This is a PDF export of the talk slides.

You can find the original slides, incl. videos, here: https://docs.google.com/presentation/d/16ZQMB7OVc1fkPRG1K7rQIWCd5zYWTj1hTaeI2rAjZSo/edit?usp=sharing

Learning Robotics Fundamentals with ROS 2 and modern Gazebo

Andreas Bihlmaier

The goal

Exploring robotics fundamentals - interactively

Hands-On Learning: Engage in practical, hands-on experience with robotics fundamentals through interactive simulation tools.

Real-Time Visualization: Gain a deeper understanding of concepts by visualizing robot movements, sensor data, and algorithms in real-time through interactive plotting.

Risk-Free Experimentation: Mitigate risks associated with hardware experimentation by utilizing simulation environments, allowing learners to experiment freely without fear of damaging physical components.

Scenario Exploration: Simulate diverse scenarios and challenges, providing learners with a broad range of experiences that may not be easily achievable in a physical setting.

Familiarity with Tools: Building familiarity with ROS 2 and Gazebo during the learning phase enables individuals to efficiently contribute to ongoing projects or start new ones.

The tools

Leveraging the ROS 2 and Gazebo ecosystem

ROS 2

- `rclpy` Python client library
- `ros2` command line tools
- rosbag2 recording & playback
- PlotJuggler advanced plotting

PlotJuggler
Juggle with data

- Not used in this talk
  - RViz 3D visualization
  - Turtlesim simple simulation

Connectors:

- URDF and SDF
- ros_gz
- gz_ros2_control

Gazebo (modern)

- gz-physics physics engines (Bullet, DART, TPE)
- gz-rendering rendering (OGRE)
- gz-sensors sensor simulation
- gz-gui GUI
- Many existing systems and example worlds
- Custom systems (plugins)
- Fuel simulation models

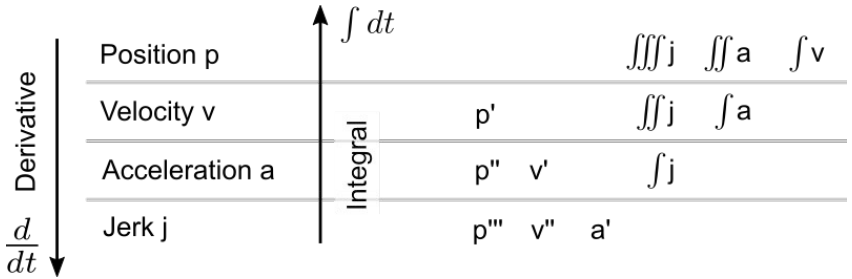
DART
Dynamic Animation and Robotics Toolkit

OGRE

Learning about math

Exploring math concepts in a robotics context

- The derivative of a function describes the “instantaneous rate of change” of the function.
- The time derivative of position is velocity
  - p′ = v
- The derivative of velocity is acceleration
  - p″ = v′ = a
- Integrals can be seen as antiderivative.
- The time integral of acceleration is velocity
  - ∫a = v
- The integral of velocity is position
  - ∫∫a = ∫v = p
- In code, derivatives and integrals are usually calculated numerically.

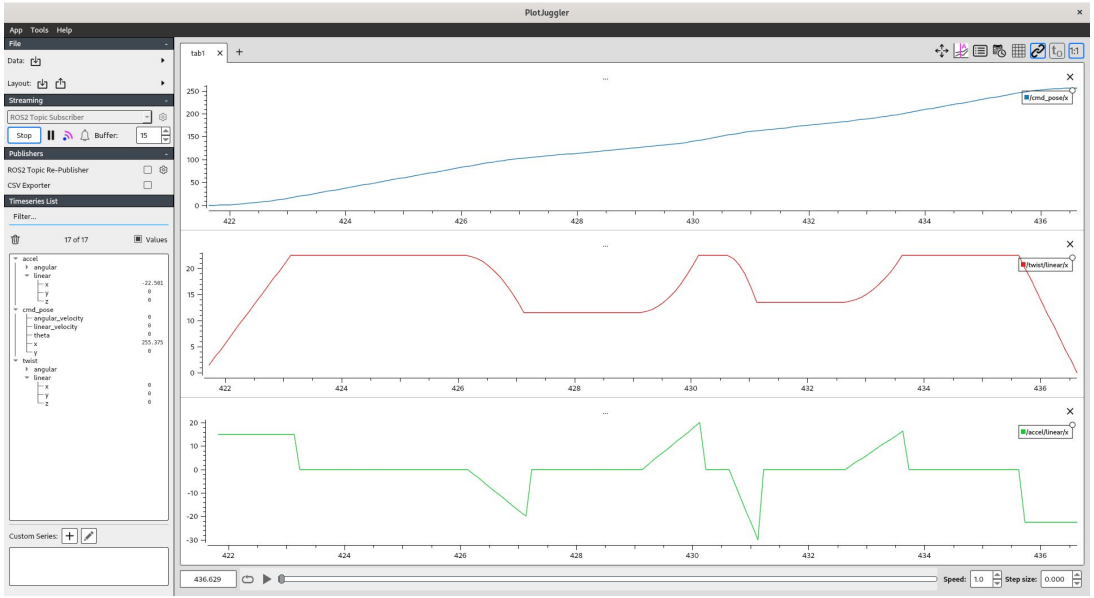


A more fun alternative to Python + matplotlib

There is nothing wrong with getting a better understanding of basic calculus using Python, e.g. numpy + matplotlib (in Jupyter notebooks).

One can get a sense for a constant acceleration resulting in linear velocity change that in turn results in quadratic position change.

But it is much more interesting and interactive using Gazebo and PlotJuggler.



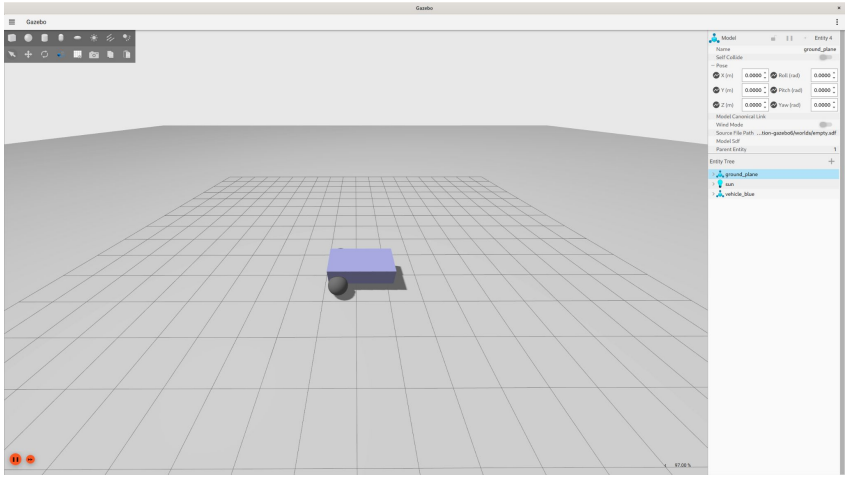
gz: Velocity v to Position x | PJ: x to v to a

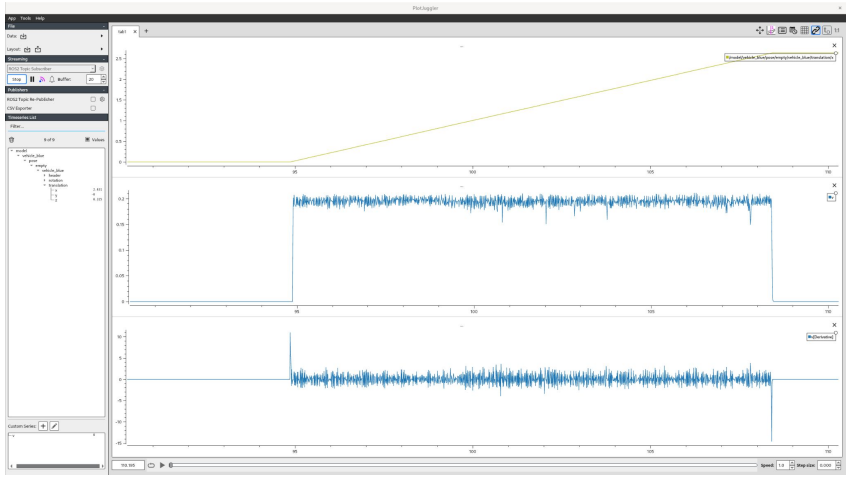
Gazebo (gz) world with a simple mobile robot with `VelocityControl` system and `PosePublisher` system.

