wheel axle -->
  <link name="base_link">
    <visual>
      <origin xyz="0 0 0.05"/>   <!-- the box is 5 cm above the axle -->
      <geometry><box size="0.4 0.3 0.1"/></geometry>
    </visual>
    <collision>
      <origin xyz="0 0 0.05"/>
      <geometry><box size="0.4 0.3 0.1"/></geometry>
    </collision>
    <inertial>
      <origin xyz="0 0 0.05"/>
      <mass value="2.0"/>
      <inertia ixx="0.016667" iyy="0.028333" izz="0.041667"
               ixy="0" ixz="0" iyz="0"/>
    </inertial>
  </link>
```

- One part, three shapes: what you see, what touches, what physics moves
- No `<pose>` on the part — the `<origin>` inside each block lifts the box above the axle
- `<inertial>` by hand — URDF has no `auto="true"`; the box, cylinder and sphere formulas are in the source

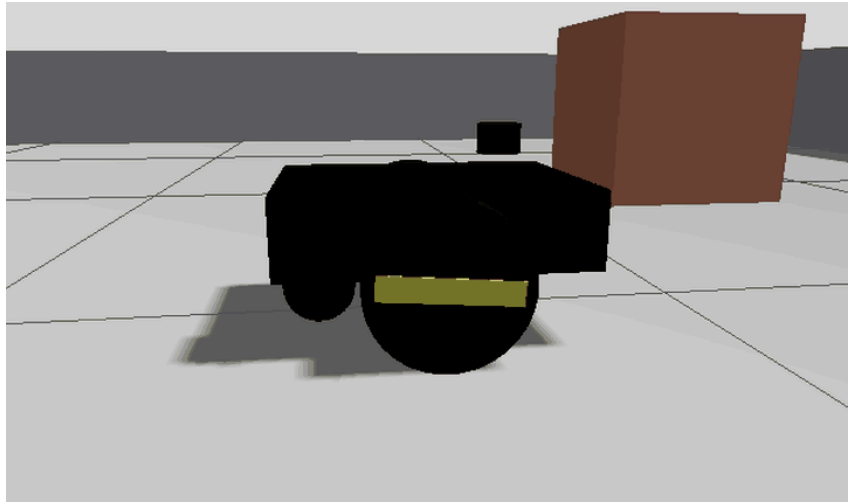
docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/urdf/setup_urdf/

URDF 2/4: ONE WHEEL

URDF · demo_bot.urdf

```
<!-- the left wheel: 17.5 cm left, the cylinder on its side -->
<joint name="left_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="left_wheel"/>
  <origin xyz="0 0.175 0" rpy="-1.5708 0 0"/>
  <axis xyz="0 0 1"/>   <!-- spins about its own axis -->
</joint>
<link name="left_wheel">
  <visual><geometry><cylinder radius="0.1" length="0.05"/></geometry></visual>
  <collision><geometry><cylinder radius="0.1" length="0.05"/></geometry></collision>
  <inertial>
    <mass value="0.5"/>
    <inertia ixx="0.001354" iyy="0.001354" izz="0.0025"
             ixy="0" ixz="0" iyz="0"/>
  </inertial>
</link>
```

- The joint places the wheel: 17.5 cm left, turned onto its side
- `continuous`: it spins without limits, about its own axis
- The right wheel: the same joint and part, mirrored in y



Our robot in Gazebo; the yellow stripe is added only to see the wheel turn.

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/urdf/setup_urdf/

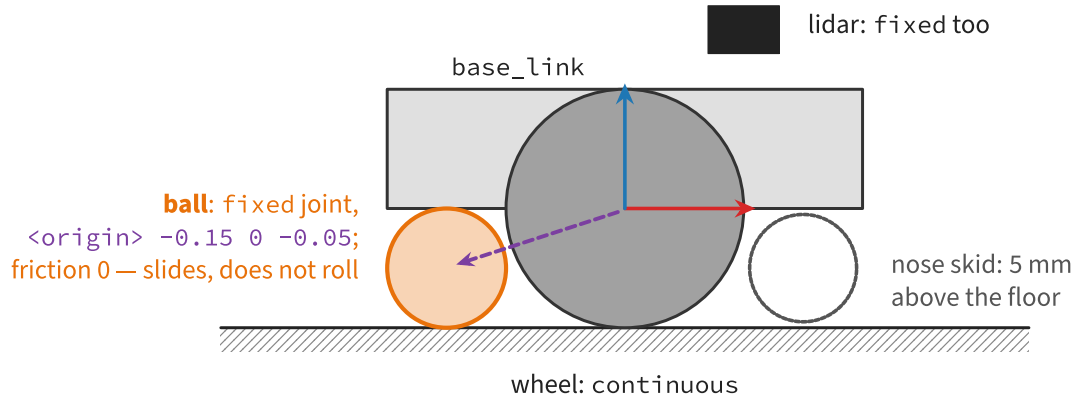
URDF 3/4: A FIXED PART — THE SUPPORT BALL

URDF · demo_bot.urdf

```
<!-- no ball joint in URDF: fixed, frictionless -->
<joint name="ball_joint" type="fixed">
  <parent link="base_link"/>
  <child link="ball"/>
  <origin xyz="-0.15 0 -0.05"/>   <!-- at the back -->
</joint>
<link name="ball"> … a 0.05 m sphere, 0.2 kg … </link>
<gazebo reference="ball">
  <mu1>0</mu1><mu2>0</mu2>
</gazebo>

<!-- inside base_link: the nose skid, a second collision -->
<collision name="nose_skid">
  <origin xyz="0.15 0 -0.05"/>
  <geometry><sphere radius="0.045"/></geometry>
</collision>
```

- A fixed joint is how every mounted thing attaches — a lidar too
- The ball slides instead of rolling — that is why its friction is zero
- One part may have several `<collision>` shapes



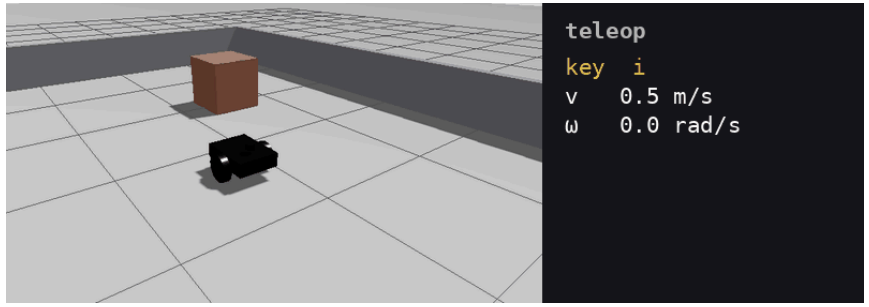
Our robot from its left side, real sizes: `<origin>` puts the ball where it touches the floor; the nose skid hangs 5 mm above it.

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/urdf/setup_urdf/

URDF 4/4: `<gazebo>` BLOCKS FOR THE SIMULATOR

URDF · demo_bot.urdf — the drive

