

GAZEBO SIMULATION - DINGO

Dingo Quadruped Gazebo Simulation Setup & Control Guide

What is Gazebo?

Gazebo is a powerful 3D simulator used to test and visualize robots in a virtual environment. It accurately simulates real-world physics, sensors, and robot movements, making it ideal for developing and debugging robotics projects before using actual hardware.

Repository Link: <https://github.com/Yerbert/DingoQuadruped/tree/master>

System Requirements

- **Ubuntu:** 20.04 LTS
- **ROS:** Noetic
- **Gazebo:** Classic (version 11)
- **Python:** 3.8 or higher

1. Install Required Packages

Open a terminal and install the necessary dependencies:

bash:

```
sudo apt update
sudo apt install ros-noetic-effort-controllers
ros-noetic-ros-controllers ros-noetic-joint-state-controller
ros-noetic-ros-control ros-noetic-controller-manager
ros-noetic-teleop-twist-keyboard
```

2. Clone the Repository

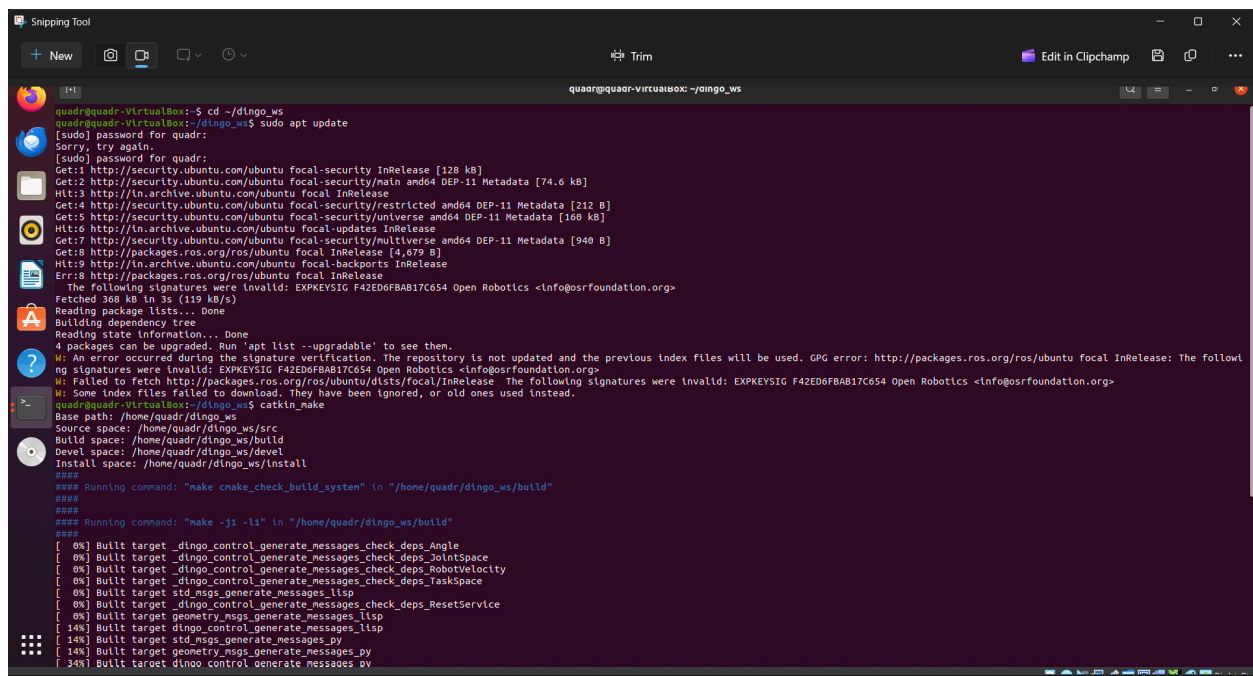
bash

```
cd ~  
git clone  
https://github.com/Yerbert/DingoQuadruped/tree/master/dingo_ws/src.git  
cd dingo_ws
```

3. Build the Workspace

bash

```
cd ~/dingo_ws  
rosdep install --from-paths src --ignore-src -r -y  
catkin_make  
source devel/setup.bash
```



The screenshot shows a terminal window titled "Snipping Tool" with a dark background. The user is in a virtual machine named "quadr@quadr-virtualbox: ~/dingo_ws". The terminal output shows the following steps:

- Running `sudo apt update` to update the package lists.
- Running `rosdep install --from-paths src --ignore-src -r -y` to install ROS dependencies.
- Running `catkin_make` to build the workspace.
- Running `source devel/setup.bash` to source the environment.

