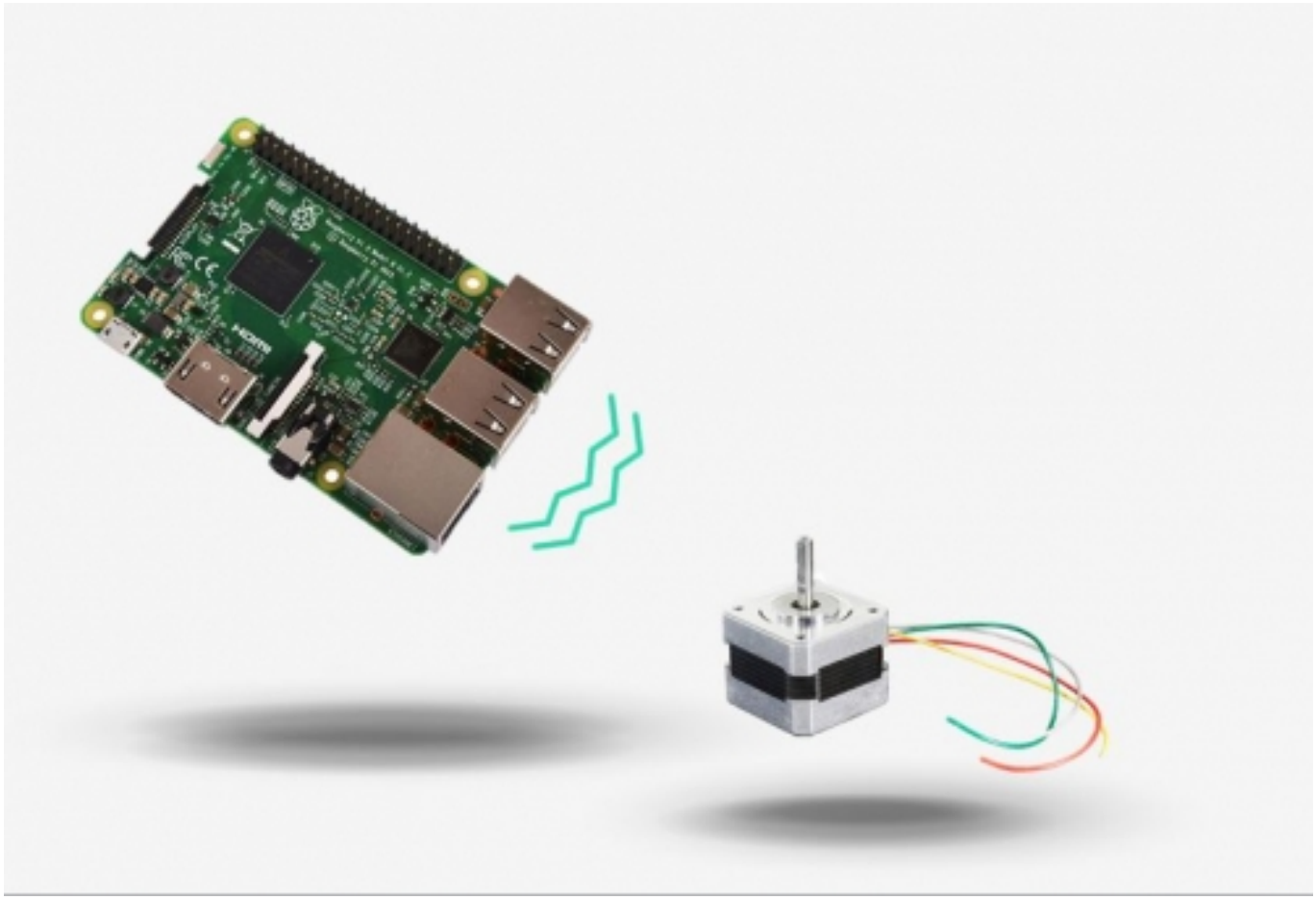
آموزش موتورهای استپر با رسپبری‌پای (قسمت سوم)

