```
===============================
 LAUNCH FILE DOCUMENTATION
 Folder: dingo_ws/src/dingo/launch/
===============================

-----------------------------------
1. File: dingo.launch
-----------------------------------
>> PURPOSE:
Main launch file for controlling the Dingo robot.
Supports both physical and simulation modes.
Enables optional input devices like joystick or keyboard.
Controls peripheral devices like LCD screen.

>> ARGUMENTS:
  - is_sim         : Default = 0 -> Use simulation mode if 1
  - is_physical    : Default = 1 -> Use physical robot if 1
  - use_joystick   : Default = 1 -> Enable joystick input
  - use_keyboard   : Default = 0 -> Enable keyboard input
  - serial_port    : Default = /dev/ttyS0 -> Serial port for Arduino
  - use_imu        : Default = 0 -> Enable IMU sensor if 1

>> NODES LAUNCHED:

[If is_physical == 1]
  - Node Name : dingo_rosserial
    Package   : rosserial_python
    Type      : serial_node.py
    Purpose   : Serial communication between Arduino & Pi

  - Node Name : dingo_LCD_node
    Package   : dingo_peripheral_interfacing
    Type      : dingo_lcd_interfacing.py
    Purpose   : Interact with LCD display

[If use_joystick == 1]
  - Node Name : JOYSTICK
    Package   : joy
    Type      : joy_node
    Purpose   : Joystick input handler (repeat rate = 30Hz)

[If use_keyboard == 1]
  - Node Name : keyboard_input_listener
    Package   : dingo_input_interfacing
    Type      : Keyboard.py
    Purpose   : Keyboard input handler

[Always launched]
```

```
  - Node Name : dingo
    Package   : dingo
    Type      : dingo_driver.py
    Args      : is_sim is_physical use_imu
    Purpose   : Main driver to control Dingo movement
```

----
2. File: dingo_gazebo_sim.launch

>> PURPOSE:
Launches the Gazebo simulation environment for Dingo.

>> CONTENT:
```
<include file="$(find dingo_gazebo)/launch/simulation.launch" />
```

>> NOTES:
Loads the simulation.launch file from dingo_gazebo package.
Spawns robot in virtual Gazebo world with physics.

----
3. File: dingo_simulator.launch

>> PURPOSE:
Same as dingo_gazebo_sim.launch.
Possibly created for clarity or future extensions.

>> CONTENT:
```
<include file="$(find dingo_gazebo)/launch/simulation.launch" />
```

>> NOTES:
Functionally identical to dingo_gazebo_sim.launch

RECOMMENDATION:
Keep only ONE of the Gazebo launch files unless you plan
to customize them differently later.

scripts
# Dingo Driver Node Documentation
# File: dingo_ws/src/dingo/scripts/dingo_driver.py

```
node:
  name: dingo
  purpose: >
    Core control script for the Dingo quadruped robot.
    Handles both manual (joystick/keyboard) and external commands via topics.
    Supports simulation and physical modes, with optional IMU integration.
```

```
arguments:
  - name: is_sim
    type: int
    default: 0
    description: Set to 1 for Gazebo simulation
  - name: is_physical
    type: int
    default: 1
    description: Set to 1 to control the real robot
  - name: use_imu
    type: int
    default: 1
    description: Set to 1 to use IMU orientation input

dependencies:
  - dingo_control
  - dingo_input_interfacing
  - dingo_servo_interfacing  # Only if physical
  - dingo_peripheral_interfacing  # Only if IMU is used
  - std_msgs
  - rospy
  - numpy
  - time
  - signal
  - socket
  - platform

subscriptions:
  - topic: /joint_space_cmd
    msg_type: JointSpace
    purpose: Receive joint angles externally
  - topic: /task_space_cmd
    msg_type: TaskSpace
    purpose: Receive foot positions externally
  - topic: /emergency_stop_status
    msg_type: Bool
    purpose: Listen for emergency stop signal

publications:
  condition: is_sim == 1
  type: Float64
  topics: >
    Publishes to 12 joint command topics like
    /dingo_controller/FR_theta1/command, etc.
    (3 joints × 4 legs = 12 topics)

components:
  class: DingoDriver
```

```
    handles:
      - Input commands (joystick/keyboard)
      - IMU data (optional)
      - Controller updates and gait logic
      - Actuation to Gazebo or hardware
      - Emergency stop monitoring
      - Mode switching (manual ↔ external)

  control_flow:
    - step: Startup
      description: Initializes subscribers, publishers, controller, and inputs
    - step: Wait for manual input
      description: Stays idle until joystick triggers external mode
    - step: Main Loop
      description: >
        Processes manual control or external commands, updates controller,
        sends commands to simulator or hardware, and checks IMU if enabled
    - step: External Mode
      description: Motion controlled via /joint_space_cmd or /task_space_cmd
    - step: Emergency Stop
      description: Halts motion if estop is triggered

  emergency_stop:
    logic: >
      If currently_estopped == 1, block all motion and log warning.
      Requires estop release to resume control.

  error_handling:
    logs:
      - External commands during manual mode
      - Attempted motion during emergency stop

  signal_handling:
    interrupt: SIGINT
    handler: signal_handler()
    behavior: Gracefully shuts down on Ctrl+C

  entry_point: |
    if __name__ == '__main__':
        main()
```

