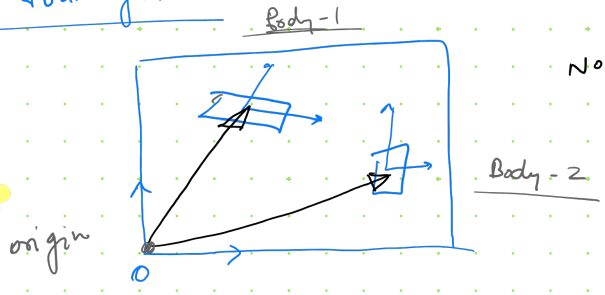
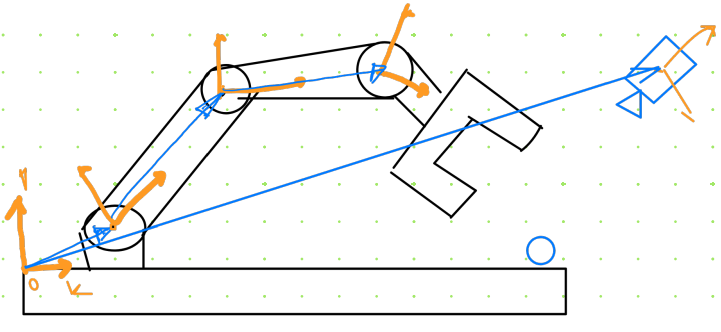
class 4

ROS-6

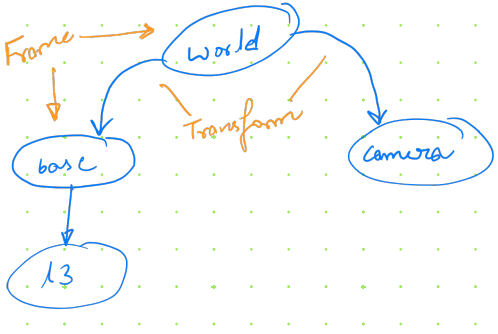
Coordinate transformation



No closed loop



ROS TF → Broadcast transform



base & camera are defined w.r.t world

Broadcaster transform static & dynamic

Broadcasting & Listening

Example of static transform

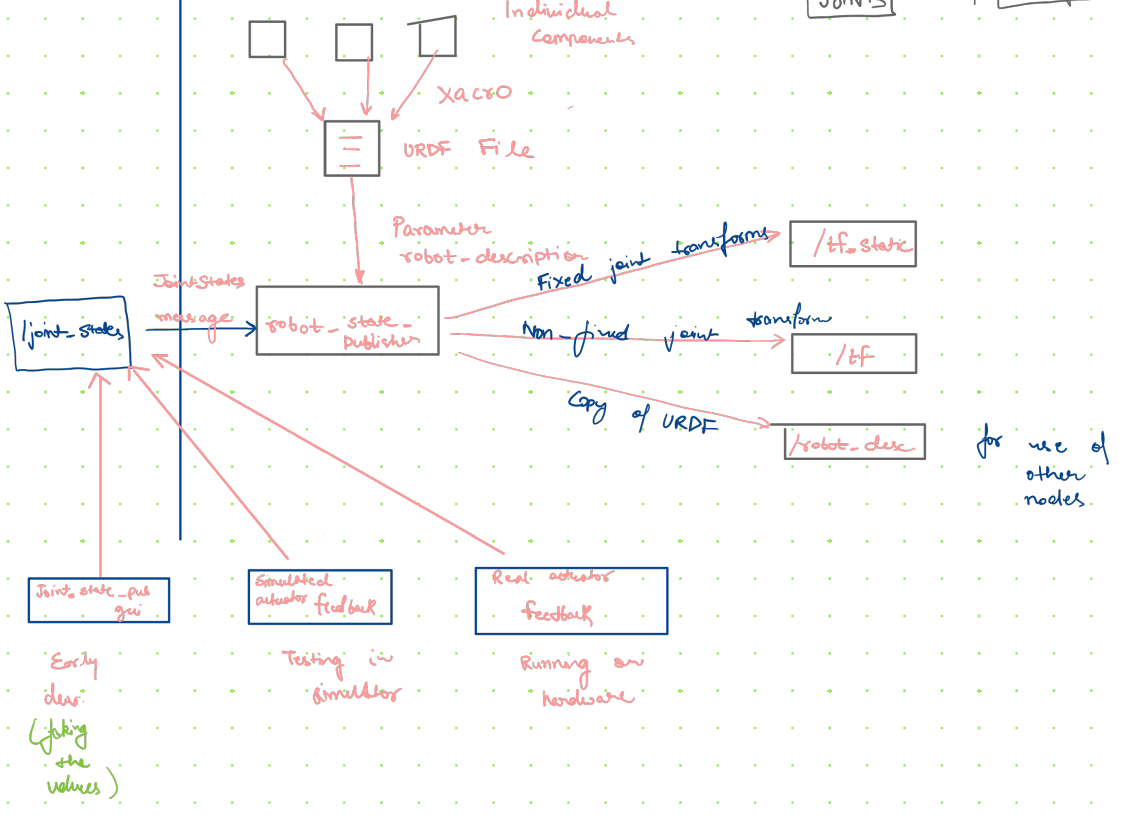
- ros2 run tf2_ros static_transform_publisher ------ world robot1
- " " " " " ------ robot1 robot2

