Jetson Wireless Setup and Open-Loop Control

Table of Contents

Introduction

Objective

Open-Loop Motor Control & Remote Access

Objective: Establish wireless control of your robot and implement a open-loop motor driver. This lab provides the essential hardware and software foundation for all subsequent labs using the GoBilda chassis.

Key Goals:

- Configure wireless remote access to the Jetson Orin Nano using SSH.
- Assemble and wire the robot's motor control components.
- Develop a C++ application to send PWM signals to the motors, enabling the robot to drive forward/backward and turn left/right.

Note: In this lab, we will not use the ROS2 stack. Instead, we will create a basic motor driver that interfaces directly with the Orin's GPIO pins and the GoBilda Motor Controllers.

High-Level Procedure

Phase 1: Hardware and Network Setup

1. Establish a Wireless Connection
    a. Connect your laptop to the same network as the Orin Nano.
    b. Use the SSH tool to establish a remote command-line connection to the Orin.
2. Assemble and Wire the Robot
    a. Follow the provided diagrams to assemble the GoBilda's chassis and parts.
    b. Wire the motors to the motor controllers, and the controllers to the appropriate GPIO pins on the Orin Nano.

Phase 2: Software Environment Configuration

3. Configure the Jetson OS Environment
    a. Set up the operating system permissions to grant your user account access to the GPIO pins.

Phase 3: Application Development

4. Develop and Demo the Motor Controller
    a. Write a C++ program (motor_driver.cpp) that toggles the GPIO pins to generate PWM signals. Your program should command the robot to:
        i. Drive forward for 2 seconds.
        ii. Drive backward for 2 seconds.
        iii. Turn left for 2 seconds
        iv. Turn right for 2 seconds
5. Compile and run your code on the Orin Nano to demonstrate this sequence.

Required Equipment Checklist

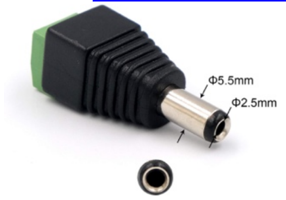
Ensure you have the following items before starting:

- x1 Jetson Orin Nano Developer Kit
- x2 Gobilda Motors

- x2 Gobilda Motor Controllers

- x1 Gobilda Power Distribution Board

- x1 12V Battery (charged)
- Personal Laptop
- x1 Barrel Jack Adapter



- x1 Gobilda Lead Cable

Networking

Orin Access Options

This quarter, we will use one of three methods to access and configure the Jetson Orin Nano (JON):

1. Wired Connection (Recommended Initial Steup)
    - Connect an Ethernet cable directly from the JON to your computer to establish an *SSH* or *VNC* connection.

2. **Wireless Connection** (Required for Robot Operation)
    - Establish an *SSH* connection to the JON over the network. This requires the JON to be on the *CP-IoT-Secure* network while your laptop is connected to *eduroam*.

3. Direct Peripheral Connection (Use Sparingly)
    - Connect a monitor, keyboard, and mouse directly to the JON.

Lab Focus

For this lab, you must use **Method 2: Wireless Connection**. A stable remote connection is necessary to program the robot and command it to move. This step establishes the foundation for all subsequent work.

Wireless Setup (CP-IoT-Secure & eduroam)

This section guides you through establishing a wireless SSH connection to your Jetson Orin. Prerequisites:

- Your Orin must be powered on and connected to the CP-IoT-Secure network (from the previous lab).
- Your laptop must be connected to the eduroam network.

Obtaining the Orin's CP-IoT-Secure IP address

Your Orin has a reserved IP address on the CP-IoT-Secure network. Your instructor will provide you with this specific IP address.

Troubleshooting: If the provided IP address does not work, you must connect to the Orin via a direct Ethernet cable and run the following command in its terminal to find the correct wireless IP address:

```
ip addr show
```

Look for the *wlP1p1s0* interface to find the wireless IP.

