



SWE 4211 Lab 6: Performance and Robot Start

1 Introduction

Last week's lab introduced a network-controlled lighting system, allowing us to toggle lights on and off at designated times. This week, we'll build on those concepts with an eye toward scalability—extending the architecture to support broader robotic systems.

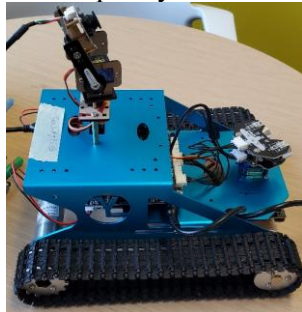
We'll also compare the performance of Raspberry Pi 3 and Raspberry Pi 4. While the Pi 4 is expected to outperform its predecessor due to its newer hardware, we'll quantify that difference to understand its practical impact.

By the end of this lab, you'll not only be able to control the robot's lights, but you'll also have a more robust and extensible infrastructure - laying the groundwork for a well-architected robotic system.

2 Lab Objectives

- Compare the real-time performance of two embedded computers
- Control a robot remotely using simple network protocols
- Develop embedded code using network sockets
- Reuse a previously developed queue for aiding in the development of a system.
- Construct software which uses periodic tasks to complete a simple task.
- Integrate simple network-based controls for a robot

3 Laboratory Equipment Needed

1 Micro SD Card with Rasperry PI OS Installed 	Yahboom Robot for Raspberry Pi Smart Car AI Starter Kit 
Robot Charger 	1 Screwdriver 



4 Deliverables / Submission

When completed, record a short video, using either your phone or windows, about 30 seconds long, showing the ability to change the colors of lights on the robot remotely. Commit the final code and push it to the server.

In addition, submit a report to Canvas with the following:

1. Title Page
 - a. Assignment
 - b. Date
 - c. Name / Link to your GitHub classroom repository
 - d. Video link
2. Introduction
 - a. What are you trying to accomplish with this lab? This section shall be written in multiple, grammatically correct sentences in your own words. Clearly describe what is intended in this lab.
3. Diagrams
 - a. Sketch a block diagram for the robot system, as was given for the light controller lab, identifying the inputs and outputs to the software system as it exists right now. (Hint: Keep this somewhere as you will be expanding upon it in future labs)
4. Data Tables / Explanations
 - a. Include the graphs of all 5 cyclic tests on the pi / virtual machine.
 - b. Include screen shots of the P key and htop.
5. Discussion
 - a. Discuss the performance of the robot
 - i. How do the results of the 5 different forms of cyclic test compare? Are there global patterns which would indicate why you would want to run in a certain mode? Be sure to reference back to the graphs in providing your answer and analysis.
 - b. Design Discussion
 - i. What is the purpose for the NetworkQueueProxy and QueueManager within the system? How do these components make communication easier or harder than in the previous lab?
6. Things gone right / Things gone wrong
 - a. This section shall discuss the things which went correctly with this lab. Include key things that came simply in completing this lab and experiences from other labs which helped to complete this exercise.
 - b. This section shall also discuss the problems that occurred completing this lab. Include issues with C++ programming that occurred, issues with networking, or other problems encountered when successfully completing the lab.
7. Conclusions
 - a. What have you learned with this experience?
 - b. What improvements can be made in this experience in the future?

This material should be submitted as a single pdf file.

If you have any questions, consult your instructor.



5 Measuring Deterministic Response on the Pi 3

The first thing we would like to do on the Pi is to run a simple experiment to measure the response times for several different systems. We will determine the performance of the stock Raspberry PI OS running in two different modes.

5.1 Experimenting with the base Raspbian Image¹

Boot up your raspberry pi and make a single ssh connection to it via the virtual machine (i.e. **ssh pi@<yourpiname>.local**). On the Raspberry PI, issue the following commands²:

```
cat /proc/cpuinfo | grep Model  
sudo uname -a3
```

Capture this output for the report to verify that your board is correct. Then continue with the following commands.

```
sudo apt-get update  
sudo apt-get install gnuplot  
sudo apt install build-essential libnuma-dev  
git clone git://git.kernel.org/pub/scm/utils/rt-tests/rt-tests.git rt-tests-current  
cd rt-tests-current  
make all4
```

```
sudo cpupower frequency-set -g powersave  
sudo ./cyclicttest -h 100 -p 80 -q -i 1000 -l 100000 --smp > histogram1.dat5
```

```
echo 'set term png; set xlabel "latency in microseconds"; set ylabel "count"; set output "20252026plotpi1.png"; plot  
"histogram1.dat" with linespoints lc rgb "blue";' | gnuplot
```

Ultimately, this will run a program, cyclicttest, on your Raspberry Pi. Per the man pages,

“Cyclicttest measures the amount of time that passes between when a timer expires and when the thread which set the timer actually runs. It does this by taking a time snapshot just prior to waiting for a specific time interval (t1), then taking another time snapshot after the timer finishes (t2), then comparing the theoretical wakeup time with the actual wakeup time (t2--(t1 + sleep_time)). This value is the latency for that timer wakeup.”

This experiment allows us to evaluate how deterministic the response time is on a given system - a key characteristic of real-time performance. In this initial run, the system operates in power-save mode, where the processor intentionally slows down to conserve energy. This reduced clock speed may lead to lower overall performance.

Our goal is to observe whether a deterministic system maintains relatively consistent latency, ideally with slightly lower latency than a general-purpose system. The output will be a histogram plot in PNG format that visualizes latency measurements.

¹ Note: You will be doing the same thing on your robot. If partners would like, one partner can do this experiment on the Pi3 with their device while the other partner does it on the robot. Note that there are some slight differences in the commands from the powersave statement on for the robot.

² Note: some may take a while to run

³ Note: Capture this for your lab report. Uname gives you details about your OS, specifically what you are running.

⁴ Note: The code is throwing a few compiler errors and warning right now, but the code does build. This is not reflective of good quality code for doing this, but this is also what happens with open-source code on occasion.

⁵ The -h option dumps the results into a histogram. The -p sets the priority of the first thread. -q causes the system to be quiet and only print a summary on exit. The -i sets the interval for thread wakeups and -l controls how many times the thread is woken up.



Once the experiment is completed, transfer both the .png image and the accompanying .dat text file to your local machine (hint: sftp⁶ is likely the most convenient method). The histogram will display latency in microseconds along the x-axis, and the frequency of each latency value on the y-axis. Be sure to save the images for inclusion in your final report.

5.2 Performance Mode Measurement Exercise

We now want to see what the impact is of running the system in performance mode. In performance mode, “The CPUfreq governor “performance” sets the CPU statically to the highest frequency within the borders of scaling_min_freq and scaling_max_freq.”⁷ This is different from the power save mode we ran in previously, in which “The CPUfreq governor “powersave” sets the CPU statically to the lowest frequency within the borders of scaling_min_freq and scaling_max_freq.”⁸

Issue the following commands inside of your VM to replicate the experiment on your Raspberry Pi in performance mode.⁹:

```
cd ~
sudo cpupower frequency-set -g performance
cd rt-tests
sudo ./cyclictest -h 100 -p 80 -q -i 1000 -l 100000 --smp > histogram2.dat

echo 'set term png; set xlabel "latency in microseconds"; set ylabel "count"; set output "20252026plotpi2.png"; plot
"histogram2.dat" with linespoints lc rgb "blue";' | gnuplot
```

5.3 Shut down your Pi

When you have finished these steps, shut down your Raspberry Pi by issuing the command `sudo shutdown now`. This will power off the Raspberry Pi. When the green light has stopped blinking, indicating there is no longer any file IO, unplug your Pi from both power and the network.

