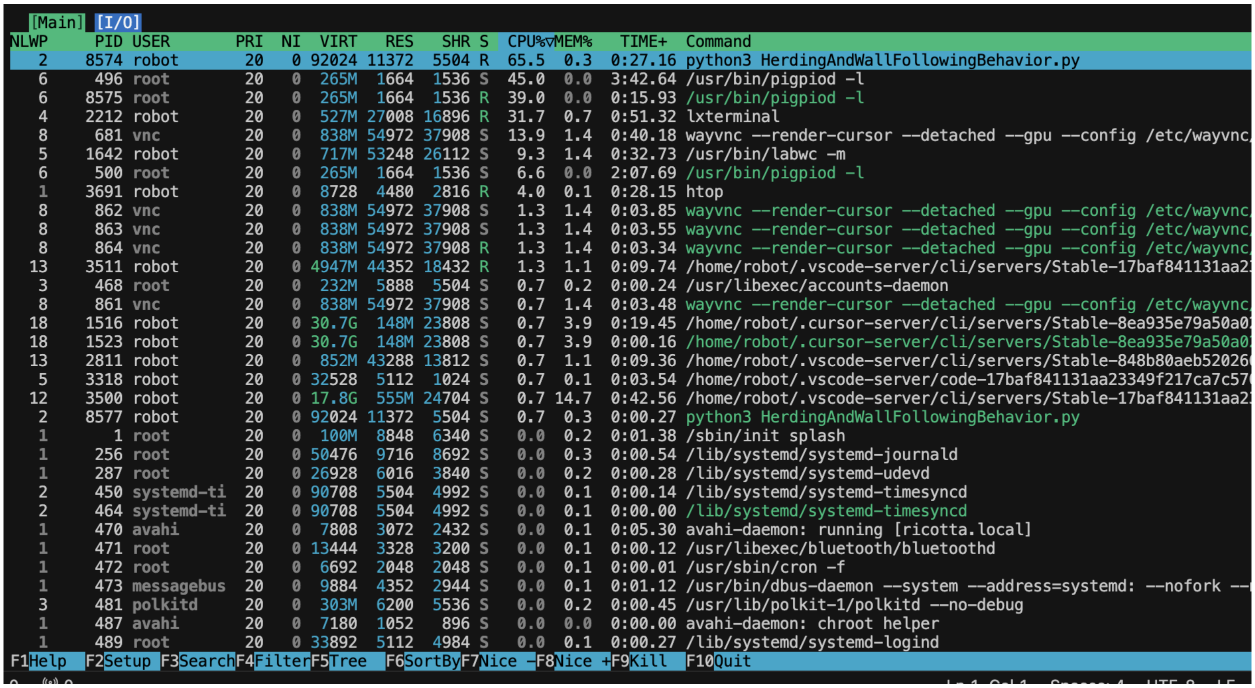
(i) the results of your htop testing in Section 2.