The terminal output shows the following messages:

```
quadr@quadr-virtualbox: ~/dingo_ws  
$ sudo apt update  
[sudo] password for quadr:  
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]  
Get:2 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [74.6 kB]  
Hit:3 http://in.archive.ubuntu.com/ubuntu focal InRelease  
Get:4 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 DEP-11 Metadata [212 B]  
Get:5 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [160 kB]  
Hit:6 http://in.archive.ubuntu.com/ubuntu focal-updates InRelease  
Get:7 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 Metadata [940 B]  
Get:8 http://packages.ros.org/ros/ubuntu focal InRelease [4,679 B]  
Hit:9 http://in.archive.ubuntu.com/ubuntu focal-backports InRelease  
Err:8 http://packages.ros.org/ros/ubuntu focal InRelease  
The following signatures were invalid: EXPKEYSIG F42ED6FBAB17C654 Open Robotics <info@osrfoundation.org>  
Fetched 308 kB in 3s (119 kB/s)  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
4 packages can be upgraded. Run 'apt list --upgradable' to see them.  
W: An error occurred during the signature verification. The repository is not updated and the previous index files will be used. GPG error: http://packages.ros.org/ros/ubuntu focal InRelease: The followi  
ng signatures were invalid: EXPKEYSIG F42ED6FBAB17C654 Open Robotics <info@osrfoundation.org>  
W: Failed to fetch http://packages.ros.org/ros/ubuntu/dists/focal/InRelease The following signatures were invalid: EXPKEYSIG F42ED6FBAB17C654 Open Robotics <info@osrfoundation.org>  
W: Some index files failed to download. They have been ignored, or old ones used instead.  
quadr@quadr-virtualbox: ~/dingo_ws$ catkin_make  
Base path: /home/quadr/dingo_ws  
Source space: /home/quadr/dingo_ws/src  
Build space: /home/quadr/dingo_ws/build  
Devel space: /home/quadr/dingo_ws/devel  
Install space: /home/quadr/dingo_ws/install  
####  
#### Running command: "make cmake_check_build_system" in "/home/quadr/dingo_ws/build"  
####  
####  
#### Running command: "make -j1 -l1" in "/home/quadr/dingo_ws/build"  
####  
[ 0%] Built target _dingo_control_generate_messages_check_deps_Angle  
[ 0%] Built target _dingo_control_generate_messages_check_deps_JointSpace  
[ 0%] Built target _dingo_control_generate_messages_check_deps_RobotVelocity  
[ 0%] Built target _dingo_control_generate_messages_check_deps_TaskSpace  
[ 0%] Built target std_msgs_generate_messages_lisp  
[ 0%] Built target _dingo_control_generate_messages_check_deps_ResetService  
[ 0%] Built target geometry_msgs_generate_messages_lisp  
[ 14%] Built target dingo_control_generate_messages_lisp  
[ 14%] Built target std_msgs_generate_messages_py  
[ 14%] Built target geometry_msgs_generate_messages_py  
[ 24%] Built target dingo_control_generate_messages_py
```


4. Launch the Robot in Gazebo

In Terminal 1, launch the Gazebo simulation:

bash

```
roslaunch dingo_gazebo simulation.launch
```

This opens the Gazebo world and spawns the quadruped robot.

 gazebo video.mp4

5. Start the State Publisher

In Terminal 2, run:

bash

```
source ~/DingoQuadruped/dingo_ws/devel/setup.bash
roslaunch dingo_description dingo_state_publisher.launch
```

