



SWE 4211 Lab 7: Basic Robot Control

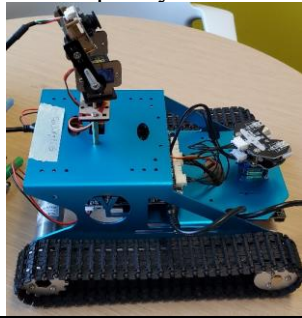
1 Introduction

In the last lab, we started doing work with the network. We were able to control the LEDs over the network, changing colors as was desired. Now, with the aid of our robot, we are going to be able to perform simple operations, making the robot drive forward and backward and turn left and right, as well as spin left and right. We also will setup the robot to During this lab, we will also setup the robot so that it can be controlled wirelessly over Wi-Fi by adding your Raspberry Pi to the Wi-Fi network if you have not already done so.

2 Lab Objectives

- Integrate simple control systems into a robot.
- 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.
- Control the creation of sound through a piezo buzzer

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 robot working with narration explaining how the robot is driving and what is going on in the background. You should demo the major robot directional operations as well as show the display showing CPU usage (i.e. press the P key). You should also demonstrate the horn going into reverse and colors changing if you choose to do that part. 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 screen shots of the P key and htop.
5. Discussion
 - a. Speed analysis
 - i. What is the lowest PWM setting that your robot moved at? What constraints within the robotic system impact this value?
 - b. Integration
 - i. Architecturally, what is the advantage of using broadcast messages for the status messages versus specific targeted messages going to individual components? What is the disadvantage of this approach?
 - ii. How difficult will it be to add new components that respond to status messages based upon the design being used?
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 Enabling WIFI on the Raspberry PI

If you have not already done so, the first step of lab will be to enable Wi-Fi on your robot. Last week's lab had instructions for this. Canvas has a slightly modified writeup with some troubleshooting added. In terms of networks, we ideally want to move to the point of having both your laptop and your robot on Wi-Fi. In the cyber lab, you have likely used one of the three networks that are segregated. If you look at the network credentials listed, there is an overall cyber network for SWE4211 that you can also use. Once you move over to Wi-Fi, you should not need to use the wired connections any more other than if debugging is necessary. But, be forewarned, debugging gets a bit more challenging.

The number one thing to remember once you move to Wi-Fi is that both your laptop and the pi must be on the same Wi-Fi and your laptop needs to be bridged to the Wi-Fi using the Wi-Fi adapter for this to work properly. If your laptop reverts to the MSOE network, then you will not be able to connect.

Your instructor can help you with this if you are having problems.

6 Key New Features

6.1 Driving the robot

As you build out this lab, the first thing you want to achieve is the ability for the robot to drive. You should be able to control your robot from the GUI, making the robot drive forward, backwards, left, and right.

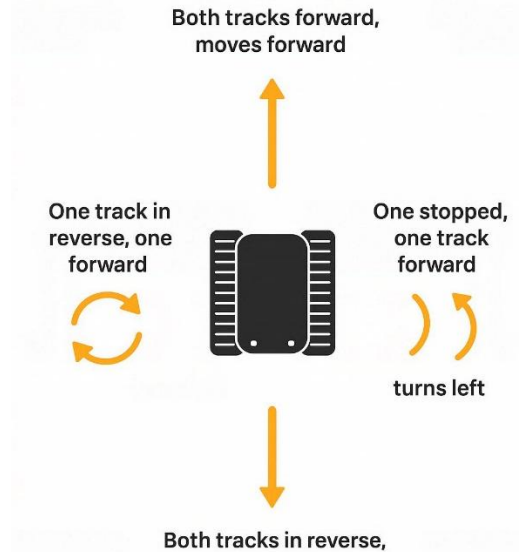


Figure 1 Steering the robot.

Tracked vehicles steer by varying the relative motion of their left and right tracks. When both tracks move forward at the same speed, the vehicle travels straight ahead. If one track moves forward while the other remains stationary, the vehicle arcs gently in the direction of the stopped track—this is often used for gradual turns. For sharper maneuvering, driving one track forward and the other in reverse causes the vehicle to spin in place around its central axis, a technique known as pivot turning. When both tracks move in reverse, the vehicle backs up in a straight line.

This differential steering method allows tracked vehicles to navigate tight spaces and rough terrain with precision. By controlling the speed and direction of each track independently, operators can execute a wide range of movements—from subtle course corrections to full rotations—without relying on traditional steering mechanisms.



Figure 2 Piezobuzzer for making noise.



6.2 Making noise

Your robot has a piezo buzzer mounted on the top. The piezo buzzer will make noise when enabled and be quiet otherwise. We do not have control over the volume or frequency emitted by the device, which makes our job somewhat easier, as the device is either making noise or silent. The piezo buzzer is connected to GPIO8.

In this lab, we will add control for the “horn” so that when the vehicle is going forward, the horn will be quiet, when the vehicle is moving in reverse, we will hear a beeping sound. The beeping sound will consist of the tone being emitted for 300ms and then being silent for 300ms.