Establish the SSH Connection

1. Open a terminal on your computer:
   - macOS: Use Terminal.
   - Windows: Use PowerShell.

2. Run the following SSH command, replacing IP_ADDRESS with the address provided to you:

```
ssh calpoly@IP_ADDRESS
```

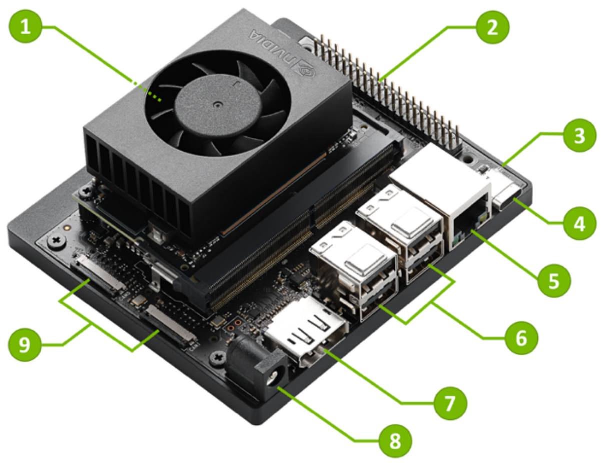
3. When prompted, enter the password (cpe1234) for the calpoly user account

Important Notes:

- Dual Connection Block: ITS policies may prevent a device from having simultaneous wired and wireless connections. You cannot be connected to the Orin via both Ethernet and SSH at the same time. Ensure any wired connection is disconnected before attempting the wireless SSH.
- Network Verification: If the connection fails, first double-check that your Orin is on CP-IoT-Secure and your laptop is on eduroam

Hardware Guide

This section provides an overview of the different hardware components at play and a guide for wiring the robot.



Jetson Orin Nano device. For this lab we will be using the GPIO pins labeled '2' in the picture above, and the Power Input labeled '8' in the picture above.

Pre-Execution Wiring Checklist

Before executing any code on the robot, verify all the following connections are secure and correct.

Wiring checklist before executing code on the robot:

Power & Motor System:

1. Motor Controller → Motors (ground and power): Connect the ground and power wires from each motor to its corresponding motor controller.

2. Motors → Power Distribution Board (PDB): Connect XT30 male connector from the motors to the PDB.

3. XT30 Lead → Barrel Jack: Connect the ground and power ends of the lead cable into the barrel jack adapter.

4. XT30 Lead → Power Distribution Board: Ensure the other end of the XT30 lead is connected to the PDB.

5. Battery → Power Distribution Board: Connect your battery to the PDB.

Compute & Control System:

6. Barrel Jack → Orin: Connect the barrel jack to the Jetson Orin. ⚠️ **Important:** After this step, the Orin will power on immediately! So, make sure you handle it with care.

Powering Down the Orin

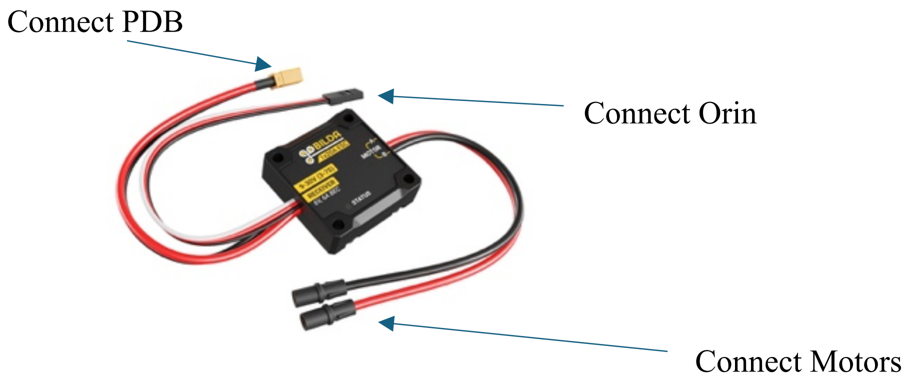
To properly shut down the Orin before disconnecting power, run the following command in a terminal on the Orin via your SSH connection:

```
sudo shutdown now
```

Wait for the Orin's green activity LED to stop blinking completely before disconnecting the barrel jack (Step 6) or the battery (Step 5).

7. Motor Controller → Jetson Orin GPIO pins

This last connection is very important and requires us to use specific GPIO pin numbers that are hard coded into the eventual ROS2 GoBilda Driver.

