Report: reproducible, design decisions

EC535 Spring 2023: Lab #5/Project

Assigned: Thursday, Apr. 6, 2023
Last Revised: Thursday, Apr. 6, 2023
Demo Days: Apr. 25 & 27 (during lecture time)
Report Due: Tuesday, May 2, 2023 @ 10:00 pm EDT

Pre-lab Preparation

Before starting this lab, you must understand the following concepts:

- Qt: en.wikipedia.org/wiki/Qt_(software)
- Framebuffer: https://en.wikipedia.org/wiki/Framebuffer
- All concepts from Labs 1-4 (e.g., how would you create an SD card image?)

Introduction

In this lab, you'll combine the skills and knowledge you've developed throughout the course to create a **userspace game**, a **kernelspace screensaver**, a **data display app**, or **another embedded system (your choice)**.

To help you make an informed decision, we'll provide a brief tutorial on Qt, a userspace graphics framework that's very widely-used in industry, which you'll use if you choose the one of the userspace options. We'll also provide some information on how the kernel handles human interface devices (HIDs) and framebuffers, which you'll need to know if you choose the kernelspace screensaver option. Finally, we're happy to answer specific questions about sensors/APIs you might be interested in using in a data display app.

You may **work in teams** of two or three, and **you must present** your completed lab on your demo day. See *Requirements* for more details.

Additional Helpful Resources

Kernelspace graphics

- “Linux Graphics Drivers: an Introduction”: https://people.freedesktop.org/~marcheu/linuxgraphicsdrivers.pdf
- “eCos (a Linux-based RTOS) Kernel Manual: Framebuffer Support”: https://doc.ecoscentric.com/ref/io-framebuf.html
- A series of patches that Sony submitted to draw modified penguin logos onho d you et a ceerho d you et a ca ae: https://lkml.org/lkml/2007/7/10/188
  - Click the “Geert Uytterhoeven” links in the sidebar to see all 4 patches
- Kernel source code for `fb_show_logo_line()`: https://elixir.bootlin.com/linux/v4.19.82/source/drivers/video/fbdev/core/fbmem.c#L464

Kernel HID subsystems

- “Linux Kernel Docs: Input Subsystems”: https://www.kernel.org/doc/html/latest/input/input_kapi.html
- Kernel source code for the `input_dev` struct: https://elixir.bootlin.com/linux/v4.19.82/source/include/linux/input.h#L121
- Kernel source code for the `evdev` driver: https://elixir.bootlin.com/linux/v4.19.82/source/drivers/input/evdev.c
- Example kernel module for manipulating the mouse: https://github.com/kushsharma/mouse-driver

Qt-based userspace games/apps

- “Getting Started with Qt”: https://doc.qt.io/qt-5/gettingstarted.html#create-your-first-applications
- “Qt 5 Examples & Tutorials”: https://doc.qt.io/qt-5/qtexamplesandtutorials.html
- “Qt Based Games”: https://wiki.qt.io/Qt_Based_Games
- “Qt5 Tutorial”: https://zetcode.com/gui/qt5/

Specifications

Your final submission must implement **one** of the four options below. Project proposals must be pre-approved by the teaching staff **by Friday, April 14, 2023**.

Option 1: Userspace game using Qt

Using the Qt graphics library and the Linux framebuffer, develop a non-trivial graphical game such as Snake, Brick-Breaker, Frogger, Minesweeper, or an original idea. Your game should be a userspace application (i.e., no kernel module required) **written in C++**. Your game should be playable using the LCD touchscreen and/or some other input method (e.g., USB keyboard, serial console, GPIO buttons, accelerometer).

Your final submission will be graded based on its functionality, creativity, and robustness. While you need not write the next *Call of Duty* or *Fortnite*, your game's graphics should be visually-attractive and polished. Your game only needs to support a single player, but you may add 2+ player support if you wish.

If you have any concerns about whether or not your game idea meets these guidelines, please contact the teaching staff.

Important: the resources and toolchain contained within the `$EC535/` folder are set up for **Qt 5.15.2**. When searching for documentation online, be sure not to use resources written for Qt 4 or Qt 6!

Requirements

Your submission must be a playable, graphical game written in C++ and using the Qt framework. Your game must compile on the lab machines and run on a BeagleBone using the provided stock image and kernel. You must provide adequate documentation for your game in your `README` and/or through in-game messages. You will lose points if your game crashes, freezes, or is otherwise unstable.