Luckily, controlling the piezo buzzer is very straight forward, as it just is a GPIO output, and unlike the LED's, it is active high, so sending a 1 will enable sound and 0 will inhibit sound.

By default, the horn is disabled in `lab_cfg.h` file when you start working on the lab. Once you have some semblance of completed horn code, enable this feature by changing the config file definition.

7 Designing our software

Overall, this lab will involve writing 3 classes and making slight modifications in a 4th. Figure 3 provides UML for the project. The color coding is consistent with previous labs. Green is provided code for you. Pink is new code you will be writing. Yellow is code you have previously written which may require slight changes. Cyan is code you have written which should not require any changes in this lab.

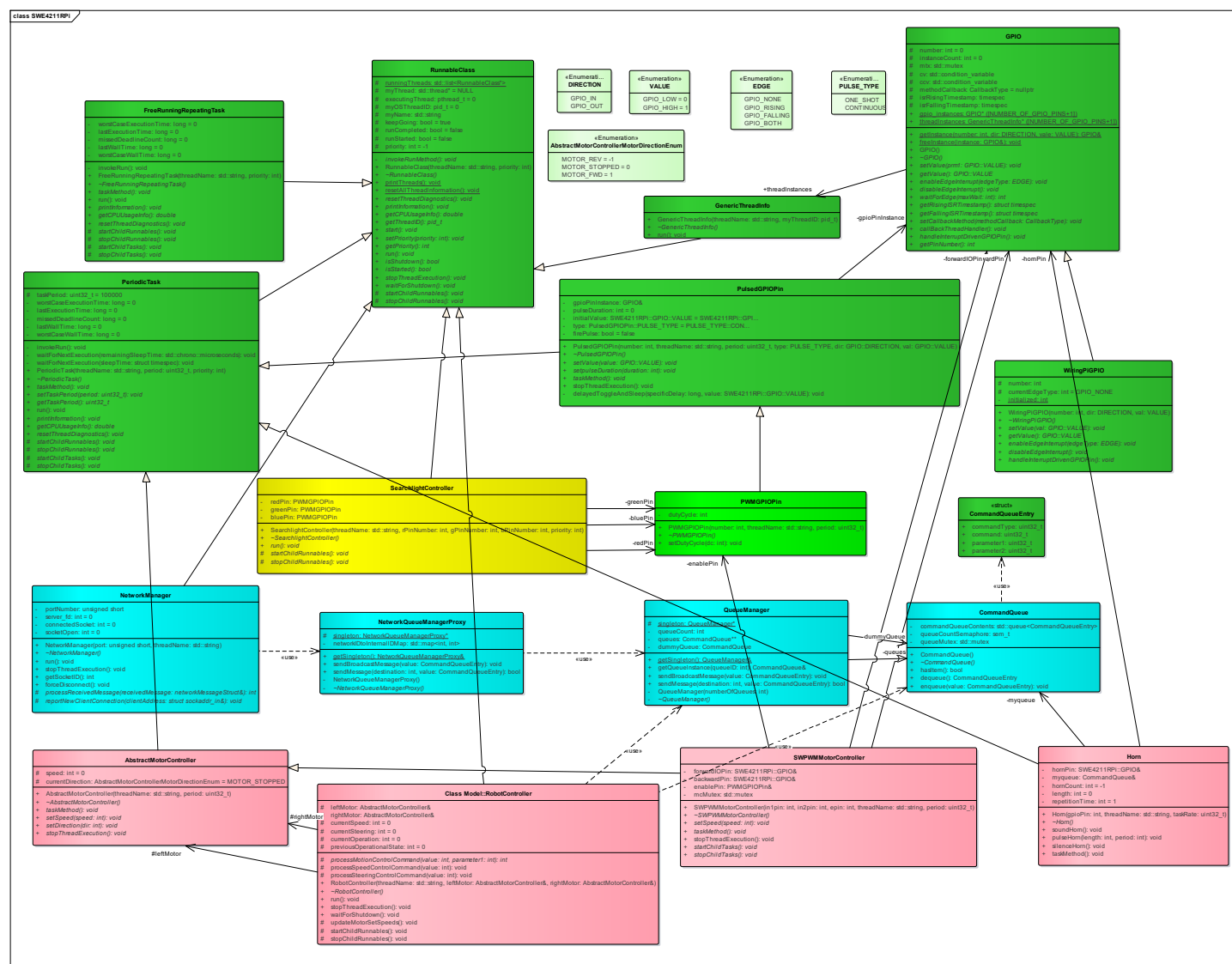


Figure 3 Project UML.



7.1 Robot Control

The robot will receive network commands from the java code via the ethernet socket. The commands will cause the robot to behave accordingly, driving, turning, and stopping. Some commands will have parameters associated with them which control how the command is interpreted. The commands that you will receive across the network are given in Table 1.

Table 1 Robot Control Commands. These are all command types that will be received by the class and processed.