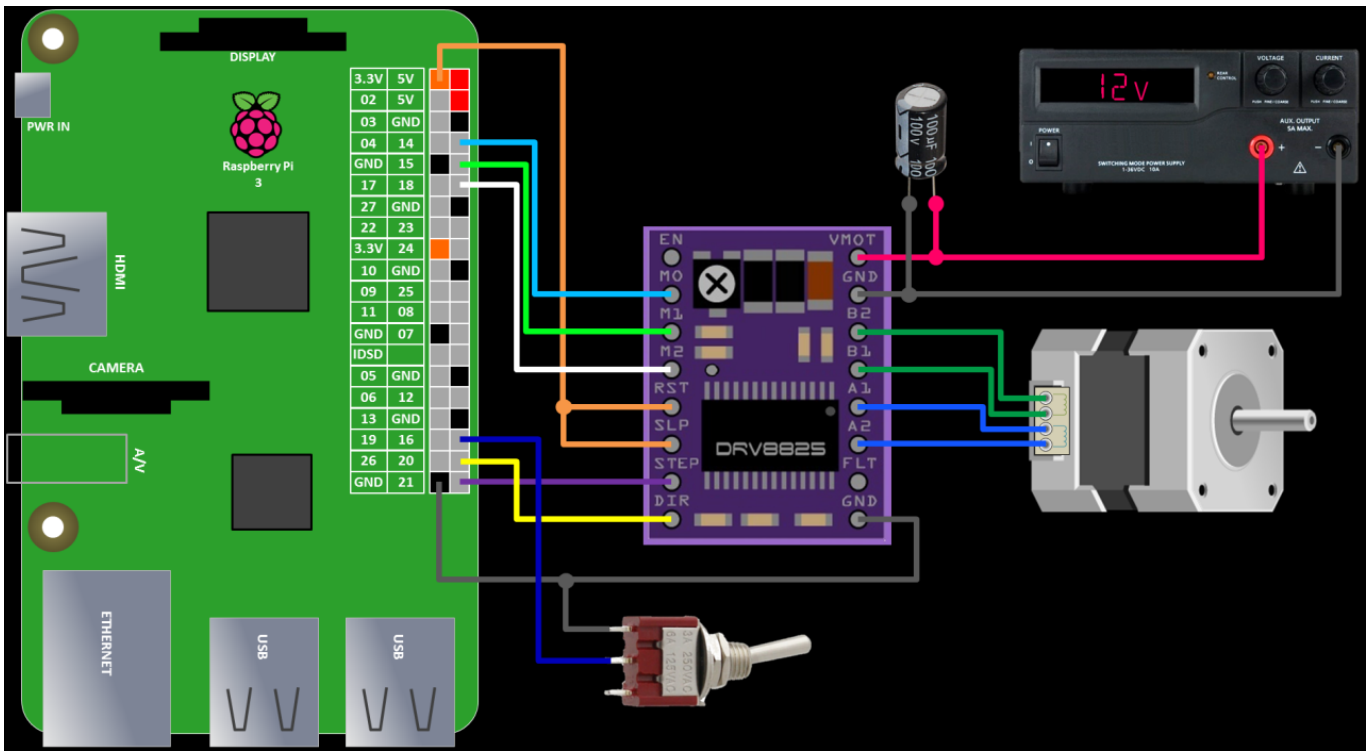


در این آموزش نشان می‌دهم که چگونه موتورهای استپر (پله‌ای) دوقطبی را روی یک رسپبری‌پای در پایتون با استفاده از یک درایور استپ موتور DRV-8825 کنترل کنیم.

در آموزش " آموزش موتورهای استپر با رسپبری‌پای (قسمت اول)" با موتور پله‌ای و درایور DRV-8825 آشنا شده و نحوه اتصال آن‌ها به برد رسپبری‌پای نشان داده شد و در " آموزش موتورهای استپر با رسپبری‌پای (قسمت دوم)" موتور پله‌ای را با رسپبری‌پای راه‌اندازی کردیم.

برای مثال بعدی، سوئیچی برای تغییر جهت می‌تواند به آموزش قبل افزوده شود. یکی از ترمینال‌های کلید به GPIO 16 می‌رود و ترمینال دیگر به زمین می‌رود.

 |



یکی از مشکلات برنامه قبلی در " آموزش موتورهای استپر با رسپبری‌پای (قسمت دوم)" این است که برای زمان‌بندی به روش sleep پایتون وابسته است که خیلی قابل‌اعتماد نیست. برای مثال بعدی، از کتابخانه PiGPIO استفاده می‌کنم که سخت‌افزاری را بر پایه زمان‌بندی PWM فراهم می‌کند.