Testing using Wall-Following (pre threading):

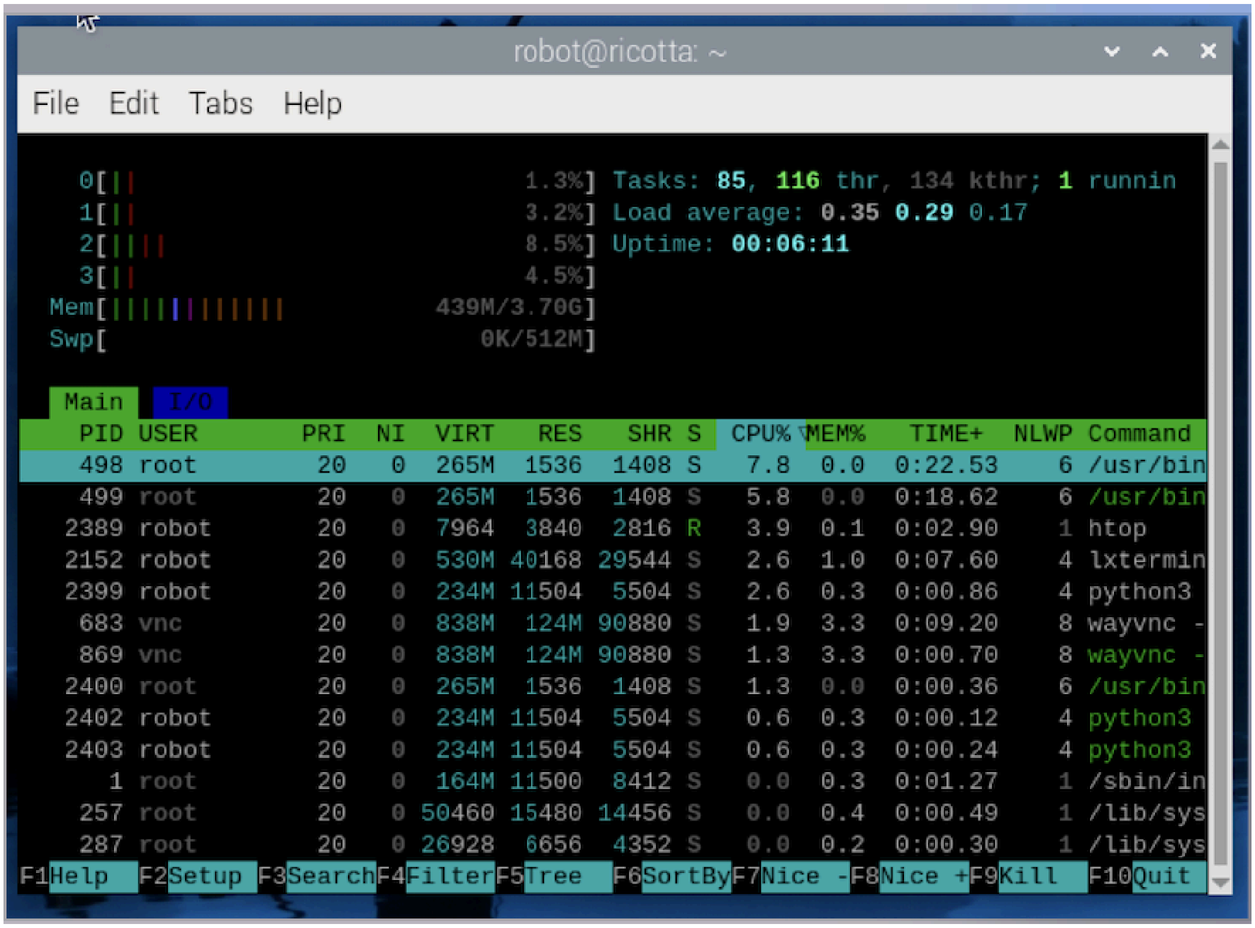


```
 [Main] [I/O]
NLWP   PID USER       PRI  NI  VIRT   RES   SHR S  CPU%▽MEM%   TIME+  Command
   2  8574 robot       20   0 92024 11372  5504 R  65.5  0.3  0:27.16 python3 HerdingAndWallFollowingBehavior.py
   6   496 root        20   0  265M  1664  1536 S  45.0  0.0  3:42.64 /usr/bin/pigpiod -l
   6  8575 root        20   0  265M  1664  1536 R  39.0  0.0  0:15.93 /usr/bin/pigpiod -l
   4  2212 robot       20   0  527M 27008 16896 R  31.7  0.7  0:51.32 lxterminal
   8   681 vnc         20   0  838M 54972 37908 S  13.9  1.4  0:40.18 wayvnc --render-cursor --detached --gpu --config /etc/wayvnc/
   5  1642 robot       20   0  717M 53248 26112 S   9.3  1.4  0:32.73 /usr/bin/labwc -m
   6   500 root        20   0  265M  1664  1536 S   6.6  0.0  2:07.69 /usr/bin/pigpiod -l
   1  3691 robot       20   0  8728  4480  2816 R   4.0  0.1  0:28.15 htop
   8   862 vnc         20   0  838M 54972 37908 S   1.3  1.4  0:03.85 wayvnc --render-cursor --detached --gpu --config /etc/wayvnc/
   8   863 vnc         20   0  838M 54972 37908 S   1.3  1.4  0:03.55 wayvnc --render-cursor --detached --gpu --config /etc/wayvnc/
   8   864 vnc         20   0  838M 54972 37908 R   1.3  1.4  0:03.34 wayvnc --render-cursor --detached --gpu --config /etc/wayvnc/
  13  3511 robot       20   0 4947M 44352 18432 R   1.3  1.1  0:09.74 /home/robot/.vscode-server/cli/servers/Stable-17baf841131aa2
   3   468 root        20   0  232M  5888  5504 S   0.7  0.2  0:00.24 /usr/libexec/accounts-daemon
   8   861 vnc         20   0  838M 54972 37908 S   0.7  1.4  0:03.48 wayvnc --render-cursor --detached --gpu --config /etc/wayvnc/
  18  1516 robot       20   0 30.7G  148M 23808 S   0.7  3.9  0:19.45 /home/robot/.cursor-server/cli/servers/Stable-8ea935e79a50a0
  18  1523 robot       20   0 30.7G  148M 23808 S   0.7  3.9  0:00.16 /home/robot/.cursor-server/cli/servers/Stable-8ea935e79a50a0