Part of your grade will be based on your game's visual appeal and graphical complexity. You should first target having a well-polished game. If you have time, you may consider ambitious features like smooth animations, shadows, and 3D rendering.

The remaining portion of your grade will be based on your game's general robustness and playability. Again, you should focus first on creating a stable game with intuitive controls. If you have time, you may consider adding advanced features like high score recording, multiplayer support, or progress saving.

Option 2: Screensaver kernel module

Develop a kernel module that automatically launches a Windows-XP-style screensaver on the framebuffer after 15 seconds of user inactivity (i.e., no touchscreen input). Your screensaver should feature moving graphics/animations, it should automatically stop upon receiving touchscreen input. Please name your module `myscreensaver` .

Your final submission will be graded based on its functionality, creativity, and visual attractiveness. While there's no need for your screensaver to show procedurally-generated fractal zooms in 8K, it should go beyond a simple moving piece of text.

If you have any concerns about whether or not your screensaver idea meets these guidelines, please contact the teaching staff.

Requirements

Your kernel module must display a graphical screensaver on the framebuffer after 15 seconds of no touchscreen input. Your module must automatically stop writing to the framebuffer upon receiving any touchscreen input. As in previous labs, your module must not crash at any time, including during module unloading.

Part of your grade will be based on your screensaver's visual appeal and graphical complexity. You should first target a well-polished, stable screensaver. If you have time, you may consider more ambitious features like smooth animations, images, and dynamic text (e.g., current time, system load, etc.).

Option 3: Data display app

Using the Qt graphics library and the Linux framebuffer, develop an application that displays data from either a sensor[1] or a web API on the BeagleBone's touchscreen (and/or an alternative display device[2]) in a visually-attractive fashion. If developing a web API-based app, think along the lines of live visualizations of useful, dynamic data: e.g., weather dashboards or beacons, street traffic maps, train trackers, and stock exchange monitors. If developing a sensor-based app, think about fun or original ways of displaying live and/or historical readings: e.g., a spirit/bubble level for accelerometer data, a tide clock for temperature/humidity/barometric data, or an ECG for heartbeat data.

Ideally, your application has some interactive components (e.g., buttons to display a 5-day weather forecast, or pinch-and-zoom controls on a traffic map), although these may not be necessary if your project requires significant effort for getting the data (e.g., you had to write a kernel module driver for your sensor). Your app should also dynamicaially update the data displayed while running, if possible.

Your final submission will be graded based on its functionality, creativity, and robustness. While you need not recreate a Bloomberg terminal, your app should be visually-attractive and polished.

If you have any concerns about whether or not your app idea meets these guidelines, please contact the teaching staff.

Option 4: Project of your choice

You are welcome to propose project ideas; but make sure to get approval from teaching staff before moving forward with your project.

Getting Started

Setting up your workspace and microSD card

frame buffer: buffer for each frame displayed on LCD

As in previous labs, we'll begin by setting up a workspace complete with a framebuffer-enabled stock kernel. We're also providing a network-enabled version of the stock rootfs with Qt pre-installed. Finally, we also provide an updated bootfs that supports device tree overlays.

```
# Run these on a lab machine (in-person in PHO 307 or via
# SSH-ing into ENG-grid and running 'qlogin -q instruction.q').
source /ad/eng/courses/ec/ec535/bashrc_ec535
export WORKSPACE=$HOME/EC535/lab5 # Feel free to edit as you see fit
mkdir -p $WORKSPACE
cd $WORKSPACE
tar -xzf $EC535/bbb/stock/stock-rootfs-net.tar.gz
```

```
ln -s $EC535/bbb/stock/stock-zImage-fb ./
ln -s $EC535/bbb/stock/stock-linux-4.19.82-ti-rt-r33-fb ./
tar -xzf $EC535/bbb/stock/stock-bootfs-net.tar.gz
chmod +x rebuild-rootfs.sh
chmod +x rebuild-bootfs.sh
./rebuild-rootfs.sh
./rebuild-bootfs.sh
```