# RUN_ROBOT.PY DOCUMENTATION

```
file: dingo_ws/src/dingo/scripts/run_robot.py
description: >
  Core script for executing and controlling the Dingo quadruped robot.
  Launches the robot in either simulation or physical mode and handles
  real-time control, input parsing, and actuator updates.
```

```
#  PURPOSE
purpose: >
  Controls Dingo robot's actuation loop, reading joystick commands, updating
  robot state, running control algorithms, and sending joint commands to
  Gazebo (if sim) or hardware servos (if physical). Optionally integrates
  IMU for orientation feedback.

# ARGUMENTS
arguments:
  is_sim:
    type: int
    default: 0
    description: Enable simulation mode (Gazebo) if set to 1.
  is_physical:
    type: int
    default: 0
    description: Enable physical mode (servo motors + hardware) if set to 1.
  use_imu:
    type: bool
    default: False
    description: Enable IMU orientation feedback if True.

# MAIN FLOW
flow:
  - Parse arguments from sys.argv using rospy.myargv.
  - Initialize ROS node as "dingo".
  - Setup signal handler for safe shutdown (Ctrl+C).
  - Setup Gazebo joint publishers if is_sim = 1.
  - Setup hardware interface if is_physical = 1.
  - Initialize controller, input interface, and input controller.
  - Print gait parameters summary.
  - Wait for joystick L1 activation to start control loop.
  - In main loop:
    - Read joystick command.
    - Read IMU orientation (if enabled).
    - Run controller update with current state and command.
    - Publish joint angles to Gazebo or hardware.
    - Loop at 50Hz until L1 is pressed again or shutdown.

#  CONNECTIONS
connects_to:
  - IMU: /dev/ttyACM0 (if use_imu is True)
  - Gazebo topics (12 Float64 joint commands) if is_sim is True.
  - Servo control (via PWM) if is_physical is True.
  - InputInterface + InputController to parse joystick data.
  - dingo_control package (Controller, State, Config, Kinematics)
  - dingo_servo_interfacing (if physical hardware)
  - dingo_peripheral_interfacing (for IMU and electrical data)
```

```
#  DEPENDENCIES
dependencies:
  - rospy
  - numpy
  - time, signal, sys, platform
  - std_msgs.msg.Float64
  - dingo_peripheral_interfacing.IMU
  - dingo_control.Controller, State, Config, Kinematics
  - dingo_input_interfacing.InputInterface, InputController
  - dingo_servo_interfacing.HardwareInterface (if physical)

#  COMPONENT DETAILS
components:
  - Configuration: Robot parameters (timing, shifts, clearances).
  - Controller: Executes gait and posture logic.
  - State: Holds joint angles, orientation, etc.
  - InputInterface: Gets command from joystick or keyboard.
  - HardwareInterface: Sends commands to servos (PWM).
  - IMU: Reads orientation data if used.
  - Publishers: 12 Gazebo joint topics (Float64) for simulation.

#  SPECIAL LOGIC
special_logic:
  - Emergency loop exit using joystick L1 button.
  - Quaternion fallback when IMU not used.
  - Handles control mode switch on-the-fly.
  - Joint angle broadcasting to simulation or real servos.
  - Real-time performance printing for debugging (commented lines).

#  OUTPUTS
outputs:
  - Sim mode: Publishes to `/notspot_controller/<joint>/command` (12 topics)
  - Physical mode: Sends servo PWM signals
  - Console logs: Gait parameters, debug prints, timing info (optional)

#  INPUTS
inputs:
  - Joystick or keyboard commands via InputInterface
  - IMU orientation data (if enabled)

# 🏁 ENTRY POINT
entry:
  function: main()
  location: Bottom of script
  condition: if __name__ == '__main__'

#  NOTE
```

```
notes: >
  This script acts as a standalone ROS node named `dingo`.
  You must set proper permissions and hardware configuration
  (e.g., correct port, joystick settings) before launching.

status_publisher.py:
  description: |
    A helper Python script in the dingo package that defines a class for publishing status messages
    on the /robot_status_messages ROS topic using std_msgs/String.

  location: src/dingo/status_publisher.py

  mandatory: false

  usage:
    - Import the class in any script: from dingo.status_publisher import StatusPublisher
    - Create an instance: status = StatusPublisher()
    - Publish a message: status.publish_message("Robot Started")

  topic:
    name: /robot_status_messages
    type: std_msgs/String
    direction: Publisher

  example_use_case:
    - Log robot status like "Started", "Stopped", "Battery Low"
    - Helpful for debugging or monitoring in terminal or GUI
    - Can be used inside run_robot.py or dingo_driver.py

  dependencies:
    - rospy
    - std_msgs.msg.String

  __init__.py:
    description: Empty file used to make dingo/ a valid Python module for imports
    mandatory: true (only for Python import system to work)

dingo_package:
  cmake_minimum_required: "3.0.2"
  project_name: "dingo"
  dependencies:
    - "rospy"
  python_setup: true
  catkin_package:
    include_dirs: []
    libraries: []
    catkin_deps: []
    system_deps: []
  include_directories:
    - "${catkin_INCLUDE_DIRS}"
  python_scripts_to_install:
    - "scripts/run_robot.py"
    - "scripts/dingo_driver.py"
  install_destination: "${CATKIN_PACKAGE_BIN_DESTINATION}"
  notes:
    - "Pure Python ROS package (no C++ code or message generation)"
    - "Python scripts installed as executables for rosrun"
```

