Multimodal Tactile Sensor Module Version 2 User Manual

Joshua Wade

May 4, 2017

1. Sensor Design

Figure 1 shows a diagram and photo of the completed sensor module. The individual sensors are attached to a 3D printed base. As shown in Figure 2, the system uses a separate circuit module with two Teensy 3.2 microcontrollers and a buck converter to provide power to and read data from the various sensors.

1.1 Fabric-Based Force Sensor: The fabric-based force sensor is based on our open-source fabric-based skin design. The force sensor's fabric-based electrodes are connected to stranded wires using conductive epoxy. Figure 3 shows the circuit used to read the sensor's signal at 1kHz.

***1.2 Contact Microphone*:** As shown in Figure 3, we used a 2 cm contact microphone and a MAX4466 amplifier with the gain set at 25 to measure audio signals resulting from contact. The Teensy 3.2 microcontroller shown in Figure 3 measures the signals from the contact microphone at 10 kHz.

***1.3 Accelerometer*:** We used an ADXL335 Accelerometer to measure acceleration in the z direction (normal to the surface of the sensor). Similarly to the force sensor, we measured the acceleration signals at 1kHz.

***1.4 Active Thermal Sensor*:** The active thermal sensor uses a self-heated 10 kΩ B57541G1103F NTC thermistor. Figure 4 details the circuit used to provide power to, and read data from the thermal sensors at 100 Hz. Note that this Teensy 3.2 is separate from the one shown in Figure 3 that is used to measure force, audio and acceleration. The active thermal sensor is self-heated with an adjustable voltage input (typically 7 - 12V) from a buck converter.

***1.5 Passive Thermal Sensor*:** Like the active thermal sensor, the passive thermal sensor uses a 10 kΩ B57541G1103F NTC thermistor. As shown in Figure 4, the circuitry for passive thermal sensing is similar to the active thermal sensor except that the sensor is powered by a constant 3.3V input from the microcontroller rather than a higher voltage buck converter. The passive thermal sensor also has a 10 kΩ reference resistor rather than a 1 kΩ reference resistor like the active thermal sensor. This higher resistance value limits the current passing through the thermistor to minimize self-heating for more accurate passive thermal sensing.