This publishes joint state and TF data for visualization and coordination.

```
quadr@quadr-VirtualBox:~/dingo_ws$ roslaunch dingo_description dingo_state_publisher.launch
... Logging to /home/quadr/.ros/log/f5b19bbe-561a-11f0-baca-4f6915d4ba6e/roslaunch-quadr-VirtualBox-7876.log
Checking log directory for disk usage. This may take a while.
Press ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

macro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://quadr-VirtualBox:38947/

SUMMARY
=====
PARAMETERS
 * /robot_description: <?xml version="1...
 * /roscpp: noetic
 * /rosversion: 1.17.3
NODES
 /
  rob_st_pub (robot_state_publisher/robot_state_publisher)
ROS_MASTER_URI=http://localhost:11311

process[rob_st_pub-1]: started with pid [7892]
[WARN] [1751333909.355905627]: The root link base_link has an inertia specified in the URDF, but KDL does not support a root link with an inertia. As a workaround, you can add an extra dummy link to you
r URDF.
```

6. Code for dingo_driver.py:

Open a new terminal.

bash:

```
cd ~dingo_ws
catkin_make
source devel/setup.bash
cd dingo_ws/src/dingo/scripts
code .
```

- This opens a folder called scripts in VS code where the following files are present.
 - dingo_driver.py
 - cmd_vel_driver.py
 - keyboard_node_input.py
- Open dingo_driver.py and paste the following:

Code:

```
import numpy as np          # ■ For numerical operations like arrays and matrices ■
import time                 # ■ Used for delays or time-based operations ■
import rospy                # ■ Main ROS Python client library ■
import sys                  # ■ For accessing command-line arguments ■
from std_msgs.msg import Float64 # ■ Message type to publish float values (like joint
angles) ■
import signal               # ■ Used to handle keyboard interrupts like Ctrl+C ■
import socket               # ■ (Optional) for network communication (not used here) ■
import platform             # ■ (Optional) to check OS (not used here) ■
from dingo_peripheral_interfacing.msg import ElectricalMeasurements # ■ Custom
message for power monitoring ■

# ■ Fetch command-line arguments: is_sim, is_physical, and use_imu ■
args = rospy.myargv(argv=sys.argv)
if len(args) != 4:
    is_sim = 0          # ■ Default: not simulation ■
    is_physical = 1     # ■ Default: physical robot is used ■
```

```

    use_imu = 1          # Default: use IMU sensor
else:
    is_sim = int(args[1]) # Get is_sim flag from arguments
    is_physical = int(args[2]) # Get is_physical flag from arguments
    use_imu = int(args[3]) # Get use_imu flag from arguments

# Import robot control modules
from dingo_control.Controller import Controller
from dingo_input_interfacing.InputInterface import InputInterface
from dingo_control.State import State, BehaviorState
from dingo_control.Kinematics import four_legs_inverse_kinematics
from dingo_control.Config import Configuration
from dingo_control.msg import TaskSpace, JointSpace, Angle
from std_msgs.msg import Bool

# Import hardware-specific modules only if using physical robot
if is_physical:
    from dingo_servo_interfacing.HardwareInterface import HardwareInterface
    from dingo_peripheral_interfacing.IMU import IMU
    from dingo_control.Config import Leg_linkage

class DingoDriver:
    def __init__(self, is_sim, is_physical, use_imu): # Constructor
        self.message_rate = 50 # 50 Hz control loop
        self.rate = rospy.Rate(self.message_rate) # ROS loop rate object

        self.is_sim = is_sim # Store simulation mode
        self.is_physical = is_physical # Store physical robot mode
        self.use_imu = use_imu # Store IMU usage flag

    # ROS Subscribers

```

```

        self.joint_command_sub = rospy.Subscriber("/joint_space_cmd", JointSpace,
self.run_joint_space_command)
        self.task_command_sub = rospy.Subscriber("/task_space_cmd", TaskSpace,
self.run_task_space_command)
        self.estop_status_sub = rospy.Subscriber("/emergency_stop_status", Bool,
self.update_emergency_stop_status)
        self.external_commands_enabled = not self.is_physical # Enable external
commands in sim

```

```

rospy.loginfo(f"DingoDriver: External commands enabled:
{self.external_commands_enabled} (derived from is_physical={self.is_physical})")

```

```

# Simulation joint controllers

```

```

if self.is_sim:

```

```

    self.sim_command_topics = [ # Topics for 12 simulated joint controllers
        "/dingo_controller/FR_theta1/command",
        "/dingo_controller/FR_theta2/command",
        "/dingo_controller/FR_theta3/command",
        "/dingo_controller/FL_theta1/command",
        "/dingo_controller/FL_theta2/command",
        "/dingo_controller/FL_theta3/command",
        "/dingo_controller/RR_theta1/command",
        "/dingo_controller/RR_theta2/command",
        "/dingo_controller/RR_theta3/command",
        "/dingo_controller/RL_theta1/command",
        "/dingo_controller/RL_theta2/command",
        "/dingo_controller/RL_theta3/command"
    ]

```

```

self.sim_publisher_array = [] # Publishers to send commands to Gazebo

```

```

for topic in self.sim_command_topics:

```

```

    self.sim_publisher_array.append(rospy.Publisher(topic, Float64, queue_size=0))

```

```

self.config = Configuration() # Load config values

if is_physical:
    self.linkage = Leg_linkage(self.config) # Linkage model for kinematics
    self.hardware_interface = HardwareInterface(self.linkage) # Hardware driver
    interface

if self.use_imu:
    self.imu = IMU() # Initialize IMU

self.controller = Controller(self.config, four_legs_inverse_kinematics) # Main
controller logic
self.state = State() # Holds robot state
self.input_interface = InputInterface(self.config) # Joystick or keyboard input
handler

rospy.loginfo("Input listener successfully initialised... Robot will now receive
commands via Joy messages")

# Print gait configuration
rospy.loginfo("Summary of current gait parameters:")
rospy.loginfo("overlap time: %.2f", self.config.overlap_time)
rospy.loginfo("swing time: %.2f", self.config.swing_time)
rospy.loginfo("z clearance: %.2f", self.config.z_clearance)
rospy.loginfo("back leg x shift: %.2f", self.config.rear_leg_x_shift)
rospy.loginfo("front leg x shift: %.2f", self.config.front_leg_x_shift)

def run(self): # Main loop function
    while not rospy.is_shutdown():
        if self.state.currently_estopped == 1:

```

```

        rospy.logwarn("DingoDriver: E-stop pressed. Controlling code now disabled
until E-stop is released")
        self.state.trotting_active = 0
        while self.state.currently_estopped == 1:
            self.rate.sleep()
        rospy.loginfo("DingoDriver: E-stop released")
        if not self.is_physical:
            self.external_commands_enabled = True
        continue

    if self.external_commands_enabled:
        rospy.loginfo_throttle(5, "DingoDriver: Robot accepting external commands
(TaskSpace/JointSpace).")
    else:
        rospy.loginfo_throttle(5, "DingoDriver: Manual robot control active. Waiting for
InputInterface (Joy messages).")
        if self.input_interface is not None:
            command = self.input_interface.get_command(self.state,
self.message_rate)
            if self.is_physical and command.joystick_control_event == 1:
                self.external_commands_enabled = True
                rospy.loginfo("DingoDriver: Joystick requested switch to External
Command Mode.")
                self.rate.sleep()
                continue
            self.controller.run(self.state, command)
            self.controller.publish_joint_space_command(self.state.joint_angles)

        self.controller.publish_task_space_command(self.state.rotated_foot_locations)
    else:
        rospy.logwarn_throttle(5, "DingoDriver: Manual control attempted but no
InputInterface initialized. Robot will stand still.")

```



```

        self.state.behavior_state = BehaviorState.REST
        self.controller.run(self.state, self.controller.get_default_command())
        self.controller.publish_joint_space_command(self.state.joint_angles)

self.controller.publish_task_space_command(self.state.rotated_foot_locations)

if self.is_physical and self.use_imu and self.imu is not None:
    self.state.euler_orientation = self.imu.read_orientation()
else:
    self.state.euler_orientation = np.array([0, 0, 0])

if self.is_sim:
    self.publish_joints_to_sim(self.state.joint_angles)

if self.is_physical and self.hardware_interface is not None:
    self.hardware_interface.set_actuator_postions(self.state.joint_angles)

self.rate.sleep()

def update_emergency_stop_status(self, msg): # ■ Callback to update E-stop ■
    if msg.data == 1:
        self.state.currently_estopped = 1
    if msg.data == 0:
        self.state.currently_estopped = 0

def run_task_space_command(self, msg): # ■ Handle TaskSpace command input
    if self.external_commands_enabled == 1 and self.state.currently_estopped == 0:
        rospy.loginfo("Received Task Space Command: " + str(msg))
        foot_locations = np.zeros((3, 4))
        foot_locations[0] = [msg.FR_foot.x, msg.FL_foot.x, msg.RR_foot.x, msg.RL_foot.x]
        foot_locations[1] = [msg.FR_foot.y, msg.FL_foot.y, msg.RR_foot.y, msg.RL_foot.y]