`ros_gz_bridge` publishes `Twist` from ROS to Gazebo and Pose from Gazebo to ROS `TFMessage`.

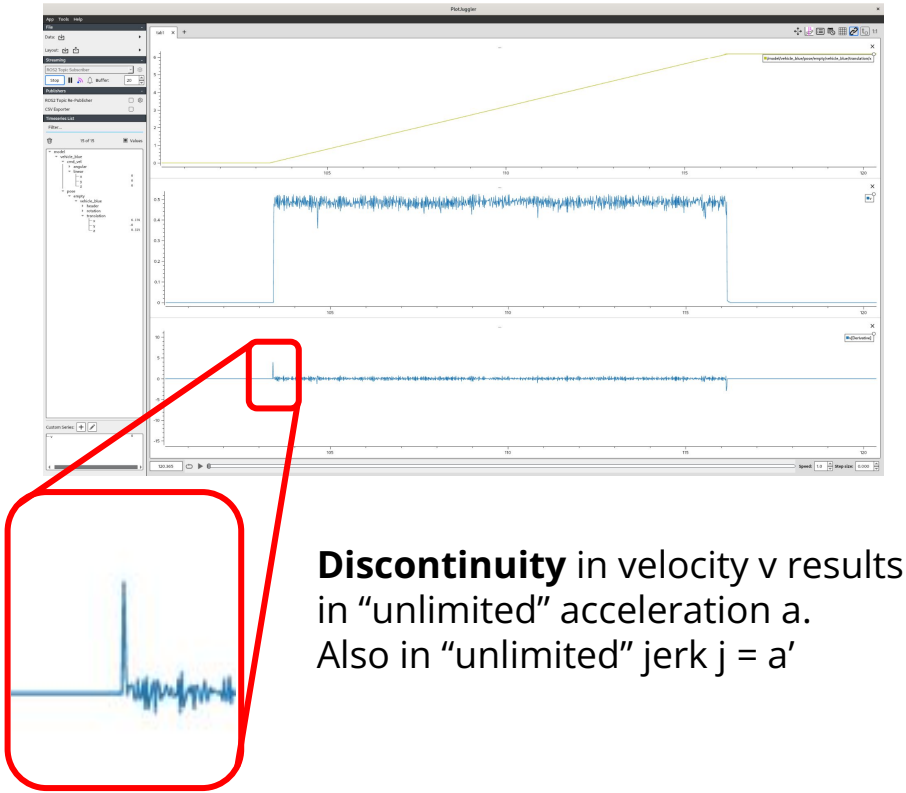
`ros2 topic pub` sends Twist command.

PlotJuggler (PJ) subscribed to ROS TFMessage topic's `translation.x` with custom series for first derivative v and filter for second derivative a.





M1: A gentle twist 1/2

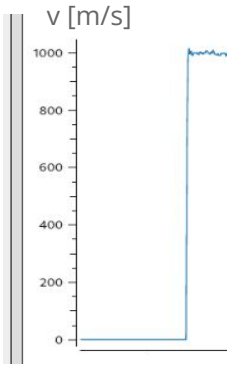


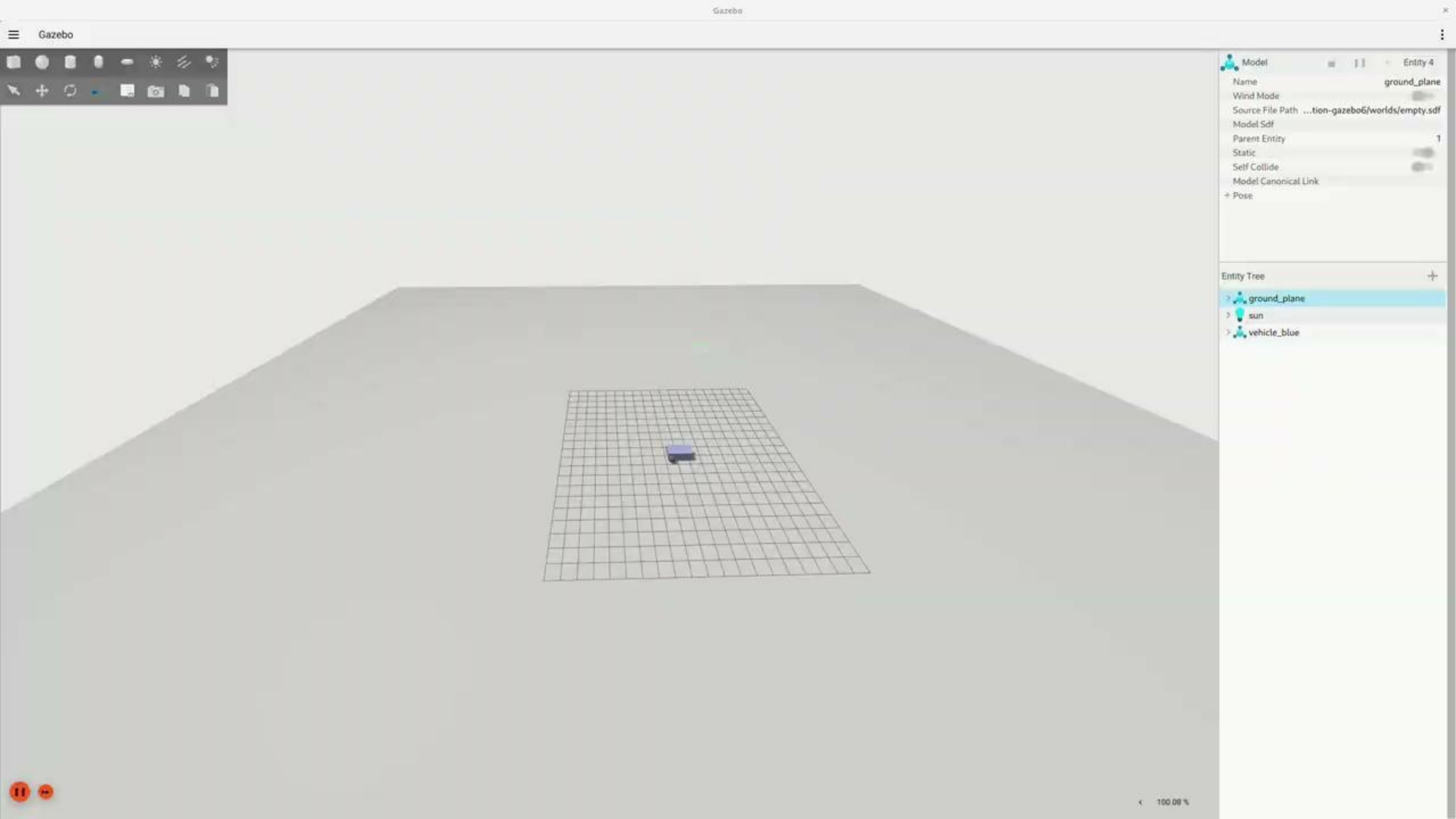
Discontinuity in velocity v results in “unlimited” acceleration a.
Also in “unlimited” jerk j = a’

Unless lower level controllers can properly handle discontinuities in commanded twist, commands should be **acceleration limited** (even better also jerk limited).