Type	Command Name	Hex Value	Description																		
COMMAND_MSG_TYPE	MOTOR_DIRECTION_CONTROL_CMD	0x02000011	This indicates that a motor direction control command has been sent across the network to the given device. The robot should respond accordingly.																		
	<table><tr><th>Parameter Name</th><th>Hex Value</th><th>Description</th></tr><tr><td>MOTOR_FORWARD</td><td>0x00000001</td><td>This indicates that the robot is commanded to drive forward.</td></tr><tr><td>MOTOR_BACKWARD</td><td>0x00000002</td><td>This indicates the robot is commanded to drive backward.</td></tr><tr><td>MOTOR_LEFT</td><td>0x00000004</td><td>This indicates the robot is commanded to steer to the left while moving.</td></tr><tr><td>MOTOR_RIGHT</td><td>0x00000008</td><td>This indicates the robot is commanded to steer to the right while moving.</td></tr><tr><td>MOTOR_STOP</td><td>0x00000010</td><td>This indicates that the robot is commanded to stop.</td></tr></table>			Parameter Name	Hex Value	Description	MOTOR_FORWARD	0x00000001	This indicates that the robot is commanded to drive forward.	MOTOR_BACKWARD	0x00000002	This indicates the robot is commanded to drive backward.	MOTOR_LEFT	0x00000004	This indicates the robot is commanded to steer to the left while moving.	MOTOR_RIGHT	0x00000008	This indicates the robot is commanded to steer to the right while moving.	MOTOR_STOP	0x00000010	This indicates that the robot is commanded to stop.
Parameter Name	Hex Value	Description																			
MOTOR_FORWARD	0x00000001	This indicates that the robot is commanded to drive forward.																			
MOTOR_BACKWARD	0x00000002	This indicates the robot is commanded to drive backward.																			
MOTOR_LEFT	0x00000004	This indicates the robot is commanded to steer to the left while moving.																			
MOTOR_RIGHT	0x00000008	This indicates the robot is commanded to steer to the right while moving.																			
MOTOR_STOP	0x00000010	This indicates that the robot is commanded to stop.																			
COMMAND_MSG_TYPE	MOTOR_SPEED_CONTROL_CMD	0x02000021	This command indicates that a motor speed command has been sent. The motor speed is given in pwm duty cycle, with a value between 0 and 1000 representing 0 to 100 percent duty cycle.																		
	<table><tr><th>Parameter</th></tr><tr><td>Motor Duty Cycle: 0 -> 0% duty cycle. 1000 -> 100% duty cycle. Values outside of the range of 0 to 1000 are invalid.</td></tr></table>			Parameter	Motor Duty Cycle: 0 -> 0% duty cycle. 1000 -> 100% duty cycle. Values outside of the range of 0 to 1000 are invalid.																
Parameter																					
Motor Duty Cycle: 0 -> 0% duty cycle. 1000 -> 100% duty cycle. Values outside of the range of 0 to 1000 are invalid.																					
COMMAND_MSG_TYPE	MOTOR_STEERING_OFFSET_CONTROL_CMD	0x02000031	This command indicates that the steering is to be adjusted accordingly. The steering parameter, which is a value between 0 and 200, with 100 representing straight forward / no adjustments to the steering. Note that the network parameter is of the range 0 to 200, which may not match the needs of the called method.																		
	Parameter: The value ranges between 0 and 200. A value of 100 represents that the robot should drive straight ahead with no adjustment. A value of 0 indicates there is a strong offset to the left, and a value of 200 represents a strong offset to the right. Values outside of the range 0 to 200 are invalid.																				

Your robot will also need to report status. Whenever the robot changes directions or states of motion, it will broadcast its state out to other components. These commands are defined in Table 2.

Table 2 Robot Status reports. These are all broadcast messages the component will send.

Type	Command Name	Hex Value	Description																		
STATUS_REPORT_MSG_TYPE	REPORT_DIRECTION_STATUS	0x08000021	This will be a report with the MOTORDIRECTIONBITMAP parameters added (LEFT, RIGHT, etc.)																		
	<table><tr><th>Parameter Name</th><th>Hex Value</th><th>Description</th></tr><tr><td>MOTOR_FORWARD</td><td>0x00000001</td><td>This indicates that the robot is commanded to drive forward.</td></tr><tr><td>MOTOR_BACKWARD</td><td>0x00000002</td><td>This indicates the robot is commanded to drive backward.</td></tr><tr><td>MOTOR_LEFT</td><td>0x00000004</td><td>This indicates the robot is commanded to steer to the left while moving.</td></tr><tr><td>MOTOR_RIGHT</td><td>0x00000008</td><td>This indicates the robot is commanded to steer to the right while moving.</td></tr><tr><td>MOTOR_STOP</td><td>0x00000010</td><td>This indicates that the robot is commanded to stop.</td></tr></table>			Parameter Name	Hex Value	Description	MOTOR_FORWARD	0x00000001	This indicates that the robot is commanded to drive forward.	MOTOR_BACKWARD	0x00000002	This indicates the robot is commanded to drive backward.	MOTOR_LEFT	0x00000004	This indicates the robot is commanded to steer to the left while moving.	MOTOR_RIGHT	0x00000008	This indicates the robot is commanded to steer to the right while moving.	MOTOR_STOP	0x00000010	This indicates that the robot is commanded to stop.
	Parameter Name	Hex Value	Description																		
	MOTOR_FORWARD	0x00000001	This indicates that the robot is commanded to drive forward.																		
	MOTOR_BACKWARD	0x00000002	This indicates the robot is commanded to drive backward.																		
	MOTOR_LEFT	0x00000004	This indicates the robot is commanded to steer to the left while moving.																		
	MOTOR_RIGHT	0x00000008	This indicates the robot is commanded to steer to the right while moving.																		
	MOTOR_STOP	0x00000010	This indicates that the robot is commanded to stop.																		