Dynamic transform

- URDF File : Inertia, joint, color

URDF
A tree of ....
LINKS
Connected by
JOINTS

TF
A tree of
Frames
connected by
Transforms



> Add display from rviz , tf & robotModel

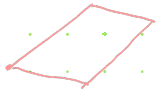
> Debugging with view_frames

- generate the pdf

-

Kor-7

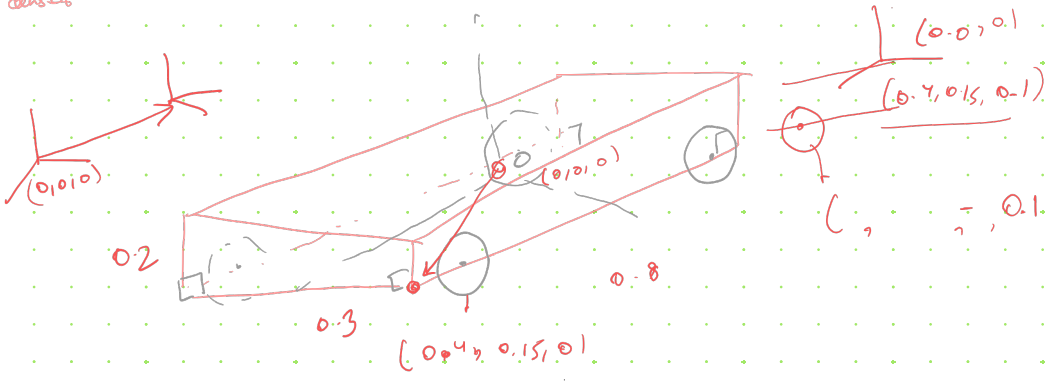
URDF



> Break physical obj into separate into components

>> If two parts move differently

>> obj.urdf



> ros2 launch urdf_tutorial display.launch.py model:=tb3_waffle.urdf

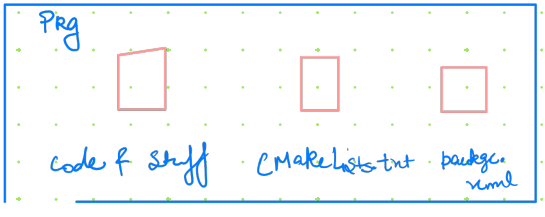
18 Nov | Morgen |

→

✓ Task for today : →

✓ I have (rough) URDF File of physical system, now i am creating testing on turtlebot3, but have to configure for SLAM, Planning, Control.

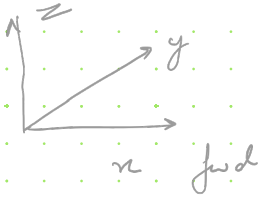
> Simulation environment / Real environment



> creating a common base for working.

---

creating a rough 3D model



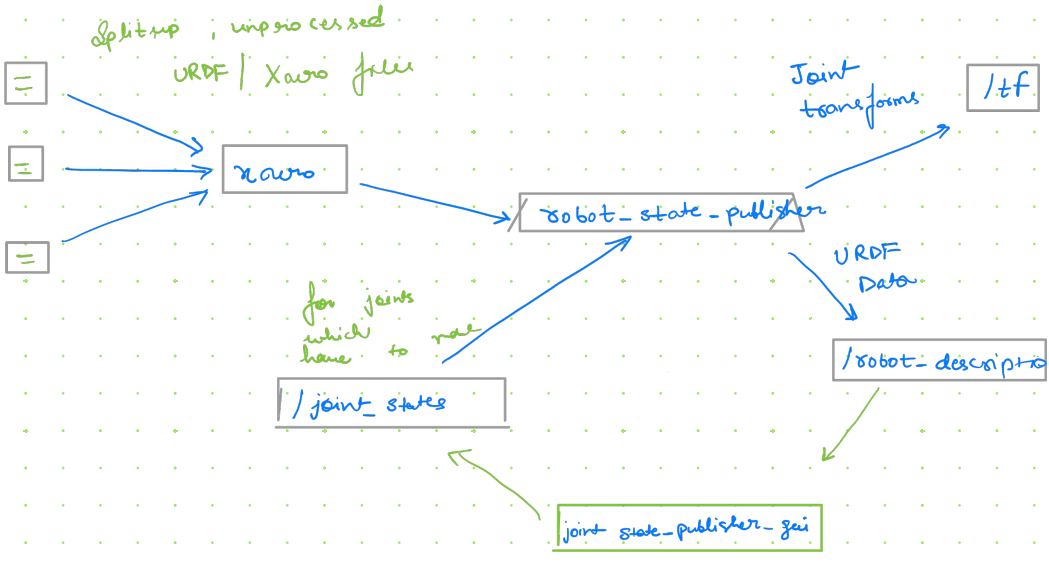


Abb.

/name : Its a topic

Rule

-- symlink-install : if used then you don't have to build xacro file again & again, if no new added

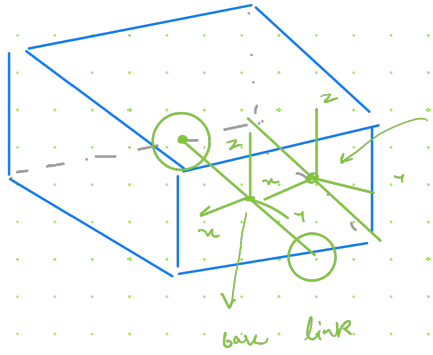
- rosdep : if rviz2 not fucking update

URDF

→ Bot , Lidar , — : different files for everything

Base link choice

It can selected anywhere COM, etc. But for differential robot its better to choose in the center of powering wheels which is also center of rotation



→ when you are going to place box, then its centered around "origin of chassis", which we don't want.

So, we will box → by xyz.

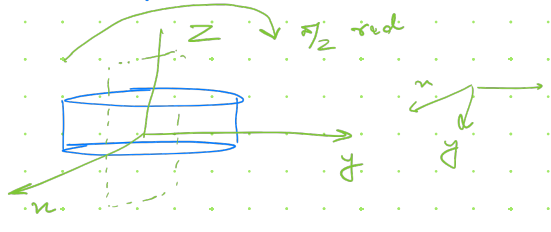
Changes in RVIZ, IN ORDER TO SEE

✓ tf, RobotModel

- change fixed frame → base_link
- In Robot Model → add description topic to → /robot_description

---

In ROS cylinders are by default oriented towards z-direction



but, we want along y-axis

→ Writing URDF, requires clarity in geometry of bot

Errors using log

> work on this ——, error message on cli, always mention something on launch files, which is not right.

After saving rviz file, we can launch that file —

> rviz2 -d path to the file

> ros2 run joint_state_publisher_gui joint_state_publisher_gui

(for launching fake joint publisher)

How to re-run?