Next, we'll create a blank `sdcard.img` and fill it with our prepared filesystems (again, just like in the previous lab).

```
cd $WORKSPACE
dd if=/dev/zero of=sdcard.img bs=1M count=512
parted --align=opt --script -- ./sdcard.img mktable msdos \
    mkpart primary fat16 0\% 63MiB \
    mkpart primary ext4 64MiB 100\% \
    set 1 boot on \
    unit MiB \
    print
dd if=bootfs.img of=sdcard.img bs=1M seek=1 conv=notrunc
dd if=rootfs.img of=sdcard.img bs=1M seek=64 conv=notrunc
```

Once your `sdcard.img` is prepared, flash it to your microSD card. **If you are in PHO 305/307, please follow the following instructions.**(If you are logging in to lab machines remotely, please download `sdcard.img` to your local computer and flash it. Detailed instructions can be found in Lab 4)

1. Plug in your microSD card/reader to the lab machine. Then run the following command to see which device path the SD card was assigned.

```
dmesg
```

Try to find `sda` , `sdb` or similar names in recent messages. Then the device path of the SD card will be `/dev/sda` or `/dev/sdb` , respectively (or something similar.)

2. Run the following command , replacing "DEVICE_PATH" with the one you discovered from `dmesg` .

```
dd if=sdcard.img of=DEVICE_PATH
```

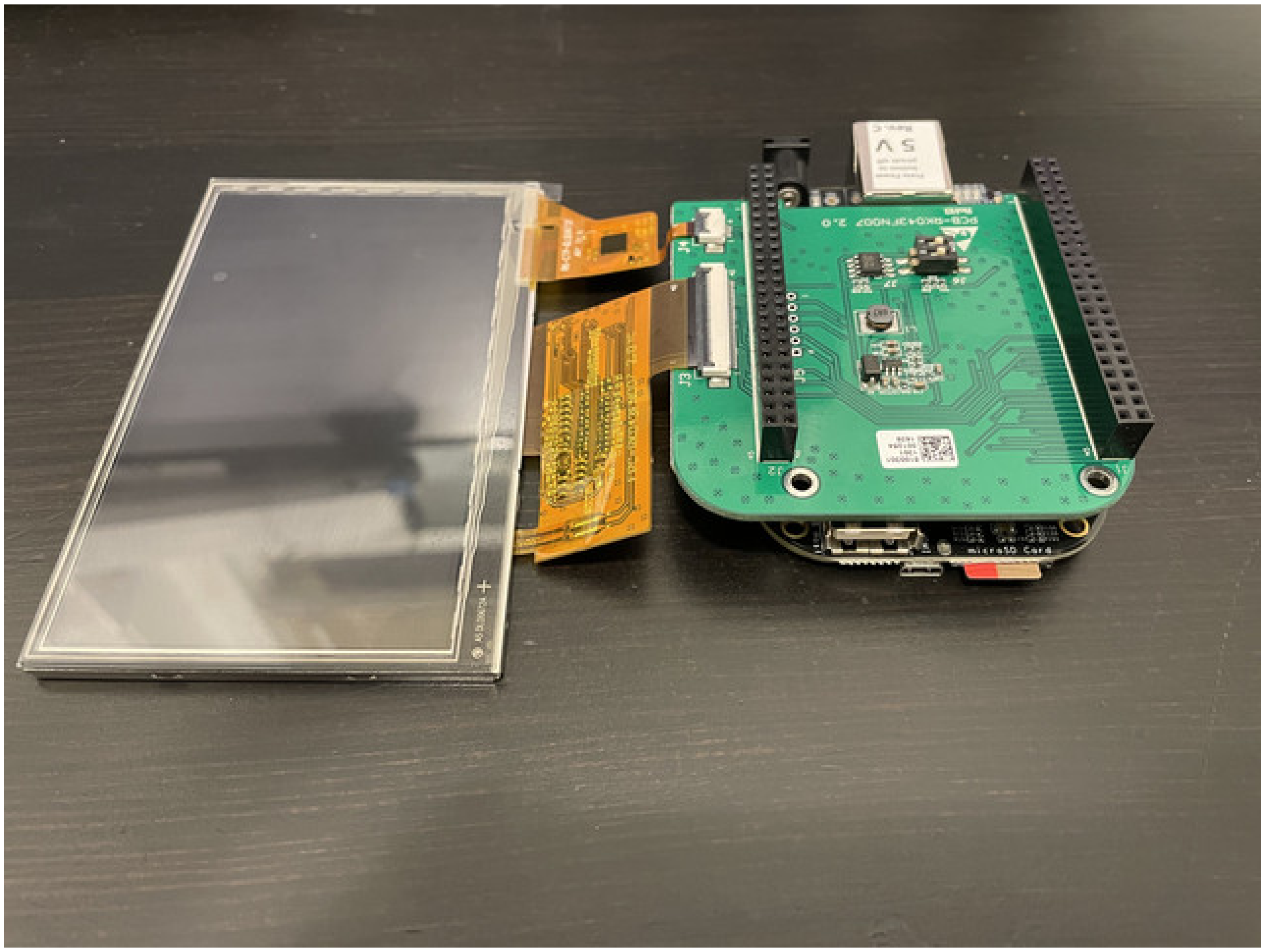
This command may not produce any output for a while.