```
<gazebo>
  <plugin filename="gz-sim-diff-drive-system"
          name="gz::sim::systems::DiffDrive">
    <left_joint>left_wheel_joint</left_joint>
    <right_joint>right_wheel_joint</right_joint>
    <wheel_separation>0.35</wheel_separation>
    <wheel_radius>0.1</wheel_radius>
    <topic>cmd_vel</topic>
    <odom_topic>odom</odom_topic>
    <frame_id>odom</frame_id>                    <!-- new: the frame names -->
    <child_frame_id>base_link</child_frame_id>   <!-- in its transform -->
  </plugin>
</gazebo>
```



Our setup: keys → `/cmd_vel` → wheels.

URDF · demo_bot.urdf — the lidar

```
<gazebo reference="lidar_link">   <!-- the part it sits on -->
  <sensor name="lidar" type="gpu_lidar">
    <topic>scan</topic>
    <update_rate>10</update_rate> <!-- scans per second -->
    <always_on>true</always_on>
    <visualize>true</visualize>   <!-- rays in Gazebo's view -->
    <lidar>
      <scan><horizontal>
        <samples>360</samples>     <!-- rays per turn -->
        <min_angle>-3.14159</min_angle>
        <max_angle>3.14159</max_angle>
      </horizontal></scan>
      <range><min>0.12</min><max>8.0</max></range>
    </lidar>
  </sensor>
</gazebo>
```

- Inside: the same `<sensor>` as in SDF · ROS ignores `<gazebo>` blocks, Gazebo pastes them in

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/odom/setup_odom_gz/

`xacro`: VARIABLES AND MACROS FOR URDF

- `xacro` turns a `.urdf.xacro` file with variables and macros into plain URDF
- **Why:** URDF has none — two identical parts are copy-paste, and a new size means editing every copy
- Properties · math in `${…}` · macros with parameters · includes
- Here: two bumpers from one macro — two wheels work the same way

```
$ xacro demo_bot.urdf.xacro > demo_bot.urdf
```

The file Gazebo and `robot_state_publisher` actually read

xacro · demo_bot.urdf.xacro

```
<xacro:property name="body_l" value="0.4"/>

<xacro:macro name="bumper" params="name x">
  <joint name="${name}_joint" type="fixed">
    <parent link="base_link"/>
    <child link="${name}"/>
    <origin xyz="${x} 0 0.02"/>
  </joint>
  <link name="${name}">
    <visual><geometry><box size="0.02 0.3 0.04"/></geometry></visual>
  </link>
</xacro:macro>

<xacro:bumper name="front_bumper" x="${body_l/2}"/>
<xacro:bumper name="rear_bumper"  x="${-body_l/2}"/>
```

📖 docs.ros.org/en/jazzy/Tutorials/Intermediate/URDF/Using-Xacro-to-Clean-Up-a-URDF-File.html

`xacro:arg`: A VALUE PASSED IN FROM OUTSIDE

xacro · demo_bot.urdf.xacro

```
<!-- declared with a default; set from outside -->
<xacro:arg name="wheel_radius" default="0.1"/>
<!-- arg → property, so it can go into ${…} -->
<xacro:property name="r" value="$(arg wheel_radius)"/>

<cylinder radius="${r}" length="0.05"/>   <!-- the wheel -->
<wheel_radius>${r}</wheel_radius>         <!-- DiffDrive -->
```

One number, used in two places — change it once

```
$ xacro demo_bot.urdf.xacro wheel_radius:=0.12 > demo_bot.urdf
```

From launch — the same `name:=value`, after the file name:

Python · launch/demo_bot.launch.py

```
Command(['xacro ', urdf, ' wheel_radius:=0.12'])
```

	SET WHERE	READ AS
`xacro:property`	inside the file	`${r}`
`xacro:arg`	outside: command line, launch	`$(arg wheel_radius)`
`xacro:include`	in another file	its properties and macros, as if written here

📖 wiki.ros.org/xacro

`gz sdf -p`: WHAT GAZEBO BUILDS FROM YOUR URDF

```
$ gz sdf -p demo_bot.urdf
  <model name='demo_bot'>
    <link name='base_link'>
      <collision name='base_link_collision'> …
      <collision name='base_link_fixed_joint_lump__nose_skid_collision_1'> …
      <collision name='base_link_fixed_joint_lump__ball_collision_2'> …
      <visual name='base_link_fixed_joint_lump__lidar_link_visual_2'> …
      <sensor name='lidar' type='gpu_lidar'> …
    <joint name='left_wheel_joint' type='revolute'> …
    <frame name='lidar_link' attached_to='lidar_joint'> …
$ gz sdf -k demo_bot.urdf
Warning … XML Element[gz_frame_id], child of element[sensor], not defined in SDF. Copying[gz_frame_id] …
Valid.
```

- Fixed joints are merged into one link: `ball` and `lidar_link` become parts of `base_link`
- **Why write URDF, then?** ROS reads the URDF; the SDF is Gazebo's private copy, rebuilt at every spawn

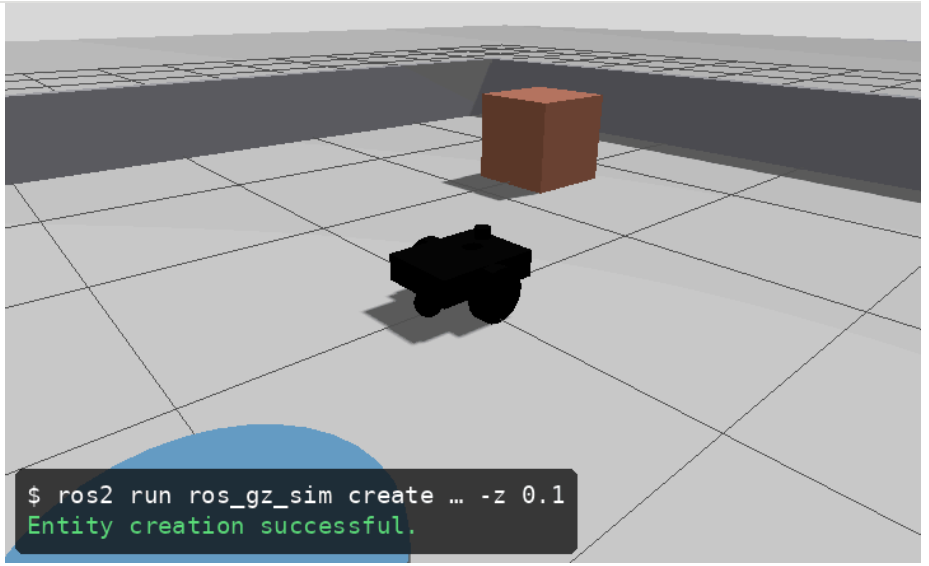
📖 gazebosim.org/docs/harmonic/spawn_urdf

`ros_gz_sim create`: SPAWNING FROM THE DESCRIPTION

	LECTURES 1-3	NOW
where the robot is written	inside the world file — a `<model>` or an `<include>`	its own URDF, apart from the world
how it gets into Gazebo	loaded with the world at start	added to the running world by `create`
who else reads it	nobody — ROS saw only topics	`robot_state_publisher`, RViz — the same text