5.4 Virtual Machine Measurement Exercise

Now open another terminal window on your VM. Issue the following commands inside of your VM (NOT ON YOUR PI) to replicate the performance experiment on your Linux VM¹⁰:

```
sudo cat /proc/cpuinfo | grep model
sudo uname -a
```

Capture this output for the report to verify that your board is correct. Then continue with the following commands.

```
cd~
sudo apt-get install git
sudo apt-get install libnuma-dev
sudo apt-get install gnuplot
git clone git://git.kernel.org/pub/scm/utils/rt-tests/rt-tests.git rt-tests-current
cd rt-tests
make all
sudo ./cyclictest -h 100 -p 80 -q -i 1000 -l 100000 --smp > histogram3.dat
```

⁶ Here is a tutorial on sftp: [How to Use SFTP to Safely Transfer Files: A Step-by-Step Guide \(hostinger.com\)](https://www.hostinger.com/tutorials/how-to-use-sftp-to-safely-transfer-files/) (https://www.hostinger.com/tutorials/how-to-use-sftp-to-safely-transfer-files/)

⁷ <https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt>

⁸ <https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt>

⁹ Note: These commands are slightly different than what you did previously.

¹⁰ Note: These commands are slightly different than on the PI.



```
echo 'set term png; set xlabel "latency in microseconds"; set ylabel "count"; set  
output "20252026plotvm.png"; plot "histogram3.dat" with linespoints lc rgb "blue";'  
| gnuplot
```

Compare the three plots. What are the differences between the plots? How does the raw latency differ between the three plots? How does the variance in latency differ across the plots? How do these differences explain why one is more useful for a real system time than the others? (Note: Keep the plots for the writeup.)

6 Yahboom Raspberry Pi Robot G1 Tank

The Yahboom Raspberry Pi Robot G1 Tank is a rugged, tank-style mobile platform equipped with multiple sensors and powered by a Raspberry Pi 3 or 4 running Linux with Real-Time Extensions. It features a rotatable USB camera for visual feedback, line-following sensors for autonomous navigation, and a front-mounted ultrasonic sensor for distance measurement.

Power is supplied via a custom battery pack mounted beneath the chassis, charged using a 12.6V 800 mA adapter. **Do not use a charger with higher voltage or current ratings—doing so may pose a serious fire hazard.**

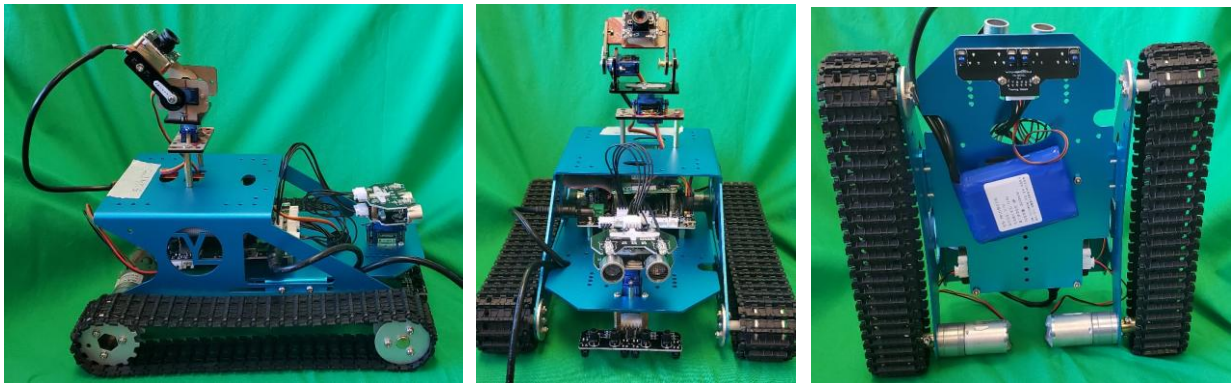


Figure 1 The robot. In the left photo, you can see the camera on the top, tilted slightly back, the distance sensor on the front, and the tank tracks. In the center, you can see the distance sensor more clearly in the front, as well as the line sensor pointing toward the ground. The camera is again in the top of the photo. On the right you can see the battery pack affixed to the bottom of the robot.

Your robot arrives fully assembled with the Raspberry Pi already mounted. However, you'll need to install your micro-SD card to complete the setup.

To do this, carefully insert the micro-SD card into the slot on the underside of the Raspberry Pi. The card should be oriented with the metal contacts facing upward and the label facing downward as you slide it in. Take care not to force the card—if aligned properly, it should seat smoothly. You should make sure the battery is unplugged when doing this.



Figure 2 Inserting the uSD card into the Raspberry Pi. On the left we see the uSD card in the middle, right above the pins and under a display cable plug. On the right we see the card being inserted, looking toward the bottom side of the board.

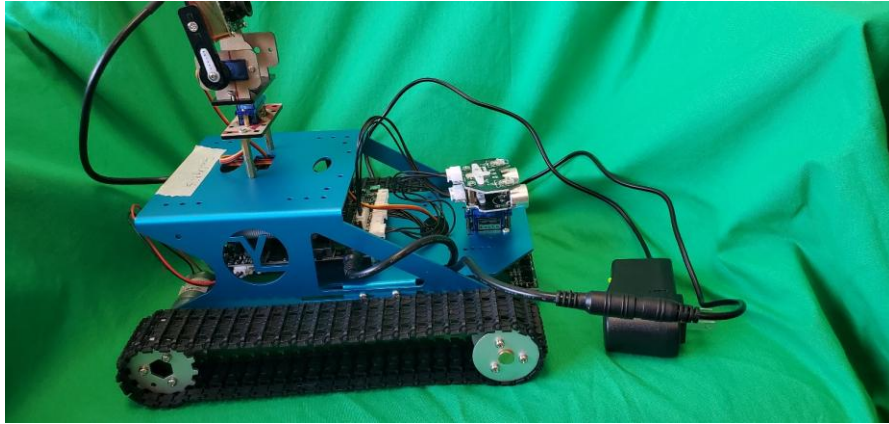


Figure 3 The robot and the charger. Plug the charger into the cable. Make sure the robot is off when doing this.

Before operating the robot, you must fully charge its battery. If the battery is completely discharged, expect the charging process to take several hours.

While charging:

- The indicator light on the charger will glow **red or orange**.
- Once fully charged, the light will turn **green**.

⚠ Important Safety Notes:

- These are lithium batteries - always charge them on a **non-flammable surface**.
- **Do not leave the robot unattended** while charging.
- Ensure the robot is **powered off** during charging. Use the small switch located just to the left of the charging plug to turn it off.