Setting up the touchscreen

Open the box labeled `4.3" LCD Display Cape` that we provide in your lab kit, and *carefully* remove the touchscreen and cape (green PCB) from their packaging (you may need scissors for this). Please save all packaging for later reuse.

Take out your BeagleBone and remove any wiring, power adapters, or other connected accessories from previous labs. Line up the cape's pins with both black headers on the BeagleBone, aligning the "RoHS" stamp on the PCB with the corner of the BeagleBone's Ethernet port. Using equal pressure applied to both of the display cape's black headers, *gently* push down on the display cape until there's less than a millimeter of all pins showing. **Please try your best to avoid bending any pins during this process, as they're very difficult to re-align.**

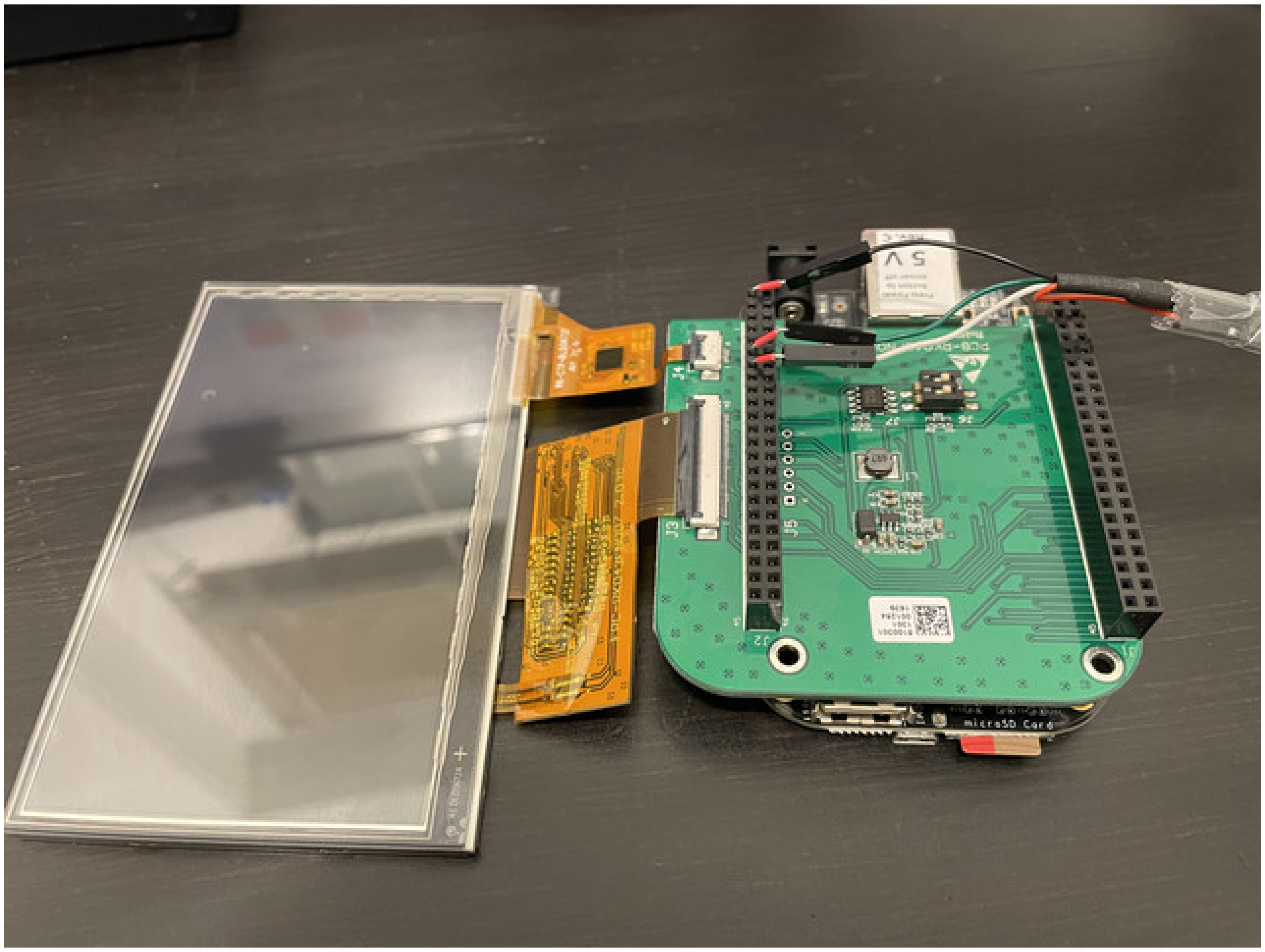
Next, take the touch screen, unfold its *very delicate* ribbon cables (removing masking tape if present), and place it screen-up on a flat surface next to your Beaglebone, lining up the wide and narrow ribbon cables with the `J3` and `J4` connectors on the display cape, respectively. Using your fingernail, gently pull up on the small black latches on `J3` and `J4` until they pop open. Slide the dark ends of each of the touchscreen's ribbon cables into `J3` and `J4`, such that closing the black latches holds the ribbon cables in place. The result should look like the picture below:



Setting up the alternate serial console

You may have noticed that the display cape blocks access to the pins to which you would normally connect your USB-to-TTL serial cable. Fear not; the BeagleBone is equipped with 3 “alternative” UART ports accessible through the black headers. Using the official pinout diagram, locate the `UART4_RXD` and `UART4_TXD` pins on the black header labeled `J2` on the display cape (shown as `P9` on the pinout diagram).

Next, take your USB-to-TTL serial cable and insert small jumper wires into the female connectors attached to the black, white, and green wires. Insert the jumper-wire-equipped black wire into a `DGND` pin (any of the pins along the top row of either header, nearest to the Ethernet port). Then connect the jumper-wire-equipped green wire into `UART4_RXD` and the jumper-wire-equipped white wire into `UART4_TXD`. As always, leave the red wire unconnected. The final result should look like this:



Connect your serial cable to your computer's USB port and open your serial console application, as in Lab 4. Insert your prepared microSD card, then connect the outlet-connected AC/DC power adapter to the BeagleBone while holding down `S2` (you may need to do this with your little finger). Once the BeagleBone has booted up, you should see a prompt like `ec535-stock-net login:`. Log-in with username `root` and password `ec535`.

Connecting to a network (optional)

This step is optional, can be completed at any time, and requires an Ethernet cable (not included in your lab kit, but we can provide it if needed).

To connect your BeagleBone to a network (and to the Internet, if accessible), simply connect your BeagleBone to a live Ethernet LAN port (e.g., on the back of your home's router, or on any of the wall boxes in PHO) using any Ethernet cable[3]. If connected before powering-on, your BeagleBone should automatically set up its connection during boot-up. Otherwise, you can manually trigger connection setup by running `ifup -a` on your BeagleBone. You can test your connectivity to Google, for example, by running `ping 8.8.8.8`.

Tip: this stock image has an SSH server with file-transfer capabilities ([dropbear]) pre-installed. The `ifconfig` command can help you determine the BeagleBone's current IP address.

Exploration

Run the commands in the subsections below *on your running BeagleBone*, unless otherwise specified.

