

Lab 5- Advanced Embedded systems

In this session you will **put Alvik on the network**. Using the **MicroPython networking stack** (`network`, `socket`, `usocket`) that ships with the Nano ESP32, you will connect the board as a *station* to a classroom **Wi-Fi router** (no USB tether required). Once both the robot and your PC share the same subnet the fun begins:

- **Bidirectional link** Run `wifiExample.py` on the robot to spin up a tiny TCP server, and `toggleLED.py` on your laptop to connect as a client, proving command-and-telemetry flow.
- **Live data logging** Stream sensor frames to a Python dashboard for plots you can save for later analysis.
- **Remote control** Send drive-speed, LED, and servo commands in real time—an essential step toward untethered Sumo competition.

By the end of the lab you will have a robust wireless pipeline for **monitoring and tuning** your robot, all routed through a standard access-point network that scales to any number of student bots.

1. How the Lab Network Fits Together?

Infrastructure Wi-Fi LAN

We use a *regular* 2.4 GHz Wi-Fi router in **access-point (AP) mode**.

The router's DHCP service hands out IP addresses (e.g., `192.168.0.x`) to every device that joins.

Both the **Nano ESP32 inside Alvik** and your **laptop/PC** connect to this same SSID, so they share one local subnet and can reach each other directly.

Sockets in one sentence

A *socket* is a software endpoint identified by an **IP address + TCP/UDP port** that lets two programs send bytes back and forth as if they were plugged together with a virtual cable.

Who plays which role

Device	Role	Key MicroPython / PC action
Nano ESP32 (Alvik)	TCP server – listens on a chosen port (e.g., 5050) using <code>socket.bind()</code> → <code>socket.listen()</code> → <code>conn, addr = accept()</code>	Runs <code>wifiExample.py</code> to advertise its IP/port and stream sensor data <code>wifiExample</code>
Laptop / PC	TCP client – initiates the connection with <code>socket.connect((ip, port))</code>	Runs <code>toggleLED.py</code> (or your dashboard script) to log data and send drive / LED commands <code>toggleLED</code>
Wi-Fi router	AP + switch + DHCP – just forwards packets between the two peers; no code needed	Provides SSID + password and assigns the IPs

In a Wi-Fi network, a **server** is the device that listens on a specific IP-address-plus-port combination, waiting for incoming socket requests and then serving data or actions, while a **client** is the program that initiates that connection to the server's IP : port, opening a two-way channel over the shared wireless medium; the **port** itself is simply the numeric tag that, alongside the IP address, pinpoints the exact software service being addressed so many independent connections can coexist without colliding.

One robot ↔ one PC

During the lab each student station claims a *dedicated* robot. DHCP still hands out the IPs automatically, but you'll write the address on a sticky label (e.g., *Robot 7* → *192.168.0.17*) so no one cross-controls another unit.

2. Initiate communication with the Alvik platform

Connect your lab pc to the lab router over wifi: Controllab_223 # Controllab

In your lab PC open command line interface and write: **ipconfig** in order to check your ip address.

e.g. 192.168.0.100

```
Wireless LAN adapter Wi-Fi:

Connection-specific DNS Suffix  . : 
Link-local IPv6 Address . . . . . : fe80::fc37:3180:9a8c:93e4%14
IPv4 Address. . . . . : 192.168.0.100
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.0.1
```

Figure 1: Local PC Ip Address 192.168.0.100

In the wifiExample.py script for the Alvik platform update the network credentials.

```
# -----
# Wi-Fi credentials
# -----
WIFI_SSID = "Controllab_223"
WIFI_PASSWORD = "Controllab"

# TCP Server settings
TCP_PORT = 5050 # TCP port
```

