01

Entendendo o projeto

Transcrição

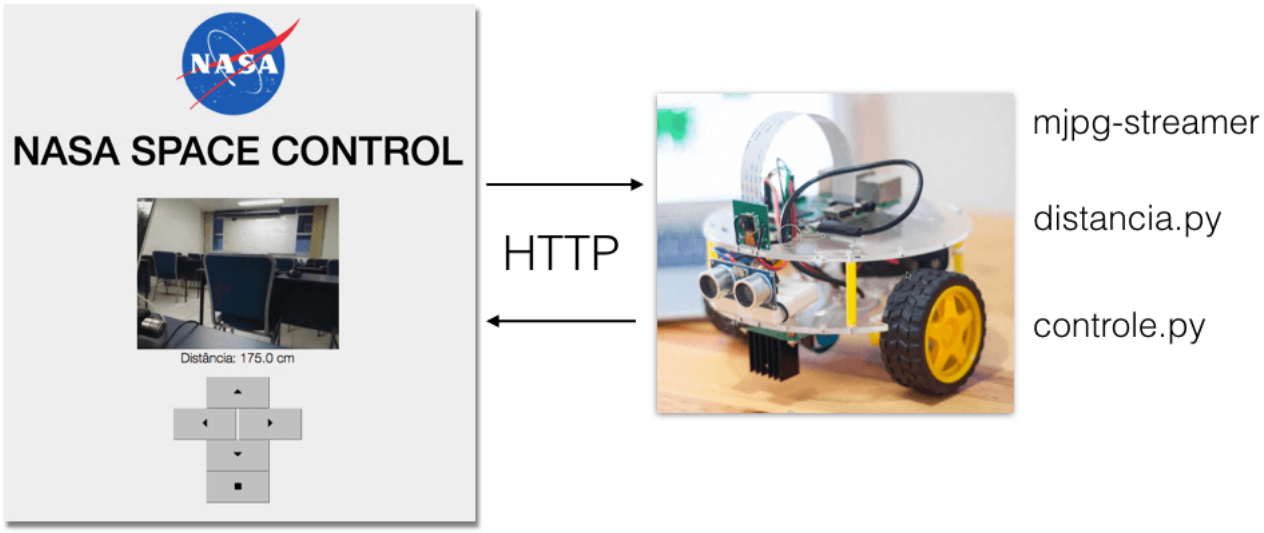
Vamos entender agora como as peças, tudo o que foi aprendido até agora, se encaixam. O nosso objetivo é ter um centro de controle do nosso carrinho, via browser. Já fizemos todas as partes eletrônicas, mecânicas do carrinho, então agora precisamos saber como todos eles vão funcionar, ser controlados através do navegador.

Quando acessarmos o IP do Raspberry Pi no navegador, na porta 5000, acessaremos o centro de controle. A página, quando estiver totalmente implementada, será semelhante a essa:

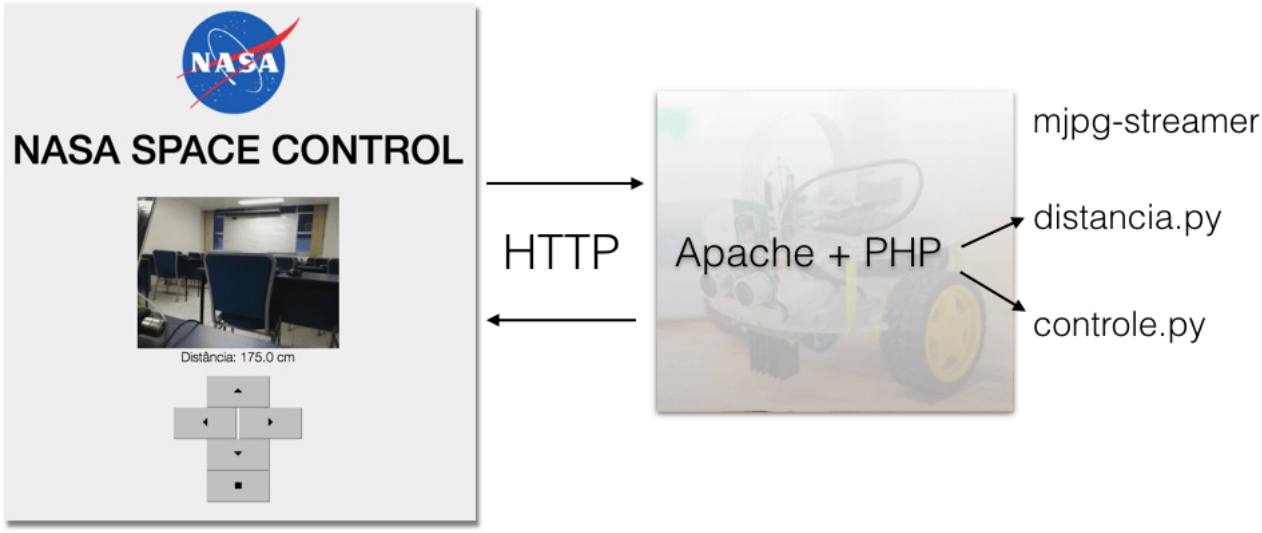
A página tem a câmera, distância, e os controles do carrinho, tudo o que construímos durante o treinamento.