1. Open Cli
2. go to ws
3. > source install/setup.bash
4. > colcon build --symlink-install
5. > ros2 launch my_bot rsp.launch.py
6. In other Cli > ros2 run joint_state_publisher_gui joint_state_publisher_gui
   fake joint publishing
7. In other cli > rviz2

Note: opening the rviz file from config doesn't work, not able to publish tf msg of joints

Driving your wheeled robot

✓ ros2 launch my_bot rsp_launch.py use_sim_time:=true

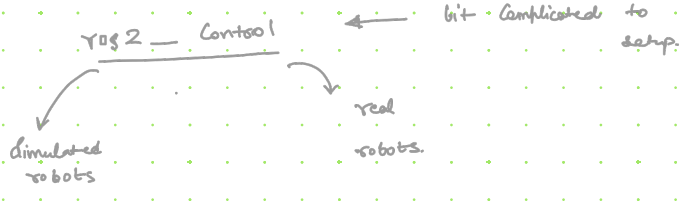
simulation synchronise time

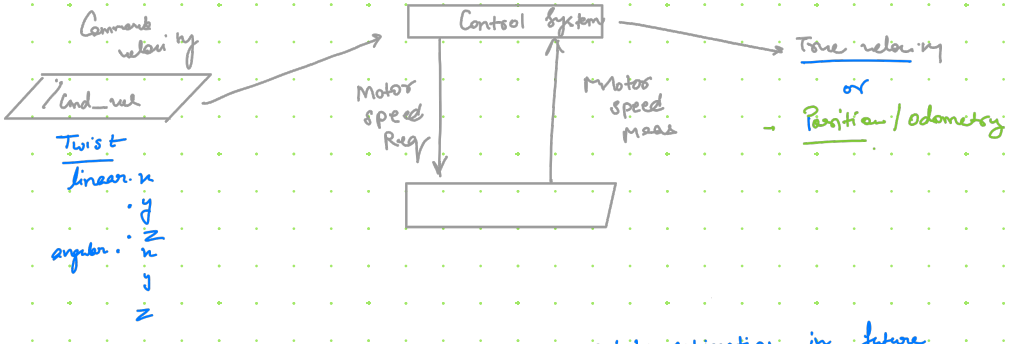
* Gazebo → error → not opening
  → uninstall & installed using binaries

launch file for launching in gazebo

> ros2 launch my_bot launch_sim.launch.py

> gazebo path setup is not correct





→ Robot position: estimating the robot estimation in future
Dead reckoning

→ Odometry: Resulting position estimate from dead reckoning

* Always pass full path starting from home → /home/user/.../... →

To launch bot & world →

> ros2 launch my_bot launch_sim.launch.py world:=pwd

How to integrate Rplidar → view results in Rviz

> setting up fixed-frame → manually →
> setting up topic
> launch file for Rplidar

Use Camera

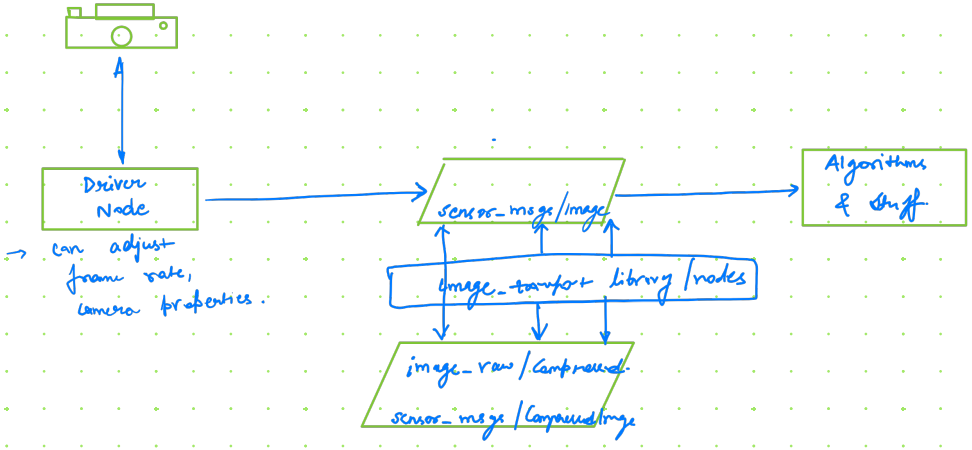
1. Intro 2. Camera → Ros2 3. Simulate virtual camera
4. Real.

1. Compression → png : lossless → exactly same (decompression)
   → jpeg : lossy → ...

focal length : dist of lens from its screen

Horizontal FOV :

$$h_{FOV} = 2\tan^{-1}\left(\frac{\text{sensor-width}}{2 \times \text{focal-length}}\right)$$



Coordinate frame

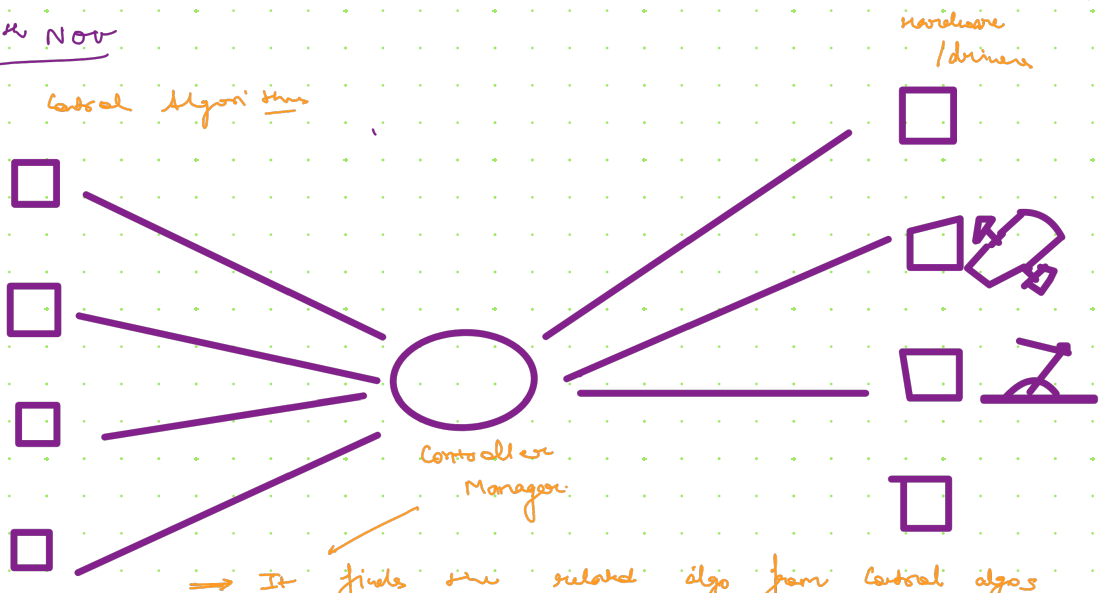
ROS Body standard

camera_link

Image standard

camera_link_optical

19th Nov



→ It finds the related algo from Control algos and links it to hardware.

→ It uses plugin system.