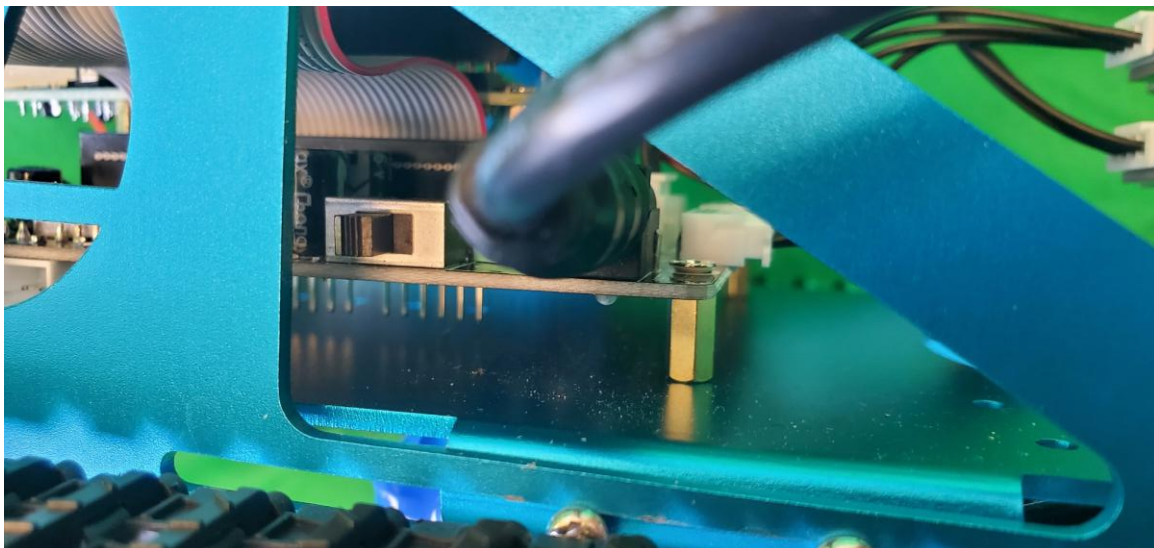


Figure 4 The power switch. Off is slid to the left.



7 Connecting to your Pi

With the uSD card moved into your robot, we want to ssh to the device. To do this, issue the command **ssh pi@<yourpiname>.local**

When doing this, you may get an error “Remote Host Identification has Changed”. This happens because, at some point in the past, you connected to the Pi or tried to connect to it, but since then something has changed in its setup. In our case, this is happening because you took the microSD card and put it into a different Pi. Basically, the message is protecting you, telling you that you should not connect to the Pi because it doesn't match what is expected.

To fix this, you can change into the /home/swestudent/.ssh/ directory and delete the known_hosts file (aka **sudo rm known_hosts**). Once you do this, you can connect.

8 Rerunning our experiment

We want to start seeing the difference between the Pi 3 and the Pi 4 in terms of performance. On the Raspberry Pi, issue the following commands:

On the Raspberry PI, issue the following commands¹¹:

```
cat /proc/cpuinfo | grep Model  
sudo uname -a12
```

Save this screen output for your report.

```
cd rt-tests-current  
sudo cpupower frequency-set -g powersave  
sudo ./cyclicttest -h 100 -p 80 -q -i 1000 -l 100000 --smp > histogram4.dat13
```

```
echo 'set term png; set xlabel "latency in microseconds"; set ylabel "count"; set output "20252026plotpi4.png"; plot  
"histogram4.dat" with linespoints lc rgb "blue";' | gnuplot
```

This will rerun the test on the pi4 in power save mode. When finished, issue the following commands inside of your VM to replicate the experiment on your Raspberry Pi in performance mode.¹⁴:

```
sudo cpupower frequency-set -g performance  
sudo ./cyclicttest -h 100 -p 80 -q -i 1000 -l 100000 --smp > histogram5.dat
```

```
echo 'set term png; set xlabel "latency in microseconds"; set ylabel "count"; set output "20252026plotpi5.png"; plot  
"histogram5.dat" with linespoints lc rgb "blue";' | gnuplot
```

Once these have completed, grab the data files and the images for future usage.

¹¹ Note: some may take a while to run

¹² Note: Capture this for your lab report. Uname gives you details about your OS, specifically what you are running.

¹³ The -h option dumps the results into a histogram. The -p sets the priority of the first thread. -q causes the system to be quiet and only print a summary on exit. The -i sets the interval for thread wakeups and -l controls how many times the thread is woken up.

¹⁴ Note: These commands are slightly different than what you did previously.

9 Designing our software

The design of the software is straightforward, as is shown in the UML below in Figure 5.