The robot controller is responsible for controlling the robot's operation. As is shown in Figure 4, the robot uses the queue to obtain command information and broadcast status information. The robot controller has two motors that are part of it,

and these motors are realized as SWM PWM motors. Each of these motors has 3 GPIO pins, a forward pin, a backward pin, and an enable pin. These pins are described in Table 3 Motor Control Pins. The robot has two motors.

Table 3 Motor Control Pins

Motor GPIO Pin	Purpose	FWD Value	Stopped Value	Backward Value
forwardIOPin	This pin will enable the motor to run forward.	HIGH	LOW	LOW
backwardPin	This pin will enable the motor to run backward.	LOW	LOW	HIGH
enablePin	This pin will control motor speed.	PWM Value between 0 and 100 percent controlling motor speed.		

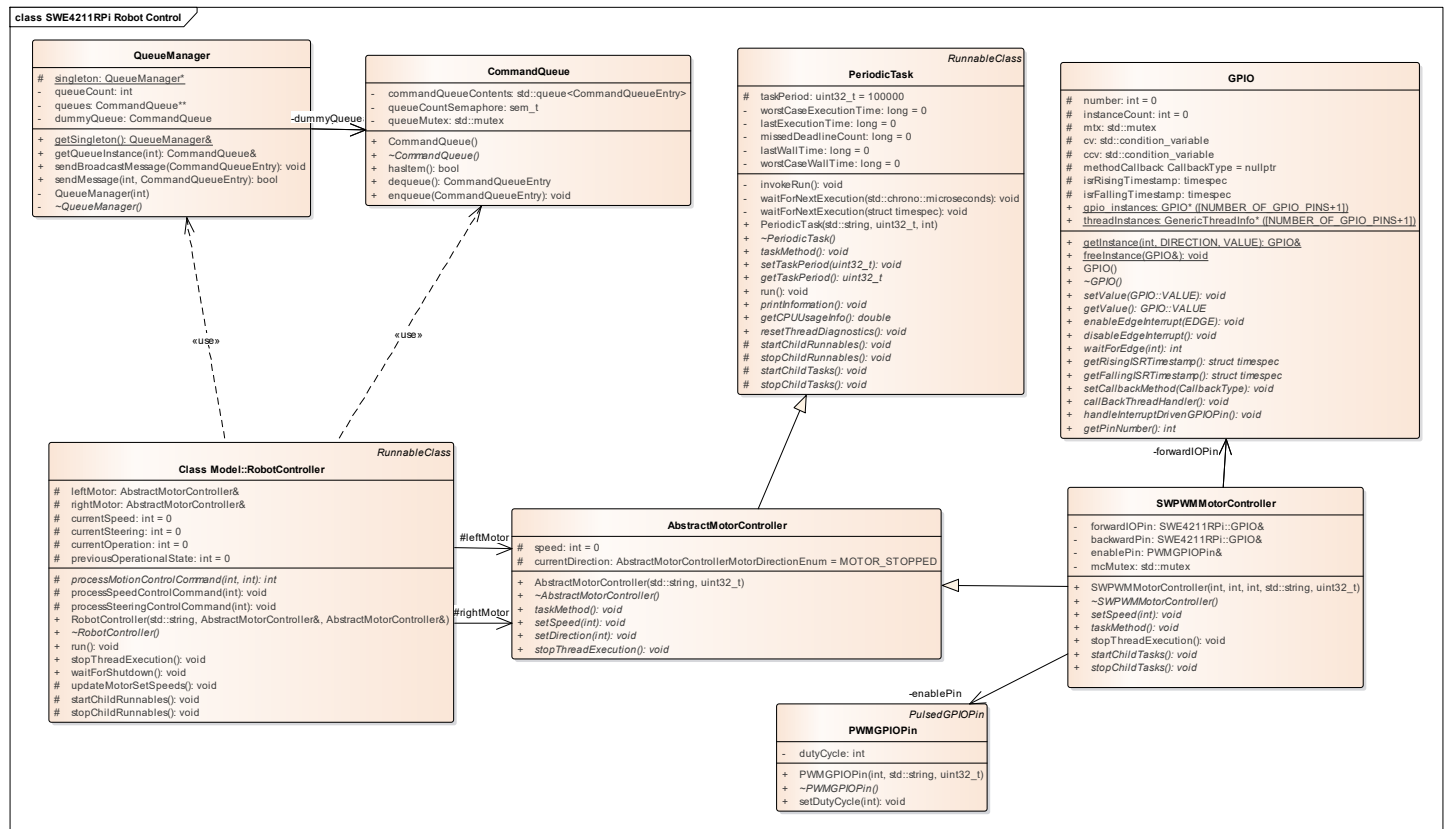


Figure 4 Robot Control subsystem.

