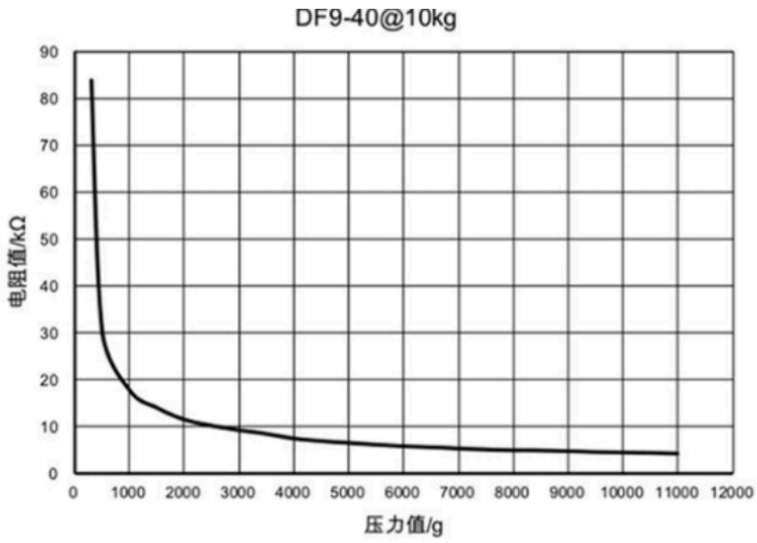
Step 1

Get the value from the FSR.

The range of value received from FSR is $[0, 1023] \cap \mathbb{Z}$. The more presure pushed on the FSR, the smaller the value received from sensor. However, the value from FSR is not linearly corresponding to the force. According to the description on the Internet, the relationship between value of FSR and force is something like this:



and the value from FSR is actually the voltage of that $10k\Omega$ resistance (This actually depends on the layout on the breadboard), and we can calculate the resistance of FSR using this formula :

$$R_{\text{fsr}} = \frac{x \times 10k\Omega}{1024 - x}$$

where x is the value read from FSR. Then, the graph above is somehow like $y = \frac{c}{x}$ (Actually this is not correct, but I can only fit it using my weak mathematical skill), then we can get the the value that is proportional to the force :

$$F \propto y = \frac{1}{R_{\text{fsr}}} = \frac{1024 - x}{x \times 10k\Omega}$$

Then we map y to a specific interval, for example $[0, 10^4]$, and I choose this formula :

$$y' = \frac{1024 - x}{x} \times 510$$

Step 2

Successfully to control the frequency of speaker using FSR.

Personally, I think the speaker can at least work well from 220Hz to 880Hz (according to the custom test) and face some issues on A5. I think it is very hard to play a melody using FSR, since that it is too difficult to controlthe force.

Step 3

Restructure the breadboard and connect the button in parallel.

It is much more easier to play the instrument compared to that of Step 2, since that we can play notes precisely. I had played *Little Star* on it.

Step 4

Restructure the breadboard again, and actually change the entire layout due to the narrow space.

We should squeeze and release the FSR periodically. It is hard to play in tune because of the unsteadiness of force conducted by human. To make it easier, we can change the algorithm to that, when the FSR is pressed and the force on it is large enough (bigger than a threshold) then add offset to the frequency of the note, and the offset is a sin function of time.

Bonus

Actually I reuse the frequency array in Step 3 and store the melody by the index of frequency and beats. and since these values are all less than 255, we can just use `uint8_t` to store them. Then use another `int` variable to store the state of the button during last time the `loop()` function executed.

Additionally, since that it is hard for user to control the beat, I add a mode that when the extra button is pressed, the instrument can automatically play the melody consistently. This mode can be switched on by using `#define AutoMode` marco when compiling.

Furthermore, to make the effect of FSR more stable, I apply the *middle value filter*.