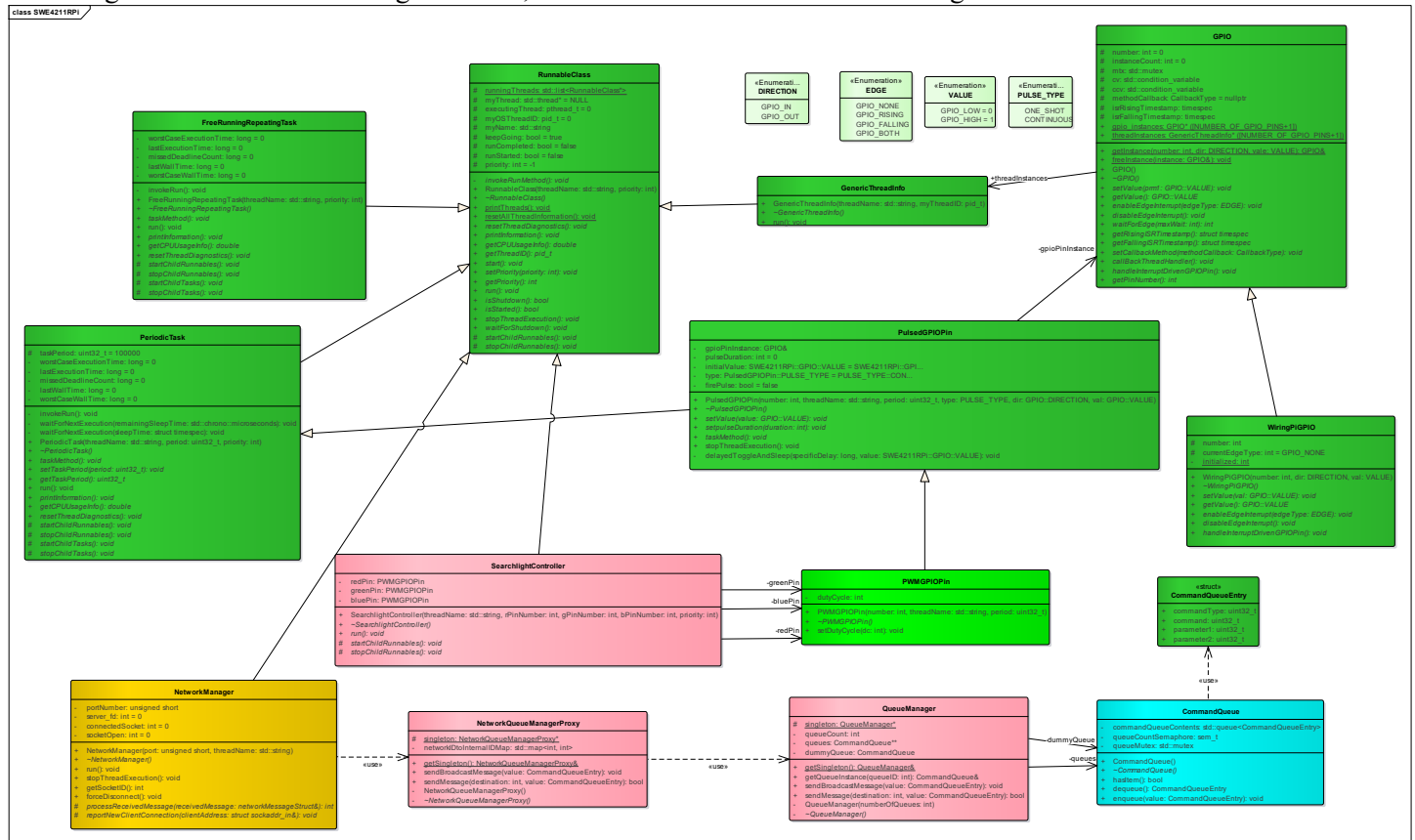


Figure 5 UML Design for the project. Classes you are provided with are shown in green. Classes you are re-using from a previous lab are shown in blue. Classes you must implement are shown in pink. Classes which you previously implemented but will require some refactoring are shown in yellow.

In this lab, your primary focus will be implementing basic event processing and robot control. While the functionality may seem straightforward, several design patterns are at play—most notably the **singleton pattern**, which is used in multiple classes to manage controlled access to shared resources. You can observe this structure in the provided UML diagrams.

9.1 Network Manager

In last week's lab, you wrote a lot of the network manager. In this week's lab, you will need to make a few changes, as the network manager will not be directly enqueueing messages any more. That functionality will be handled by other classes. The changes, however, should be rudimentary.

9.2 Queue Manager

The **Queue Manager** provides a consistent and decoupled mechanism for inter-component communication. Rather than requiring one class to know the internal details of another, a component simply requests that a message be enqueued for a target device. The Queue Manager ensures that the message is routed to the correct queue, abstracting away the implementation details and promoting modularity.

9.3 Network Proxy Manager

The **Network Proxy Manager** serves a similar purpose, but for inbound messages received from the network. It ensures that these messages are dispatched appropriately within the system. To support this architecture, you'll need to make minor adjustments to the **Network Manager**. Specifically, it will no longer enqueue events directly. These changes help decouple the networking layer from the event processing logic, improving maintainability and scalability.

9.4 Searchlight Controller

The **Searchlight Controller** is intentionally simple. It listens for events on its queue and responds by activating the robot's lights. The lights will illuminate in the color specified by the event parameters.

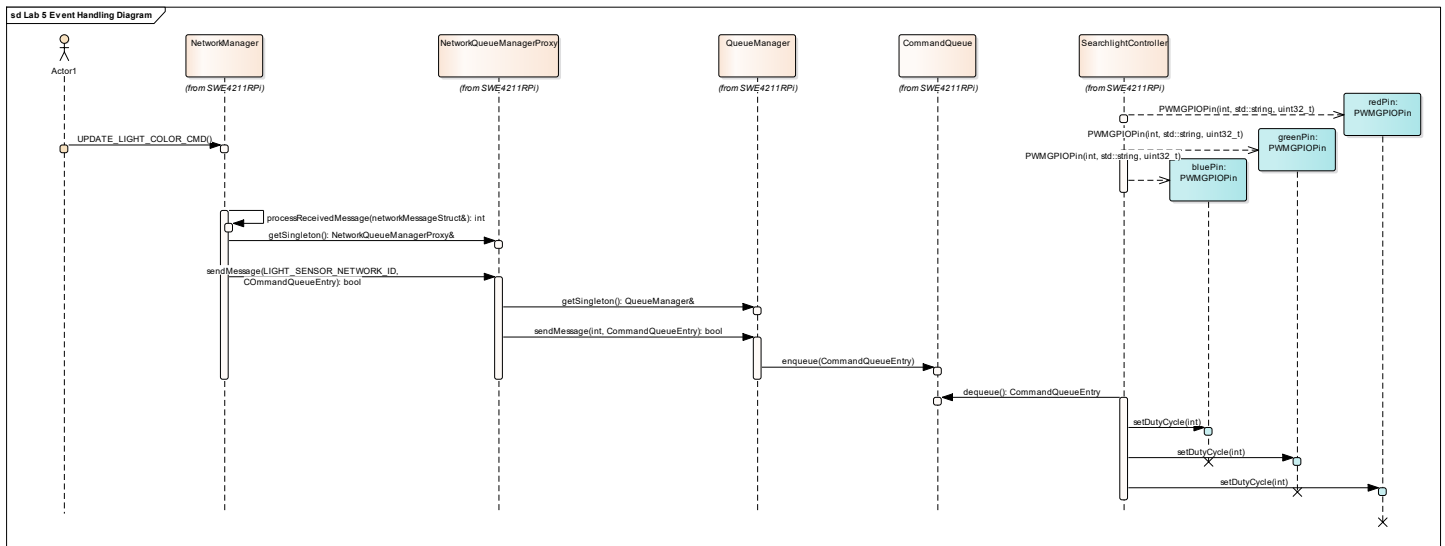


Figure 6 Communication of events.

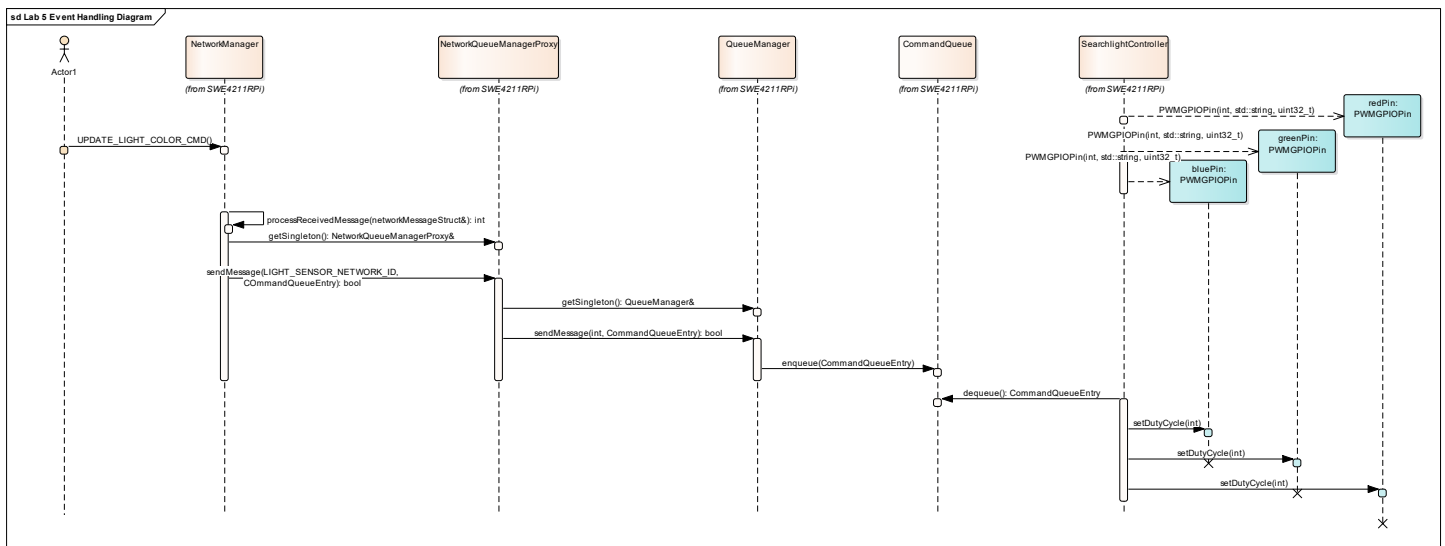


Figure 6 illustrates the event-handling flow of the system using a sequence diagram. On the left, messages are initiated by the Java program and transmitted over the network to the robot.