7.2 Horn

The horn is a simple class. The horn is a periodic task with a period typically set to 100ms. Each time the thread runs, it will determine whether the piezo buzzer should be making sounds or not.

The horn can either be sounded or pulsed. If it is sounded, it will be on constantly. If it is being pulsed, then it will be on for a period followed by a period of silence. There are two different API calls, soundHorn and pulseHorn. The sound horn will simply turn the horn on, keeping the line high until it is set to be low. The pulse horn will take a period for the horn to repeat sounding and a length. The horn will sound for length and then will be quiet until the period completes at which point it will turn on again.

For best operation, the period and length should both be multiples of the period of the task, but this is not required. The times may be slightly longer based upon the task rate selected. For example, if the on time is set to 101ms and the horn



pulse period is 400ms with a 100ms task period, the horn will be on for 200ms, not 101, and then off for 200ms until the horn period repeats again.

The horn also has its own queue. It will read events from a CommandQueue which will tell when the horn is to sound and when it is to be quiet. The commands are defined in Table 4 Horn commands and status reports transmitted via the Command Queue processed by the Horn controller.. However, it will also have to process status messages that are broadcast to the queue and sound accordingly. UML for the horn is given in Figure 5 Horn Control.

Table 4 Horn commands and status reports transmitted via the Command Queue processed by the Horn controller.

Type	Command Name		Hex Value	Description
COMMAND_MSG_TYPE	HORN_MUTE_COMMAND		0x02000041	This command will mute the horn.
COMMAND_MSG_TYPE	HORN_SOUND_COMMAND		0x02000042	This command indicates that the horn is to sound continuously.
COMMAND_MSG_TYPE	HORN_PULSE_COMMAND		0x02000043	This indicates that the command received is a command to pulse the horn.
	Parameter 2: Period in ms. This is the period of the pulsed horn, given in ms. Parameter 1: This is the on time for the pulsed horn, given in ms.			
Type	Command Name		Hex Value	Description
STATUS_REPORT_MSG_TYPE	REPORT_DIRECTION_STATUS		0x08000021	This will be a report with the MOTORDIRECTIONBITMAP parameters added (LEFT, RIGHT, etc.)
	Parameter Name		Hex Value	Description
	MOTOR_FORWARD		0x00000001	Mute horn.
	MOTOR_BACKWARD		0x00000002	Pulse Horn
	MOTOR_STOP		0x00000010	Mute Horn