Figure 2: Lab Wi-Fi credentials

Run the wifiExample.py script either from main.py or directly and notice the IP Address for the Alvik robot: e.g. 192.168.0.101

```
>OKAlvik robot setup...
Already connected to Wi-Fi: ('192.168.0.101', '255.255.255.0', '192.168.0.1', '192.168.0.1')
TCP server started on port 5050
Waiting for client to connect...
Scheduled receive_from_client() to run every 500 ms.
Starting events thread
```

Figure 3: Alvik IP Address - 192.168.0.101

In command line interface ping the robot to verify that the network is established: e.g. **ping 192.168.0.101**

```
C:\Users\arkad>ping 192.168.0.101

Pinging 192.168.0.101 with 32 bytes of data:
Reply from 192.168.0.101: bytes=32 time=94ms TTL=64
Reply from 192.168.0.101: bytes=32 time=104ms TTL=64
Reply from 192.168.0.101: bytes=32 time=107ms TTL=64

Ping statistics for 192.168.0.101:
    Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 94ms, Maximum = 107ms, Average = 101ms
```

Figure 4: Successful ping of the alvik platform

PuTTY is a lightweight, free terminal application for Windows (also available in portable form) that lets you open raw TCP, Telnet, SSH, and serial sessions from a single interface—perfect for quick Wi-Fi link checks.

Open Putty and initiate communication with the Alvik platform over that IP Address using raw communication over port 5050:

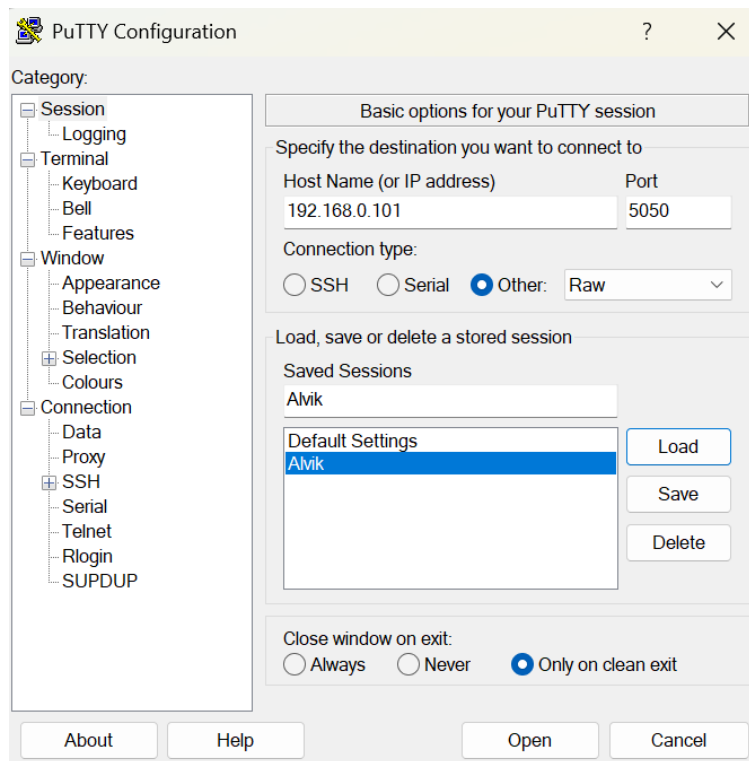


Figure 5: Putty application - Configured for Alvik platform

Send 1 and 0 over the terminal to see the interaction in live:

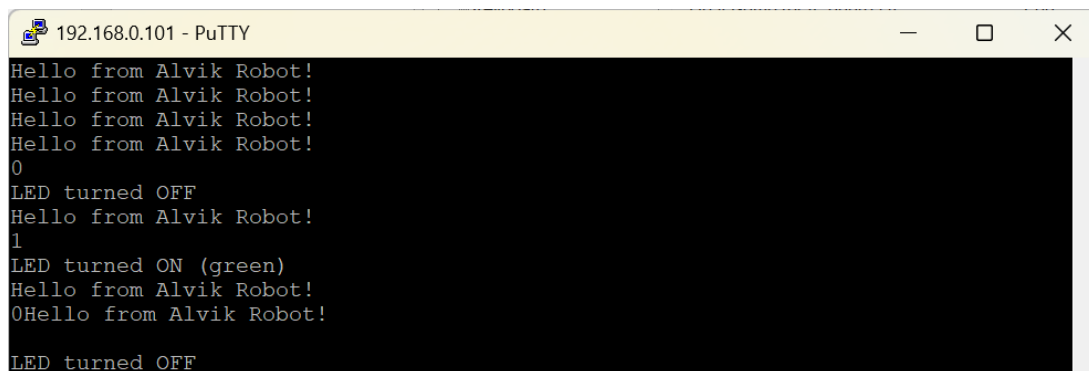


Figure 6: Succesfull interaction with the Alvik platform. Putty --> wifiExample.py

3. Toggling the led over Matlab and Python scripts

Lets proceed toggling the led over Matlab or python (chose environment you are more familiar with).

Matlab Users:

Open Matlab and run the toggleLED example from terminal. Make sure the function script is in the workspace folder.

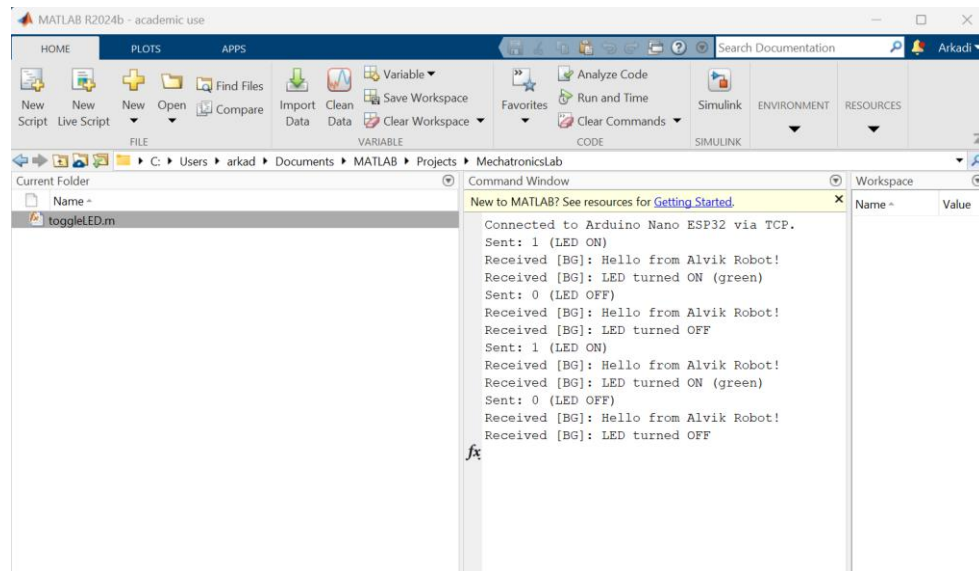


Figure 7: Matlab Enviroment toggleLED.m

Python Users:

Open VSCode or any other environment you are familiar with and run toggleLED.py

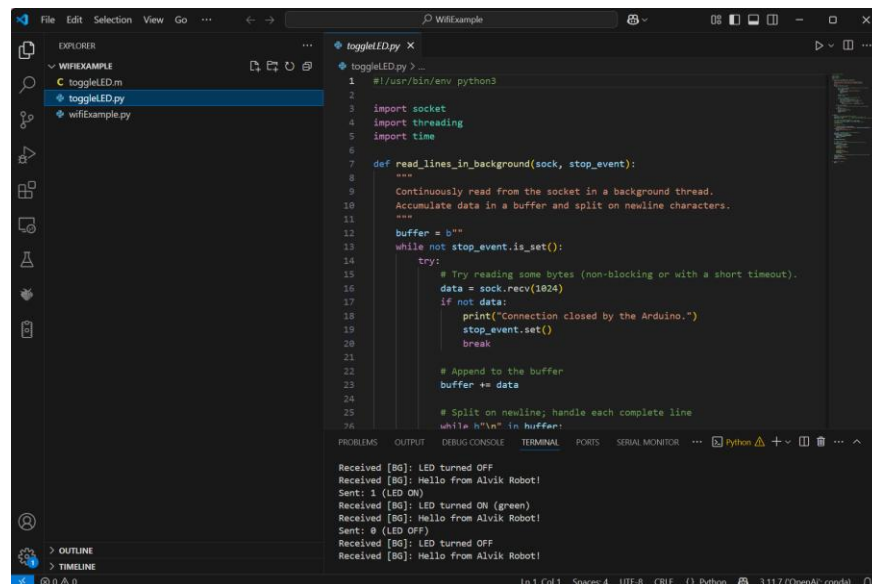


Figure 8: VSCode with Python Enviroment toggleLED.py

How the scripts interact:

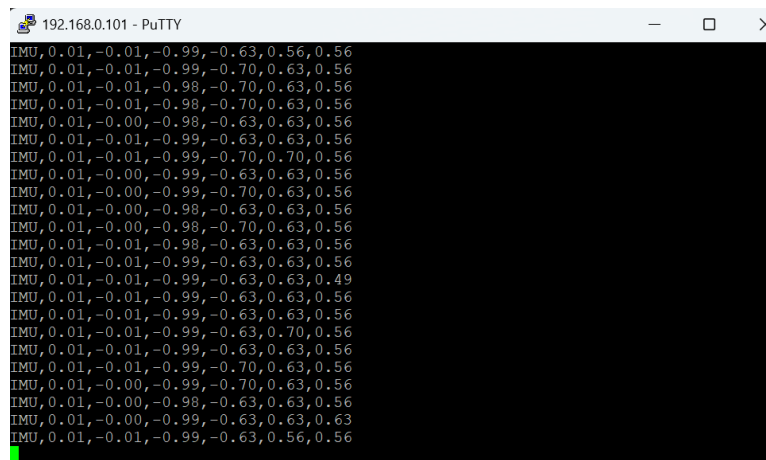
Step	What the PC script does	What the Alvik server (MicroPython) does	Result
Connect	Creates a TCP socket and calls <code>connect((robot_ip, 5050))</code> .	<code>wifiExample_telnet.py</code> (renamed) already executed <code>socket.bind(('0.0.0.0', 5050))</code> , <code>listen()</code> , and <code>accept()</code> , returning <code>client_conn</code> .	A raw, full-duplex byte pipe opens over Wi-Fi.
Background receive thread	<code>read_lines_in_background()</code> continuously calls <code>recv(1024)</code> , buffers bytes, splits on <code>\n</code> , and prints each complete ASCII line.	Whenever the robot invokes <code>send_to_client("...\r\n")</code> , those bytes flow into the socket.	Status lines such as “LED turned ON (green)” appear in the PC console.
Command loop	Alternately sends <code>b"1\r\n"</code> (LEDs on) and <code>b"0\r\n"</code> (LEDs off), pausing two seconds between sends.	<code>parse_message()</code> strips CR/LF, checks for "1" or "0", toggles the left & right LEDs, and echoes confirmation via <code>send_to_client()</code> .	LEDs flash green → off → green in sync with the PC's “Sent:” messages and the robot's replies.
Timeout / non-blocking	Sets <code>sock.settimeout(2.0)</code> ; if no data arrives, it simply loops, keeping the UI responsive. Uses a <code>threading.Event</code> to terminate cleanly.	Server socket and client socket both run with <code>settimeout(0)</code> , so networking never blocks robot tasks.	Neither side freezes; both can handle other work while the link is idle.
Cleanup	Signals the background thread to stop (<code>stop_event.set()</code>), joins it, then closes the socket.	Next <code>recv()</code> raises an exception; robot calls <code>close_client()</code> and resumes <code>accept()</code> to await the next PC.	Connection ends gracefully, ready for a new raw-TCP session on 5050 .