  13  2811 robot       20   0  852M 43288 13812 S   0.7  1.1  0:09.36 /home/robot/.vscode-server/cli/servers/Stable-848b80aeb52026
   5  3318 robot       20   0 32528  5112  1024 S   0.7  0.1  0:03.54 /home/robot/.vscode-server/code-17baf841131aa23349f217ca7c57
  12  3500 robot       20   0 17.8G  555M 24704 S   0.7 14.7  0:42.56 /home/robot/.vscode-server/cli/servers/Stable-17baf841131aa2
   2  8577 robot       20   0 92024 11372  5504 S   0.7  0.3  0:00.27 python3 HerdingAndWallFollowingBehavior.py
   1     1 root        20   0  100M  8848  6340 S   0.0  0.2  0:01.38 /sbin/init splash
   1   256 root        20   0 50476  9716  8692 S   0.0  0.3  0:00.54 /lib/systemd/systemd-journald
   1   287 root        20   0 26928  6016  3840 S   0.0  0.2  0:00.28 /lib/systemd/systemd-udevd
   2   450 systemd-ti  20   0 90708  5504  4992 S   0.0  0.1  0:00.14 /lib/systemd/systemd-timesyncd
   2   464 systemd-ti  20   0 90708  5504  4992 S   0.0  0.1  0:00.00 /lib/systemd/systemd-timesyncd
   1   470 avahi       20   0  7808  3072  2432 S   0.0  0.1  0:05.30 avahi-daemon: running [ricotta.local]
   1   471 root        20   0 13444  3328  3200 S   0.0  0.1  0:00.12 /usr/libexec/bluetooth/bluetoothd
   1   472 root        20   0  6692  2048  2048 S   0.0  0.1  0:00.01 /usr/sbin/cron -f
   1   473 messagebus  20   0  9884  4352  2944 S   0.0  0.1  0:01.12 /usr/bin/dbus-daemon --system --address=systemd: --nofork --
   3   481 polkitd     20   0  303M  6200  5536 S   0.0  0.2  0:00.45 /usr/lib/polkit-1/polkitd --no-debug
   1   487 avahi       20   0  7180  1052   896 S   0.0  0.0  0:00.00 avahi-daemon: chroot helper
   1   489 root        20   0 33892  5112  4984 S   0.0  0.1  0:00.27 /lib/systemd/systemd-logind
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

There are 2 threads running

PID	Command	CPU %	Threads
8574	HerdingAndWallFollowingBehavior.py	65.5	2
496	pigpiod	15.0	4

Post Wall-following Thread:



(ii) your prediction and final results for Section 3.

