SWE4211 Lab 12: Collision Sensing Streaming Robot

1 Introduction

In this lab, our goal is to integrate the distance sensor into our robot code to detect collisions and to integrate streaming so we can see what our robot is doing. We will also take a look at how well the robot is performing internally, because while it may function OK, it may not be optimal. We won't make any changes this lab, but we do want to see how it is doing.

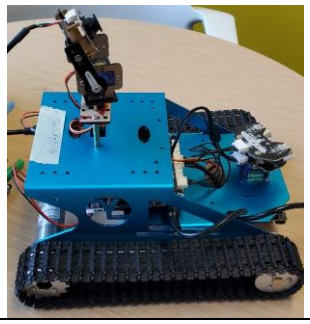
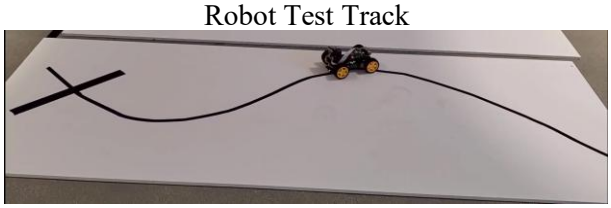
Previously, we implemented distance measurement software. This week, we want to build on it so that the robot will properly stop before running into an obstruction. To do this, we are going to integrate collision detection into our robot's code by adding a periodic task. This task will routinely poll the distance sensor and read the distance to any obstruction that is in front of the robot. If an obstruction is found, the task will enqueue a stop command to the robot controller, forcing the robot to stop.

When all is finished, your robot should be able to drive and follow lines, stop when reaching a cross line, drive forward and stop before running into an object at any speed, and stream video to the laptop in real time. The resolution and frame rate will be fixed based upon your results from previous labs.

2 Lab Objectives

- Integrate multiple queues into a control system.
- Use input sensors to manage a robotic system.
- Add threads to an existing system to extend basic functionality.
- Using experimentally captured data, design a solution to a real-world problem.
- Measure CPU utilization.

3 Laboratory Equipment Needed

Robot Charger 	Yahboom Robot for Raspberry Pi Smart Car AI Starter Kit 
Robot Test Track 	1 Micro SD Card with Raspberry Pi OS Installed 

4 Deliverables / Submission

Each team should submit a short video showing the following:

1. The robot streaming video while driving.
2. The robot stopping prior to touching a wall, using the distance sensor, at different speeds.
3. The robot stopping when it crosses a black line on a white surface.
4. The robot following a line while streaming video of the line following (ok, this may be a bit jerky to watch).

In your explanation, provide details as to the frame rate and resolution being streamed.

Each team will be responsible for submitting one report with the following contents:

1. Introduction -> What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Screen captures / Data Analysis
   a. Include a screen capture of your baseline diagnostics data, with all tasks set to default priority and default task rates. Reset the diagnostics after the robot starts to clear out any transient startup issues.
   b. Show an htop with the robot streaming.
   c. Include any graphs that you may have created (e.g. best fits) to design your minimum acceptable distance (MAD) calculation.
3. RMA Analysis
   a. Convert your P key output to a sorted spreadsheet.
   b. Using the formulas from class, calculate the max CPU utilization using the RMA criterion for the number of tasks in the system.
   c. Identify any tasks which have the wrong priority relative to their execution periods or other tasks. Fix any obvious mistakes.
4. Discussion
   a. How did you approach determining the MAD from the duty cycle? What data did you use and what was your analysis process? How did you account for latency and the need to detect the MAD twice before stopping?
   b. Once you deployed your code, how close was the performance of the code to your initial design? How much did you need to adjust your codebase / calculations in order to get the robot to stop reliably without hitting the test jig?
   c. How hard was it to integrate the camera into the code base?
   d. What was the resolution you captured from the camera? What was the resolution you transmitted? What was your frame rate? What was your quality? Why did you select these?
   e. If you needed to further improve the performance of this robot, where would you start?
5. Things gone right / Things gone wrong
   a. This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
6. Conclusions
   a. What have you learned with this experience? 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 Changes to codebase

There are a few changes you will need to make to your existing codebase that should help normalize the performance of your systems. In addition, as you go through the lab, you will need to make changes to the config files.