In short: the PC script **sends ASCII “1” or “0” terminated with CR-LF** to the robot, the robot's MicroPython server **decodes that single-character command to switch its LEDs**, and then **returns a human-readable status line** that the PC's background thread prints—demonstrating bidirectional Wi-Fi communication with minimal code on both ends.

4. Practical demonstration for data logging:

Systematic **data logging** turns each test-drive into a reproducible experiment: by capturing time-stamped sensor readings, actuator commands, and error flags you can trace why a line-following algorithm drifted, quantify battery sag over a match, or tune a PID loop with real numbers instead of guess-and-check. Streaming those logs over **Wi-Fi** means the robot stays untethered—its on-board processing is not blocked by file writes, and a laptop can visualize plots in real time, flag anomalies instantly, and store gigabytes of data without filling the microcontroller's flash. In short, wireless logging converts every run into actionable insight while preserving the robot's full mobility.

- Upload and run wifiIMU.py on the Alvik platform
- Open Putty and see the IMU logs:



A screenshot of a PuTTY terminal window titled "192.168.0.101 - PuTTY". The window displays a continuous stream of IMU data logs. Each line represents a timestamped reading of acceleration (ax, ay, az) and angular velocity (wx, wy, wz). The data is formatted as "IMU, timestamp, ax, ay, az, wx, wy, wz". The values for acceleration range from approximately -1.00 to 0.56 g, and angular velocity values range from approximately -0.99 to 0.63 rad/s. The logs are displayed in a black background with white text.

Figure 9: IMU Logging over Putty

- A more advanced application using wifiLogger.py

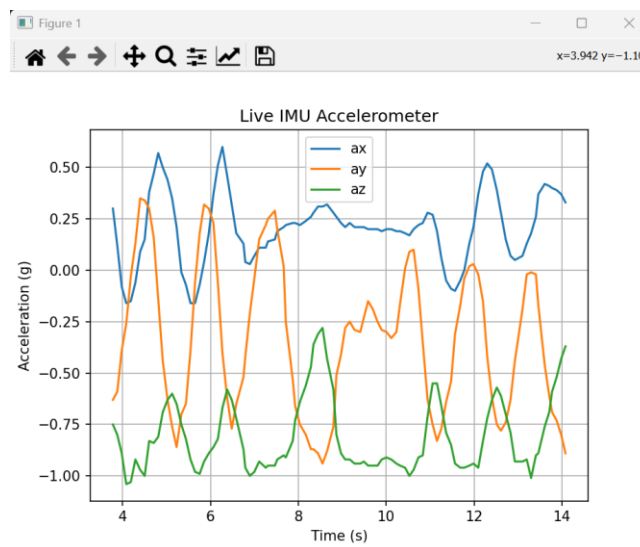


Figure 10: IMU logging using wifiLogger.py

wifiLogger.py not only plots the data it also saves it for later analysis.

5. Teleoperation

Tele-operating your Alvik over Wi-Fi is an effortless on-ramp to robotics control. A lightweight TCP console lets you drive, steer, and stream live IMU data from your PC, instantly revealing how throttle, turning radius, and sensor readings map to real-world motion. Because these keystroke commands traverse the same network stack you'll later use for autonomy—broadcasting odometry and receiving path-planning targets—each session builds familiarity with both the robot's mechanics and its communications pipeline. This hands-on “joystick” practice sharpens your feel for wheel slip, latency, and heading drift, while real-time sensor feedback clarifies what those numbers mean in motion. In short, Wi-Fi teleop is a risk-free sandbox for rapid experimentation: explore, iterate, and internalize the robot's quirks before you hand control over to code.