1. **Message Reception:** The **Network Manager** receives incoming messages. If the data structure is complete and valid, the message is accepted for processing.

2. **Message Routing:** The Network Manager forwards the message to the **Queue Proxy**, which acts as an intermediary. The proxy then passes the message to the **Queue Manager**.
3. **Queue Assignment:** The Queue Manager analyzes the message parameters to determine the appropriate destination queue. It then enqueues the message accordingly.
4. **Event Processing:** Once the message is enqueued, the corresponding thread, such as the one running in the **Searchlight Controller**, is notified. The thread wakes up, dequeues the event, and processes it.
5. **Action Execution:** In this case, the Searchlight Controller interprets the event and changes the robot's light color based on the parameters provided.

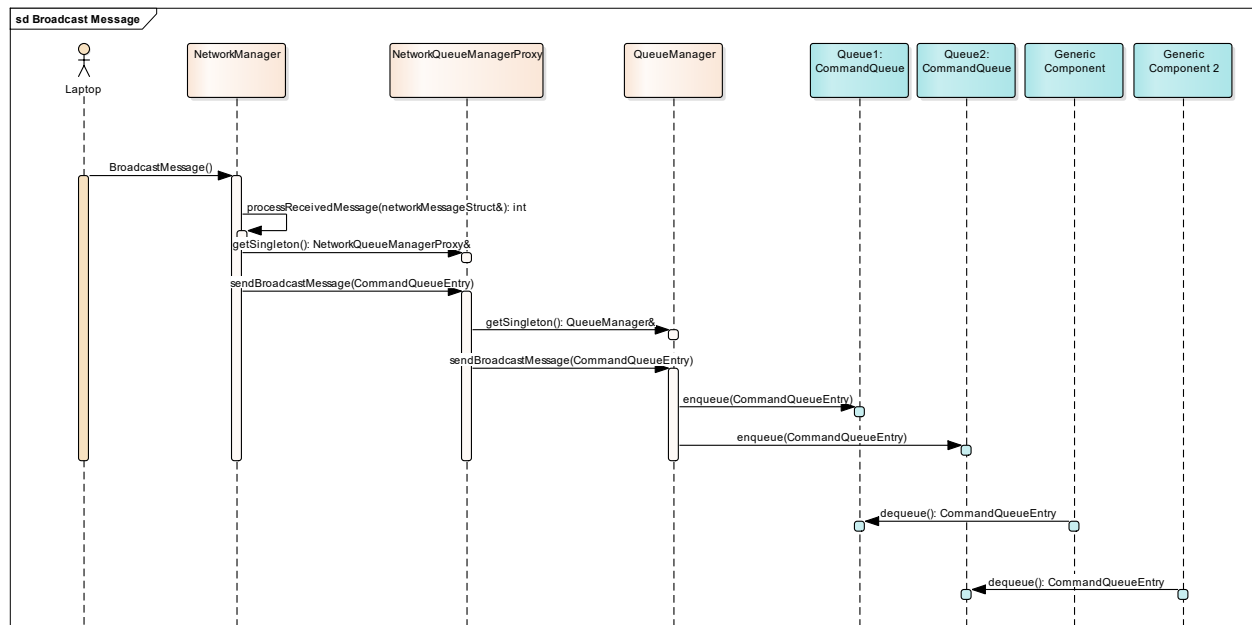


Figure 7 A broadcast message example.

In addition to device-specific events, the system architecture supports **broadcast messaging**, allowing a single event to reach all components.

Figure 7 illustrates this process:

1. **Message Origin:** A broadcast message is initiated from the laptop and sent to the robot's software over the network.
2. **Queue Distribution:** Upon receiving the message, the **Queue Manager** identifies it as a broadcast and ensures that the event is enqueued to **all active device queues**, not just a single target.
3. **Component Activation:** On the right side of the diagram, each generic component receives the broadcast event from its respective queue. This enables synchronized behavior across multiple subsystems without direct coupling.

This design promotes modularity and scalability, allowing new components to respond to broadcast events without altering the core messaging logic.

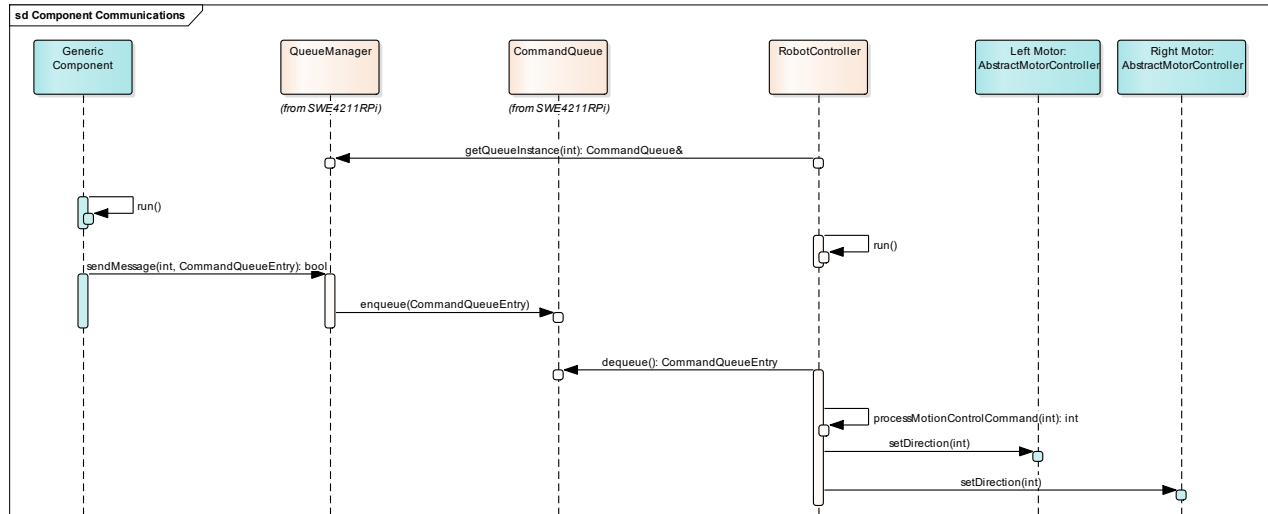


Figure 8 Components communicating using the queue.

Beyond handling external events, the queue system also facilitates communication between internal components. Figure 8 illustrates a generic component interacting with the **Robot Controller** through the queue mechanism.