```
$ gz sim -r walls.sdf      # the world: ground, light, walls — no robot
$ ros2 run ros_gz_sim create -topic robot_description -name demo_bot -z 0.1
[INFO] [ros_gz_sim]: Requesting list of world names.
[INFO] [ros_gz_sim]: Waiting messages on topic [robot_description].
[INFO] [ros_gz_sim]: Entity creation successful.
```

- `-topic robot_description` — Gazebo takes the URDF that ROS publishes: one text for both
- `-z 0.1` lifts it by the wheel radius, so the wheels do not start inside the floor



In our setup: the world has no robot until `create` builds it from `/robot_description`.

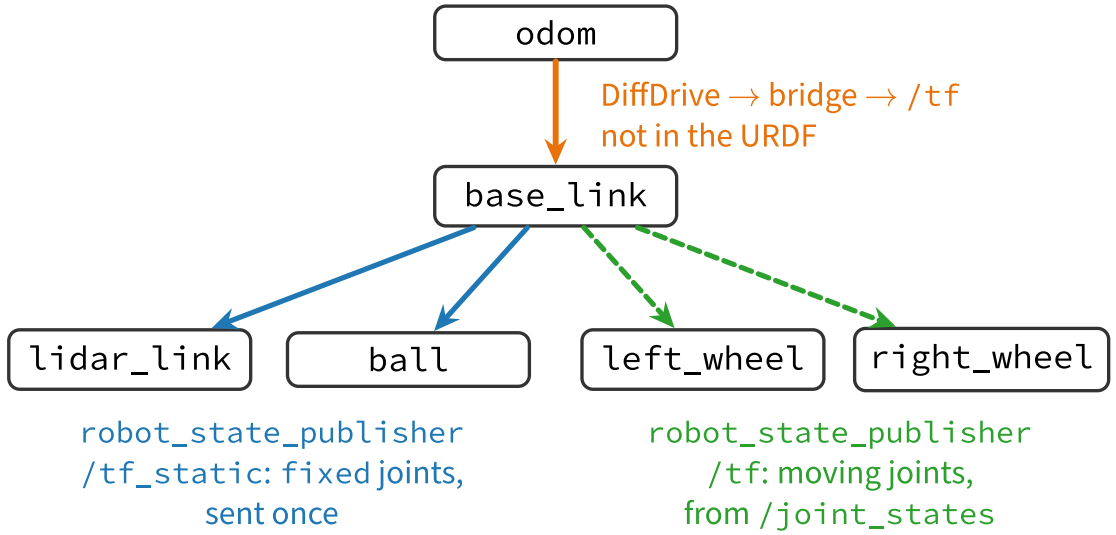
youtu.be/fH4gkIFZ6W8

`robot_state_publisher`: THREE TOPICS FROM ONE URDF

TOPIC	WHAT IS INSIDE	WHEN
`/robot_description`	the URDF text, as is — for RViz's robot model and for `create`	once; late subscribers still get it
`/tf_static`	one transform per `fixed` joint: `base_link → ball`, `base_link → lidar_link`	once; late subscribers still get it
`/tf`	one transform per moving joint: `base_link → left_wheel`, …	each time `/joint_states` brings the angles

```
$ ros2 topic echo /tf_static --once | grep child_frame_id
  child_frame_id: ball
  child_frame_id: lidar_link
$ ros2 topic echo /tf | grep child_frame_id | sort -u
  child_frame_id: base_link
```

- `/tf_static`: both fixed joints, once — from `robot_state_publisher`
- `/tf`: only `base_link` — the drive's `odom → base_link`, via the bridge. No wheels: nothing sends `/joint_states`



docs.ros.org/en/jazzy/Tutorials/Intermediate/URDF/Using-URDF-with-Robot-State-Publisher-py.html

`robot_state_publisher`: STARTING IT FROM LAUNCH

```
$ ros2 run robot_state_publisher robot_state_publisher --ros-args \
    -p robot_description:="$(xacro demo_bot.urdf)" -p use_sim_time:=true
```

Python · launch/demo_bot.launch.py

```
from launch.substitutions import Command
from launch_ros.actions import Node
from launch_ros.parameter_descriptions import ParameterValue

description = ParameterValue(Command(['xacro ', urdf]), value_type=str)  # str, or launch reads the URDF as YAML
Node(package='robot_state_publisher', executable='robot_state_publisher',
     parameters=[{'robot_description': description, 'use_sim_time': True}])
```

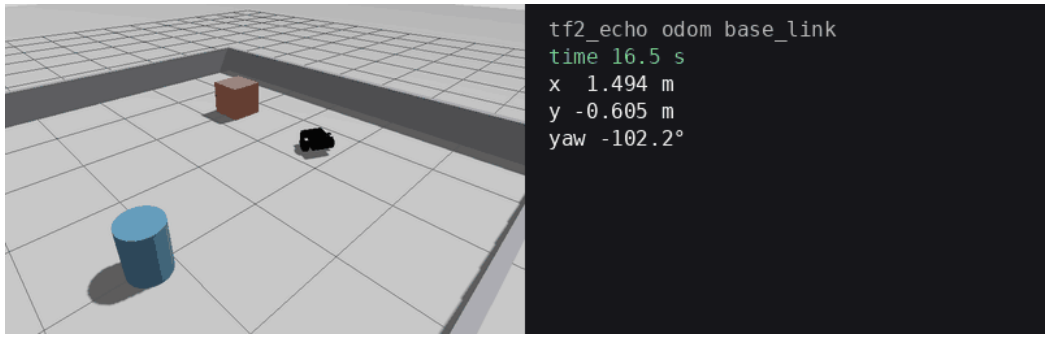
- RViz → RobotModel → Description Topic: `/robot_description`

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/urdf/setup_urdf/

DIFFDRIVE: THE TRANSFORM `odom → base_link`

- Not `robot_state_publisher`: it covers everything below `base_link`; where the robot is in `odom` comes from the drive
- DiffDrive always publishes it, on the Gazebo topic `/model/demo_bot/tf` — the bridge brings it to `/tf`
- The names inside are its `frame_id` and `child_frame_id`: `odom` and `base_link`, matching the URDF — otherwise two trees
- `tf2_echo` only reads: it prints what the drive sent