```

```

foot_locations[2] = [msg.FR_foot.z, msg.FL_foot.z, msg.RR_foot.z, msg.RL_foot.z]

joint_angles = self.controller.inverse_kinematics(foot_locations, self.config)

if self.is_sim:
    self.publish_joints_to_sim(joint_angles)
if self.is_physical and self.hardware_interface is not None:
    self.hardware_interface.set_actuator_postions(joint_angles)
elif self.external_commands_enabled == 0:
    rospy.logerr("ERROR: Robot not accepting commands. Please deactivate manual
control before sending control commands")
elif self.state.currently_estopped == 1:
    rospy.logerr("ERROR: Robot currently estopped. Please release before trying to
send commands")

def run_joint_space_command(self, msg): #  Handle JointSpace command input

    if self.external_commands_enabled == 1 and self.state.currently_estopped == 0:
        joint_angles = np.zeros((3,4))
        for i in range(3):
            joint_angles[i] = [msg.FR_foot[i], msg.FL_foot[i], msg.RR_foot[i], msg.RL_foot[i]]
        if self.is_sim:
            self.publish_joints_to_sim(joint_angles)
        if self.is_physical:
            self.hardware_interface.set_actuator_postions(joint_angles)
    elif self.external_commands_enabled == 0:
        rospy.logerr("ERROR: Robot not accepting commands. Please deactivate manual
control before sending control commands")
    elif self.state.currently_estopped == 1:
        rospy.logerr("ERROR: Robot currently estopped. Please release before trying to
send commands")

```

```

def publish_joints_to_sim(self, joint_angles): # Publish joint angles to Gazebo
    controllers
    rows, cols = joint_angles.shape
    i = 0
    for col in range(cols):
        for row in range(rows):
            self.sim_publisher_array[i].publish(joint_angles[row, col])
            i += 1

def signal_handler(sig, frame): # Clean shutdown on Ctrl+C
    sys.exit(0)

def main(): # Entry point
    rospy.init_node("dingo_driver")
    signal.signal(signal.SIGINT, signal_handler)
    dingo = DingoDriver(is_sim, is_physical, use_imu)
    dingo.run()

main()

```

7. Launch dingo_driver.py:

bash

```

cd ~/dingo_ws
catkin_make
source devel/setup.bash
roslaunch dingo_driver.py 1 0 0

```

- 1 0 0 means: is_sim=1, is_physical=0, use_imu=0
- Make sure dingo_driver.py is executable or adjust path if needed.

8. Send Task Space Commands

In a new terminal, Use the following command to make the robot move its legs:

bash

```
cd ~dingo_ws
catkin_make
source devel/setup.bash
rostopic pub -r 10 /task_space_cmd dingo_control/TaskSpace
"FR_foot:

  x: 0.1

  y: -0.1

  z: -0.2

FL_foot:

  x: 0.1

  y: 0.1

  z: -0.2

RR_foot:

  x: -0.1

  y: -0.1

  z: -0.2

RL_foot:

  x: -0.1

  y: 0.1

  z: -0.2"
```

 This moves the robot to the specified foot positions using inverse kinematics.

Output:  dingo moving video.mp4

If You See Controller Errors Like:

Could not load controller 'FR_theta1' because controller type 'effort_controllers/JointPositionController' does not exist.

Fix with:

bash

```
sudo apt install ros-noetic-effort-controllers
```

```
ros-noetic-ros-controllers
```

```
source /opt/ros/noetic/setup.bash
```

```
cd ~/DingoQuadruped/dingo_ws
```

```
catkin_make
```

```
source devel/setup.bash
```