For example:

- A **line-following module** may generate steering commands and enqueue them to the Robot Controller as the system attempts to track a line.
- A **collision-sensing module** might enqueue distance measurements or alerts when a potential obstacle is detected.

This design allows components to remain decoupled. Each module can send messages without needing to know the internal details of the controller or other subsystems.

Although this functionality is not implemented in the current lab, it will be introduced in future labs to support more advanced behaviors and modular system design.

10 Controlling our robot

10.1 Safety Reminder Before Starting Code Execution

Before running your program, always be mindful of your robot's physical environment. Ensure that:

- Someone is securely holding the robot, **or**
- The robot is placed on the floor—not on a table—so it cannot accidentally drive off and fall.

Although motor control code has not yet been implemented, real-time systems can behave unpredictably if something goes wrong. Unexpected movement is unlikely but still possible, so take precautions to avoid damage or injury.

10.2 Program Startup Behavior

When the program begins execution, diagnostic information will be printed to the console. This includes:

- The **IP address** of the Raspberry Pi
- A list of **runnable classes** that have been initialized

This output helps verify that the system is configured correctly and that key components are active.

```
[?2004hpi@schillingpi3u51:~$ gdbserver -u :2345 /home/pi/lab6new; exit
[?2004l

Process /home/pi/lab6new created; pid = 847
Listening on port 2345
Remote debugging from host ::ffff:192.168.4.132, port 38912
Main thread id is : 847
lo IP Address 127.0.0.1
eth0 IP Address 192.168.4.121
lo IP Address ::1
eth0 IP Address fe80::999b:6d7f:c420:f69c
Starting Runnable named: NetworkManager
Starting Runnable named: Robot On Screen Stat...
Starting Runnable named: Search light Thread
Starting Runnable named: Red LED PWM
Starting Runnable named: Green LED PWM
Starting Runnable named: Blue LED PWM
```

Figure 9 The startup screen.

To operate the system using the UI shown in Figure 10:

1. **Start the Program**¹⁵
 - Launch the control program on the robot.
 - Then start the Java-based UI on your PC.
2. **Connect to the Robot**
 - Enter the robot's IP address into the connection field.
 - Press **Connect**.
 - The robot's **Network Manager** will accept the connection, enabling the Java program to control the robot.

At this stage, control is limited to changing the robot's light colors.

3. **Open the Color Panel**

¹⁵ Note that for best performance, you should start this program on your native windows machine using Java 11 or newer. It has been tested with IntelliJ.

- From the **Tools** menu, select **Open Color Chooser**.
- This opens the color selection panel.
- 4. **Change the Light Color**
 - Choose a color from the panel.
 - Press **Apply** to send the color change command to the robot.
- 5. **Disconnecting**
 - Once connected, a **Disconnect** button will appear.
 - Pressing it will terminate the session, allowing another PC to connect and take control of the robot.

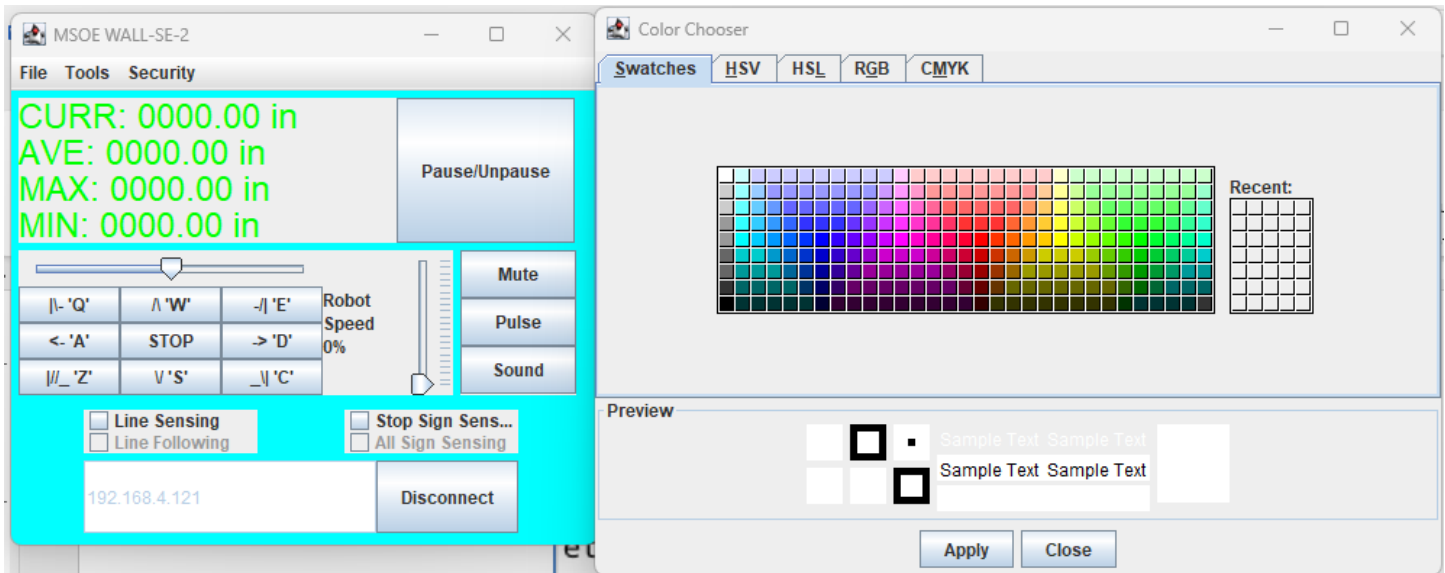


Figure 10 Control UI. Note that in this case, the robot is connected to IP address 192.168.202.122.

The communication protocol is the same as was used with the light and is shown in Table 1.

If you run the program in IntelliJ, the commands that are being sent will be dumped out to the console, as is shown in Figure 11. You will need to interpret the hex a bit, but these are the specific values being sent in the structure.

Table 1 Network message and structure definition.

Bytes and offsets	Type	Description
0	int32_t securityMode	This is a flag for the security mode. Overall, it is 0 unless the message is encrypted in some format. If it is nonzero, then some form of encryption is being used.
4	int32_t messageID	This is a unique ID for this message. It should always increase by 1 each time a message is sent. There never should be a duplicate message ID until the system rolls over after $2^{32}-1$ messages, which should take a while.
8	int32_t timestampHigh	This is the timestamp for this message. It is the number of ms since the start of the Epoch.
12	int32_t timestampLow	
16	int32_t messageType	This is the type of message that is being transmitted. COMMAND_MSG_TYPES are the only supported message types right now. Anything else should be ignored.
20	int32_t messageDestination	This is the destination. This is a numerical destination on the given device for the message (i.e. the queue). The first queue is 1. The second queue is 2, etc. This should never be negative.
24	int32_t message	This is the message that is to be sent.
28	int32_t parameter1	This is the first parameter that is to be sent.
32	int32_t parameter2	This is the second parameter that is sent.
36	int32_t xorChecksum	This is the checksum which will verify that the message has been properly received. It is a simple XOR of the words in the message.

```

Run  Main x
[Icons]
↑ Sent: AES 0 MID 48 TS 199 a80aa2f8 T 100 D 7 MSG 2000061 CK aaf5a27c
↓ Sent: AES 0 MID 49 TS 199 a80ab24c T 100 D 7 MSG 2000061 CK aa397efa
⇅ Sent: AES 0 MID 4a TS 199 a80ab5bd T 5aa55aa5 D 8 MSG 4000020 CK f6afeee3
⇅ Sent: AES 0 MID 4b TS 199 a80abef6 T 100 D 7 MSG 2000061 CK aa6c8dbd
⇅ Sent: AES 0 MID 4c TS 199 a80ac946 T 5aa55aa5 D 8 MSG 4000020 CK f6af921e
⇅ Sent: AES 0 MID 4d TS 199 a80ac946 T 100 D 7 MSG 2000061 CK aa0aaf92
⇅ Sent: AES 0 MID 4e TS 199 a80ad95c T 100 D 7 MSG 2000061 CK aaf51512
⇅ Sent: AES 0 MID 4f TS 199 a80adcd0 T 5aa55aa5 D 8 MSG 4000020 CK f6af878b
⇅ Sent: AES 0 MID 50 TS 199 a80ae7c8 T 100 D 7 MSG 2000061 CK aa0a2bfe
⇅ Sent: AES 0 MID 51 TS 199 a80af058 T 5aa55aa5 D 8 MSG 4000020 CK f6afab1d
⇅
⇅
⇅

