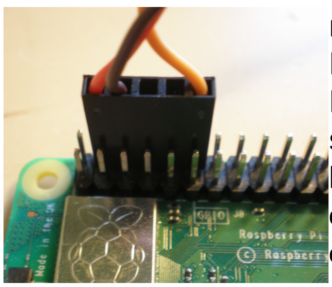
Seattle Raspberry Jam
A Raspberry Pi Meetup for Beginners to Experts

RASPBERRY JAM

Hardware Project #4: Servo Dial

Assembly

First, 3D-print the Dial.stl file (from github.com/alexmous/Seattle-Raspberry-Jam). Push the servo into the dial and screw in the screws as shown in the picture to the right. Next, place the needle/servo arm onto the servo's shaft. Do not screw in the screw on the servos shaft, as we will need to calibrate the direction of the needle later. Finally plug the socket from the servo into the Raspberry Pi's header on pins 4, 6, 8, 10 and 12 as shown in the picture to the left. **WARNING! Incorrectly installing the servo's connector could damage the Raspberry Pi. Please have one of the organisers check that it is plugged in correctly before applying power.**



Servo Software Setup

Note: gpio may not be installed; you can use sudo apt-get install wiringpi to install it. To setup the servo, we need to run a specific sequence of commands. First, configure GPIO pin 18 for pwm:

```
gpio -g mode 18 pwm
```

Next, we will set the postion of the servo to 90 degrees:

```
gpio -g pwm 18 150
```

Now run the following series of commands:

```
gpio pwm-ms
gpio pwmc 192
gpio pwmr 2000
```

These set up some important parameters. The first command sets the PWM peripheral to use the M/S algorithm. The second command sets the clock divider to 192. The last command sets the range to 2000. See the box on How Does the Raspberry Pi's PWM Work? for more information on these parameters. Now ready to move the servo!

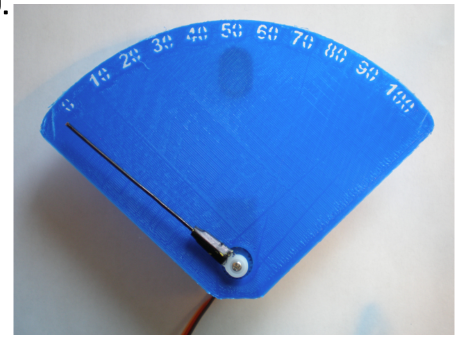
Moving the Servo

```
gpio -g pwm 18 PULSE
```

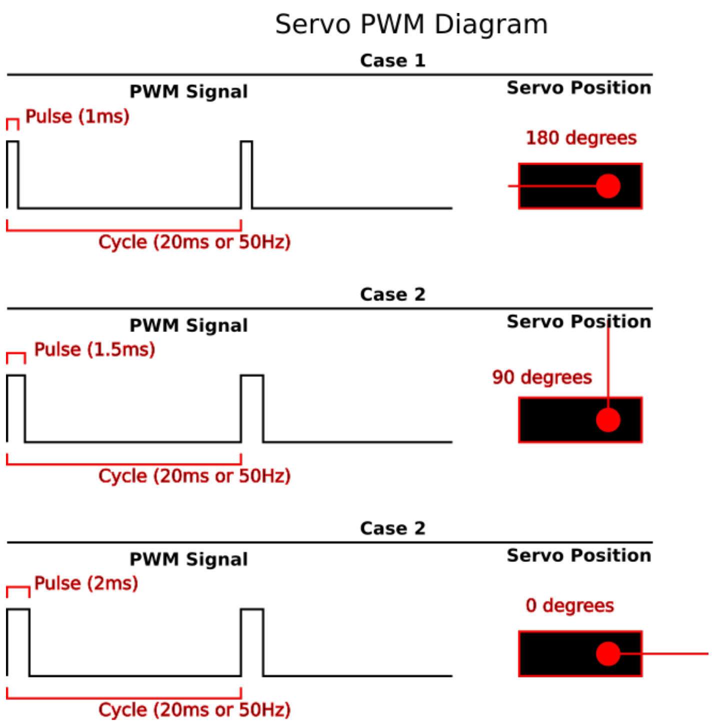
We can use the following command to control the servo's arm:

Continued overleaf

To move the servo to the left, put 200 in for PULSE. For right, use 100. Halfway in between, 150, makes the arm point straight up. Now we can aclibrate and screw the needle on to the servo. (See the picture to the right for what the completed servo dial should look like.) To do this, put 200 in for PULSE. After the servo has finished moving, line the needle up approximately with the 0 on the dial and press it down onto the servo's shaft. Finally, screw in the screw to secure the needle. We can now create a shell script to check the processor temperature and adjust the servo accordingly.