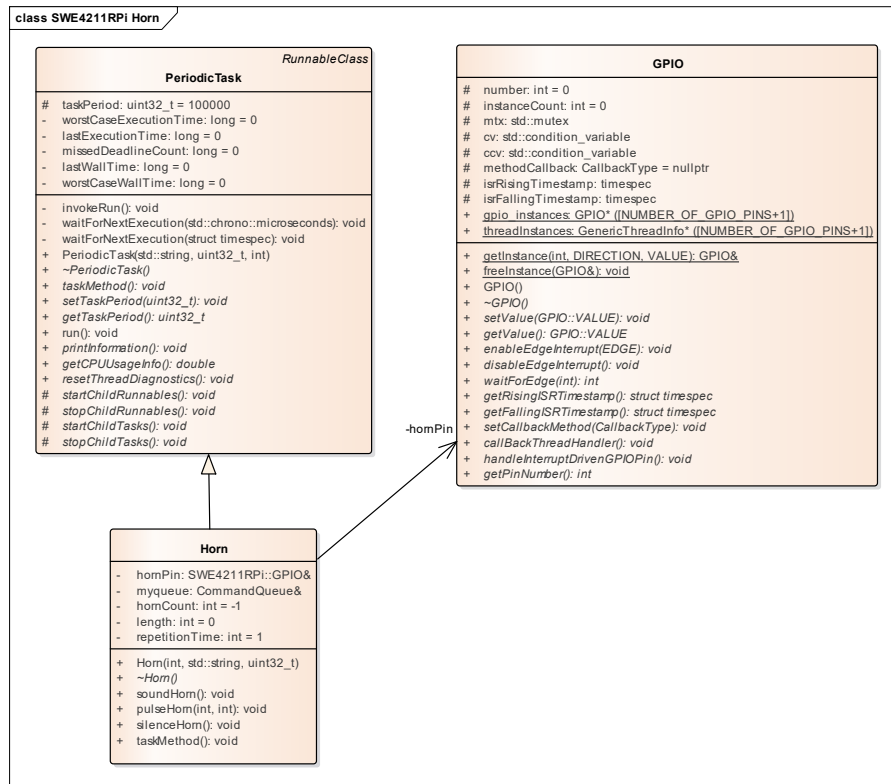


Figure 5 Horn Control



7.3 Lighting Control

Last week, you implemented lighting control on the robot. In essence, you could send a light control to the robot via the network, and the lights would change to that color. In this week's lab we'd like to make the lights automatically change color when the motion of the robot changes. To do this, the light control code will need to be changed to handle lighting commands. Essentially, the light controller will respond to status messages and change color accordingly. It also, however, will respond to specific colors that are sent to it, as it did last week.

Table 5 Color for robot motion

State	Color
Motor Stopped	RED
Forward	White
Left	Blue
Right	Green
Backward	Cyan
Forward Left	Magenta
Backward Left	Magenta
Forward Right	Yellow
Backward Right	Yellow

7.4 Putting it all together

All of the components communicate using messages that are enqueued on the various queues. Right now, the main message set is status reports from the robot controller. Changes in robot status may cause different components to change their behaviors.

Figure 6 shows a couple of events and how they are handled within the system. The sequence begins with a command for the robot to drive forward. This is sent via the network, which enqueues the message in the queue for the robot controller. From there, the event is dequeued and processed by the robot controller. In doing so, the robot controller will report its new status as a broadcast message which is enqueued to all the queues. The diagram then shows how two components will respond to this. In the case of the horn controller, it will receive this event and stop the horn from sounding. The searchlight controller will receive this event and will set the lights accordingly.

Later in the sequence, the robot controller receives a reverse command. In this case, you will see that the horn will start beeping as the robot moves in reverse, and the colored lights will change.