Detalhando o centro de controle

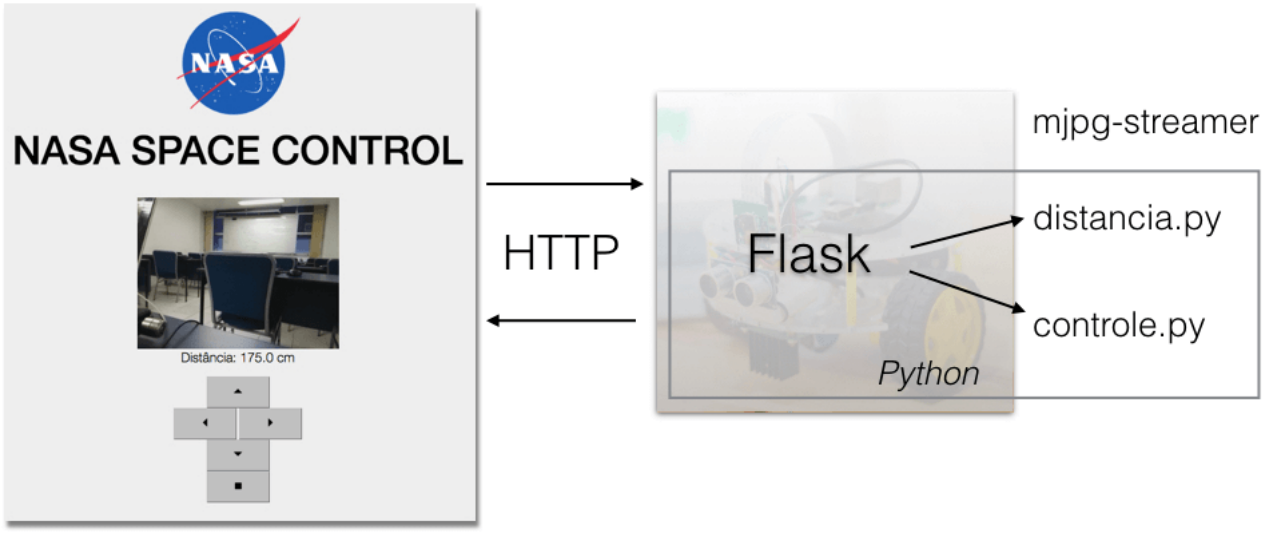
Para o carrinho funcionar via web, nós fazemos uma chamada HTTP até o mesmo, fazemos isso via navegador. E por baixo dos panos, temos 3 programas rodando, o **mjpg-streamer**, responsável por gerar as imagens da câmera e fazer *streaming* para a nossa tela, o **distancia.py**, que mede a distância dos objetos à frente do nosso carrinho, e o **controle.py**, responsável pela mobilidade do nosso carrinho.



No curso **Raspberry Pi: Controlando o mundo com GPIO** (https://cursos.alura.com.br/course/raspberrypi-controlando-o-mundo-com-gpio), quando fizemos o projeto de automação, para que pudéssemos integrar isso tudo, nós fizemos um pouco diferente. Nós tínhamos o navegador, que se comunicava via HTTP com o servidor web (Apache) e no meio do caminho ainda havia o PHP, porque precisávamos de uma programação para interagir com os scripts, programas, já que o servidor web por si só não fazia isso.