The noise in velocity and acceleration is due to (double) numerical differentiation.

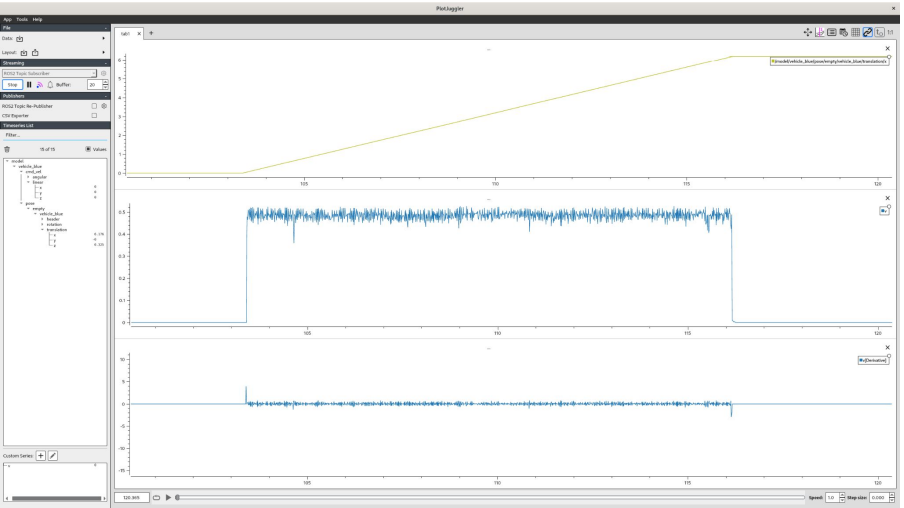
Aside: It is easy to create the world’s fastest robot - in simulation.

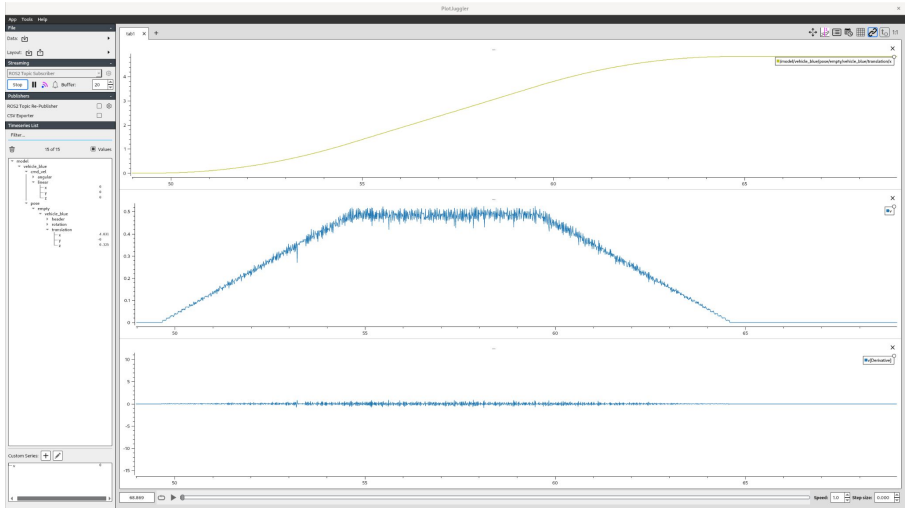



Gazebo
Model
Entity 4
Name
ground_plane
Wind Mode
Source File Path
...tion-gazebo6/worlds/empty.sdf
Model Sdf
Parent Entity
1
Static
Self Collide
Model Canonical Link
Pose
Entity Tree
ground_plane
sun
vehicle_blue
100.00 %

M1: A gentle twist 2/2

```
def timer_callback(self):
    if self.state == self.State.ACCELERATE:
        twist.linear.x = min(self.linear_velocity, self.prev_twist.linear.x + self.linear_acceleration * self.timer_period)
    elif self.state == self.State.DECELERATE:
        twist.linear.x = max(0.0, self.prev_twist.linear.x - self.linear_acceleration * self.timer_period)
```





Learning about physics

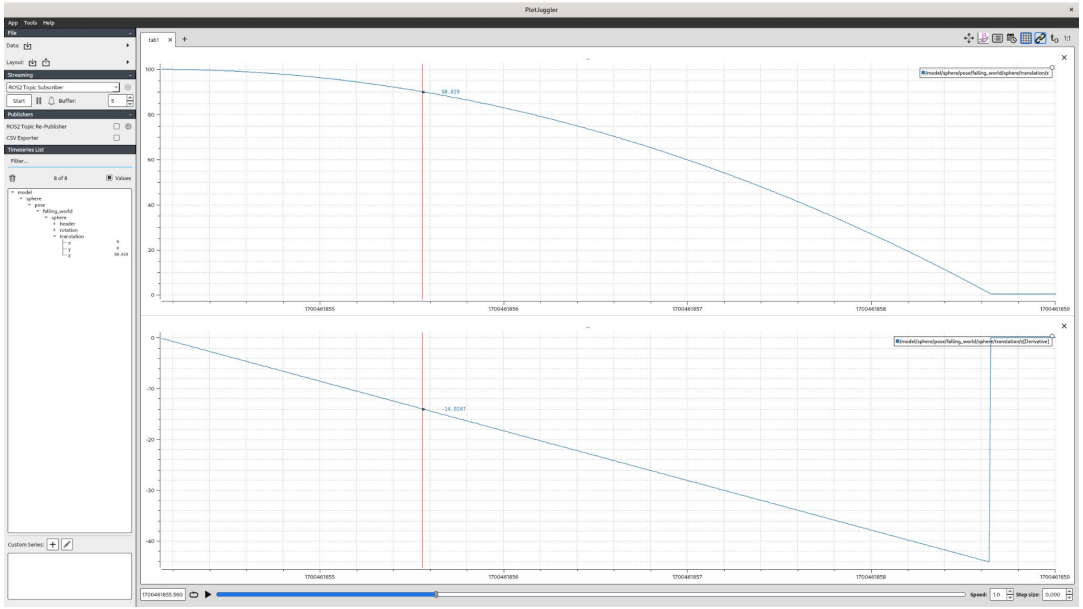
P1: Watch the ball drop in free fall

Paused “empty” world in Gazebo with a sphere at z = 100 m.

`PosePublisher` system and `ros_gz_bridge` as before.

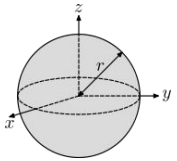
`ros2 bag record` the topic.

Open bag file in PlotJuggler, configure plot(s) and explore data with the time tracker.