package.xml:

```
package:
  format: 2
  name: dingo
  version: "0.0.0"
  description: "The dingo package"
  purpose: >
    This package.xml file is used in a ROS 1 catkin-based workspace to define the
    metadata, dependencies, and build configuration for the 'dingo' robot software package.
    It tells the ROS build system (catkin) how to compile and run the package, and ensures
    that all required packages (like rospy, numpy) are available.

  usage: >
    1. When you run `catkin_make` or `catkin build`, this file is read to resolve all
       dependencies and include them during the build process.
    2. ROS tools like `roslaunch`, `rosdep`, and `rospack` also use this file to understand
       the package structure and dependency tree.
    3. This file is mandatory in every ROS package to make it discoverable and usable
       within a catkin workspace.

  maintainer:
    name: alex
    email: alex@todo.todo

  license: "TODO"  # Replace with an actual license like MIT, BSD, or GPLv3

  build_tool_depend:
    - catkin

  dependencies:
    build_depend:
      - rospy
    build_export_depend:
      - rospy
    exec_depend:
      - rospy
    depend:
      - numpy
      - time  # Note: 'time' is part of Python stdlib, not needed here

  export: {}

setup_py:
  purpose: >
    Defines how the Python part of the ROS package should be installed and structured.
    Required if your ROS package contains a Python module (like dingo inside src/).
    Works with catkin's catkin_python_setup().

  required_when:
    - You have custom Python modules in `src/`
    - You are using `catkin_python_setup()` in CMakeLists.txt
    - You want the package to be installable and importable as a Python package

  code_blocks:
    - from distutils.core import setup:
        description: Loads the Python packaging system
    - from catkin_pkg.python_setup import generate_distutils_setup:
        description: Gets setup metadata from ROS package.xml
    - generate_distutils_setup:
        packages: ['dingo']
        package_dir: {'': 'src'}
        description: Specifies that `src/dingo/` is the Python package
    - setup(**d):
        description: Executes setup with all collected metadata

  notes:
    - No need to modify unless you rename your Python package or change its folder
    - It's automatically used during catkin build processes
```