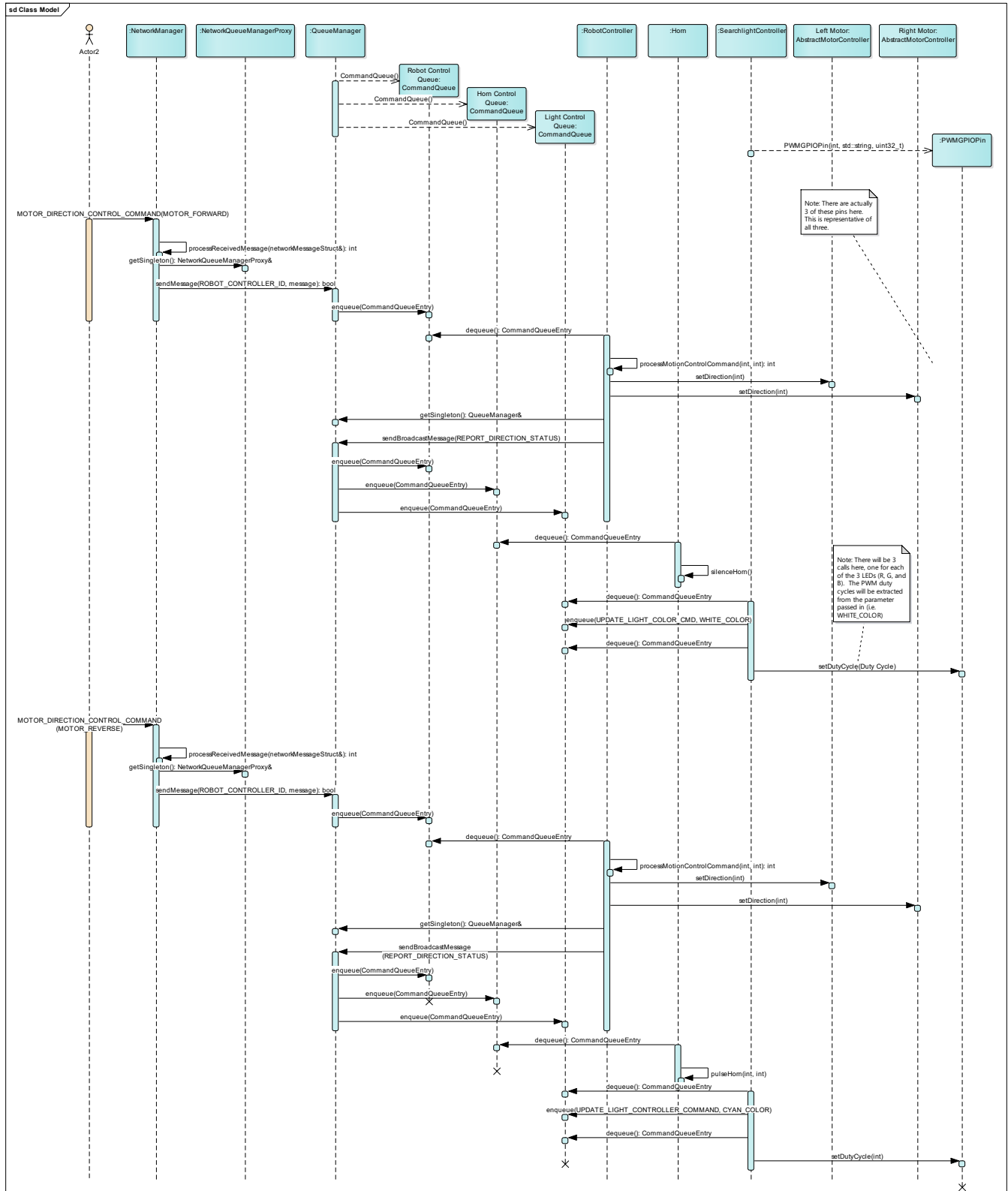


Figure 6 This sequence diagram shows the handling of two events. The first is a forward driving command sent from the Network to the robot. Subsequently, a reverse command is sent.

8 Controlling our robot

Before going into the instructions, one safety note. **Always be cognizant of your robot when starting code up.** Either make sure someone is holding the robot so that it does not run away or that it is sitting on the floor so that it will not drive off the table and fall to the floor. With real time systems, if there is a problem, the problem may manifest itself in very strange manners.

With the safety note completed, as with the light controller last week, you are being given a Java program that you can use to control the robot. The Java program has complete source code available that you can use to see exactly how the communications protocol works and how the messages are constructed.

The UI for this program is shown below in Figure 7. To operate the system, the user starts the program on the robot and then starts the java program running on the PC.¹ The user enters the IP address of the robot and presses connect. The network manager of the robot will accept the connection and allow the connected Java program to control the robot.

Pressing the $\wedge$ or $\vee$ will cause the robot to move forward or backward at a speed set by the robot speed slider. Pressing the $\leftarrow$ or $\rightarrow$ buttons will cause the robot to spin to the left or right. Pressing the $\swarrow$ or $\searrow$ buttons will cause the robot to move to the left or right in a forward direction and $\nwarrow$ or $\nearrow$ will cause the robot to move to the left or right in a reverse direction. For right now, the odometer will be inoperative. You can also use the Q, W, E, A, D, Z, S, and C buttons on your keyboard to control the robot. (Yes, these are not the most intuitive.) The vertical slider adjusts the robot speed. The horizontal slider adjusts the steering alignment so that if the robot “pulls” to the left or right when driving, you can straighten it out.

Once connected, you can also disconnect by pressing the disconnect button which will appear when a connection to the robot is made. This will allow any other PC to connect to the robot and take control.

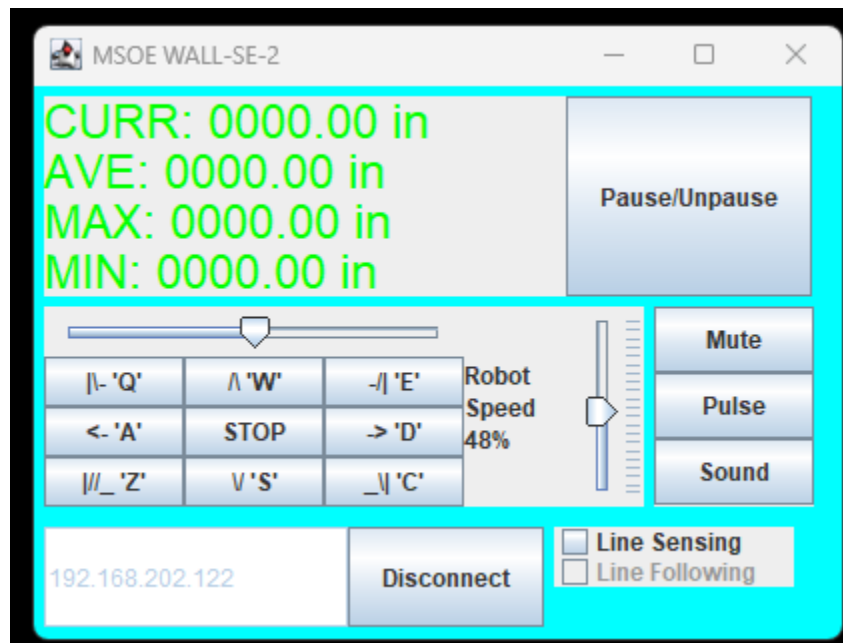
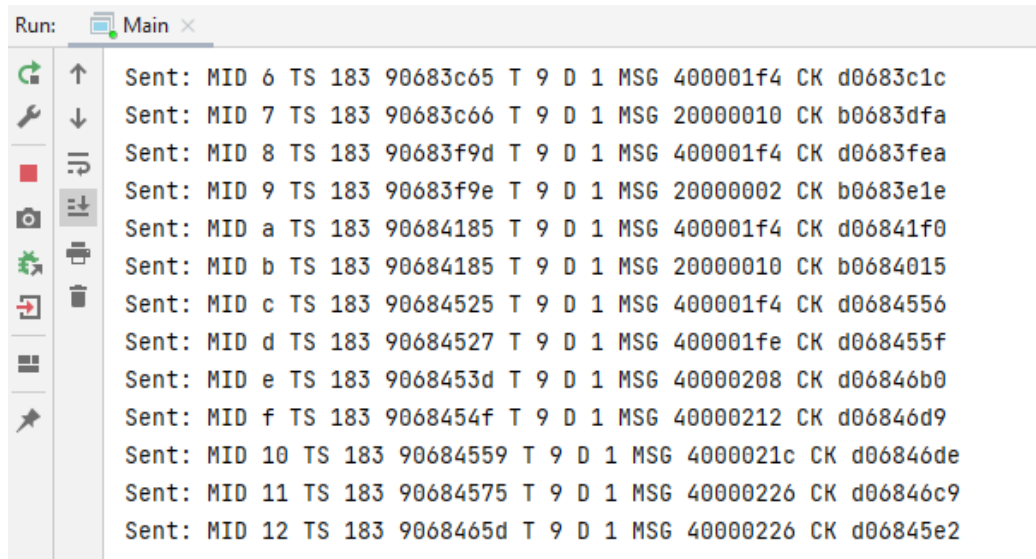