5.1 Camera Changes

The first required change is in the camera controller, and it involves the addition of a single line of code. Right now, OpenCV may use as many threads as there are cores on the processor when manipulating images. The Raspberry Pi 4 is a quad core processor, meaning that OpenCV may add as many as 4 threads to the system. This is not always beneficial, however, and in our case, the extra threads can cause performance issues. So, to gain more consistent performance, we are going to limit OpenCV to a single thread. This may seem non-optimal, but it results in less performance variance.

To do this, add the following line to the constructor for your camera:

cv::setNumThreads(0);

5.2 Changes to the Robot Controller

To make this project work properly, you will also need to make a change to the robot controller. Right now, the robot controller broadcasts direction changes, which are used to control the horn and the searchlights. To make collision sensing work, we will also need to broadcast whenever a PWM change occurs. A new report, REPORT_NEW_PWM_SETTING, has been added to the code base, and you will need to broadcast it from your robot controller any time a new PWM setting is received.

5.3 Changes to searchlight controller

Now that we have the camera, a race condition sometimes appears during the shutdown of the searchlight controller, which can cause the program to hang on shutdown. To fix it, you will need to add some code to your searchlight controller to respond to the shutdown event that is broadcast. Here is the code snippet:

```
/**
 * <li>Otherwise, if the command from the queue is a broadcast message type and it is a shutdown
command</li>
 * <ol>
 */
} else if (commandToProcess.commandType == BROADCAST_MESSAGE_TYPE) {
        if (commandToProcess.command == BROADCAST_SHUTDOWN_COMMAND) {
        /**
         * <li>Stop the threads from executing.</li>
         * </ol>
         */
        this->stopThreadExecution();
}
```

6 Designing our software

6.1 Collision Detection

Collision detection will be built upon the DistanceSensor code we developed previously. A new CollisionSensor task will be executed periodically, reading the current distance from the DistanceSensor. If the measured distance is below the minimum acceptable distance (MAD) for two consecutive readings, the task will stop the robot. To do this, it will enqueue a stop event to the robot controller, enqueue a horn activation event on the horn queue, and enqueue an event to disable line following.

There is one catch, however. Issuing a stop command silences the horn, so to keep the horn sounding while the robot remains stopped, the task will re-enqueue the horn event if a third consecutive reading is below the threshold.

The class also includes a mutator method to set the minimum acceptable distance, though right now the user interface does not provide a way to adjust this value. The UML diagram for this design is shown in Figure 1.

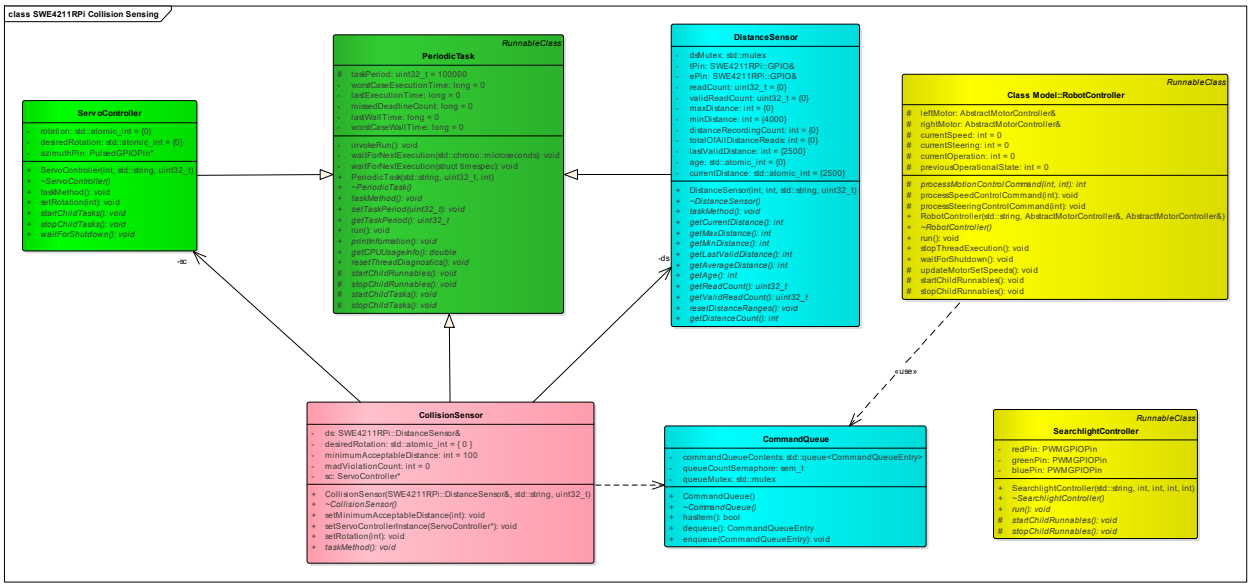


Figure 1 UML for the collision sensing aspect of the robot.