```
$ ros2 run tf2_ros tf2_echo odom base_link      # after 3 s at 0.3 m/s
At time 41.497000000
- Translation: [0.901, 0.000, 0.000]
- Rotation: in RPY (degree) [0.000, -0.000, 0.000]
```



Live in our setup: the transform the drive publishes, printed once a second.

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/odom/setup_odom_gz/

THE BRIDGE: GAZEBO TYPES TO ROS TYPES

WHAT	GAZEBO TOPIC · TYPE	ROS TOPIC · TYPE
clock	`/clock` · `gz.msgs.Clock`	`/clock` · `rosgraph_msgs/msg/Clock`
command	`cmd_vel` · `gz.msgs.Twist`	`/cmd_vel` · `geometry_msgs/msg/Twist`
odometry	`odom` · `gz.msgs.Odometry`	`/odom` · `nav_msgs/msg/Odometry`
`odom → base_link`	`/model/demo_bot/tf` · `gz.msgs.Pose_V`	`/tf` · `tf2_msgs/msg/TFMessage`
scan	`scan` · `gz.msgs.LaserScan`	`/scan` · `sensor_msgs/msg/LaserScan`

YAML · config/bridge.yaml

```yaml
- ros_topic_name: /tf
  gz_topic_name: /model/demo_bot/tf      # renamed on the way
  ros_type_name: tf2_msgs/msg/TFMessage
  gz_type_name: gz.msgs.Pose_V
  direction: GZ_TO_ROS
```

- `/tf_static` from `robot_state_publisher` is ROS-only — nothing to bridge

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

OUR ROBOT'S TF: EVERY EDGE AND WHO SENDS IT

JOINT	PARENT → CHILD	<origin>	TYPE	TOPIC · SENT BY
`ball_joint`	`base_link → ball`	`-0.15 0 -0.05`	fixed	`/tf_static` · `robot_state_publisher`, once
`lidar_joint`	`base_link → lidar_link`	`0.1 0 0.15`	fixed	`/tf_static` · `robot_state_publisher`, once
`left_wheel_joint`	`base_link → left_wheel`	`0 0.175 0`, rpy `-1.5708 0 0`	continuous	`/tf` · needs wheel angles — not today
`right_wheel_joint`	`base_link → right_wheel`	`0 -0.175 0`, rpy `-1.5708 0 0`	continuous	`/tf` · needs wheel angles — not today
— not in the URDF	`odom → base_link`	—	—	`/tf` · DiffDrive, through the bridge · 50 Hz



Every link is a frame; a joint is the edge between two — its name never appears in TF

- **The error:** RViz's robot model looks up every link in TF. `left_wheel`, `right_wheel` are not there: nothing sends `/joint_states`, so `robot_state_publisher` has no wheel angles to publish
- **What it breaks:** only the picture — RViz puts both wheels at `odom`'s origin. Driving, odometry and the scan never use the wheel frames
- **The fix:** Gazebo sends the angles — the `JointStatePublisher` system in a `<gazebo>` block — and one bridge line brings them to `/joint_states`. Then `/tf` carries `left_wheel` and `right_wheel` too: measured in our setup

🎯 github.com/ros/robot_state_publisher/blob/jazzy/README.md

`view_frames`: CHECKING YOUR ROBOT'S TREE

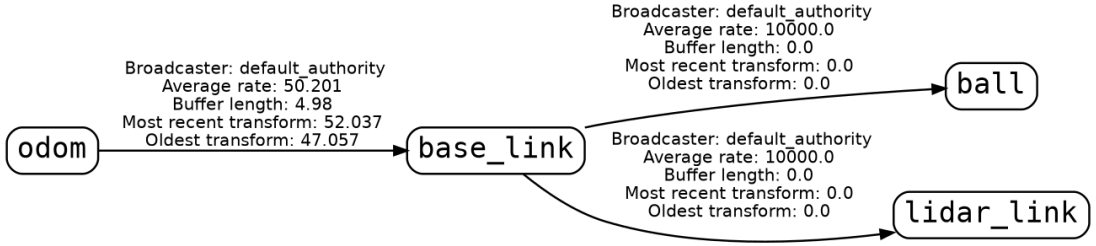
```
$ ros2 run tf2_tools view_frames    # listens to TF for 5 s → frames_<date>.pdf
```

Check

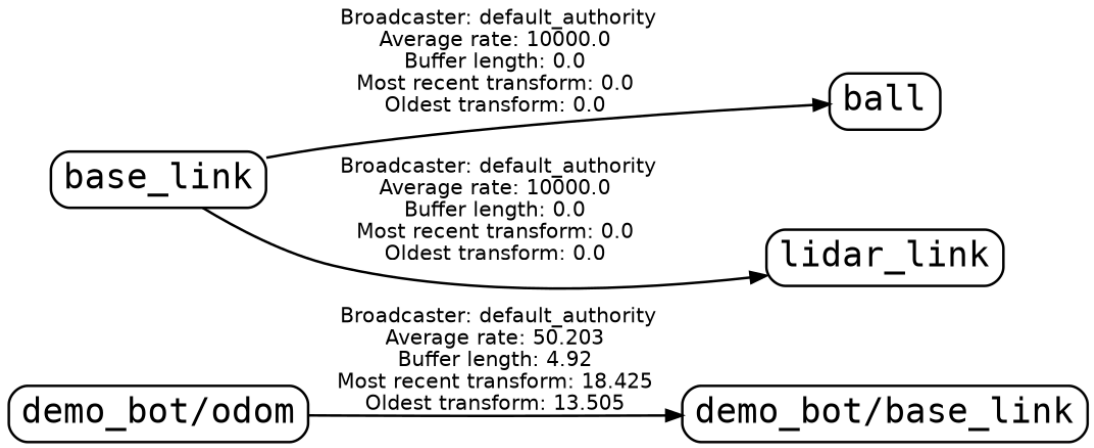
- one root: `odom`
- `base_link`, `ball`, `lidar_link` present
- names as in the URDF — no `demo_bot/…`
- `odom → base_link` updating at the drive's rate
- fresh stamps

Faults

- two trees side by side → a name mismatch
- a stale stamp → its publisher stopped
- the robot jumps between two poses → two publishers of one transform



Ours: one tree, root `odom`.



DiffDrive without `frame_id`, `child_frame_id`: two trees — the drive's `demo_bot/…` and the URDF's.

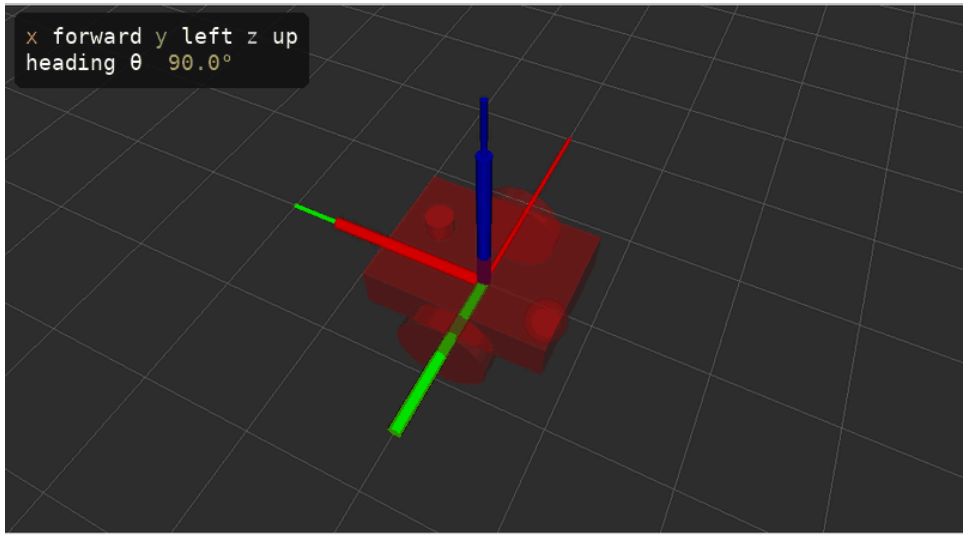
github.com/MOGI-ROS/Week-3-4-Gazebo-basics

CONVENTIONS: NAMES AND AXES EVERYONE ASSUMES

- **Names:** exactly `odom`, `base_link`, `map` — another spelling is another frame, and the tools will not find it
- **Axes:** x forward, y left, z up; anticlockwise is positive; metres, radians, seconds
- **One publisher per transform** — find who sends `odom → base_link`:

```
$ ros2 topic info /tf --verbose    # publishers only, their node names
Publisher count: 2
Node name: robot_state_publisher
Node name: ros_gz_bridge
$ # stop the bridge, then
$ ros2 run tf2_ros tf2_echo odom base_link
… Invalid frame ID "odom" passed to canTransform argument target_frame - frame does not exist
```

The transform vanished with the bridge — so it came through the bridge, from the drive



Thin: `odom`; thick: `base_link`. A left turn is anticlockwise, so the heading grows.

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/transformation/setup_transforms/

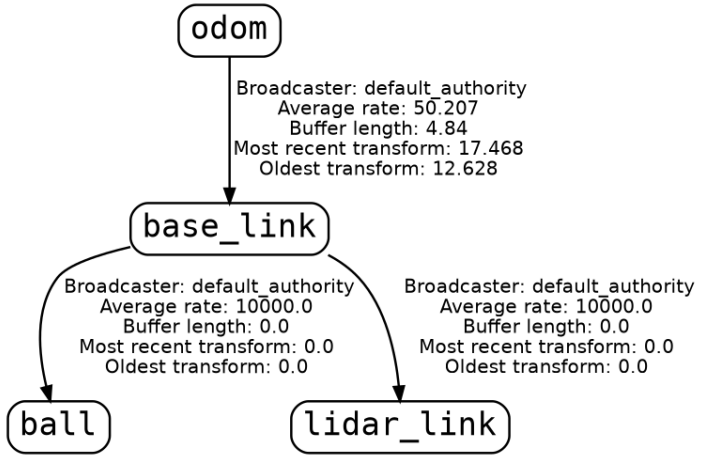
THE LIDAR PUBLISHES. RVIZ SHOWS NO SCAN. WHY?

ANSWER: THE SCAN'S FRAME IS NOT IN THE TREE

```
$ ros2 topic echo /scan --field header --once
stamp:
  sec: 6
  nanosec: 100000000
frame_id: demo_bot/base_link/lidar
```

- The scan says: *I was measured in* `demo_bot/base_link/lidar`
- TF knows `lidar_link` — and no frame by that name
- RViz cannot place the scan, so it draws nothing — no crash, **no red**: LaserScan says *Status: Ok, Showing [0] points*
- Only RViz's terminal says why:

```
[INFO] [rviz]: Message Filter dropping message: frame 'demo_bot/base_link/lidar'
  at time 4.700 for reason 'discarding message because the queue is full'
```



The tree of that run: `lidar_link` is there, `demo_bot/base_link/lidar` is not.

	THE SCAN'S `frame_id`	IN THE TREE?	RVIZ
ours now	`demo_bot/base_link/lidar`	no	draws nothing
as it must be	`lidar_link`	yes — from the URDF	the scan on the walls

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

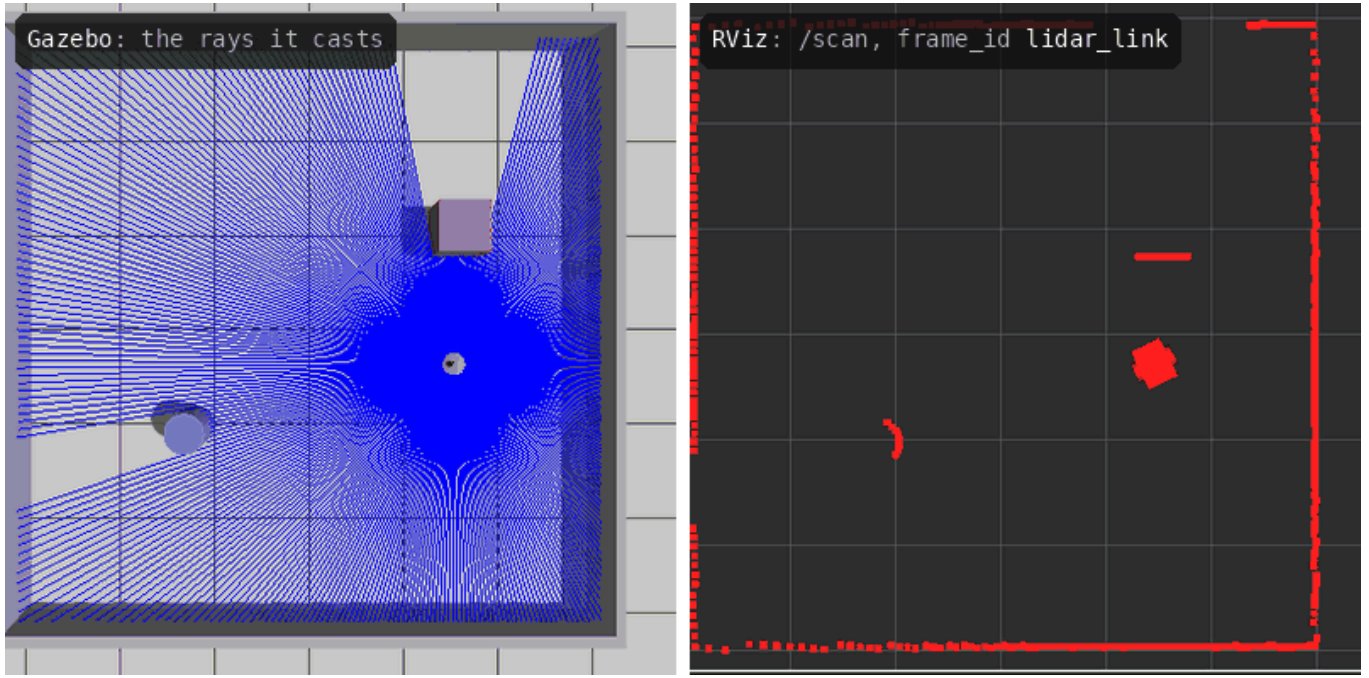
`gz_frame_id`: NAMING THE SENSOR'S FRAME

- Gazebo names a sensor's frame `<model>/<link>/<sensor>` — and after merging, the link is `base_link`
- One line gives the scan the name the tree already has:

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

```
<sensor name="lidar" type="gpu_lidar">
  <gz_frame_id>lidar_link</gz_frame_id>
  …
```

- The path of that name: `<gz_frame_id>` → each scan's `header.frame_id` → RViz asks TF where `lidar_link` is → draws the points in its Fixed Frame, `odom`
- You do not switch RViz to `lidar_link`: the Fixed Frame stays `odom`



After the fix, one drive: left, the rays Gazebo casts; right, the scan they make, stamped `lidar_link` and placed by TF.

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

`use_sim_time`: ONE CLOCK FOR EVERY NODE

- Every transform carries a time — TF looks it up for the message's stamp, so every stamp must come from one clock
- **Where:** `/clock` — once, a bridge line from Gazebo · `use_sim_time: true` — on **every** node you start: `robot_state_publisher`, RViz, your nodes, the tools
- **When:** always with a simulator · a bag: `ros2 bag play --clock` · a real robot: never
- The growing lag on the right is not the error — Gazebo just runs slower than real time. The error is a node on the other clock:

```
$ ros2 run tf2_ros tf2_monitor                                    # wall clock
Frame: base_link, … Average Delay: 1.7924e+09, Max Delay: 1.7924e+09
$ ros2 run tf2_ros tf2_monitor --ros-args -p use_sim_time:=true   # Gazebo's
Frame: base_link, … Average Delay: 0.000691932, Max Delay: 0.09
```

Python · launch/demo_bot.launch.py — the right way

```
Node(package='robot_state_publisher', executable='robot_state_publisher',
     parameters=[{'robot_description': description, 'use_sim_time': True}])
Node(package='rviz2', executable='rviz2',
     parameters=[{'use_sim_time': True}])
# or once, for every node after it: SetParameter(name='use_sim_time', value=True)
```