(things we control) (things we measure)

→ Command interface vs. State interface

✓ Resource Manager / Controller Manager { for multiple motors / actuators this offers all command & state interface and exposes it to hardware interface

✓ Controllers

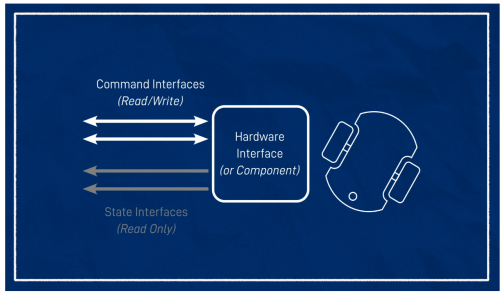
Q. How to write a hardware interface / hardware component for your robot?

Interacting with Controller Manager

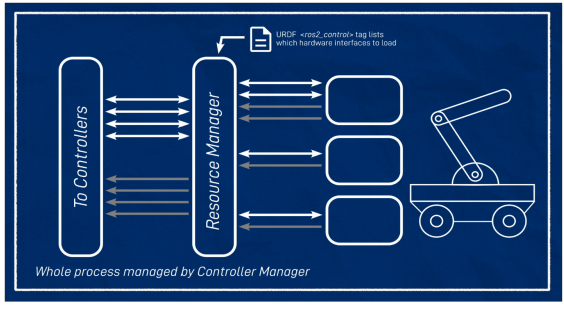
1. via ROS Services
2. via ros2 Control CLI tool
3. via specialised nodes / scripts

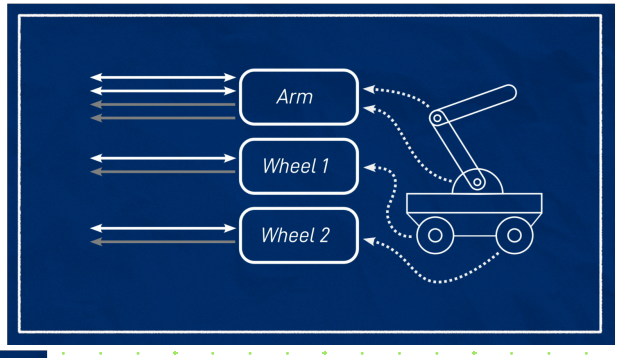
Q. How to write ROS2_ Control file?

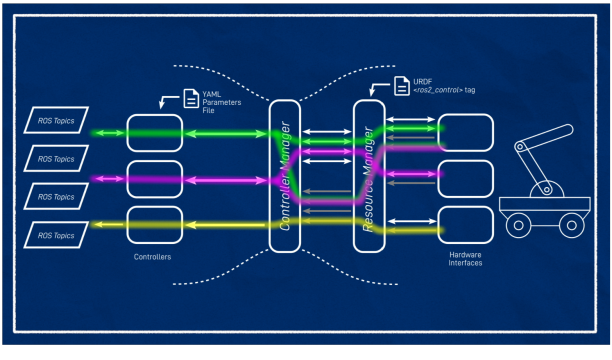
- ✓ plugin name →
- ✓ hardware interface
- ✓ command interface is the way through which we control the hardware
- ✓ state interface are used to read the hardware values.




```
dev@dev-PC:~/dev_ws$ ros2 control list_hardware_interfaces
command interfaces
        left_wheel_joint/velocity [unclaimed]
        right_wheel_joint/velocity [unclaimed]
state interfaces
        left_wheel_joint/position
        left_wheel_joint/velocity
        right_wheel_joint/position
        right_wheel_joint/velocity
dev@dev-PC:~/dev_ws$
```







> when not using Gazebo, we will need to run "ros2 run controller_manager ros2_control_node"

> To check for controllers

- ros2 control list_hardware_interfaces
  - ✓ command interfaces
    - left_wheel_joint/velocity [unclaimed]
    - right_wheel_joint/velocity [unclaimed]
  - ✓ state interfaces
    - left_wheel_joint/position
    - left_wheel_joint/velocity
    - right_wheel_joint/position
    - right_wheel_joint/velocity
- ros2 control list_controllers
  only shows controllers which are running
- ros2 run control [tab]
- ros2 screen controller_manager spawner diff_cont

Running things

ROS2 - CONTROL EXTRA BITS (for implementing in real bot)

How to launch basic setup of indoor bot with controller & Map ?

T1 → dev_ws $ source install/setup.bash

T1 → ros2 launch my_bot launch_sim.launch.py world:= Complete path

T2 → rviz 2

T3 → ros2 run teleop_twist_keyboard teleop_twist_keyboard

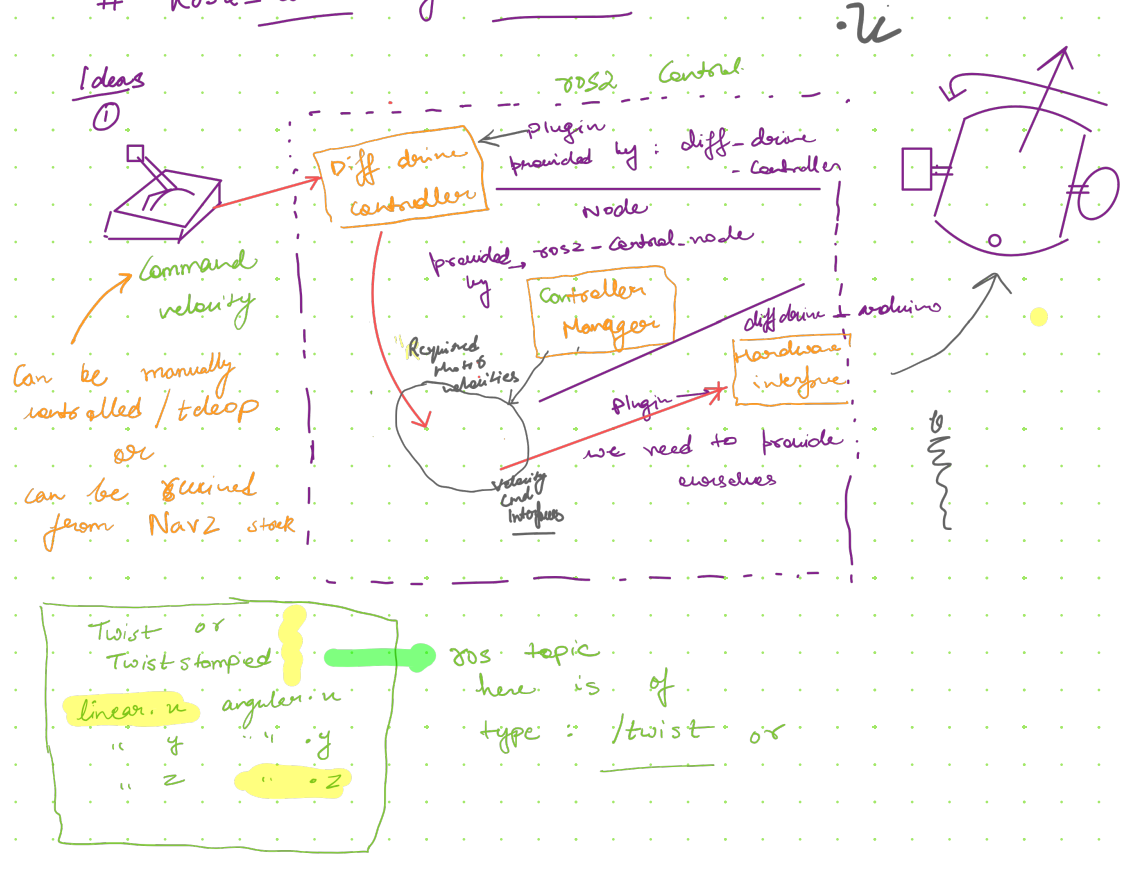
--ros-args -r /cmd_vel:=/diff_cont/cmd_vel_unstamped