Thread Name	Purpose	Predicted	Activity
MainThread	Robot control logic (wall-following or brain loop)	30-60%	Executes behavior loop (wall-following, driving decisions, map updates, turn, line-following,
TriggerThread	Ultra sound sensor triggering every 50ms	Low < 15%	Runs the trigger method periodically used for the ultrasound
UIThread	Takes user input, and updates mode flag	Low < 15 %	Waits on input()

Pigpio callbacks	Handles ultrasonic rise/fall and GPIO	Low < 15%	Background thread managed by pigpio daemon. Only active during echo return.

Shared Data	**Thread 1**	**Thread 2**	**How do you ensure thread-safety?**
Self.distance	Pigpio callback threads on falling and rising of the ultrasound	Main behavior thread reads the sensor values	Each distance value is a simple atomic float?
Last_trigger_time	Trigger threads updates 50ms	Main behavior thread	Safe because its a single boolean variable and therefore atomic. We can avoid a threadlock
Mode for UI control	Read in wall following loop	Set by UI (input()	We can use threadlocking so safely share

(iii) a specification of your shared data in Section 5 and how the commands set the appropriate flags/modes to accomplish the commands.

Modes:

"manual" — user can input left, right, straight, goto, etc.

"goto" — follow the Dijkstra path to the goal.

"herding" — start herding behavior based on proximity sensors.

“Wall_following” – start wall following behavior

“Stop” – freeze the robot into place

“Resume” – resume from the frozen spots

“calibrate “ - calibrates the magnetometers (we could either make this a mode or do this automatically

(iv) a specification of you intersection class and brain logic, allowing for obstacles in Section 6.
As always, please submit the report and code (via GitHub) in Gradescope, linking teammates.

1. Integrating the Ultrasound Sensors

First, we’ll incorporate the ProximitySensor class into my code so that we can detect obstacles in front of the robot. Initialize this sensor inside the main function of brain.py, just like we did for the line and angle sensors. Then we’ll pass this sensor into the Behaviors class so it can be used inside high-level movement logic. If we don’t already have a clean method for getting just the front-facing distance reading, we’ll add one, probably called something like read_front().

2. Extending the Intersection Data Structure

Next, we’ll modify the Intersection class in MapBuilding.py to track blocked streets. Right now it only stores the status of each of the 8 surrounding streets (like CONNECTED or UNKNOWN), but that doesn't account for whether a street is physically blocked. So we’ll add a new list of 8 booleans, one per heading, where True means there’s an obstacle preventing travel in that direction. This list will live alongside the streets list and give me a second layer of information for planning.

3. Creating a Blockage-Checking Behavior

Inside Behaviors.py, we'll write a new method to check if the street directly in front of the robot is blocked. It will just read the front ultrasound sensor and compare the distance to a threshold, probably around 50 to 70 cm. If the sensor detects something too close, we'll return True to signal that the path ahead is blocked. This will let the brain react to obstacles right after a movement.

4. Calling the Blockage Check After Every Movement

Whenever the robot finishes pulling forward into a new intersection, we'll call this check_blockage() method. If the method returns True, we'll record that in the current intersection's blocked list at the current heading. That way, every time the robot completes an action, it updates the map with whether the next direction is physically drivable or not. We'll do this in every if result == "intersection" block — both in manual exploration and in automatic mode.

5. Avoiding Blocked Streets During Exploration

To make sure the robot doesn't try to drive into a known blocked street, we'll update how I generate the list of directions to explore. Right now I look for directions that are UNKNOWN or UNEXPLORED, but now we'll also filter out any directions that are marked as blocked. That means in both the unknown and unexplored phases of exploration, the robot will skip over blocked directions and only consider safe, open paths.