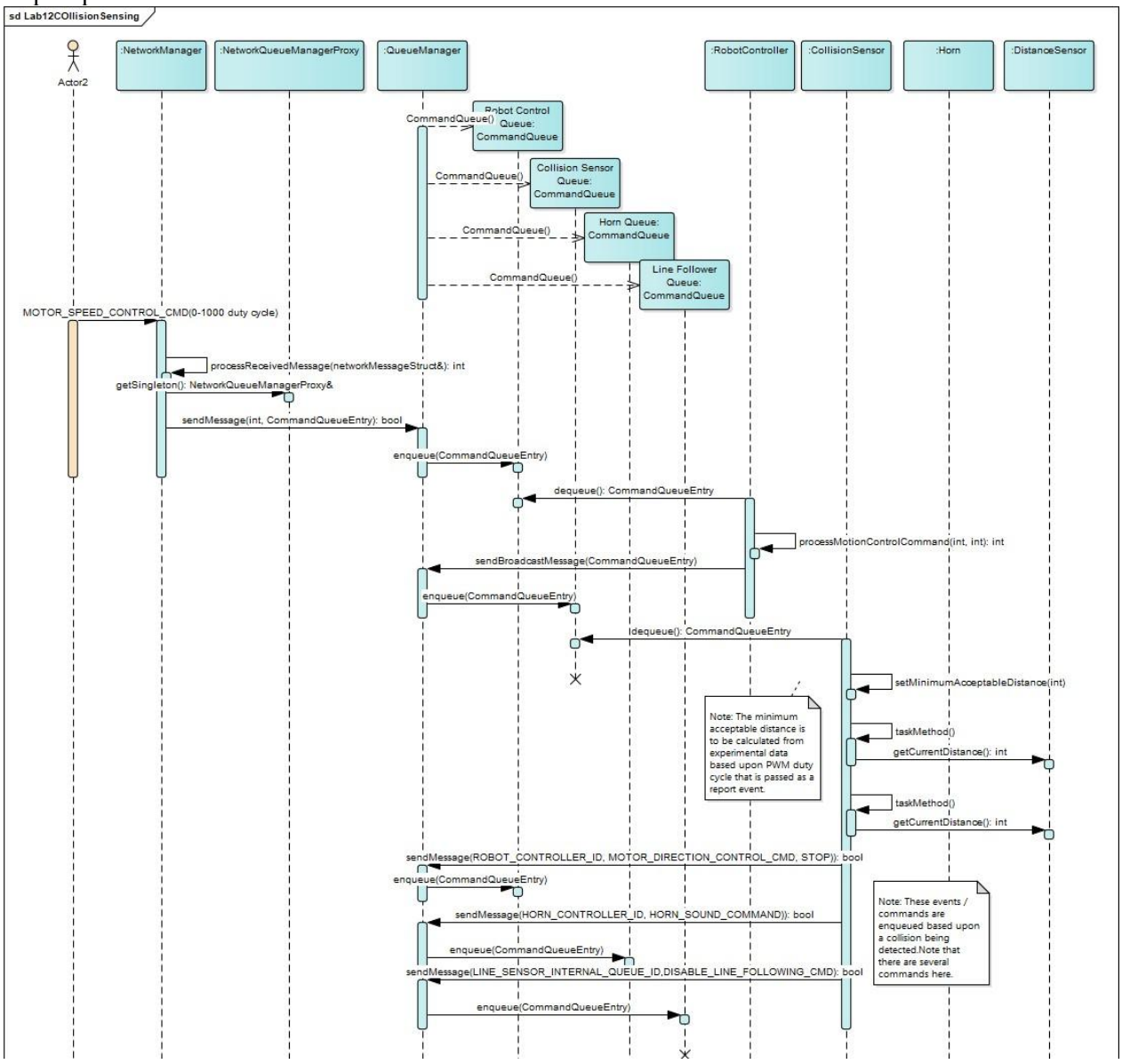
6.1.1 Minimum Acceptable Distance Calculation