T4 → ros2 topic list

Sol~ • Adjust the update rate very high by creating gazebo_sim.yaml file and then updating launch file

1. wheel slippage

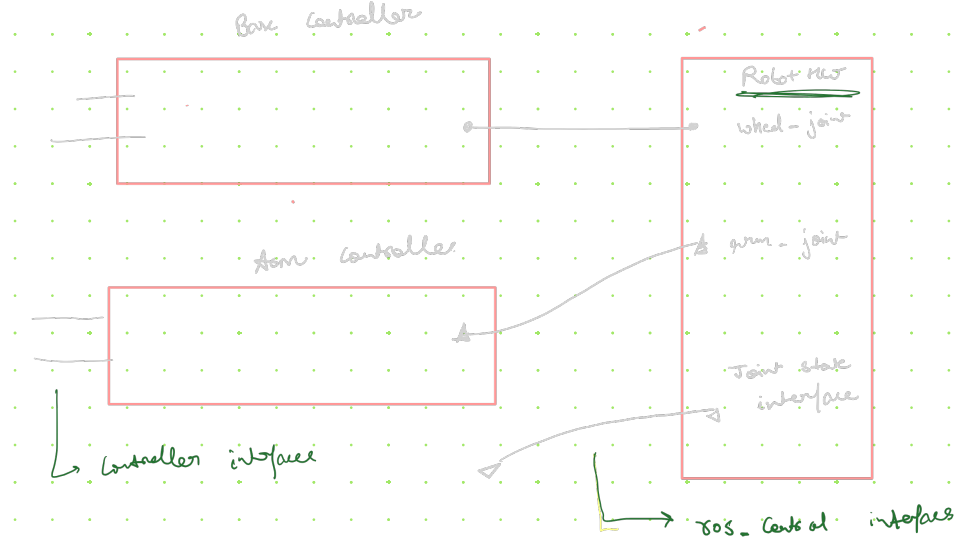
Ros2_Control for real robot



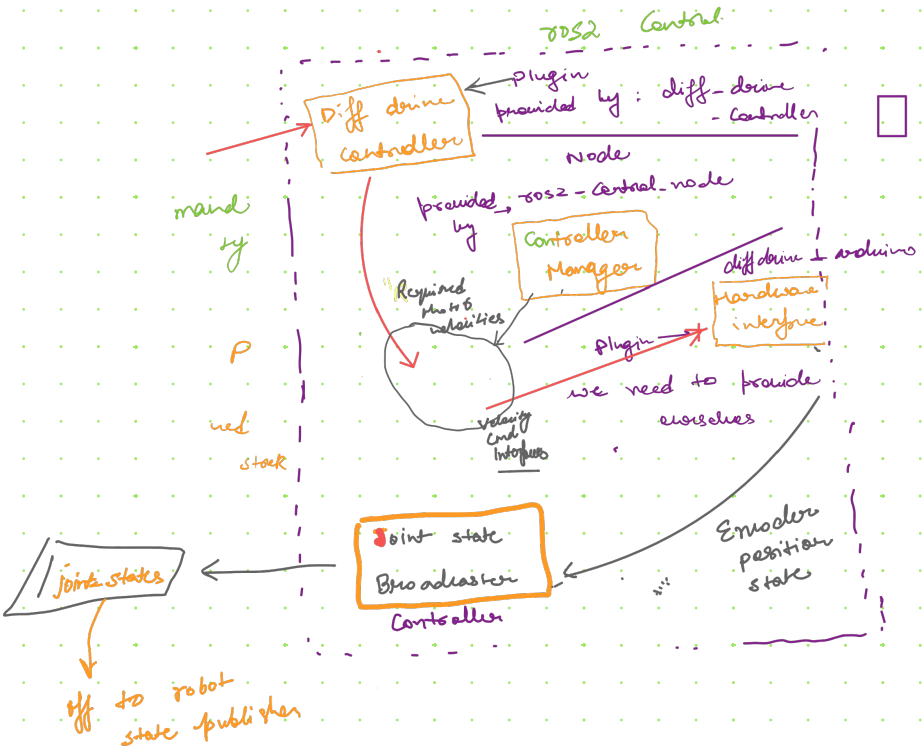
Jeam février 2014

1. Controllers

2. Robot requires abstraction



✓



Hardware Interface (diff_drive_arduino)

- ✓ for Ros2 ← something new
- ✓ vs editor remote deploy

✓ Steps for allocating Custom hardware interface

1. update ros2_control.xacro , add parameters
2. update launch file → launch_robot.launch.py
   so to call controller manager
   pkg : controller_manager
   exe : ros2_control_node
   param : [ ]

URDF → Controller Manager ← param.yaml

* gazebo_does

add line of robot_description = ( )
and controller_params_file = my_controller.yaml

3. Time_delay or Timer → for delaying launch of controller manager
4. spawn problem delay.
5. launch rviz2 from (Mer configuration { TF, Robot Model, LaserScan, Camera
6. teleop (Moving in rviz also)
7. encoder check → 360°

my_bot

8. Camera check

- ✓ source ~/.bashrc
- ✓ ros2 launch my_bot camera_launch.py

9. LiDAR

10. odometry issue → encoder_count_per_revolution = change

- ✓ Game Controller
- ✓ Camera feed on phone

Controller plugin : which Ackermann based steering is good for the hardware work?