قبل از استفاده، PigPIO daemon باید با استفاده از sudo pigiod مربوط به یک ترمینال شروع شود.

```
sudo pigpiod
```

قسمت اول کدی که در ادامه می‌آید مشابه مثال اول است. ترکیب بقیه قسمت‌ها با استفاده از کتابخانه PiGPIO اصلاح‌شده است. روش set_PWM_dutycycle برای تنظیم PWM dutycycle استفاده‌شده است. این درصدی از پالس است که بالا و پایین است. مقدار 128، این را به 50% تنظیم می‌کند؛ بنابراین بخش‌های روشن و خاموش چرخه مشابه هستند. روش set_PWM_frequency تعداد پالس‌ها بر ثانیه را تنظیم می‌کند. مقدار 500، فرکانس را 500 هرتز تنظیم می‌کند. یک حلقه while بینهایت سوئیچ را چک می‌کند و جهت را به‌طور مناسب تغییر می‌دهد.

```
from time import sleep
import pigpio

DIR = 20      # Direction GPIO Pin
STEP = 21     # Step GPIO Pin
SWITCH = 16   # GPIO pin of switch

Connect to pigpiod daemon #
pi = pigpio.pi()

Set up pins as an output #
pi.set_mode(DIR, pigpio.OUTPUT)
pi.set_mode(STEP, pigpio.OUTPUT)
```



```
Set up input switch #
(pi.set_mode(SWITCH, pigpio.INPUT
(pi.set_pull_up_down(SWITCH, pigpio.PUD_UP

MODE = (14, 15, 18)   # Microstep Resolution GPIO Pins
,(RESOLUTION = {'Full': (0, 0, 0
,(Half': (1, 0, 0'
,(0 ,1 ,0) :'1/4'
,(0 ,1 ,1) :'1/8'
,(1 ,0 ,0) :'1/16'
{(1 ,0 ,1) :'1/32'
:(for i in range(3
([pi.write(MODE[i], RESOLUTION['Full'][i

Set duty cycle and frequency #
pi.set_PWM_dutycycle(STEP, 128)  # PWM 1/2 On 1/2 Off
pi.set_PWM_frequency(STEP, 500)  # 500 pulses per second

:try
:while True
pi.write(DIR, pi.read(SWITCH))  # Set direction
(sleep(.1

:except KeyboardInterrupt
("...print ("\nCtrl-C pressed.  Stopping PIGPIO and exiting
:finally
pi.set_PWM_dutycycle(STEP, 0)  # PWM off
()pi.stop
```

یک مشکل استفاده از PiGPIO set PWM frequency این است که این دستور به مقادیر خاص فرکانسی در هر نرخ نمونه‌برداری محدود است که به‌طور خاص در جدول زیر آمده است:

Sample Rate	Hertz								
1:	40000	20000	10000	8000	5000	4000	2500	2000	1600
	1250	1000	800	500	400	250	200	100	50
2:	20000	10000	5000	4000	2500	2000	1250	1000	800
	625	500	400	250	200	125	100	50	25
4:	10000	5000	2500	2000	1250	1000	625	500	400
	313	250	200	125	100	63	50	25	13
5:	8000	4000	2000	1600	1000	800	500	400	320
	250	200	160	100	80	50	40	20	10
8:	5000	2500	1250	1000	625	500	313	250	200
	156	125	100	63	50	31	25	13	6
10:	4000	2000	1000	800	500	400	250	200	160
	125	100	80	50	40	25	20	10	5

Default sample rate of 5 is set when PiGPIO daemon is started. -s to specify a different rate

شما می‌توانید نرخ مشابه را با استفاده از گزینه پیکربندی S هنگام شروع PiGPIO daemon تغییر دهید. نرخ نمونه پیش‌فرض 5 است که مقادیری بین 8000 و 10 هرتز دارد (ردیف سبزرنگ بالا را ببینید). برای مثال، در کد اینجا 500 هرتز را انتخاب می‌کنیم که یکی از این مقادیر است. اگر مخصوصاً 600 را انتخاب کنید، به‌طور خودکار آن را تا 500 کم می‌کند و 700 یک‌باره تا 800 اضافه می‌شود. اگر شما نیاز دارید از فرکانسی استفاده کنید که در جدول وجود ندارد، PiGPIO به شما اجازه می‌دهد که از PWM ساخته‌شده در سخت‌افزار پای استفاده کنید که این تنها در پین GPIO 18 قابل‌دسترس است. به‌جای استفاده از روش‌های set_pwm شما می‌توانید از روش hardware_PWM استفاده کنید. این روش پارامترهایی را برای GPIO، فرکانس و duty cycle می‌پذیرد.

```
(pi.hardware_PWM(18, frequency, duty_cycle
```