Figure 7 Control UI. Note that in this case, the robot is connected to IP address 192.168.202.122.

¹ Note that for best performance, you should start this program on your native windows machine using Java 8 or newer. It has been tested with IntelliJ.

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 8. You will need to interpret the hex a bit, but these are the specific values being sent in the structure.



```

Run: Main x
Sent: MID 6 TS 183 90683c65 T 9 D 1 MSG 400001f4 CK d0683c1c
Sent: MID 7 TS 183 90683c66 T 9 D 1 MSG 20000010 CK b0683dfa
Sent: MID 8 TS 183 90683f9d T 9 D 1 MSG 400001f4 CK d0683fea
Sent: MID 9 TS 183 90683f9e T 9 D 1 MSG 20000002 CK b0683e1e
Sent: MID a TS 183 90684185 T 9 D 1 MSG 400001f4 CK d06841f0
Sent: MID b TS 183 90684185 T 9 D 1 MSG 20000010 CK b0684015
Sent: MID c TS 183 90684525 T 9 D 1 MSG 400001f4 CK d0684556
Sent: MID d TS 183 90684527 T 9 D 1 MSG 400001fe CK d068455f
Sent: MID e TS 183 9068453d T 9 D 1 MSG 40000208 CK d06846b0
Sent: MID f TS 183 9068454f T 9 D 1 MSG 40000212 CK d06846d9
Sent: MID 10 TS 183 90684559 T 9 D 1 MSG 4000021c CK d06846de
Sent: MID 11 TS 183 90684575 T 9 D 1 MSG 40000226 CK d06846c9
Sent: MID 12 TS 183 9068465d T 9 D 1 MSG 40000226 CK d06845e2
  
```

Figure 8 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.

9 Testing our robot

Testing at this point will be arbitrary. Make sure the robot can go in all 8 of the supported directions (forward backward, left, right, etc.). Make sure that speed control works as well, with the robot running faster the faster the speed is set. Note that the robot may not move at some of the minimum PWM settings. As you do your tests, figure out what the minimum PWM value is that the robot will reliably start moving at.

In addition, look at your CPU usage. In a second window, open htop and filter for only the threads that are part of your robot. Grab a screen shot and look at the CPU usage. At the same time, in the terminal, press the P key on your keyboard and press enter. This will show you a slightly different view of CPU usage, built into our periodic task system, as is shown in Figure 9. Do the numbers somewhat match? Should they?

```

=====
Thread Diagnostic Information:
Thread Task Prio. Type period (us) Last Exec. (us) WCET (us) Last Wall Time (us) WCWT (us) CPU Usage CPU Usage
814 Blue LED PWM 40 PER 25000 17 62 26 22561 0.248%
813 Green LED PWM 40 PER 25000 16 64 26 22556 0.256%
812 Red LED PWM 40 PER 25000 19 65 22528 22606 0.260%
815 Search light Thread 1 RUN
809 Robot Controller 24 RUN
811 Horn Controller 10 PER 101000 4 43 7 86 0.043%
810 Robot On Screen Stat... 0 RUN
807 Motor PWM 54 PER 5000 22 82 1131 2571 1.640%
808 Right Motor Ctrl 50 PER 10000 5 50 8 56 0.500%
805 Motor PWM 54 PER 5000 20 89 1127 2564 1.780%
806 Left Motor Ctrl 50 PER 10000 5 48 8 52 0.480%
804 NetworkManager 29 RUN
803 main 0 RUN
Total CPU Usage: 5.207
=====
  
```

Figure 9 The output of the P key. This shows each thread, its description, priority, the type, the period (if periodic) , and performance information. More details about the columns will come later on.