TEB RPP MPPI VP

Controller	Dynamic obstacle Handling	Path Accuracy	Speed Handling	computational demand	Comment
TEB	Excellent	Moderate	Moderate	High	dynamic obstacle avoidance & smooth navigation
RPP	Limited	Excellent	Moderate	Low	Exact path following in static environment

Controller	Dynamic obstacle Handling	Path Accuracy	Speed Handling	computational demand	Comment
MPPI	Excell	Exc.	high	very high	Complex dynamic environment & precise Controls.
VP	Mod.	Mod.	Exc.	Mod.	High speed path tracking in open environment

Micro-ROS

→ a light-weight adaption of ROS2 designed for micro controllers.

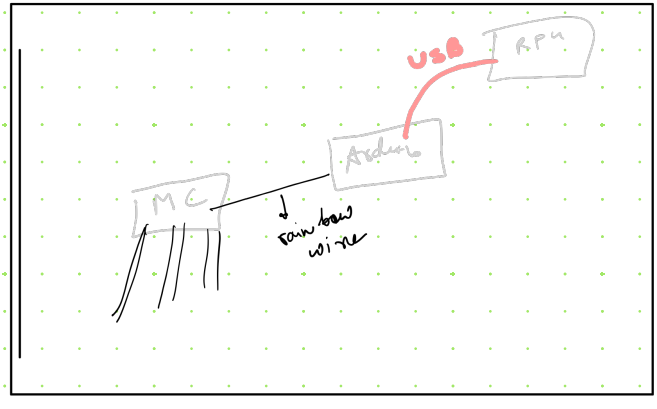
You will have as a host (Ubuntu), here ROS2 agent act as bridge for ROS2 full stack to microcontroller.

hardware-interface, ← pkg

Software representation of a robot

6 dof robot

- ros2 - Control overview
  - > writing a URDF
  - > writing a hardware interface
  - > writing a controller



Testing carlike robot ✓

① ✓ folder structure

WS
└ src
  └ ackermann_vehicle
    ├ launch
    │ ├ view_vehicle.launch.py rviz
    │ ├ simulation.launch.py sim
    │ └ hardware_control.launch.py hard
    ├ urdf
    │ ├ ackermann_vehicle.urdf.xacro
    │ ├ gazebo_plugins.xacro
    │ └ materials.xacro
    ├ config
    │ ├ controllers.yaml
    │ ├ gazebo_ros_control.yaml
    │ └ navigation.yaml
    ├ src
    │ ├ ackermann_vehicle_hardware.cpp
    │ └ main.cpp
    ├ include
    │ └ ackermann_vehicle utilities [header for .cpp]
    ├ CMakeLists.txt
    └ package.xml

Start a fresh

wrong right

I have to do what ?

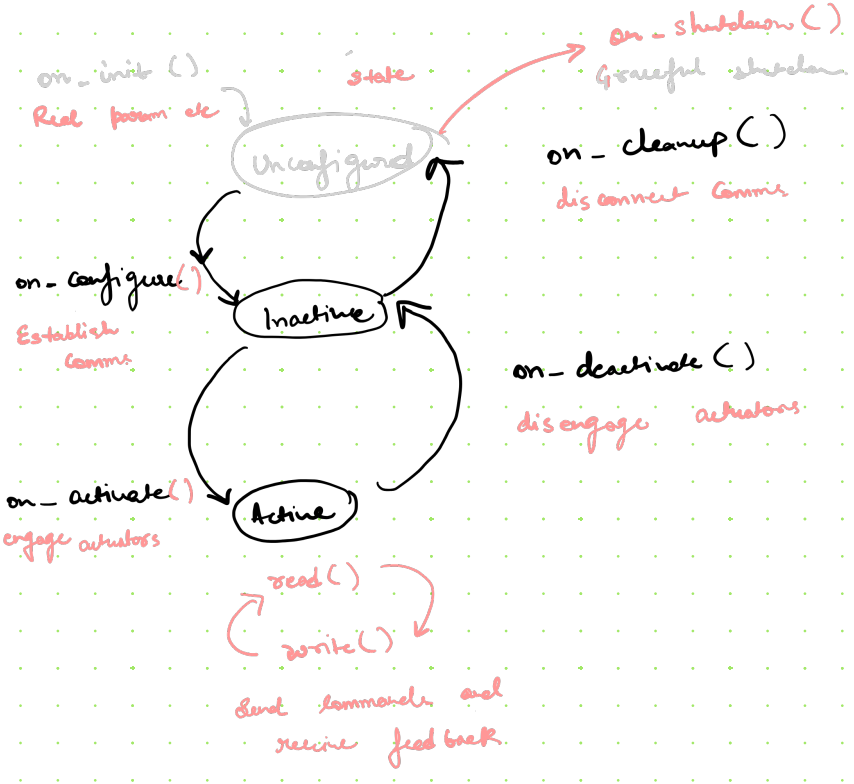
diff tutorials Carlike_bot

.cpp .cpp

copy from here paste

Compare with Carlike_bot

video → 1 ros2 lifecycle node → Lifecycle State



There are functions & states. Functions transist from one state to another.

→ Use ros2_demo as it is as a pkg, when using just the name of plugin

→ virtual motor, when running in RViz using teleop, we are using this.