You will need to design the calculation for the minimum acceptable distance yourself. Start with the experimental stopping distance data from previous labs, which will tell you how much clearance the robot needs from an obstruction. From there, your calculations will need to account for the duty cycle, the speed of the robot, and the latency of the system, including the fact that multiple sensor readings are required before a stop decision is made. The result of these calculations will then be used to set the MAD.

Work with your lab partner to define your approach for these calculations. The goal is to ensure that the robot stops safely based on real-world performance data, and the approach you take will greatly impact how smoothly the debugging of this feature goes.

Keep in mind that all of this depends on the broadcast status message described earlier, which the robot controller uses to notify all components of a new PWM duty cycle.

You can see how the system will behave over time by looking at the sequence diagram in Figure 2. The sequence begins with a new robot speed being set. The speed is broadcast out to everyone, including the collision sensor, which kicks off a calculation of the new minimum acceptable distance based upon the robot's characteristics. If the robot ever has two reads in a row that are under the MAD, it will initiate a stop sequence for the robot.



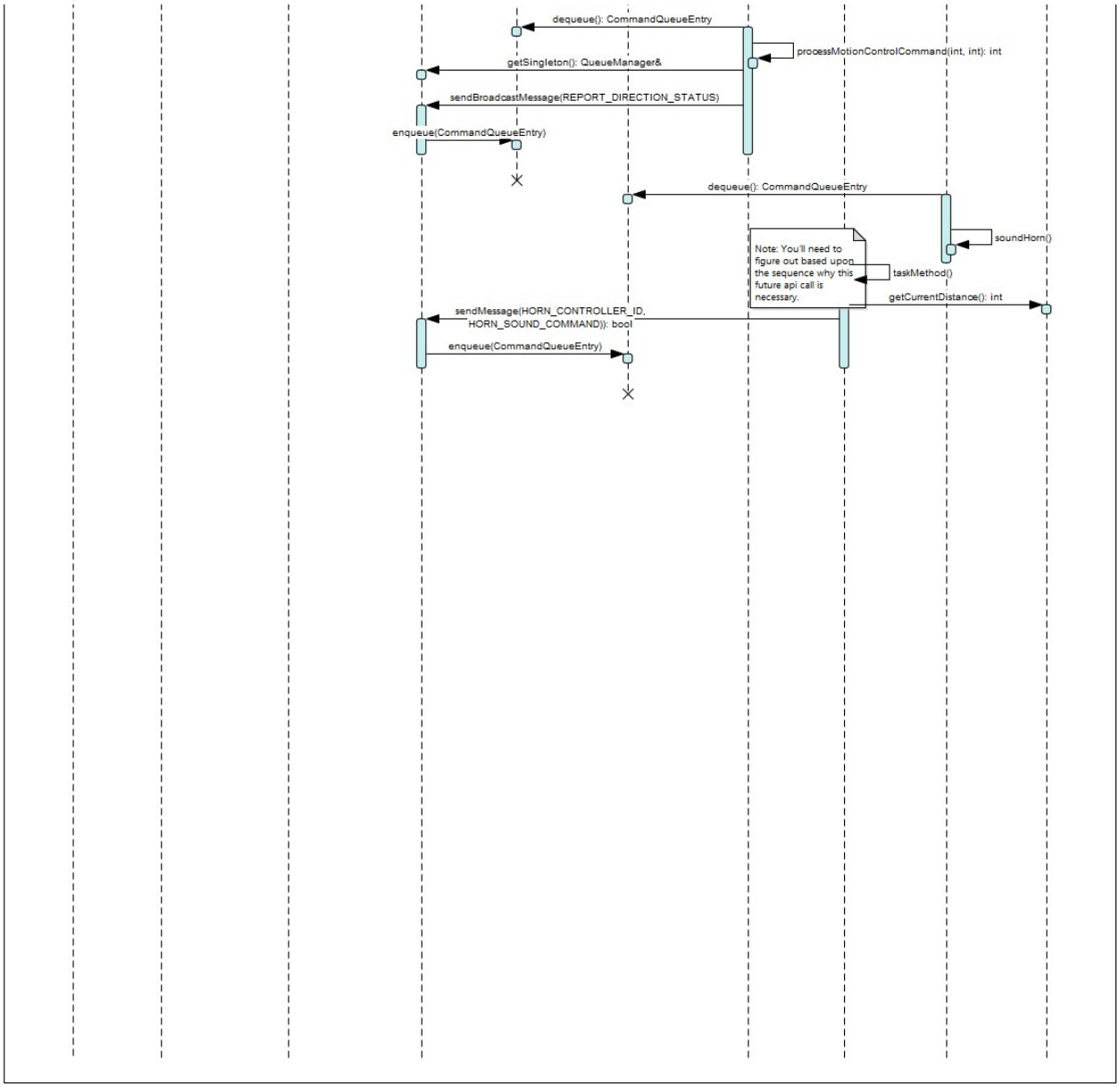


Figure 2 The sequence which will control collision sensing.

6.2 Camera Integration

The next step is to integrate the camera into the robot. This should be relatively straightforward, as you simply need to link in the necessary files, build them, and enable the camera functionality within the robot system.

To get smooth and reliable video streaming, you will need to determine appropriate frame rate and resolution settings for both the camera and the stream. These parameters significantly affect performance, and their impact may not be linear. For example, increasing the resolution from 640 × 480 to 1024 × 768 may have a greater performance impact than switching to 1280 × 480, even though the latter has more pixels. The JPEG compression quality also influences both performance and the reliability of the stream.