How do servos work?



Servos are useful for all sorts of applications because they are highly efficienct and precisely control the rotation of the shaft, usually from 0 to 180 degrees. But how do these little plastic boxes work? Essentially, a servo is a motor, a control circuit, a potentiometer and some gears. The potentiometer and motor are connected to each other, so when the motor turns, the potentiometer's resistance changes. This resistance is how controller knows where the motor is at all times. The controller also takes data from outside of the servo on the data wire. Servos typically have three wires: one for positive voltage, one for ground and one data wire. To control the servos, Pulse Width Modulation (PWM) data is sent to the servos control wire. The controller than reads this data and adjusts the motor accordingly. PWM can be best demonstrated by a diagram, so see the diagram above for an illustration. Interestingly, the width of each pulse on the PWM dictates the servos position. A 1 millisecond pulse is all the way to the left, whereas a 2 millisecond puls is all the way to the right. The amount of time for each cycle is also important for servos, and is usually 20 milliseconds (50Hz).

Create A Shell Script

First, we need to create and open a new file:

```
nano servo_temp.sh
```

Enter the following text into the file (Note: the code below is without comments; you can download the commented code from https://github.com/alexmous/Seattle-Raspberry-Jam in the *Hardware Project #4* folder):

Continued overleaf

How does the Raspberry Pi's PWM work?

To understand the `gpio` commands, we need to know how the Raspberry Pi's PWM works. Refer to the diagram in the How do Servos Work? box for a diagram of the pulse and the cycle. Firstly, PWM can run in two modes: mark/space or balanced. The `pwm-ms` command sets mark/space mode. This is standard PWM; a description of balanced mode is beyond the scope of this article. The two other parameters, `pwmr` and `pwmc`, are range and clcok divider, respectively. The first, the range, is essentially the resolution of the duty cycle (the duty cycle is the ratio of length of pulse to total cycle, for example a 1ms pulse in a 2ms cycle would be a duty cylce of 50%). We set this value to 2000, so we get a high resolution and therefore we can control the servo accurately. The next parameter, the clock divider, is the frequency of each resolution of the range. For example, we use 192, so the frequency of each "resolution" is 100,000Hz (since the PWM clock is 19.2MHz, divided by 192). So over 2000 cycles of that (since the range is 2000), dividing range by frequency, we get a 20ms. Thus, the PWM frequency can be calculated with 19200000/pwm_clock/pwm_range. For servos, this value needs to be 50Hz, and with 192 as the clock divider and 2000 as the range, this will work. See the BCM2835 Arm Peripherals datasheet for more informatiuon.

```
#!/bin/bash

declare -A convert=( [0]=200 [1]=190 [2]=180 [3]=170 [4]=160 [5]=150
[6]=140 [7]=130 [8]=120 [9]=110 [10]=100 )
PREV_TEMP=-1
setup_pin () {
     gpio -g mode 18 pwm
     gpio -g pwm 18 150
     gpio pwm-ms
     gpio pwmc 192
     gpio pwmr 2000
}
get_temp () {
     RAW=`cat /sys/class/thermal/thermal_zone0/temp`
     RAW=$[(($RAW/1000)+5)/10]
     return $RAW
}
main () {
     setup_pin
     while :
     do
          get_temp
          TEMP=$?
          ANGLE=${convert[$TEMP]}
          if [ $TEMP -ne $PREV_TEMP ]; then
               gpio -g pwm 18 200
               sleep 1
               gpio -g pwm 18 $ANGLE
          fi
          sleep 15
          PREV_TEMP=$TEMP
     done
}
main
```

Save and quit by typing Ctrl-X, then Y and finally press Enter. Before we can run this script, we need to make it executable. This can be accomplished with the following command:

```
chmod +x servo_temp.sh
```

Now we can run the script with:

```
./servo_temp.sh
```

After a few seconds, the servo will calibrate itself and then point towards the current temperature on the dial. If you want to test how the dial works, we have created another piece of code that will heavily load the CPU, therefore raising the temperature. The file is called *stress_test.py* and can be downloaded from the same place as the servo_dial.sh. Run this code with `python stress_test.py` (when you want to quit, type Ctrl-Z). Congratulations! You have completed this project. The dial we designed is multipurpose, so you could also use it to measure the humidity or ambient temperature.

Challenge: Modify the shell script to measure the current processor speed instead of the temperature. Remember: you will need to scale the speed correctly so that it is in the range 0 to 100. Hint: the current speed is in the */sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq* file.