- Upload and run `wifiTeleoperate.py` on the Alvik platform
- Open Putty and send over commands to make Alvik move

Command summary for `wifiTeleoperate.py`

Key sent from PC	Category	Action on Alvik
2	Speed preset	Set <code>current_speed</code> = 20 RPM
3	Speed preset	Set <code>current_speed</code> = 30 RPM
4	Speed preset	Set <code>current_speed</code> = 40 RPM
5	Speed preset	Set <code>current_speed</code> = 50 RPM
w	Motion	Drive forward at <code>current_speed</code> on both wheels
s	Motion	Drive backward at <code>current_speed</code> on both wheels
d	Motion	Rotate right (in-place): left wheel <code>+current_speed</code> , right wheel <code>-current_speed</code>
a	Motion	Rotate left (in-place): left wheel <code>-current_speed</code> , right wheel <code>+current_speed</code>
1	LEDs	Both LEDs ON (green, RGB 0 1 0)
0	LEDs	Both LEDs OFF

*Line-feed (`\n`) and carriage-return (`\r`) characters are ignored, so pressing **Enter** alone has no effect.*

- Upgraded version of the logger, compatible with simple teleoperation: `wifiConsoleLogger.py`

6. “What’s Next?” - Expanding Your Alvik Experience

#	Enhancement	Quick synopsis	Skills to exercise
1	Wi-Fi servo control	Add a micro servo to the PWM header and map new console keys (<code>u/j</code>) to absolute or incremental angles.	PWM generation, payload encoding, mechanical calibration, latency awareness
2	Stream the color + line sensors	Extend the IMU packet with <code>COL</code> , <code>R</code> , <code>G</code> , <code>B</code> and <code>LINE</code> , <code>L1</code> , <code>L2</code> frames; plot reflectance vs. floor patterns in the PC logger.	Multi-sensor time-stamping, packet framing, data visualization, surface-classification intuition
3	Scan-and-log with the TOF sensor	Sweep the robot slowly while logging distance; build a 1-D lidar plot on the PC side.	I ² C device management, range filtering, basic mapping concepts
6	Macro recording & autonomous replay	Capture a user’s key-press sequence + sensor traces, save to CSV, then replay it untethered as an ‘automation demo.’	File I/O, timing reproduction, safety interlocks, introduction to scripting autonomy

For the Lab report:

1. Check-Your-Understanding Questions

- Distinguish “server” and “client” in your Wi-Fi setup. Which device plays each role during LED toggling, and why?
- What is a TCP **port** and why did we choose **5050**?
- Explain the difference between **blocking** and **non-blocking** sockets, and give one line-number example from each of the two Python scripts.
- List every ASCII command understood by `wifiTeleoperate.py` and state its effect.
- Describe one runtime error you encountered (e.g., `NameError` on `client_conn`). What debugging steps led you to the fix?
- Propose a test to verify that IMU sampling at 10 Hz
- Describe one runtime error you encountered (e.g., `NameError` on `client_conn`). What debugging steps led you to the fix?

2. Post-Lab Reflection Tasks

- Experience summary (≤150 words)** - narrate the most important thing you learned and the biggest surprise you encountered while tele-operating Alvik.
- Block diagram** – Draw a labelled block diagram of the complete data path—from your laptop keyboard to Alvik’s wheel motors and IMU feedback—showing:
 - Device blocks (PC, Wi-Fi AP/Router, Alvik MCU, motor drivers, IMU).
 - Interfaces/protocols (TCP-IP, UART, PWM, I²C).
 - Direction of each signal (Tx/Rx).
- Code review**– Review thoroughly both the `wifiExample.py` and `toggleLED.py` or `toggleLED.m` and prepare a block diagram to represent the codes and their interactions. Use the help of the documentation table in the manual.
- Post lab analysis**- prepare a plot for one of the logs taken during the lab
- Suggest one enhancement from the “**What’s Next**” table you’d priorities for the Sumo competition, and justify your choice in ≤100 words.