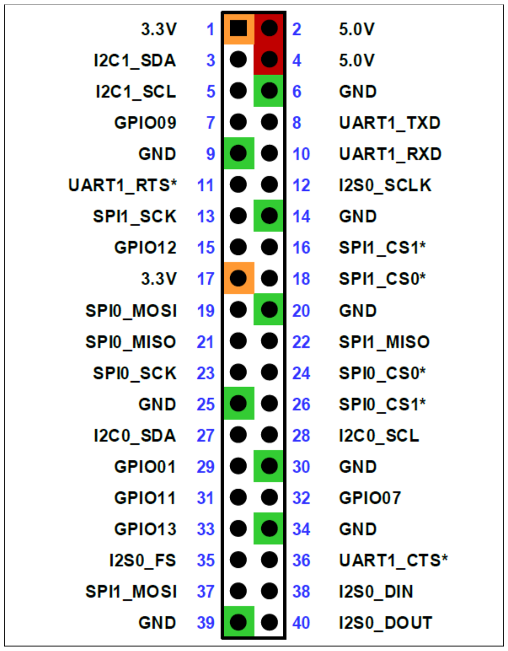


Note the "Ground Wire" is the black wire on the motor controller and the "Signal Wire" is the white wire. You should NOT use the red power wire for any part of the robot. Motor controllers will receive power from the PDB.

Pin Assignment

The following pins on the Orin's 40-pin GPIO header are used:

Motor	PWM Signal Pin	Ground Pin
Left Motor	Pin 15 (GPIO12)	Any Ground (e.g., Pin 6, 9, 14. etc)
Right Motor	Pin 32 (GPIO07)	Any Ground (e.g., Pin 6, 9, 14. etc)



3.3V	1	2	5.0V
I2C1_SDA	3	4	5.0V
I2C1_SCL	5	6	GND
GPIO09	7	8	UART1_TXD
GND	9	10	UART1_RXD
UART1_RTS*	11	12	I2S0_SCLK
SPI1_SCK	13	14	GND
GPIO12	15	16	SPI1_CS1*
3.3V	17	18	SPI1_CS0*
SPI0_MOSI	19	20	GND
SPI0_MISO	21	22	SPI1_MISO
SPI0_SCK	23	24	SPI0_CS0*
GND	25	26	SPI0_CS1*
I2C0_SDA	27	28	I2C0_SCL
GPIO01	29	30	GND
GPIO11	31	32	GPIO07
GPIO13	33	34	GND
I2S0_FS	35	36	UART1_CTS*
SPI1_MOSI	37	38	I2S0_DIN
GND	39	40	I2S0_DOUT

Map of the Jetson Orin Nano's 40-pin GPIO expansion header. For this lab we will be using pins 15 & 32 to send PWM signals to the Gobilda motor controllers.

With respect to the robot's frame make the following connections:

1. Left Motor Controller:
   - PWM Signal Wire → Pin 15
   - Ground Wire → Any available Ground pin
2. Right Motor Controller:
   - PWM Signal Wire → Pin 32
   - Ground Wire → Any available Ground pin

Software Guide

Jetson Orin Nano Setup

The section ensures your Orin's permissions and environment are correctly configured to control the robot.

1. Clone the Driver Repository

First, obtain the source code by cloning the provided repository. Run the following command in your terminal on the Orin:

```
git clone -recursive https://github.com/ambulantelab/gobilda.git
```