Mas qual a diferença do nosso projeto atual para o projeto do curso anterior? Sabemos que para rodar os nossos scripts, precisamos do Python, então ao invés de usarmos o conjunto Apache + PHP, usaremos um servidor web nativo em Python, o **Flask**. Com o Flask, nós temos um servidor web, para se comunicar com o navegador, e como ele nativamente roda Python, nós temos condições de acionar diretamente os nossos scripts.



Repare que na imagem o **mjpg-streamer** não está dentro do retângulo do Flask. Isso porque ele já é uma aplicação pronta, que já possui um servidor web, podendo assim se comunicar diretamente com o navegador.

No Flask, teremos o arquivo **pibot-web.py**, que irá concentrar as chamadas aos nossos scripts, **controle.py** e **distancia.py**. Esse arquivo já está pronto, você pode baixá-lo aqui (https://s3.amazonaws.com/caelum-online-public/raspberry3/files/pibot-web.py). Ele possui algumas funcionalidades, por exemplo, como ele entrega o a página de controle para o navegador?

No momento que acessamos no navegador o IP do Raspberry Pi, na porta 5000, estamos fazendo uma requisição HTTP do tipo GET em um formulário, dentro do Flask. E esse formulário monta toda a página no navegador para nós. Isso é feito através do seguinte código:

```
@app.route('/', methods=['GET'])
def form():
    return render_template('form.html')
```

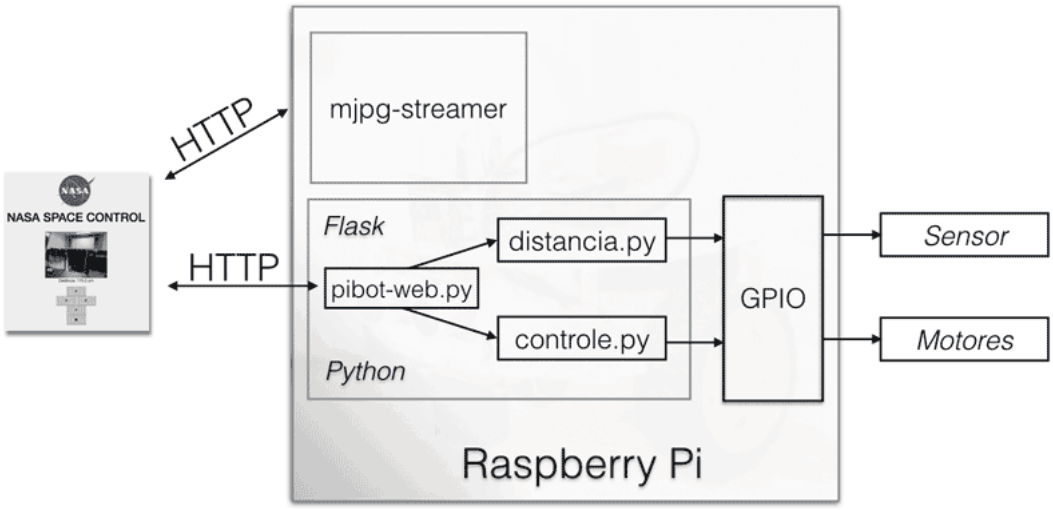
Temos também uma outra função, que faz uma requisição HTTP GET para uma URL específica, nos retornando a distância:

```
@app.route('/distancia', methods=['GET'])
def distancia():
    return str(get_distancia())
```

Falta agora detalhar como é feito o controle do carrinho. Isso é feito através de uma requisição HTTP POST para o próprio formulário. Por baixo dos panos fazemos e verificação das teclas (W, A, S ou D) e chamamos as suas respectivas funções direcionais:

```
@app.route('/', methods=['POST'])
def submit():
    comando = request.form['comando']
    if(comando == 'w'):
        move_frente()
    ...
```

Com isso, todas as chamadas, de controle e distância, ficando reunidas dentro o **pibot-web.py**.



Analisadas todas as peças do nosso projeto, a imagem acima representa bem o seu funcionamento e como o mesmo está estruturado. Com tudo isso entendido, vamos voltar ao nosso carrinho e terminá-lo!