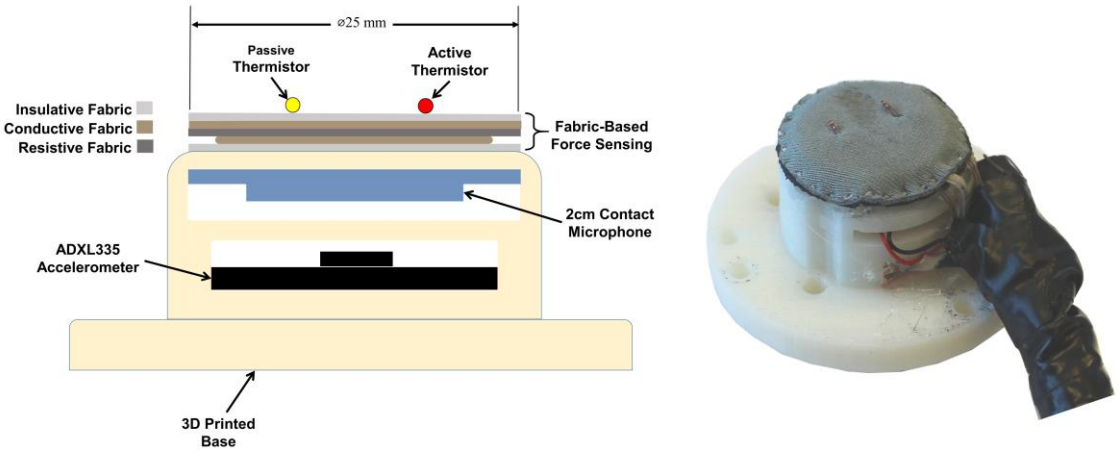

Figure 1: Multimodal sensor module

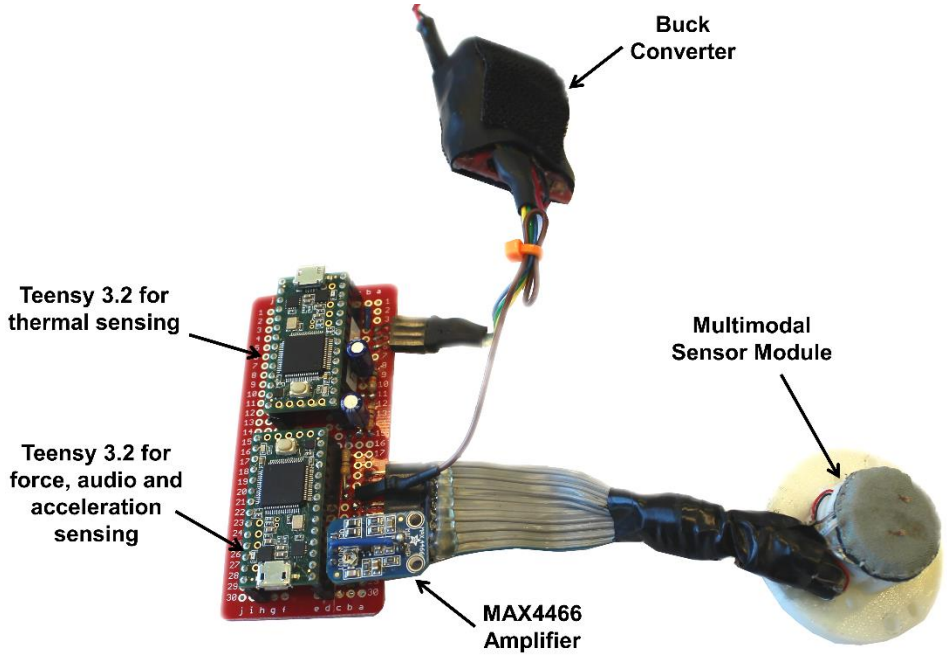

Figure 2: Multimodal sensor complete system

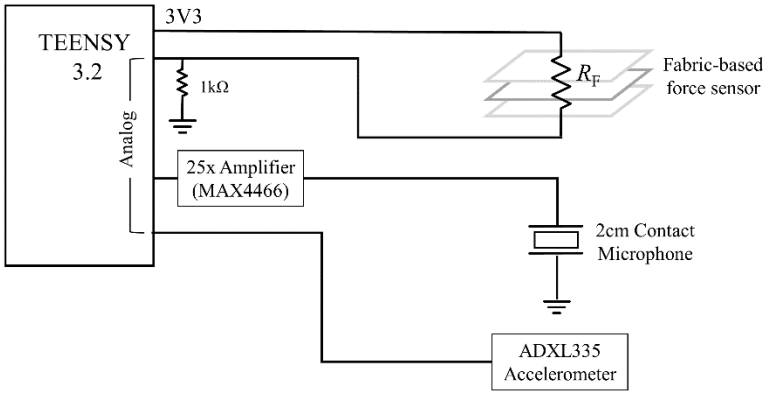

Figure 3: Circuit for force, audio and acceleration sensing

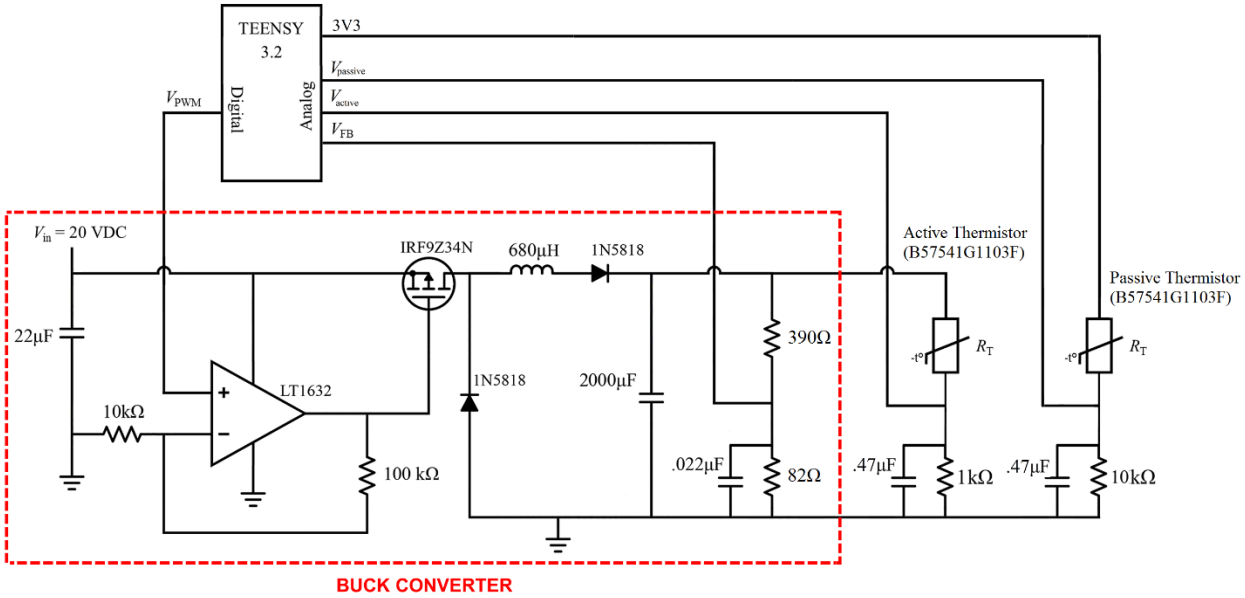

Figure 4: Circuit for active and passive thermal sensing

2. Microcontroller Code

2.1 Code for measuring force, audio and acceleration: The code for the Teensy 3.2 shown in Figure 3 that measures force, audio and acceleration is available here. When flashed with this code, the Teensy communicates over serial with a baudrate of 500000. It will send sensor data over serial at a rate of 1 kHz.

Each serial message is structured as follows:

<Force (N)>, <Acceleration (m/s^2)>, <Audio_1 (V)>, < Audio_2 (V)>, …, < Audio_10(V)> \n

Because the audio is sampled 10 times faster than the data is sent (10 kHz vs. 1 kHz), each serial message includes the last 10 audio samples. When the microcontroller is first powered on, it automatically biases the force and audio sensors. To re-bias the force sensor only, simply send a single character ‘B’ to the Teensy via serial.

2.2 Code for measuring active and passive thermal sensors: The code for the Teensy 3.2 shown in Figure 4 that measures active and passive thermal sensor data is available here. When flashed with this code, the Teensy communicates over serial with a baudrate of 115200. It will send sensor data over serial at a rate of 100 Hz. Because the thermal signals measured have much slower dynamics compared to the force, acceleration and audio signals, this slower sample rate of 100 Hz is sufficient.

Each serial message is structured as follows:

<Active (°C)>,<Passive (°C)>,<Desired Voltage (V)>, <Actual Voltage (V) >\n

This code includes a feedback controller to regulate the initial temperature of the active thermal sensor, T, to a desired value, T_{des} which is 55.0 °C by default. Figure 5 provides a block diagram of the feedback controller. As shown in the figure, an outer PID controller is used to specify the desired buck converter voltage, V_{des}, to adjust the sensor temperature. A second, much faster PID controller adjusts the PWM duty cycle, D, supplied to the buck converter by the microcontroller to adjust the buck converter output voltage, V, to the desired voltage. When the active thermal sensor temperature is within 0.5 °C of the setpoint, the orange light on the Teensy will flash, indicating that the desired temperature is achieved.