Based on your findings from last week's work, set the resolution in the camera configuration file and adjust the frame rates to match what your robot can handle consistently.

Remember that **robot reliability takes priority over stream quality**. If necessary, sacrifice resolution or frame rate to maintain stable control and responsiveness.

7 Measuring Performance

There are several ways to evaluate how well the system is performing in this lab. Ultimately, the robot should follow lines as before while streaming video, stop before hitting an obstruction at any PWM duty cycle, and stream images reliably, without significant breakup and without negatively impacting control performance.

7.1 Monitoring System Behavior

Once integration is complete, run htop while driving the robot. Observe the CPU usage across the threads and identify which tasks consume the most resources. You will also want to capture the P key output, since with more tasks now active, it will help you analyze their behavior.

7.2 Thread Prioritization Analysis

Until now, we have not examined thread prioritization. While you do not need to fix any issues this week, it is important to verify that the priorities are set correctly. To do this, export the P key output as a tab-delimited data table and import the data into a spreadsheet for analysis.
To capture the output during runtime, save your terminal output to a file, as shown in Figure 3.

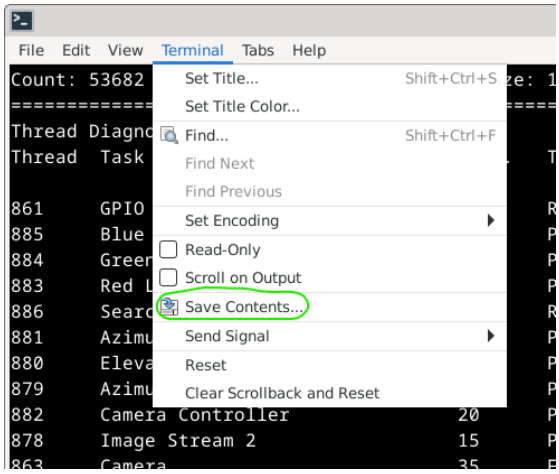


Figure 3 Saving console output to a file.

7.3 Analyzing Task Prioritization

After importing the data into your spreadsheet, begin by sorting the tasks by period, from the shortest to the longest. The only free-running task with a defined period is the camera task. If the camera is configured for 30 frames per second, its period should be approximately 1/30 of a second, or 33,333 microseconds.

Next, check the priorities. Tasks with the same period should share the same priority, tasks with shorter periods (i.e. those which execute more frequently) should have higher priorities, and tasks with longer periods should have lower priorities. Remember that task priorities range from 1 to 54, with 54 being the highest.
For now, simply document any inconsistencies you find in the prioritization. We will address and correct these in a later step. An example of this analysis is shown in Figure 4.