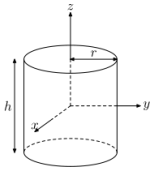
P2: Inertia and the inclined plane

The 3D inertia tensor for a solid sphere of radius r and mass m is



$$I = \begin{bmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{bmatrix}$$

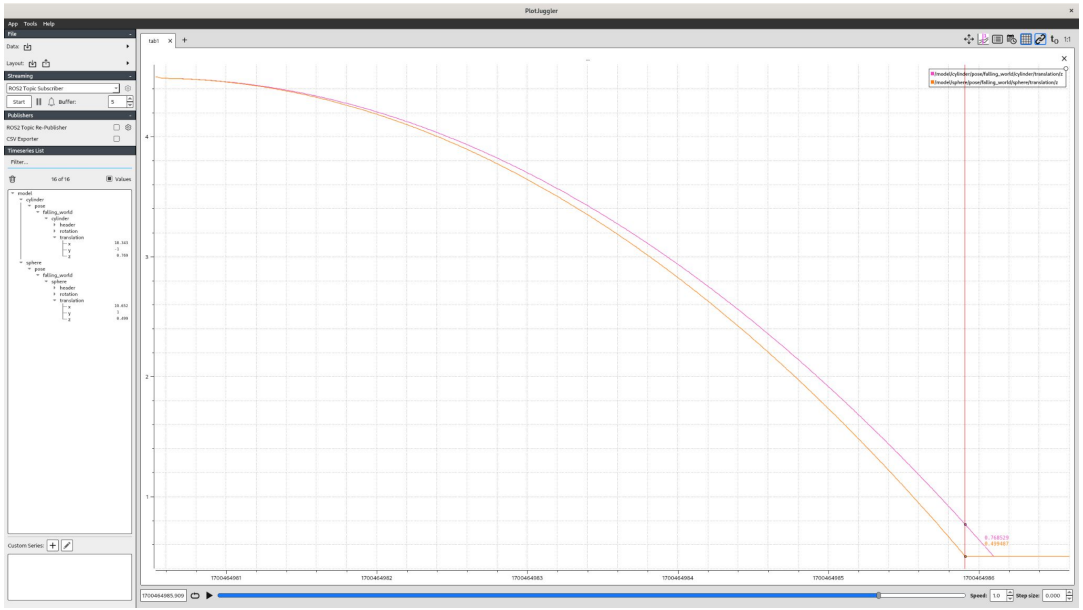
The inertia for a solid cylinder of radius r, height h and mass m is



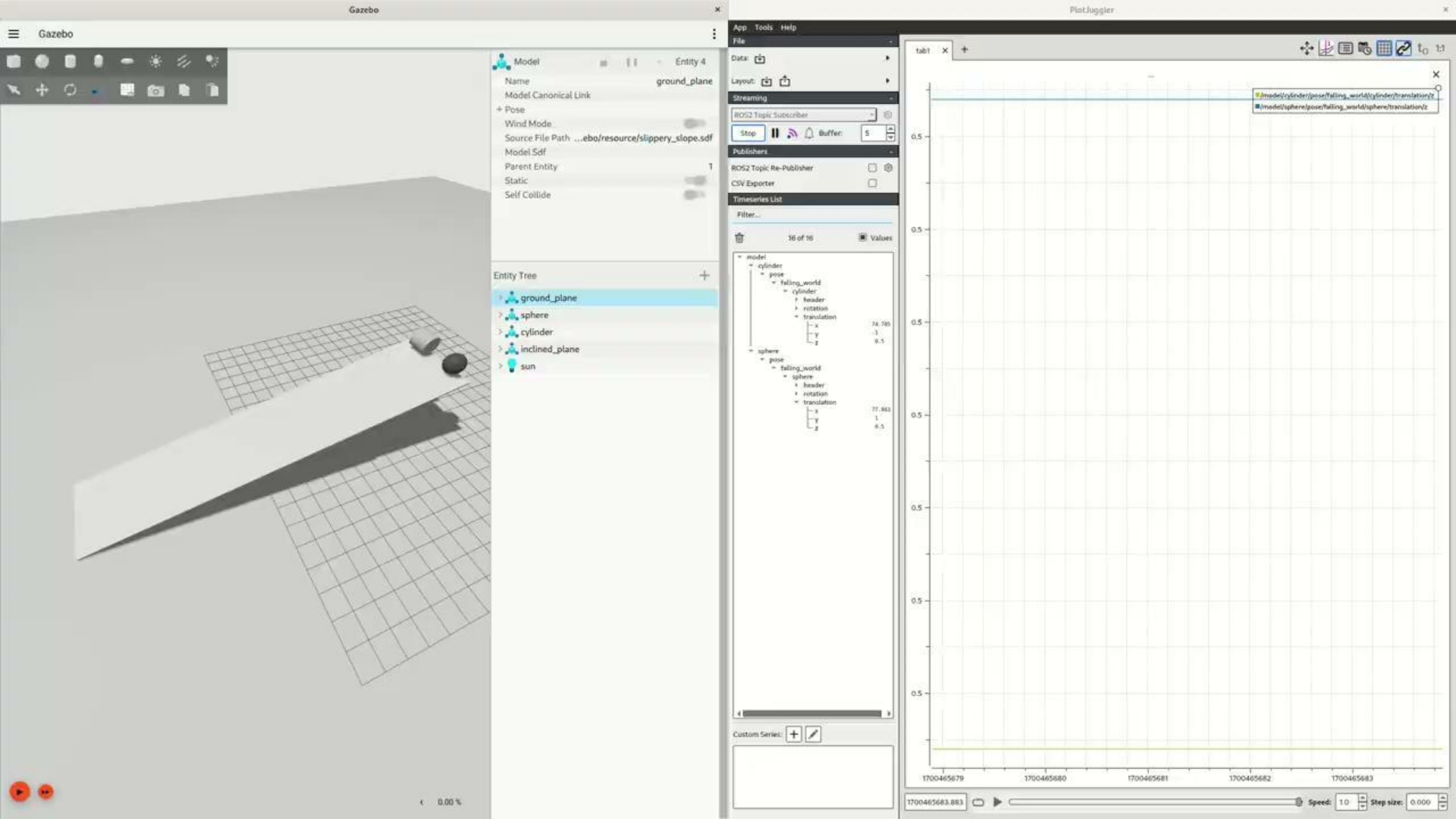
$$I = \begin{bmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix}$$

via List of moments of inertia

What happens when both roll down an inclined plane?



Aside: The unit matrix is **not** a good default value for robot links.

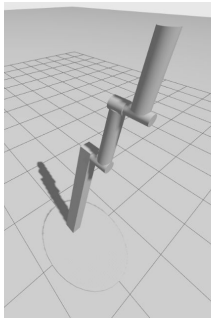

Gazebo
Gazebo
Model
Entity 4
Name
ground_plane
Model Canonical Link
Pose
Wind Mode
Source File Path
...ebo/resource/slippery_slope.sdf
Model Sdf
Parent Entity
1
Static
Self Collide
Entity Tree
ground_plane
sphere
cylinder
inclined_plane
sun
0.00 %
PlotJuggler
App
Tools
Help
File
Data:
Layout:
Streaming
ROS2 Topic Subscriber
Stop
Buffer:
5
Publishers
ROS2 Topic Re-Publisher
CSV Exporter
Timeseries List
Filter...
16 of 16
Values
model
cylinder
pose
falling_world
cylinder
header
rotation
translation
x
y
z
74.785
-1
9.5
sphere
pose
falling_world
sphere
header
rotation
translation
x
y
z
77.463
1
6.5
Custom Series:
tab1
1:1
/model/cylinder/pose/falling_world/cylinder/translation/z
/model/sphere/pose/falling_world/sphere/translation/z
0.5
1700465679
1700465680
1700465681
1700465682
1700465683
1700465683.883
Speed:
1.0
Step size:
0.000

Learning about robot physics

R1: The grandfather pendulum

Aside: Inertia parameters in standard model are completely off.
I'm sure there are reasons for the default - but they are not simulation realism.