```

Figure 11 Console output from the program. MID is the message ID, always increasing by 1. TS is the timestamp of when the message was enqueued. The type is following the T. D is the destination queue. MSG is the message that was sent, and CK is the checksum.



11 Testing our robot

Testing at this point will be arbitrary. To make sure the robot is working, start the program and verify that the light colors change accordingly. Note that the colors will not likely match the color swatch exactly due to how the system works. But, you should be able to see the red, green, and blue components of the colors changing accordingly.

With your robot code running, enter a capital P into the terminal where the debug session is running and press enter. You should see a debug output similar to that shown in the bottom of this image. This is a diagnostic screen built into the pi code that will allow us to see how different runnable and periodic tasks are performing on our robot. For now, not all of the information will be useful, but you should be able to see info about CPU usage. How close is this to what htop is showing? Note that the PIDs are the same across windows.

```
Terminal - pi@schillingpi3u61: ~
File Edit View Terminal Tabs Help

0[|
1[|
2[|
3[|
Mem[|||||
Swp[|

0.7% Tasks: 32, 13 thr, 122 kthr; 1 running
0.7% Load average: 0.00 0.01 0.00
0.0% Uptime: 06:18:39
0.7%
127M/3.70G
0K/2.00G

Main 1/0
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
3372 root -41 0 182M 3736 0 S 0.7 0.1 0:00.95 /home/pi/debug/lab6
3360 root 20 0 20604 6680 5540 S 0.0 0.2 0:00.05 sudo gdbserver localhost:2345 /home/pi/debug/lab6
3364 root 20 0 20604 2628 1492 S 0.0 0.1 0:00.00 sudo gdbserver localhost:2345 /home/pi/debug/lab6
3365 root 20 0 15056 4828 4228 S 0.0 0.1 0:00.76 gdbserver localhost:2345 /home/pi/debug/lab6
3367 root 20 0 182M 3736 3368 S 0.0 0.1 0:00.00 /home/pi/debug/lab6
3368 root -30 0 182M 3736 0 S 0.0 0.1 0:00.01 /home/pi/debug/lab6
3369 root 20 0 182M 3736 0 S 0.0 0.1 0:00.00 /home/pi/debug/lab6
3371 root -41 0 182M 3736 0 S 0.0 0.1 0:00.96 /home/pi/debug/lab6
3373 root -2 0 182M 3736 0 S 0.0 0.1 0:00.00 /home/pi/debug/lab6
F1Help F2Setup F3Search F4FILTER F5Tree F6SortBy F7Nice F8Nice + F9Kill F10Quit

File Edit View Terminal Tabs Help
3367 main 0 RUN
Total CPU Usage: 1.284
=====
R
P
=====
Thread Diagnostic Information:
Thread Task Prio. Type period (us) Last Exec. (us) WCET (us) Last Wall Time (us) WCWT (us) CPU Usage CPU Usage
3372 Blue LED PWM 40 PER 25000 16 77 27 341 0.308%
3371 Green LED PWM 40 PER 25000 19 23 30 38 0.092%
3370 Red LED PWM 40 PER 25000 17 25 29 41 0.100%
3373 Search light Thread 1 RUN
3369 Robot On Screen Stat... 0 RUN
3368 NetworkManager 29 RUN
3367 main 0 RUN
Total CPU Usage: 0.500
=====
```



12 Enabling WIFI on the Raspberry PI (if not already done)

At some point during this lab, you will want to enable Wi-Fi access on the raspberry pi (if you have not done so already). This can be done early in the lab or later. It really does not matter when. If done earlier, then it may make driving the vehicle easier. However, it may also lead to more challenges with connectivity. With wired connections, it tends to be a bit easier to diagnose problems than with Wi-Fi connections, as you can easily ping devices and use Wireshark to monitor the network more easily. But driving a tethered robot is not as much fun.

The tasks described here will need to be completed each time you work on a different network. In general, you will need to complete this once for the cybersecurity lab network, but if you work in your apartment or in a different location on a different Wi-Fi setup, then you will need to complete this task again for that network. The beauty is that you can always revert to a wired connection, even once this is completed.

12.1 Before you begin

Start with your Pi connected by Ethernet and an SSH session open to it from your laptop. Every step through Part 5 is done over this wired session, so you never lose access while setting up Wi-Fi.

You will need:

- A Raspberry Pi running Raspberry Pi OS Trixie Lite, with a working SSH login over Ethernet.
- The SSID (network name) and passphrase of the lab Wi-Fi network, provided by your instructor.
- Your laptop connected to that same Wi-Fi network, so it can reach the Pi once the cable is removed.
- A place to record information. You will write down your Pi's wired and Wi-Fi IP addresses.

Before starting, record the wired IP address you are currently using to SSH to the Pi:

`hostname -I`

12.2 Part 1: Clear existing Wi-Fi connections

Remove every saved Wi-Fi connection so your Pi starts from a clean state. This only removes wireless connections; your wired Ethernet connection, and your SSH session, are left alone.

1. List the connections currently saved on the Pi:

`nmcli connection show`

Each row is one saved connection. The TYPE column shows wifi for wireless connections and ethernet for wired ones. Note how many Wi-Fi connections are listed. It may be zero.

2. Delete all saved Wi-Fi connections at once:

`nmcli -t -f UUID,TYPE connection show | awk -F: '{ $2=="802-11-wireless" } { print $1 }' | xargs -r sudo nmcli connection delete`

This pipeline lists every connection by its unique ID (UUID) and type, keeps only the wireless ones, and deletes each one. If there are no Wi-Fi connections, it does nothing.

3. Confirm that only wired connections remain:

`nmcli connection show`

You should see no rows with type wifi. A wired connection and possibly a loopback (lo) entry will still be listed. That is expected.



Do not delete the wired connection. If you do, you will likely lose your SSH session and need a keyboard and monitor to recover.

12.3 Part 2: Prepare the Wi-Fi radio

The Pi keeps its Wi-Fi radio disabled until a country code is set, because legal radio channels differ by country. Set the country and make sure the radio is turned on.