Interacting with the framebuffer in userspace

Before we get started with Qt-based GUIs, let's try a straightforward method of interacting with the framebuffer in userspace. Like everything else in Linux, the framebuffer is represented by a file, `/dev/fb0`. Before we can play around with it though, we need to prevent the *framebuffer console* ("Please press enter to activate this console") from overwriting our framebuffer changes using the following command.

```
echo 0 > /sys/class/vtconsole/vtcon1/bind
```

Now let's try writing some random data to the framebuffer and see what happens on the display.

```
dd if=/dev/urandom of=/dev/fb0
```

Now observe what happens when you write all zeros to the framebuffer.

```
dd if=/dev/zero of=/dev/fb0
```

Let's try displaying something more meaningful. We'll use the `fbv` utility to draw some images directly on the framebuffer. Run these commands one at a time, observing the display, and then pressing `q` to quit `fbv` before running the next.

```
fbv -f /root/test-images/seal.png
fbv -f /root/test-images/beagle.png
fbv -f /root/test-images/flowers.jpg
```

Finally, restore the framebuffer console with the command below.

```
echo 1 > /sys/class/vtconsole/vtcon1/bind
```

Interacting with the framebuffer in kernelspace

While it's fun and easy to play with the framebuffer in userspace, we love kernel modules in this course. We've provided a demonstration module, `hellofb`, for you to try. Unbind the framebuffer console like you did in the last step, then load the module, observing the display.

```
echo 0 > /sys/class/vtconsole/vtcon1/bind
insmod /root/hellofb.ko
```

You should see a red rectangle appear on the display. Now remove the module.

```
rmmod hellofb
```

And that's it! You can inspect the source code for `hellofb` under `$EC535/examples/other/hellofb`.

Launching the Qt demo apps

Qt is a graphical framework that makes it easy for developers to draw graphical displays via the framebuffer device `/dev/fb0`, much like you did above. The network-enabled stock image includes several applications under `~/qt-examples/` that demonstrate Qt's abilities. Try running some of these applications and interacting with them via the touchscreen.

```
cd /root
./qt-examples/widgets/touch/fingerpaint/fingerpaint
./qt-examples/widgets/painting/pathstroke/pathstroke
./qt-examples/quick/animation/animation
./qt-examples/charts/qmlweather/qmlweather
./qt-examples/charts/donutbreakdown/donutbreakdown
```

Note that the source code for these demos is located alongside each binary. Also note that Qt applications have their own build system, `qmake`, that you'll need to use in addition to `make`. Try it out by making a copy of the `basicdrawing` example and compiling it yourself on ENG-Grid:

```
# Run these commands on ENG-Grid, not on your BeagleBone
cd $WORKSPACE/rootfs/root/qt-examples/widgets/painting/basicdrawing
rm basicdrawing
ls
# Notice how there's no Makefile, but there is a .pro file...
qmake
ls
# Now there's a Makefile!
make
ls
# And there's the binary!
```

You can learn more about the `qmake` build system on the Qt 5 website: https://doc.qt.io/qt-5/qmake-tutorial.html

Reading HID inputs from userspace

Screensavers need a way of determining when the system is "idle" in order to know when to launch and when to exit. If you choose to create a kernelspace screensaver, you'll do this using the kernel's HID subsystems. Before you attempt this, however, it's helpful to examine how HIDs work in userspace.

Your BeagleBone has one HID connected at the moment: the touchscreen. Like everything else in Linux, the touchscreen is represented in userspace as a device file. Device files that represent HIDs are known as event devices or "evdevs". Try running the command below, and touch the display immediately after. Press `Ctrl-C` to stop monitoring the evdev.

```
cat /dev/input/event0 | hexdump
```

Now how could you intercept and use this data in kernelspace? See *Additional Helpful Resources* above for some resources to help you answer this question.

Additional Requirements

Collaboration

As stated in the introduction, you may work in teams of 2-3 on this lab. Teams of 3 people will be held to higher standards of complexity, robustness, and visual appeal during grading.

If you choose to work in a team, each member must outline their division of work in the `README`. All students are also required to use a version control system such as Git.

Report

You must submit a project report by the deadline listed at the top of this page. See Blackboard > Content > Lab Assignments > Lab 5/Project Additional Materials > Lab 5 Report Template for more details.

Grading

See Blackboard > Content > Lab Assignments > Lab 5/Project Additional Materials > Lab 5 Rubric for details on how your submission will be graded.

---

1. Please see Piazza post for a list of sensors and other hardware that's available to students. ↩
2. E.g., you could use the traffic light from Lab 4 to show the speed of traffic on a particular highway. Creativity is welcome, but be sure to run your idea past course staff first. ↩
3. We assume your network automatically assigns IP addresses to new devices using DHCP (this is extremely common). If you know this is not the case for your network, contact the TA. ↩