مشکل بعدی در استفاده از مثال انجام داده‌شده این است که ما تنها روی سرعت و جهت کنترل داریم. این نمی‌تواند تعداد گام‌ها را برای حرکت موتور تعیین کند. همچنین، سوئیچ کردن خیلی سریع و یا ناگهانی موتور می‌تواند منجر به از دست دادن گام‌ها شود. تمرین بهتر این است که به‌آرامی شتاب گیرد و از شتاب آن کاسته شود. این معمولاً به‌عنوان ramping اشاره می‌شود. هردوی این مسائل می‌توانند به کمک PiGPIO waveforms آدرس‌دهی شوند.

امروزه یک باگ در کتابخانه PiGPIO در رابطه زمان‌بندی شکل موج وجود دارد. یک‌راه حل این است که PiGPIO daemon را با گزینه t برابر صفر برای pwm شروع کنید که برخلاف حالت پیش‌فرض PCM است.

```
sudo pigpiod -t 0
```

این کار زمان‌بندی شکل موج را اصلاح می‌کند. خوشبختانه این باگ به‌زودی اصلاح خواهد شد. لطفاً توجه داشته باشید بااینکه این کار زمان‌بندی شکل موج را اصلاح می‌کند با زمان‌بندی PWM در مثال قبلی تداخل دارد و از استفاده از سخت‌افزار PWM() جلوگیری می‌کند.

روشی که در ادامه می‌آید generate_ramp توسط Joan نویسنده PigPIO به فروم رزبری‌پای فرستاده‌شده است. این روش یک ramp متغیر را می‌گیرد که فهرستی از فرکانس‌ها و جفت گام‌هاست. سپس یک زنجیره‌ای از شکل موج‌های مربوط به مقادیر منتخب تولید می‌کند.

```
def generate_ramp(ramp):
    """Generate ramp wave forms.
    ramp:  List of [Frequency, Steps]
    """
    pi.wave_clear()     # clear existing waves
```

```
    length = len(ramp)  # number of ramp levels
    wid = [-1] * length

    # Generate a wave per ramp level
    for i in range(length):
        frequency = ramp[i][0]
        micros = int(500000 / frequency)
        wf = []
        wf.append(pigpio.pulse(1 << STEP, 0, micros))  # pulse on
        wf.append(pigpio.pulse(0, 1 << STEP, micros))  # pulse off
        pi.wave_add_generic(wf)
        wid[i] = pi.wave_create()

    # Generate a chain of waves
    chain = []
    for i in range(length):
        steps = ramp[i][1]
        x = steps & 255
        y = steps >> 8
        chain += [255, 0, wid[i], 255, 1, x, y]

    pi.wave_chain(chain)  # Transmit chain
```

نگران نباشید اگر روش به نظر پیچیده می‌آید، چراکه این روش برای استفاده بسیار آسان است. برنامه زیر generate_ramp با 6 سطح ramp را فراخوانی می‌کند تا به‌آرامی از 320 هرتز تا 2000 هرتز شتاب بگیرد. گام‌ها در هر سطح ramp برای نشان دادن هدفمان اغراق‌شده است. اغلب یک فرکانس خاص موجب ایجاد رزونانس می‌گردد و ضروری است که شتاب را به‌سرعت افزایش داد تا به‌سرعت از درون فرکانس رزونانس عبور کند.

```
# Ramp up
generate_ramp([[320, 200],
               [400, 500],
               [500, 800],
               [700, 1000],
               [900, 1600],
               [10000, 2000]])
```

نظرات، پیشنهادات و انتقادات خود را براي بهتر شدن محتواي مطالب با ما در ميان بگذاريد...

ترجمه شده توسط تیم الکترونیک صنعت بازار | منبع: سایت rototron.info