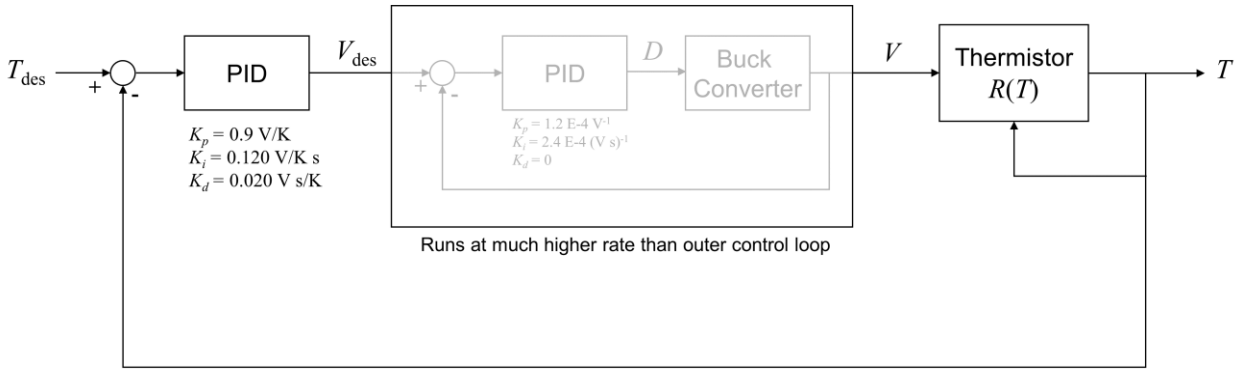

Figure 5: Controller block diagram for buck converter and active thermal sensor

When the sensor is contacting an object, it advisable to turn off the temperature feedback controller by holding V_{des} constant. This ensures that the feedback controller does not cancel informative temperature changes from the active thermal sensor. To do this simply send a single ‘H’ character

to the microcontroller over serial. After the sensor is no longer in contact with an object, send a single ‘C’ character re-activate the temperature controller.

The microcontroller also allows you to specify a new temperature setpoint (other than 55.0 °C) by sending a command in the form ‘T <temperature setpoint in millidegrees Celsius> ’ (note space after number). Valid temperature setpoints range from 0 to 55.0 °C. This command will also activate the temperature controller like sending ‘C’. For example, to set the temperature setpoint to 45 °C and turn on the temperature controller:

```
Send_over_serial('T 45000 ')
```

Finally, the microcontroller also allows you to specify a new voltage setpoint and turn off the temperature controller. This is done by sending a command in the form ‘V <voltage setpoint in mV> ’ (note space after number). Valid voltage setpoints range from 1 to 14.0 V. For example, to turn off the temperature feedback controller and set the buck converter voltage to a constant 9 V:

```
Send_over_serial('V 9000 ')
```

3. Data Collection

3.1 Udev Rules: If using Ubuntu it is convenient to add a udev rule so that each Teensy 3.2 connected to the computer via USB will show appear with a persistent name related to the Teensy’s manufacturer serial number. This rule file must be placed at /etc/udev/rules.d/49-teensy.rules (preferred location) or /lib/udev/rules.d/49-teensy.rules (required on some broken systems). To install, type this command in a terminal:

```
sudo cp 49-teensy_modified.rules /etc/udev/rules.d/49-teensy_modified.rules
```

if there is a file called 49-teensy.rules in /etc/udev/rules.d/, move it out of this directory to a backup location. After this file is installed, physically unplug and reconnect Teensy.

3.2 Python Script to Save Sensor Data: A python code that reads data from the microcontrollers and saves it in pickle files is provided here: save_sensor_data.py. The original script and the following instructions were written for use with Ubuntu but could also be used with other operating systems. To run the code, enter:

```
python save_sensor_data.py myFolder
```

Now when the sensor is pressed to an object with a force exceeding the threshold of 0.1 N, the script will create a folder (named ‘myFolder’ in this example) in the same location where the script is saved. Within this folder it will create subfolders called ‘trial_1’and save 5 pickle files containing the sensor data and a file called graph.png that provides a visualization of the data.

The pickle files are as follows:

Active thermal sensor data:	`myFolder/trial_1/a.pkl`

Passive thermal sensor data:	`myFolder/trial_1/p.pkl`
Force data:	`myFolder/trial_1/f.pkl`
Acceleration data:	`myFolder/trial_1/ac.pkl`
Audio data from contact microphone:	`myFolder/trial_1/m.pkl`

Each pickle file contains a numpy array where the zeroth column is time and the first column is the sensor data. By default, the script saves data from 0.5 s before contact to 5 s after contact. Once the trial is saved the script will once again wait for contact to occur and save additional trials as ‘trial_2’, ‘trial_3’, etc.