Understanding from Josh — Cpp

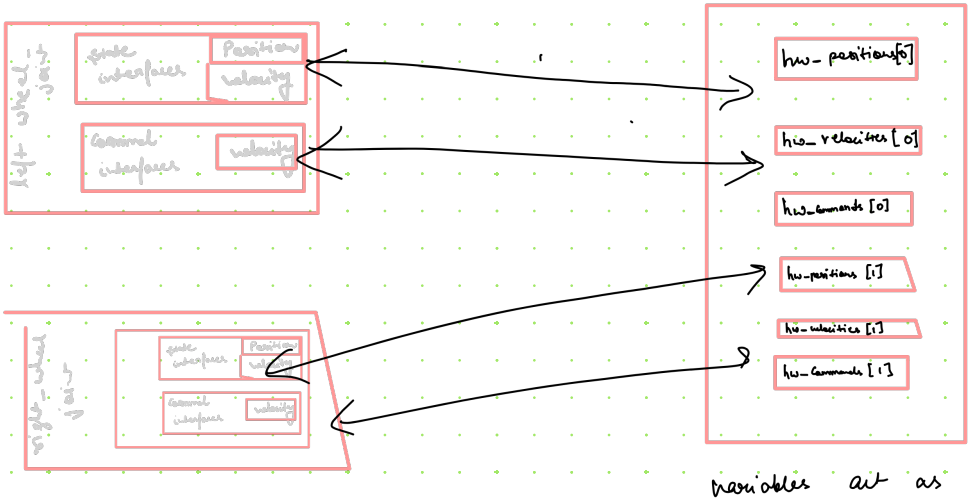
1. Start with .hpp, you see header files, which describes about interfaces or state transition functions.

2. Then, .cpp

3. init() → means as loading & checking joints → number
   → command interface
   → state interface

4. export_state_interface()
   &
   export_command_interface()
   → Its a way to tell ros2_control what hardware interface has accessibles.
   Things can be commanded & things we can get feedback on

Also, it interlinks of interface with variables inside the plugin



variables act as temporary container for write & read

5. on_activate : Its saying to awake the hardware & initializing some variables for use

6. on_deactivate : Shutdown of system

7. the last line of file says how to use plugin

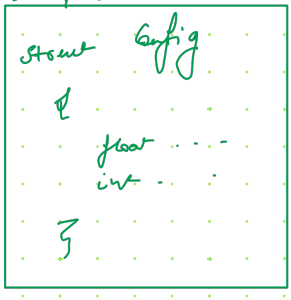
⇒ Connecting to hardware using SSH and coding using VS Code remote development tool.

Protocol

1. Start with ~~header~~ header file .hpp
2. create a [arduinoComms] object handles all communication from arduino.
3. Now we will call , in the state transition functions.
4. Now, edit in .cpp
5. four things to do → Connect to hardware
   disconnect to hardware
   write ~~from~~ to hardware
   read to hw.
6. on_activate() → // Comms_.Connect();
7. on_deactivate() → // Comms_.disconnect();
8. " →
9. " →

10. creating your own structure of storing parameter in .typ file

In class, add



left_wheel_name =
right1_wheel_name =
right2_wheel_name =

Now make a instance of Config.

11. on_init paste the param

12. Now, with the variable mentioned, we are assigning hardware param that in turn what to search for

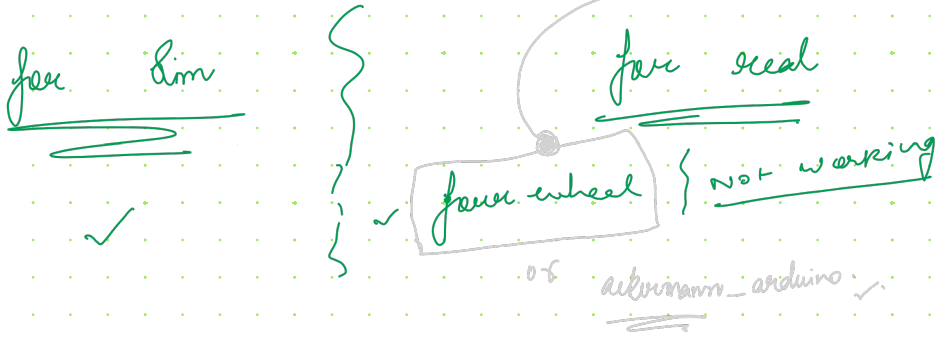
13. on_activate

In - wheel data → initialize

Tsrive → for real : diff drive_ arduino

for Sim : example 2 & 11

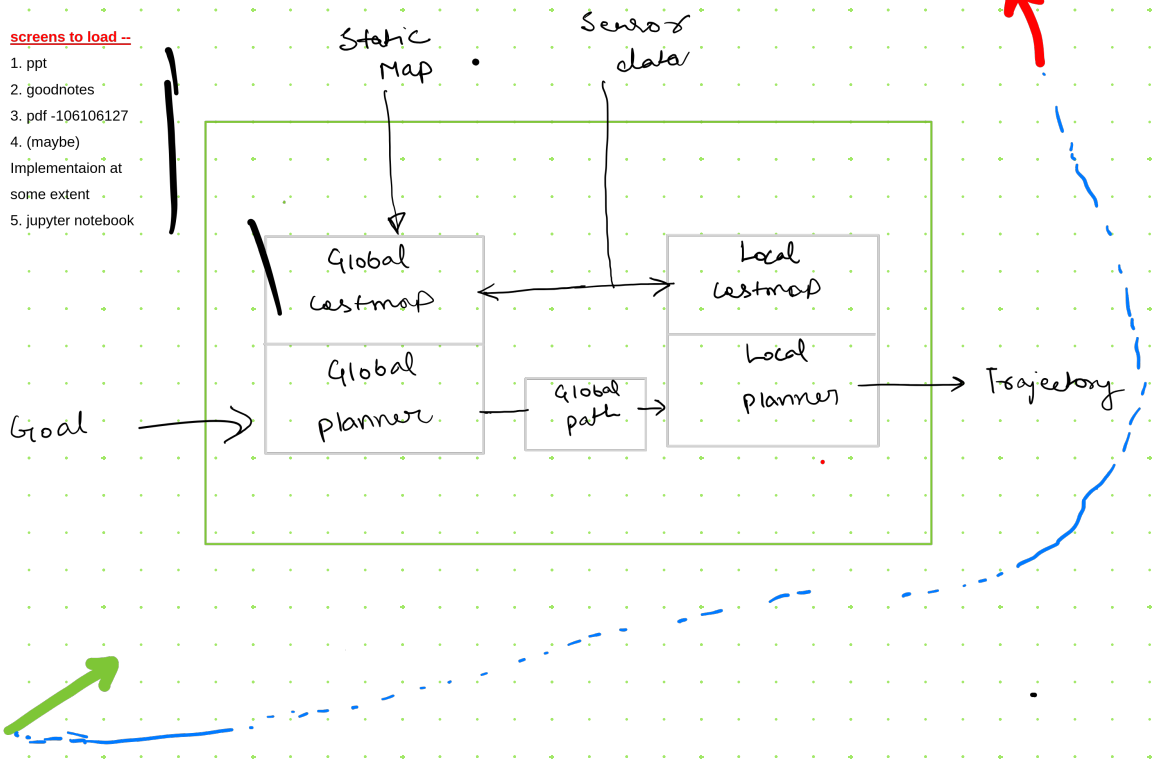
'DW' 'like_ours'



Internal Imp. of ROS Navigation

screens to load --

1. ppt
2. goodnotes
3. pdf -106106127
4. (maybe) Implementaion at some extent
5. jupyter notebook



what goes into costmap ?