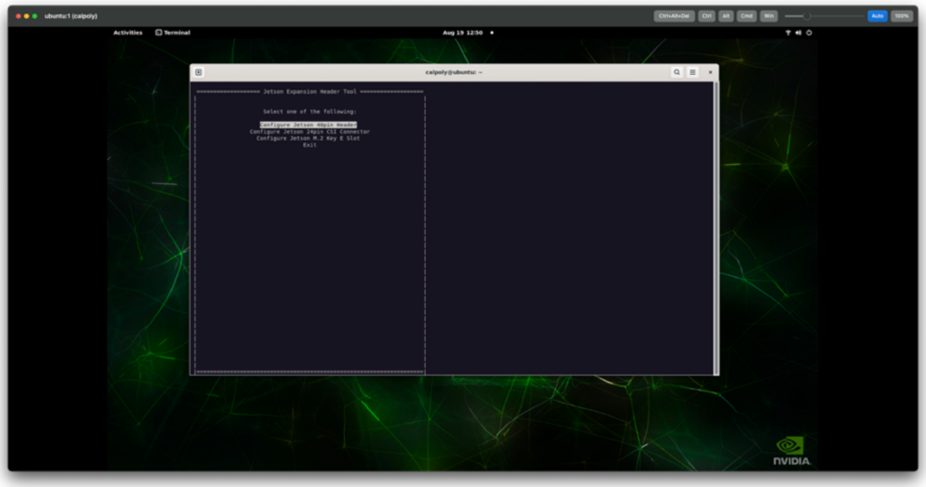
2. Configure GPIO Pins for PWM signals

The motor controllers require Pulse Width Modulation (PWM) signals from specific GPIO pins. We will use NVIDIA's provided configuration tool to assign this function.

- Run the configuration tool with the following command (from terminal in on Jetson):

```
sudo /opt/nvidia/jetson-io/jetson-io.py
```

This will launch an interactive text-based menu.

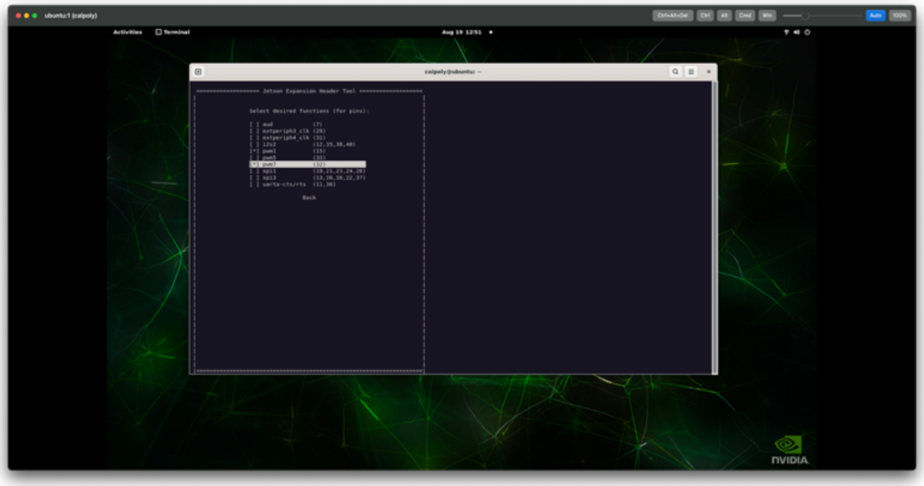


```
==================== Jetson Expansion Header Tool ====================

                  Select one of the following:

                  Configure Jetson 40pin Header
                Configure Jetson 24pin CSI Connector
                  Configure Jetson M.2 Key E Slot
                               Exit
```

- In the menu, navigate to the option to configure the *40-pin GPIO expansion header*
- Find and select pin 15 and 32, and set their function to PWM
- Save the configuration and exit the tool. You will be prompted to reboot the Orin – select 'Yes'.



```
==================== Jetson Expansion Header Tool ====================

                Select desired functions (for pins):

                [ ] aud             (7)
                [ ] extperiph3_clk  (29)
                [ ] extperiph4_clk  (31)
                [ ] i2s2            (12,35,38,40)
                [*] pwm1            (15)
                [ ] pwm5            (33)
                [*] pwm7            (32)
                [ ] spi1            (19,21,23,24,26)
                [ ] spi3            (13,16,18,22,37)
                [ ] uarta-cts/rts   (11,36)

                              Back
```

Note: This configuration may already be set up on your Orin from the previous year's labs. If your pins are already configured as PWM, you can skip this step.

3. Configure User Permissions

To allow your user to control the PWM hardware without the *sudo* command, you must add it to the correct system group.

- From within the *gobilda* directory, run the provided setup script:

```
sudo bash env_scrips/jetson_orin.sh
```

(Note: this script will add our calpoly user to the 'pwm' group)

- After the script completes, you must reboot the Orin a final time for the group changes to take effect

After rebooting, verify that your user *calpoly* has been added to the *pwm* group by running the following command: *groups.* In the list of groups that appears, you should see *pwm* included.

Testing Our Robot With *simple_pwm.cpp*

As outlined earlier, this lab uses a standalone C++ implementation instead of ROS2. The final step before beginning the main assignment is to compile and run the *simple_pwm.cpp* test program located in the *jetson_gpio_sys* directory.