1. Set the Wi-Fi country to the United States:

```
sudo raspi-config nonint do_wifi_country US
```

2. Make sure the radio is not blocked:

```
sudo rfkill unblock wifi
```

3. Check the status of the network devices:

```
nmcli device status
```

Look for the wlan0 row. Its STATE should be disconnected. If it says unavailable, repeat steps 1 and 2. If there is no wlan0 row at all, ask your instructor, as your Pi model may not have built-in Wi-Fi or there may be a hardware failure.

12.4 Part 3: Scan for available networks

Ask the Pi to scan for nearby wireless networks and find the lab network in the results.

1. Trigger a fresh scan:

```
sudo nmcli device wifi rescan
```

2. Wait about 5 seconds, then list what the Pi found:

```
nmcli device wifi list
```

Each row is one network. The most useful columns are:

Column	Meaning
SSID	The network name. Must be typed exactly, including capitalization and spaces.
CHAN	The radio channel. Channels 1–14 are 2.4 GHz; 36 and above are 5 GHz.
SIGNAL	Signal strength from 0 to 100. Above 50 is good.
SECURITY	The security type. WPA2 or WPA3 means a passphrase is required.

3. Find the lab network in the list and record its details:

SSID: _____ Signal: _____ Security: _____

If the lab network does not appear, run the rescan again and wait a little longer. Some older Pi models only support 2.4 GHz and will not see 5 GHz-only networks.

12.5 Part 4: Connect to the Wi-Fi network

Connect to the lab network using the SSID you recorded in Part 3. NetworkManager saves the connection and will reconnect to it automatically whenever it is in range, including after a reboot.



1. Connect, replacing LabNetwork with the exact SSID. Keep the quotation marks, especially if the SSID contains spaces:

```
sudo nmcli --ask device wifi connect "LabNetwork"
```

The `--ask` option makes nmcli prompt you for the passphrase. Type it and press Enter. The passphrase will not be shown on screen.

2. Wait for the result. On success you will see a message similar to:

Device 'wlan0' successfully activated with '...'

If you see an error such as Secrets were required, but not provided, the passphrase was wrong. Run the command again. You may see guides that put the passphrase directly on the command line with password "...". That works, but the passphrase is then saved in your shell history (`~/.bash_history`) in plain text. Using `--ask` avoids this.

Either way, NetworkManager stores the passphrase in plain text in a root-only file under `/etc/NetworkManager/system-connections/`. Anyone with physical access to your uSD card can read it. Keep this in mind before saving your home network's passphrase on the Pi.

12.6 Part 5: Verify the connection and record the Wi-Fi IP address

Confirm the Pi is on Wi-Fi and find the IP address it was given. This address is different from your wired address, and it is the one you will use once the cable is unplugged.

1. Check the device status:

```
nmcli device status
```

Both `eth0` and `wlan0` should now show connected.

2. Show the Wi-Fi interface's address:

```
ip -4 addr show wlan0
```

Find the line beginning with `inet`. The address before the `/` is your Wi-Fi IP address. For example, in `inet 192.168.1.57/24`, the address is `192.168.1.57`.

3. Record it:

Wi-Fi IP address: _____

4. Test that the Pi can reach the network over Wi-Fi specifically:

```
ping -c 4 -I wlan0 8.8.8.8
```

The `-I wlan0` option forces the ping out through the wireless interface. You should see 4 replies and 0% packet loss. If the lab network blocks internet access, ping your laptop's IP address instead.

5. Check the Pi's hostname, which gives you a second way to reach it:

12.7 Part 6: Switch from the wired connection to Wi-Fi

Prove the Wi-Fi connection works before removing the cable, then unplug and log in over Wi-Fi only. Testing first means that if something is wrong, you still have your wired session to fix it.



1. Leave your current wired SSH session open. On your laptop, open a second terminal window.
2. Make sure your laptop is connected to the same Wi-Fi network as the Pi.
3. From the second terminal, ping the Pi's Wi-Fi IP address recorded in Part 5:
`ping -c 4 <wifi-ip-address>`
(On Windows, use `ping -n 4` instead of `ping -c 4`.)
4. Still in the second terminal, open a new SSH session to the Pi's Wi-Fi address, replacing username with your Pi login:
`ssh username@<wifi-ip-address>`
If this works, you now have two sessions to the same Pi: one wired, one wireless. If it does not work, stop here and go to Troubleshooting. Do not unplug the cable.
5. Type `exit` in the wired SSH session to close it.
6. Unplug the Ethernet cable from the Pi.
7. Return to your Wi-Fi SSH session and confirm it is still responsive:
`nmcli device status`
`eth0` should now show unavailable and `wlan0` should show connected. Your session survived because it never used the cable.
8. Close the session with `exit`, then reconnect from scratch to show you can reach the Pi wirelessly from a cold start:
`ssh username@<wifi-ip-address>`
You can also try connecting by hostname, which keeps working even if the IP address changes later:
`ssh username@<hostname>.local`

You can return to a wired connection at any time by plugging the cable back in and using your wired IP address. Both connections can be active at once.

12.8 Troubleshooting

Most problems come from the radio being blocked, a mistyped SSID or passphrase, or the laptop and Pi being on different networks.

Symptom	Likely cause	What to try
<code>wlan0</code> shows unavailable	Country not set or radio blocked	Repeat Part 2. Check with <code>rfkill list</code> ; Wi-Fi should show <code>Soft blocked: no</code> .
No <code>wlan0</code> device at all	Pi has no built-in Wi-Fi	Ask your instructor about a USB Wi-Fi adapter.
Lab network missing from scan	Out of range, 5 GHz-only network, or scan too soon	Run <code>sudo nmcli device wifi rescan</code> , wait 10 seconds, list again. Move closer to the access point.
Secrets were required, but not provided	Wrong passphrase	Run the Part 4 connect command again and retype carefully.
No network with SSID ... found	SSID typed incorrectly	Copy the SSID exactly from <code>nmcli device wifi list</code> , including capitalization and spaces.



Connect command asks for a username or fails on a campus network

Network uses WPA2-Enterprise (username and password login)

The simple connect command does not support this. Ask your instructor for the lab network settings.

12.9 Command quick reference

Task	Command
List saved connections	<code>nmcli connection show</code>
Show device status	<code>nmcli device status</code>
Delete one saved connection	<code>sudo nmcli connection delete "<name>"</code>
Set Wi-Fi country	<code>sudo raspi-config nonint do_wifi_country US</code>
Unblock the Wi-Fi radio	<code>sudo rfkill unblock wifi</code>
Scan for networks	<code>sudo nmcli device wifi rescan</code>
List nearby networks	<code>nmcli device wifi list</code>
Connect to a network	<code>sudo nmcli --ask device wifi connect "<SSID>"</code>
Connect to a hidden network	<code>sudo nmcli --ask device wifi connect "<SSID>" hidden yes</code>
Disconnect without forgetting	<code>sudo nmcli connection down "<name>"</code>
Show the Wi-Fi IP address	<code>ip -4 addr show wlan0</code>
Show all IP addresses	<code>hostname -I</code>
Menu-based alternative	<code>sudo nmtui</code>