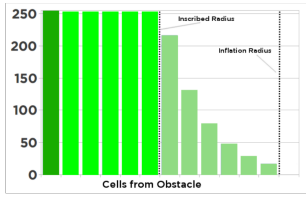
- ✔ static map created using G-Map
- ✓ presents obstacles
- ✓ contains sensors info

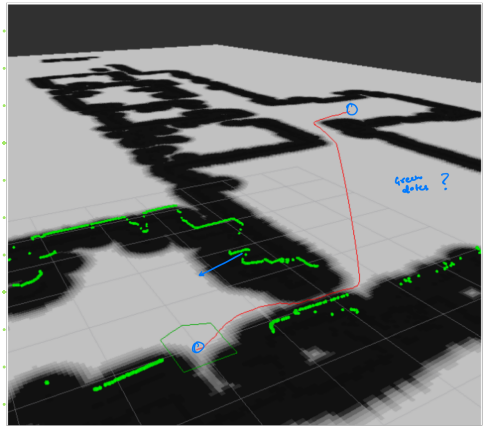
Obstacle inflation ← Comp. shortcut

X

- ✓ 

✓ using robots as a single point useful for working graph planning algo



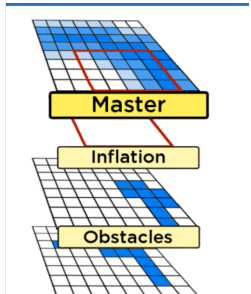


2 ways to approach — but which takes lesser time

works well with static scenario use case, which we will consider for our case, however in the use case of human interaction. The behaviour is based on considering human as a obstacle, for eg. robot drives straight towards the person to the front of person, then will start circumventing (closely) around a human , which is not nice. 05:41

Navigation illumination: Shedding light on the ROS navstack

different layers are just like plugin , which we can load and use by mentioning it in parameter file.
"Implementing a layer"

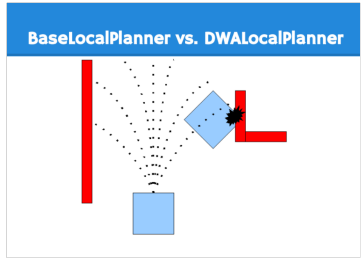


Costmap Layers

- Obstacles Layer
- Inflation Layer
- Static Layer
- Range Sensor Layer
- Proxemic Layer
- Claustrophobic Layer

commonly used global_planner is NavFn, which is based on dijsktra algrothium using gradient descent.

Local planner create a number of candiadate traje. based on the range of velocity given by user in the direction. Then score a traje. which one does the best.



Scoring Trajectories
Weighted Sum =
oscillation_cost
+ costmap_cost
+ goal_distance_cost
+ path_distance_cost
+ goal_alignment_cost
+ path_alignment_cost

← Now choosing the weighs and cost function is based on the behaviour required and its tricky.

The above different cost is also based on plugin which we can utilise and add based on the use case.

what is the difference b.w global and local costmap implementation?
Entire, not static map in local, reference frame one is fixed, other based on odom

ref: https://roscon.ros.org/2014/wp-content/uploads/2014/07/ROSCON2014_DLu.pdf

Check it!

MoveBase,

local planner implementaions ,

using SLAM there is a possibility whole space doesnt need to studied before starting navigation ?

Screens to l

IMU

✓ Error modelling

- Sources of error

→ Misalignment of axis & scale factor : Ma & Mg (sensor sensitivity & axis)

→ bias : ba , bg

→ Random noise : Wa & Wg

Eqn of error modelling →

Accelerometer

$$\tilde{f} = b_a + (I_3 + M_a) f + w_a$$

Gyroscope

$$\tilde{\omega} = b_g + (I_3 + M_g) \omega + w_g$$

Terms :

1. $\tilde{f}$, $\tilde{\omega}$: measured o/Ps
2. f, ω : true values (force & angular)
3. ba & bg : Const offset
4. Ma & Mg : Matrices for scale factor errors & misalignment errors

$$M_a = \begin{pmatrix} s_{a,x} & m_{a,xy} & m_{a,xz} \\ 0 & s_{a,y} & m_{a,yz} \\ 0 & 0 & s_{a,z} \end{pmatrix}$$

$$M_g = \begin{pmatrix} & & \\ & — & \\ & & \end{pmatrix}$$

a —

g —

S → scale factor errors

$m_{xa}, g_{\xi}, \alpha\beta$: cross - coupling errors { non - orth... of β & α

$$R = I_3 + [\epsilon]_\times \} \rightarrow \text{skew-symm. matrix}$$

misalignment rotation matrix

Convert gyroscope reading into acceleration ✓

$$w^a = R^{-1} w^g$$

Assmp:

1. Quantization error assumed small $\Rightarrow$ neglected
2. Non-linearities of IO/OT curves $\Rightarrow$ "

Axis	Time start	Time end
+x	14:07:18	14:07:38
	14:07:57	14:08:15
-x	14:08:30	14:08:45
+y	14:09:17	14:09:30
	14:09:43	14:10:01
-y	14:10:27	14:10:44
	14:11:03	14:11:20
+z	14:11:52	14:12:09
	14:12:23	14:12:39
-z	14:12:58	14:13:15
	14:13:30	14:13:46
	14:14:06	14:14:23

Rot

Gyro

	Start	end
x	14:37:25	14:38:14
y	14:38:30	14:39:04
z	14:39:16	14:39:48

Acc

	6238	6447	
+x			
-x	6958	7116	
+y	7687	7876	
-y	8488	8646	
+z	9287	9456	
-z	9957	10126	

Gyro

+x	11543	12042	
+y	12193	12542	
+z	12653	12982	

```
ros2 topic echo /odom --csv > odom_data.csv
rviz2 -d slam_map.rviz
ros2 bag record -o slam_debug /scan /imu -
```