```
wall clock (the computer)        19.9 s
/clock    (Gazebo)               15.0 s
Gazebo is behind by               5.0 s
measured in our container: real-time factor 0.75
```

Measured in our container: `/clock` falls behind the wall clock — every node on `/clock` falls behind with it, together.

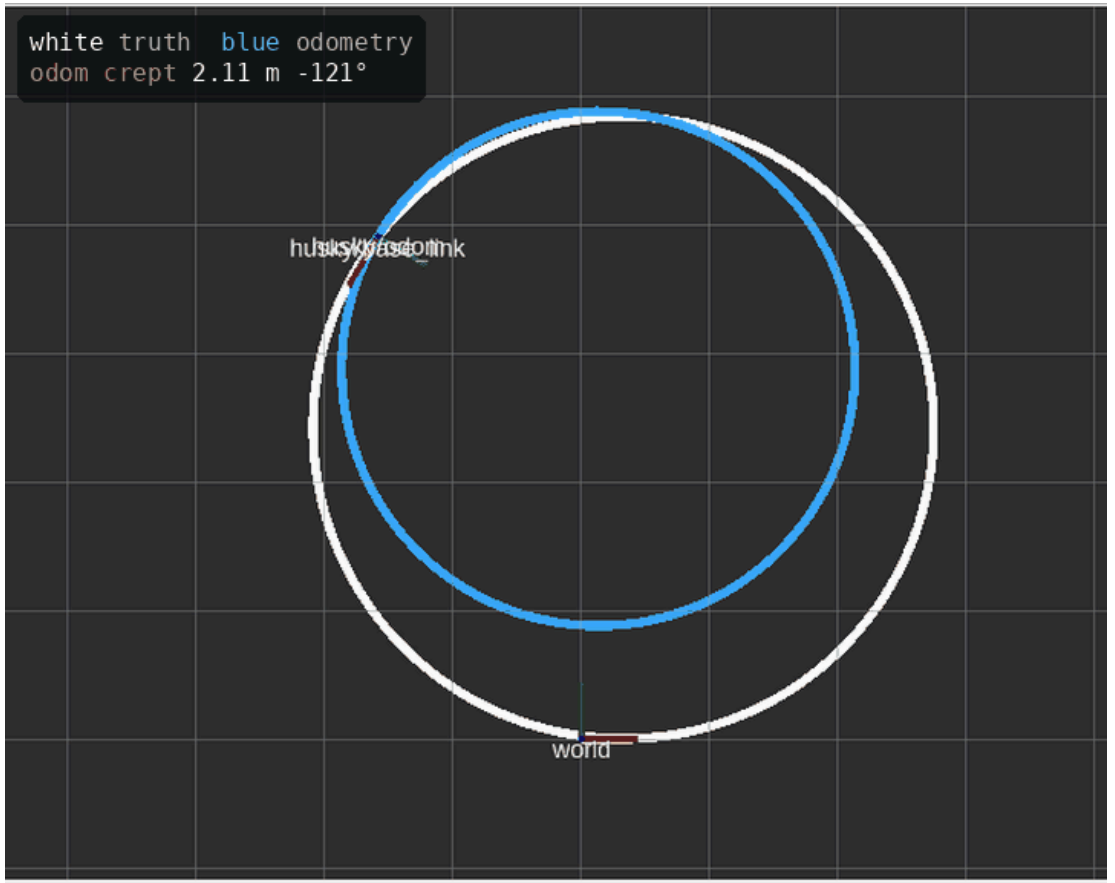
📖 design.ros2.org/articles/clock_and_time.html

MEASURING DRIFT: HOW FAR OFF ODOMETRY IS

YOU HAVE SEEN ODOMETRY DRIFT. NOW — BY HOW MUCH?

DRIFT: THE `odom` FRAME CREEPS AWAY

- `odom → base_link` is smooth — but the `odom` frame itself slides away from the world
- Two full circles, measured:
  - two-wheel robot: odometry ends at the start; Gazebo puts it 1.7 cm and -1.6° from it
  - four-wheel robot: odometry ends at the start; Gazebo puts it 2.1 m and -121° from it
- The wheels count what they turned, not where the robot went



The Husky, two circles: `odom` drawn where it really is creeps away — the overlay is measured.

docs.nav2.org/jazzy/configuration_and_development/first_time_robot_setup_guide/odom/setup_odom_gz/

HEADING ERROR: WHY DRIFT GROWS WITH DISTANCE

$$e_{\perp} \approx d\,\Delta\theta$$

How far sideways the robot ends up after driving d with its heading off by $\Delta\theta$

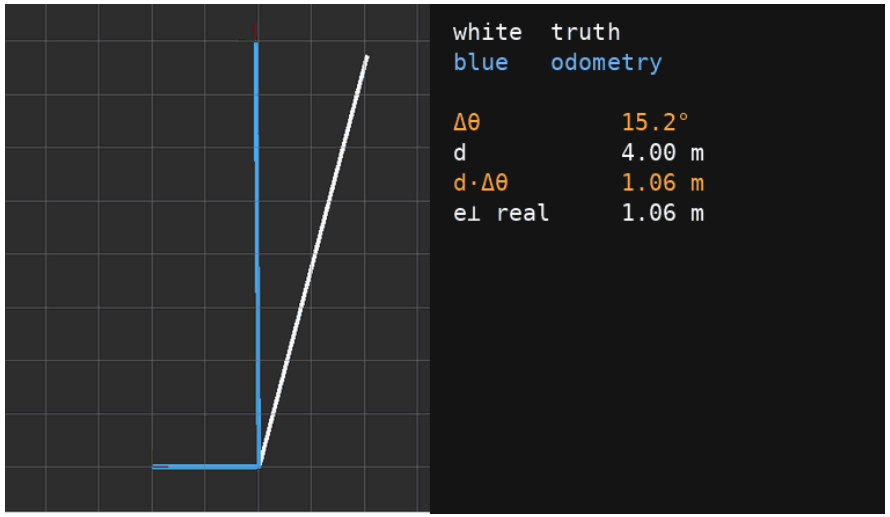
- 1° of heading error → 17 cm sideways after 10 m, 1.7 m after 100 m
- Every turn adds heading error — so position error grows faster than distance

LEGEND

$e_{\perp}$ — sideways error, m

d — distance since the heading error, m

$\Delta\theta$ — heading error: the real heading minus odometry's, rad (1° = 0.01745 rad)



The Husky in our setup: one bad turn, then the sideways error grows like d·Δθ — 1.06 m by the formula, 1.06 m measured.

youtu.be/IVTV7vJgIkU

GROUND TRUTH: THE TRUE POSE FROM GAZEBO

Not there by default. **It arrives** only if (each checked in our setup):

1. OdometryPublisher sits in the model — a `<gazebo>` block in the URDF
2. its own `<odom_topic>` — else `/model/<name>/odometry`, the drive's default too
3. the simulation runs — paused, it publishes nothing
4. the bridge carries it: `/odom_gt@nav_msgs/msg/Odometry[gz.msgs.Odometry`

It works as the truth only if:

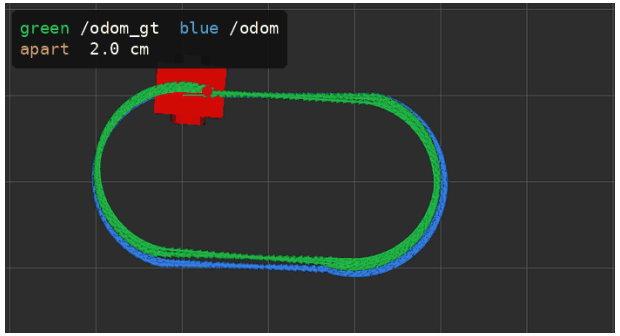
5. it agrees with Gazebo's own pose, `gz model -m demo_bot -p` — ours: to the last digit
6. its own frame names — `world`, `base_link_gt`, never the drive's `odom`, `base_link`
7. one clock: its stamps are simulator time — your measuring node runs with `use_sim_time`
8. to draw both in RViz: one static `world → odom` — they start at one pose

📖 gazebosim.org/api/sim/8/classgz_1_1sim_1_1systems_1_1OdometryPublisher.html
📖 troubleshooting: gazebosim.org/docs/harmonic/troubleshooting/

URDF · demo_bot.urdf