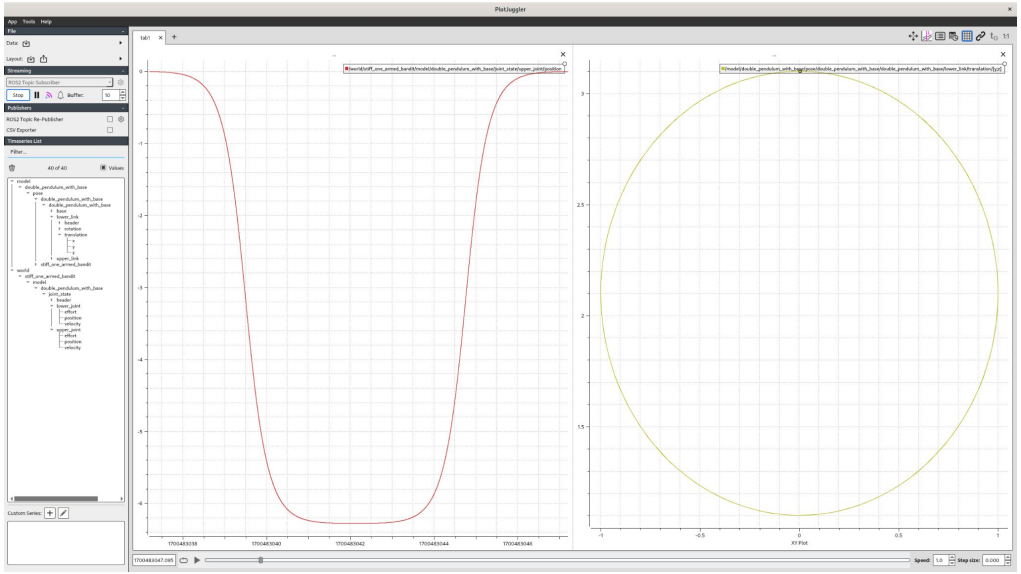
Modified `double_pendulum_with_base` with `ApplyJointForce`, `JointStatePublisher` and `PosePublisher` systems.

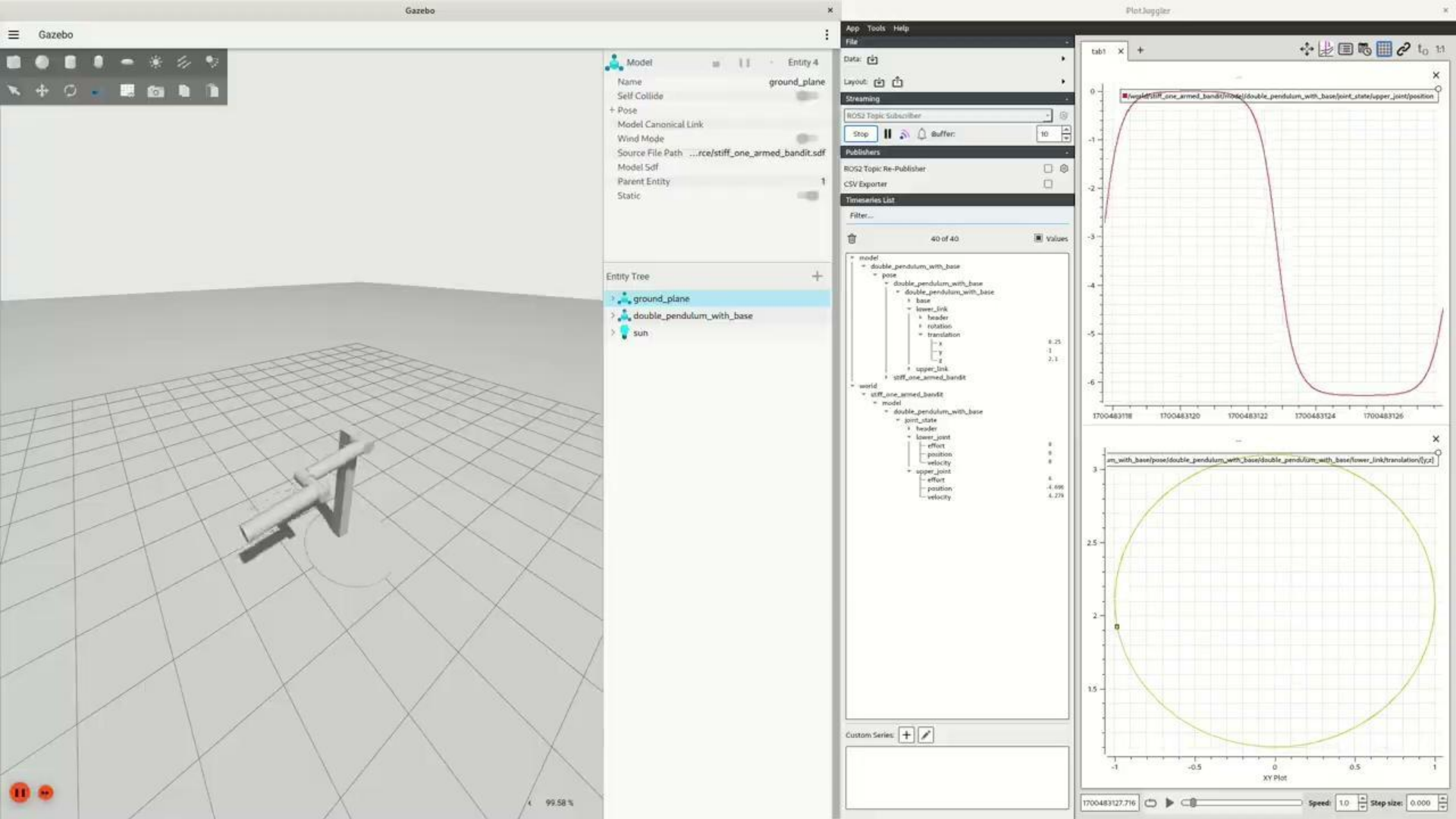


Here the `lower_joint` has been made a `fixed` joint. *Single* pendulum here.

The joints have no friction or damping. Perpetuum mobile!

PlotJugger visualizing the periodic joint position and XY plotting the `lower_link`.




Gazebo
Gazebo
Model
Entity 4
Name
ground_plane
Self Collide
Pose
Model Canonical Link
Wind Mode
Source File Path
...rce/stiff_one_armed_bandit.sdf
Model Sdf
Parent Entity
1
Static
Entity Tree
ground_plane
double_pendulum_with_base
sun
99.58 %
PlotJuggler
App
Tools
Help
File
Data:
Layout:
Streaming
ROS2 Topic Subscriber
Stop
Buffer:
10
Publishers
ROS2 Topic Re-Publisher
CSV Exporter
Timeseries List
Filter...
40 of 40
Values
model
double_pendulum_with_base
pose
base
lower_link
header
rotation
translation
x
y
z
upper_link
stiff_one_armed_bandit
world
joint_state
lower_joint
effort
position
velocity
upper_joint
Custom Series:
tab1
/world/stiff_one_armed_bandit/model/double_pendulum_with_base/joint_state/upper_joint/position
1700483118
1700483120
1700483122
1700483124
1700483126
XY Plot
1700483127.716
Speed:
1.0
Step size:
0.000