Thread Diagnostic Information:									
Thread	Task	Prio.	Type	period	Last Exec.	WCET	Last Wall	WCWT	CPU Usage
				(us)	(us)	(us)	Time (us)	(us)	CPU Usage
872	Line Sensor Task		PER	5000					
867	Motor PWM		PER	5000					
865	Motor PWM		PER						
868	Right Motor Ctrl		PER						
866	Left Motor Ctrl		PER						
873	Line Following Contr...		PER	15000					
881	Azimuth Thread 2		PER						
880	Elevation Thread		PER						
879	Azimuth Thread		PER						
876	Azimuth Thread		PER	20000					
885	Blue LED PWM		PER	25000					
884	Green LED PWM		PER						
883	Red LED PWM		PER						
863	Camera		FRT	33333					
875	Collision Sensor Thread		PER	50000					
882	Camera Controller		PER	65000					
877	DS Servo Controller		PER						
874	Distance Sensor		PER						
878	Image Stream 2		PER						
871	Horn Controller		PER	101000					
861	GPIO 0		RUN						
886	Search light Thread		RUN						
869	Robot Controller		RUN						
870	Robot On Screen Stat...		RUN						
864	NetworkManager		RUN						
860	main		RUN						
Total CPU Usage: 49.167									

Figure 4 Sample excel sheet showing CPU execution data (i.e. P Key data).

8 Configuration Parameter Setting

In the file labconfig.h, there are multiple configuration parameters that may be of assistance as you complete this lab.

These switches decide which capabilities the robot has when it is built and run. You can enable one feature, test it, and then add the next, so an unfinished feature can't interfere with the ones that already work. At startup, the program prints a line to the console for most enabled features (for example, "Distance Sensing Integrated." or "Collision Sensing Enabled."), so you can confirm what the running robot includes.

8.1 DISTANCE_SENSOR_FEATURE_ENABLED

Controls the robot's ultrasonic distance sensor. When enabled, the robot measures the distance to whatever is in front of it several times per second. When disabled, the robot has no sense of distance and drives with no awareness of obstacles. If collision sensing is disabled but the distance sensor is enabled, the robot prints the current distance reading to the console about twice a second, which is useful for checking the

sensor before building on top of it. Collision sensing and status reporting both use the distance readings, so treat this feature as required whenever either of them is enabled.

8.2 COLLISION_SENSING_FEATURE_ENABLED

Turns on the robot's automatic safety stop. When enabled, the robot keeps checking the distance sensor, and if an object stays closer than a minimum distance (100 mm by default) for more than one reading, it stops forward motion on its own, even if the operator is still commanding it to drive forward. This feature also controls the servo that rotates the distance sensor, so the sensor can be aimed in different directions. When disabled, the robot doesn't stop by itself, the operator is fully responsible for avoiding collisions, and the console printout of raw distance readings appears instead.

8.3 ROBOT_STATUS_REPORTING_FEATURE_ENABLED

Controls whether the robot sends information back to the base station (the Java program on the PC). When enabled, the robot regularly reports its current distance reading, converted to hundredths of an inch, over the network. If collision sensing is also enabled, it reports collision sensor information too. The operator can then watch what the robot "sees" from the PC. When disabled, the robot still accepts commands but sends nothing back, so the operator only knows what the robot is doing by watching it.

8.4 CAMERA_STREAM_FEATURE_ENABLED

Controls the live video feed. When enabled, the robot captures images from its camera, compresses them as JPEGs, and streams them to the base station. The operator can also aim the camera remotely with its pan (azimuth) and tilt (elevation) servos. When disabled, there's no video and no camera aiming, and the robot uses less processor time and network bandwidth. The program must then be started with no command-line arguments.

8.5 CAMERA_VAR_SETTINGS_FEATURE_ENABLED

Only matters when the camera stream is enabled, and decides where the camera's settings come from. When disabled, the robot uses the defaults in Cameracfg.h, and you start the program with no arguments. When enabled, you must supply seven command-line arguments every time you start the program: camera width, camera height, transmit width, transmit height, frames per second, camera task priority, and JPEG quality (0-100). If you don't, the program prints a usage message and exits. This mode is handy for experimenting with the trade-off between image quality, frame rate, and system load without rebuilding, but it makes everyday startup less convenient.