```
<gazebo>
  <plugin filename="gz-sim-odometry-publisher-system"
          name="gz::sim::systems::OdometryPublisher">
    <odom_topic>odom_gt</odom_topic>   <!-- not the drive's topic -->
    <odom_frame>world</odom_frame>
    <robot_base_frame>base_link_gt</robot_base_frame>
    <dimensions>3</dimensions>         <!-- z, roll, pitch too -->
  </plugin>
</gazebo>
```

Nothing arrives? In this order: `gz topic -l` → `gz topic -e -t /odom_gt -n 1` → `ros2 topic hz /odom_gt`



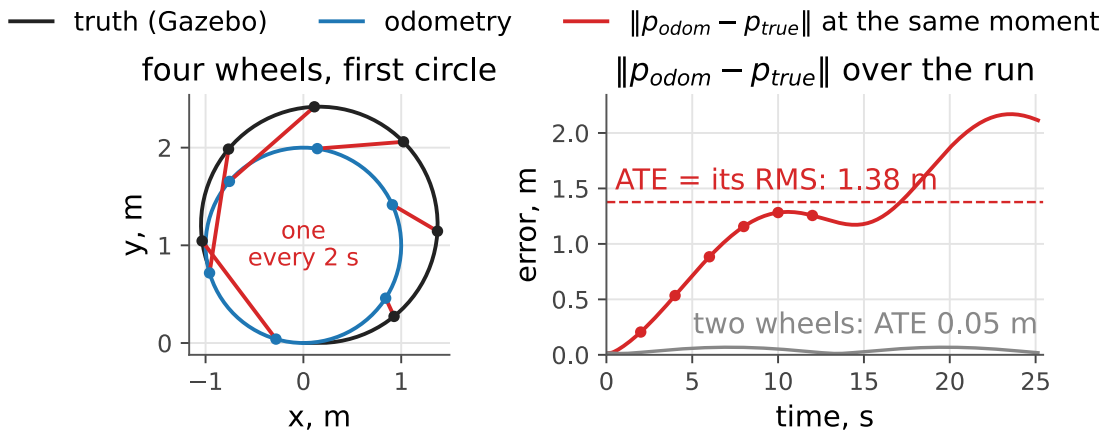
Three fast loops in our setup: green `/odom_gt`, blue `/odom`.

ATE: HOW FAR OFF, OVERALL

$$\mathrm{ATE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left\|p_{odom}(t_i) - p_{true}(t_i)\right\|^2}$$

One number per run: how far, typically, odometry puts the robot from where it really is

- Both positions at the same times, start on start: the root mean square of their distances



From our runs: one distance every 2 s on the left, all of them over time on the right.

LEGEND

ATE — Absolute Trajectory Error

$p_{odom}(t_i)$ — position by odometry at t_i, m

$p_{true}(t_i)$ — position by ground truth at the same time, m

N — matched samples

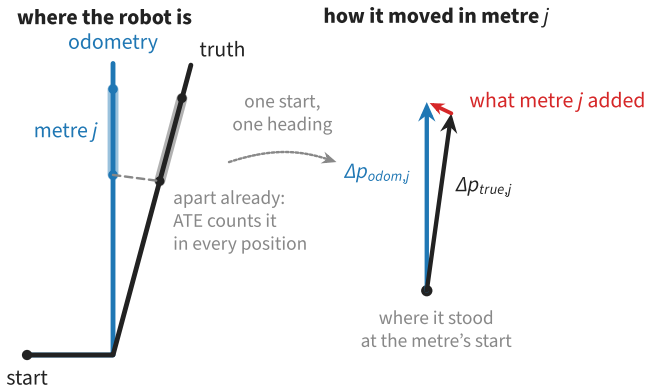
github.com/MichaelGrupp/evo/wiki/Metrics

RPE: HOW MUCH WORSE PER METRE

$$\mathrm{RPE} = \sqrt{\frac{1}{M}\sum_{j=1}^{M} \left\| \Delta p_{odom,j} - \Delta p_{true,j} \right\|^2}$$

How much error one metre of driving adds — wherever the earlier error came from

- Each metre's motion by odometry and truly, both seen from where the robot stood at the metre's start — the old offset drops out



Our drawing: moved to one start and one heading, only the red arrow is metre j's own error.

github.com/MichaelGrupp/evo/wiki/Metrics

LEGEND

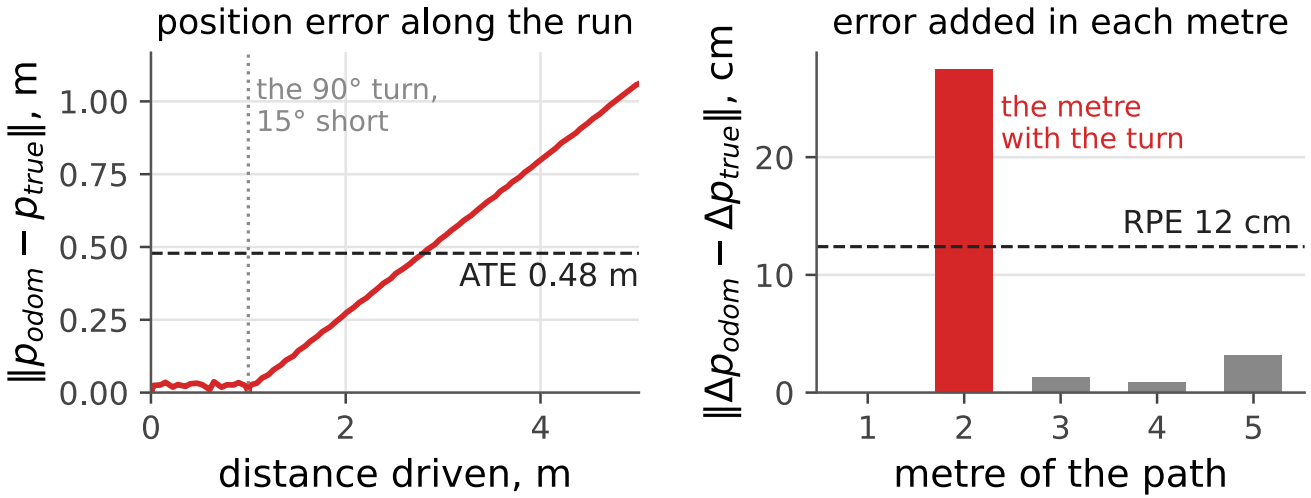
RPE — Relative Pose Error

$\Delta p_{odom,j}$, $\Delta p_{true,j}$ — how the robot moved during metre j by odometry and truly, seen from where it stood at the metre's start, m

M — metres driven

ATE VS RPE: WHERE IT IS VS HOW IT DRIVES

	ATE — ABSOLUTE TRAJECTORY ERROR	RPE — RELATIVE POSE ERROR
compares	positions, at the same moments	each metre's motion, from where it began
an early error	counted again in every later position	counted once, in its own metre
answers	how far off is odometry, typically?	how much does each metre of driving spoil it?
good for	one run, one number	comparing drives, robots, settings



Our run with one bad turn: ATE counts it in every later position, RPE only in the metre where it happened.

📖 rpg.ifi.uzh.ch/docs/IROS18_Zhang.pdf

WHAT'S NEXT

Lecture 5 — Sensors and Noise

What each sensor really measures · how a lidar sees · how noisy it is, measured · where a 2D lidar is blind

gazebosim.org/docs/harmonic/sensors