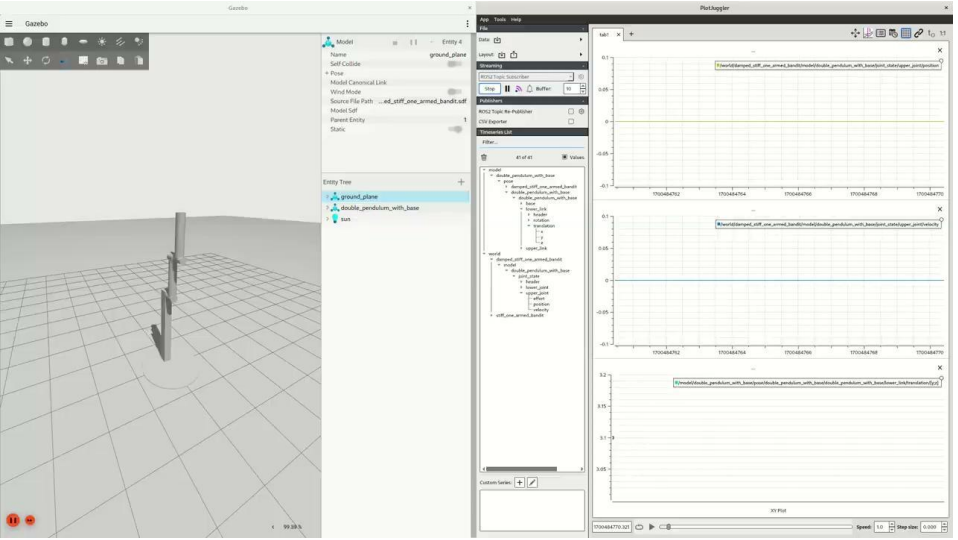
R2: Friction and damping

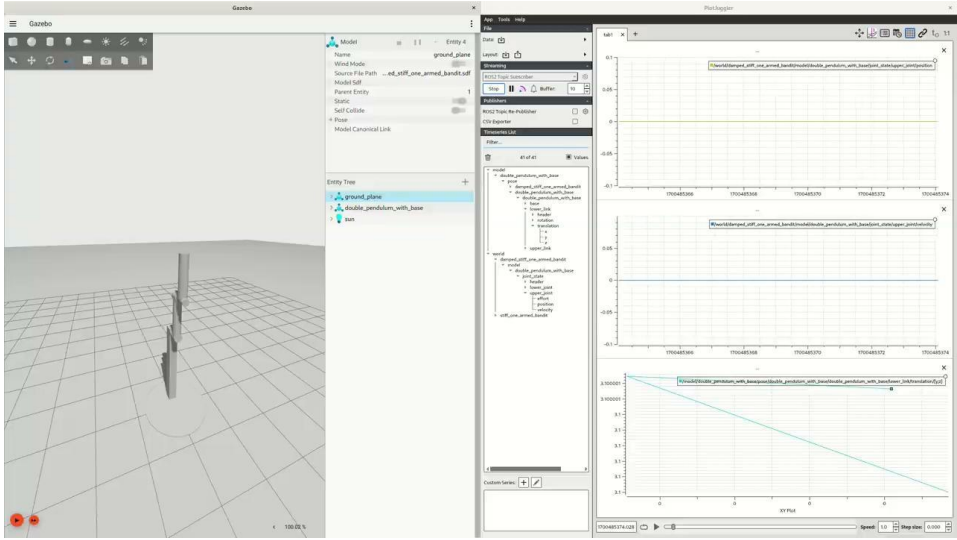
`<friction>`: Static friction

Independent of joint velocity.

`<damping>`: Viscous damping

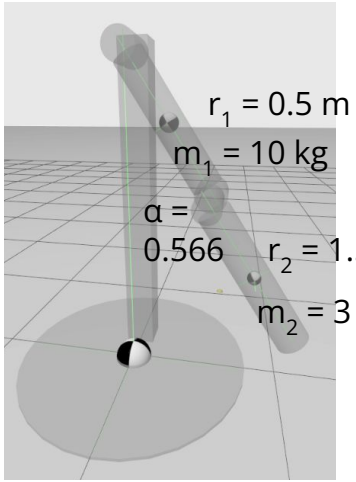
Relative to joint velocity.





R3: Inertia dependent on joint positions ⟶ M(q) 1/2

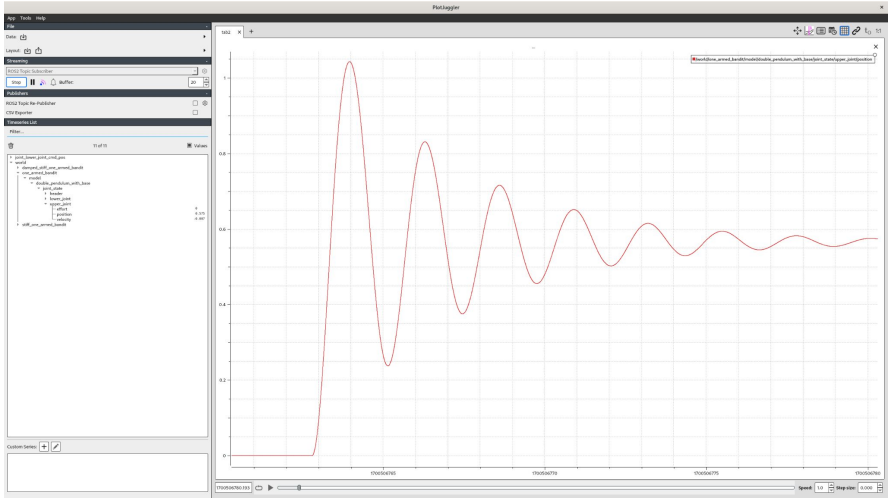
A heavily damped version of the double pendulum with two `revolute` joints. `upper_joint` as before, `lower_joint` controlled via `JointPositionController` system. Lower joint at position $q_2 = 0$.



$$\tau = (r_1 * m_1 * \sin(\alpha) + r_2 * m_2 * \sin(\alpha)) * g \cong 50 \text{ Nm}$$

Starting from bottom position, applying a torque of 50 Nm to the upper joint.

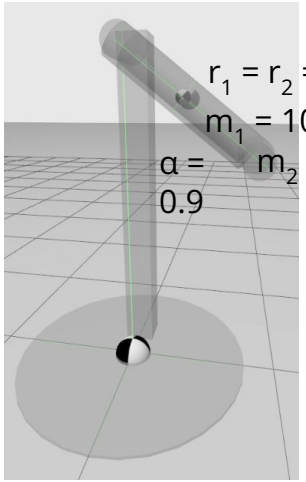
Settles at an upper joint position of 0.566 rad.



R3: Inertia dependent on joint positions ⟶ M(q) 2/2

Lower joint at position $q_2 = \pi$.

Settles at an upper joint position of 0.9 rad.



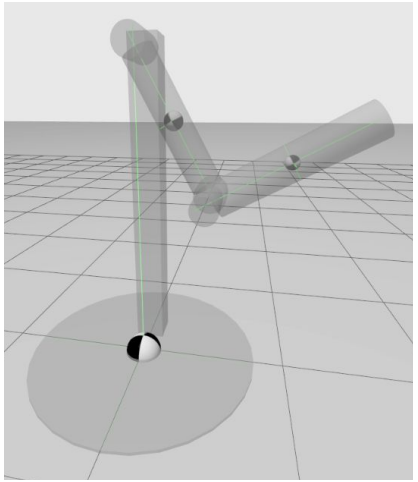
$\tau = (r_1 * m_1 * \sin(\alpha) + r_2 * m_2 * \sin(\alpha)) * g \cong 50$ Nm