This test verifies that our entire hardware and software toolchain is functioning correctly.

Safety Check: Raise the Robot
Before running the executable, you should ensure the robot is raised so its wheels are off the ground.

- Why: This prevents the robot from suddenly driving off your desk.
- How: Place the robot on a box, book or stand. This will be sufficient for now as we are just testing the C++ PWM code.

Compilation and Execution

1. Navigate to the correct directory
```
cd ~/gobilda/jetson_gpio_sys
```

2. Compile the *simple_pwm.cpp* source code into an executable
```
g++ simple_pwm.cpp -o simple_pwm
```

3. Run the executable
```
./simple_pwm
```

Expected Result:
After running the executable, you should observe both motors spin. The goal of this step ultimately is system validation, not code understanding. You do not need to analyze the low-

level details of the C++ code. If both motors spin, your GoBilda system is operational, and you are ready to proceed with the lab assignment!

Lab Assignment

Your primary coding task is to implement the *InverseKinematics* function within the *diff_drive_kin.cpp* file. This function is the core of a simple differential drive control system, translating a desired robot velocity into individual wheel commands.

Task: Implement the *InverseKinematics* function.

Function signature:

- std::pair<double, double> InverseKinematics(const double v, const double w)

Parameters:

- v: The desired linear velocity of the robot (in meters per second). A positive value means forward motion.
- w: The desired angular velocity of the robot (in radians per second). A positive value means counter-clockwise rotation.

Returns:

- A std::pair<double, double> containing the computed angular velocities for the wheels, in the order: <left_wheel_velocity, right_wheel_velocity>. These values should be in radians per second.

Your Implementation:
Inside the function, you will calculate the left (*wl*) and right (*wr*) wheel angular velocities using the standard kinematic model for a differential drive robot. The key concept is that the robot's overall velocity is a combination of the individual wheel speeds.

Key Resources:

- Course Textbook, Chapter 4: The sections on differential drive kinematics.
- Lecture Notes: Review the material on differential drive modeling.

The constants for the robot's wheel radius (R) and wheelbase (L) are provided for you in the code.

Important Code Guidelines

- Your detailed understanding of the code outside the *InverseKinematics* function is not required. Key for this lab is implementing the kinematic equations correctly.
- There are sections of the provided code, that you should not alter.

Validation and Demonstration
Code Verification (Pass the Tests)

- Before you run the code on the robot, you must verify its correctness. Compile your code and run the generated executable.
- The program contains assert statements that will test your InverseKinematics function with known values. If your implementation is correct, the program will output test passed. If it is

incorrect, the program will fail abruptly, indicating a logic error that you must fix. Do not proceed to physical testing until your code passes the assertions.

Demonstration (Demo):
Once your code passes the tests, you are ready for the physical demonstration.

The main function is programmed to execute the following sequence of movements automatically:

- Drive Forward for 2 seconds.
- Drive Backward for 2 seconds.
- Turn Left (in-place) for 2 seconds.
- Turn Right (in-place) for 2 seconds.

To receive full credit, you must show your robot successfully performing this entire sequence of movements.

It is important to understand that this lab uses open-loop control. This means the robot commands wheel velocities without any sensor feedback to correct for errors. In the real world, factors like slight differences in motor performance, friction, and battery voltage mean the robot will not move with good precision.

Therefore, **you will not be graded on the exact precision of the motion**. For example, the robot may not drive in a perfectly straight line, or it may not turn a perfect 90 degrees. You will be graded on the successful execution of the logic—i.e., the robot clearly drives forward, backward, turns left, and turns right as commanded. Significant deviations from the expected behavior (e.g., only one wheel spinning, the robot moving in the wrong direction for a command) should be debugged, but minor imperfections are expected and acceptable.

In short: Focus on getting the logic and kinematics correct in your code. The assert statements are your primary guide for correctness. The demo is about showing functional, purposeful movement, not pinpoint accuracy.

Submission

You are expected to demo your code in the next lab section. Make sure that I get your name down after your successful demo.

After you've demo'ed your code and passed, I will need you to submit your source code.

1. Submit your *diff_drive_kin.cpp* source code onto Canvas