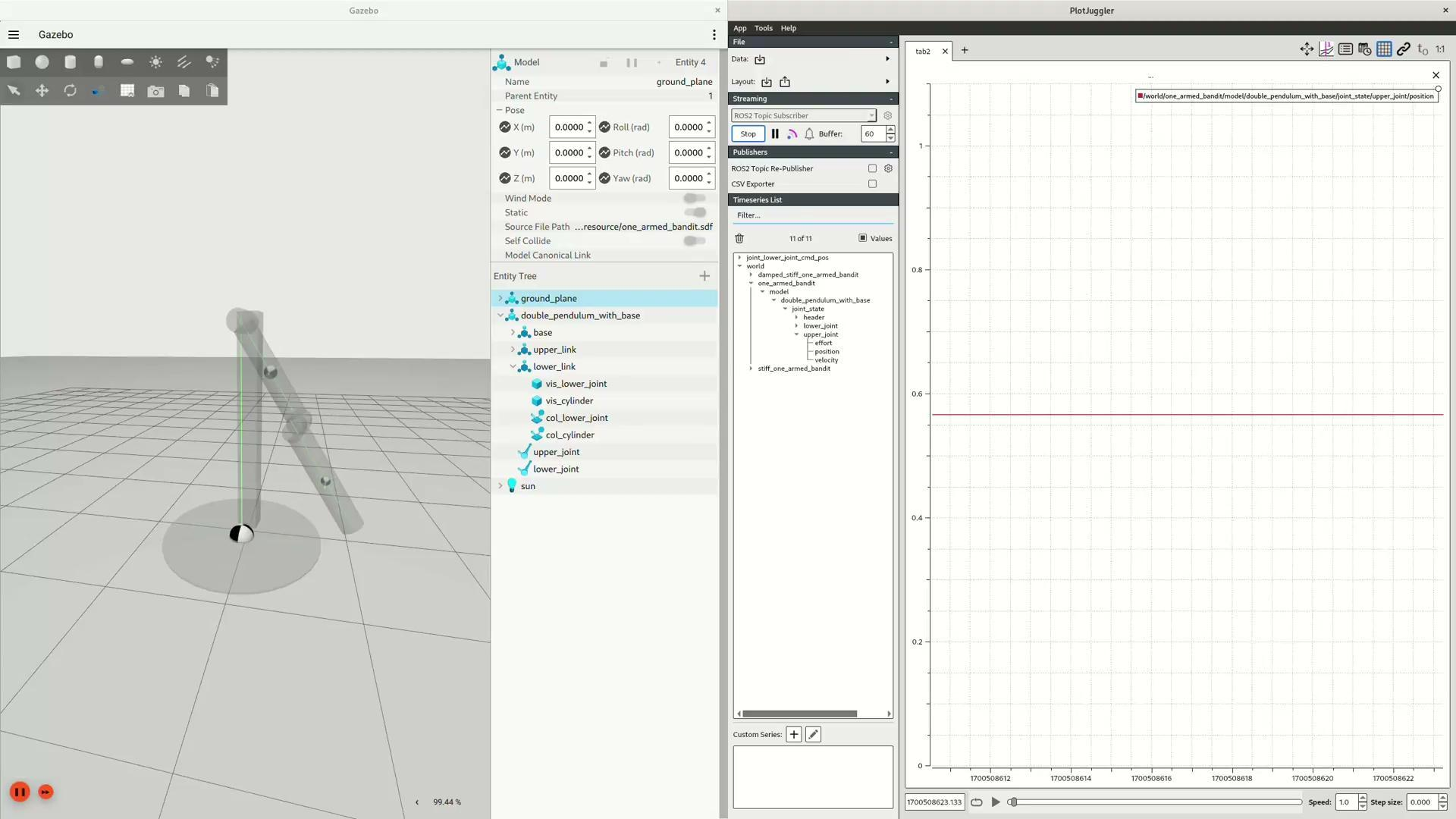
Lower joint at position $q_2 = \pi/2$.

Settles at an upper joint position of 0.49 rad.

In summary, seen from the upper joint, the robot inertia depends on other joint positions and the relationship is non-linear:

$q_2 = 0 \longrightarrow \alpha = 0.566$
$q_2 = \pi/2 \longrightarrow \alpha = 0.49$
$q_2 = \pi \longrightarrow \alpha = 0.9$




Gazebo
Gazebo
Model
Entity 4
Name
ground_plane
Parent Entity
1
Pose
X (m)
0.0000
Roll (rad)
0.0000
Y (m)
0.0000
Pitch (rad)
0.0000
Z (m)
0.0000
Yaw (rad)
0.0000
Wind Mode
Static
Source File Path
...resource/one_armed_bandit.sdf
Self Collide
Model Canonical Link
Entity Tree
ground_plane
double_pendulum_with_base
base
upper_link
lower_link
vis_lower_joint
vis_cylinder
col_lower_joint
col_cylinder
upper_joint
lower_joint
sun
99.44 %
PlotJuggler
App
Tools
Help
File
Data:
Layout:
Streaming
ROS2 Topic Subscriber
Stop
Buffer:
60
Publishers
ROS2 Topic Re-Publisher
CSV Exporter
Timeseries List
Filter...
11 of 11
Values
joint_lower_joint_cmd_pos
world
damped_stiff_one_armed_bandit
one_armed_bandit
model
double_pendulum_with_base
joint_state
header
lower_joint
upper_joint
effort
position
velocity
stiff_one_armed_bandit
Custom Series:
tab2
/world/one_armed_bandit/model/double_pendulum_with_base/joint_state/upper_joint/position
1
0.8
0.6
0.4
0.2
0
1700508612
1700508614
1700508616
1700508618
1700508620
1700508622
1700508623.133
Speed:
1.0
Step size:
0.000

Learning about control

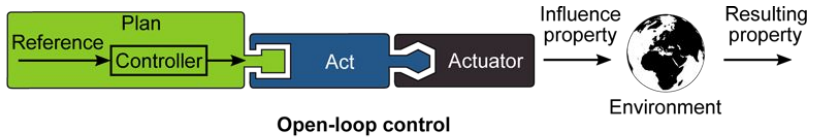
Closing the loop on control

Thus far we have sent commands to Gazebo to cause actions and observed the results.

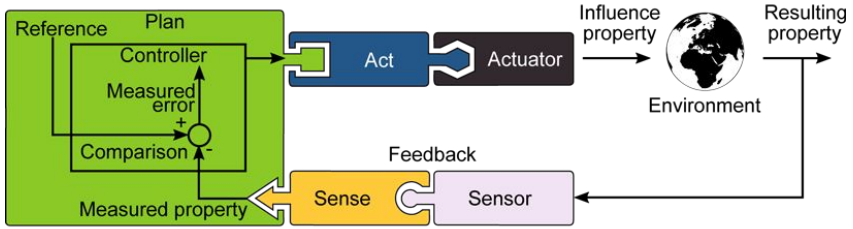
Now we make use of data about the effects to influence our actions.

Instead of combining already seen Gazebo systems, we use `ros2_control`, more specifically `gz_ros2_control`.

Note: As a consequence, the following example requires much more infrastructure.



Open-loop control

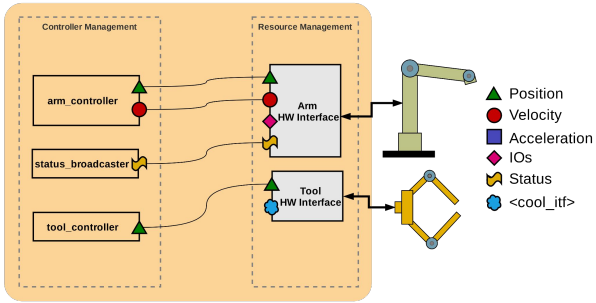


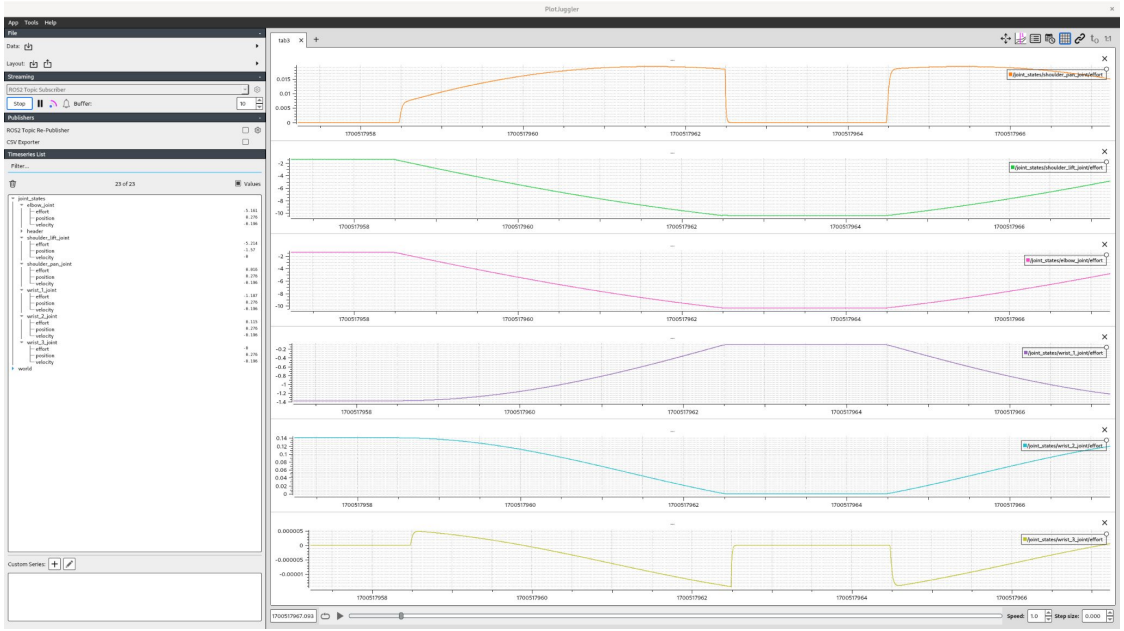
Closed-loop control

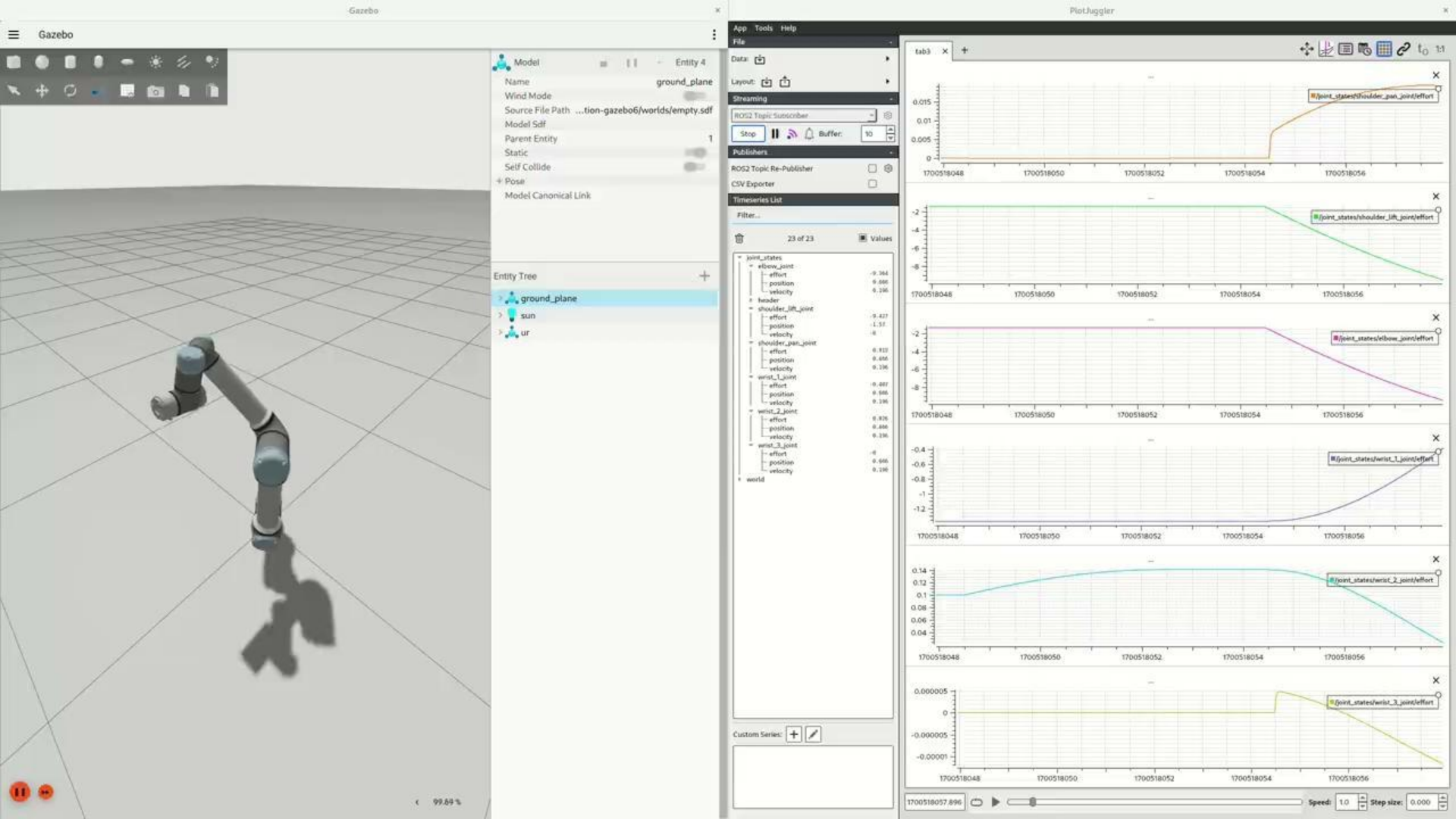
C1: Robots or stable pendulums

The `ur_simulation_ignition` package uses the `ign_ros2_control/IgnitionSystem` (aka `gz_ros2_control::GazeboSimROS2ControlPlugin`) for joint control.

It acts as ros2_control HW interface. Commanding positions, velocities or efforts and providing actual values back to the (closed-loop) controller.





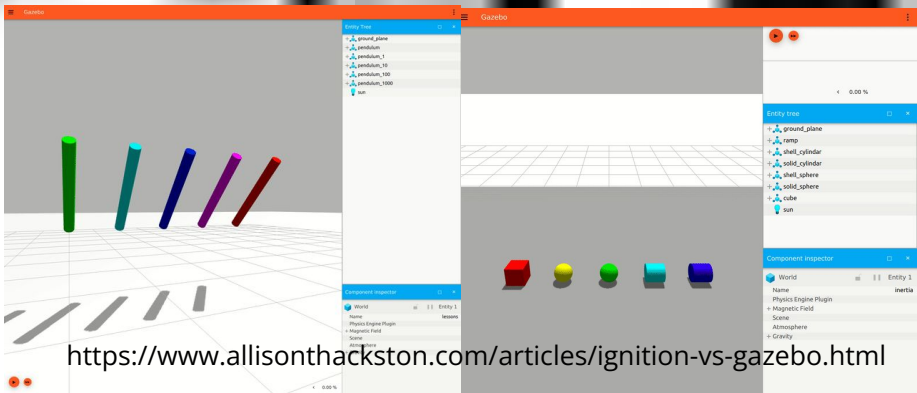

Gazebo
Gazebo
Model
Entity 4
Name
ground_plane
Wind Mode
Source File Path
...tion-gazebo6/worlds/empty.sdf
Model Sdf
Parent Entity
1
Static
Self Collide
Pose
Model Canonical Link
Entity Tree
ground_plane
sun
ur
99.69 %
PlotJuggler
App
Tools
Help
File
Data:
Layout:
Streaming
ROS2 Topic Subscriber
Stop
Buffer:
10
Publishers
ROS2 Topic Re-Publisher
CSV Exporter
Timeseries List
Filter...
23 of 23
Values
joint_states
elbow_joint
effort
position
velocity
header
shoulder_lift_joint
effort
position
velocity
shoulder_pan_joint
effort
position
velocity
wrist_1_joint
effort
position
velocity
wrist_2_joint
effort
position
velocity
wrist_3_joint
effort
position
velocity
world
Custom Series:
tab3
/joint_states/shoulder_pan_joint/effort
/joint_states/shoulder_lift_joint/effort
/joint_states/elbow_joint/effort
/joint_states/wrist_1_joint/effort
/joint_states/wrist_2_joint/effort
/joint_states/wrist_3_joint/effort
1700518048
1700518050
1700518052
1700518054
1700518056
1700518057.896
Speed:
1.0
Step size:
0.000

There is much more to explore at the intersection of robotics fundamentals, ROS 2 and Gazebo ... enjoy exploring it!

Thanks!

https://gazebosim.org/api/physics/[illegible]physics_concepts.html

https://github.com/andreasBihlmaier/robotics_fundamentals_ros_gazebo



https://www.allisonthackston.com/articles/ignition-vs-gazebo.html