Acionamento de comandos do carro via controle de voz

Projeto Final

Matheus Jericó Palhares
Universidade de Brasília - UnB
Faculdade Gama - FGA
Brasília, Brasil.
matheusjerico1994@hotmail.com

Lucas Raposo Souza Carvalho
Universidade de Brasília - UnB
Faculdade Gama - FGA
Brasília, Brasil.
lucas.raposo1995@gmail.com

Resumo: Dispositivo controlado por voz capaz de acionar faróis, central de música e vidros elétricos, tendo como objetivo atuar no carro.

Palavras chaves: RaspBerry Pi, controle de voz, controle e automação de carro.

I. INTRODUÇÃO

Com o aumento populacional, aumentou-se a quantidade de carros principalmente em grandes centros urbanos, causando grandes congestionamentos, estresse durante o caminho devido à quantidade de carros na pista, entre outros. Nessa linha de raciocínio, nota-se que é necessário cada vez mais prestar no trânsito enquanto dirige e muitas vezes não é possível realizar tal tarefa com excelência, visto que o motorista pode ser distraído devido à necessidade de ligar o farol do carro, aumentar o volume do som ou até escolher uma música infantil para acalmar a criança no banco de trás. Nesses pequenos momentos de distrações podem ocorrer acidentes como batidas e atropelamentos.

Um dispositivo que é controlado por voz (Jasper/Judy) e atua em vários sistemas do carro, resolveria grande parte dos problemas e ainda traz um conforto associado ao luxo extra ao motorista e aos passageiros do automóvel.

Será possível realizar o acionamento dos faróis, central de música e os vidros elétricos, a partir do controle feito por comando de voz.

II. DESENVOLVIMENTO

O desenvolvimento é composto pela descrição de hardware e software:

II. Descrição de hardware:

O Hardware do projeto conta com a Raspberry Pi 3 como recebedor de dados, processamento e atuador para os periféricos. Um microfone já com conversor AD integrado, pois a Raspberry não conta com este recurso, alto falantes para que possa ouvir a interface de voz e as músicas, por fim módulos Relé que recebem sinais do Raspberry para abrir e fechar, atuando cargas 12v do carro como faróis e vidros elétricos. Para o controle dos vidros elétricos, é preciso realizar o controle do motor elétrico que acionada os vidros, para isso, é necessário utilizar uma ponte H, com 4 transistores, capaz que realizar o controle do motor elétrico

O diagrama de blocos e a montagem do circuito são apresentadas na figura abaixo:

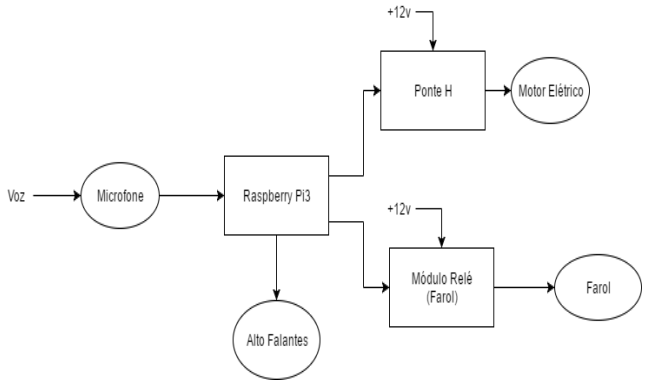

Imagem 1: Diagrama de Blocos Hardware do Projeto.

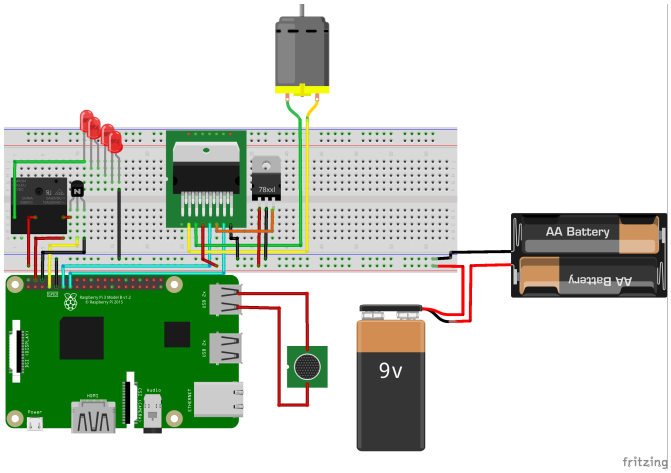

Imagem 2: Montagem do circuito

Para montagem do circuito no Fritzing, necessitou-se de algumas alterações por falta de recurso do programa. A primeira delas foi a falta de uma fonte 12v, necessitando ligar em série uma fonte 9v com uma fonte 3v. A segunda alteração foi a falta de um módulo pronto de relé, logo montou-se um módulo na protoboard (Circuito mais à esquerda da imagem) que contém o relé e o transistor para proteger o pino de saída do RaspberryPi. Também no circuito de acionamento de faróis, não foi encontrado nenhum tipo de carga luminosa que seja alimentada com 12v, logo por representação foi colocado 4 LEDs em série para representar as lâmpadas dos faróis.

No circuito de acionamento do motor DC (Vidros elétricos), o Fritzing não tem na biblioteca um módulo pronto do acionador L298N, logo foi montado um módulo na protoboard (Circuito mais ao centro da imagem). Para deixar a imagem mais limpa, foi feito o acionamento de apenas um motor DC, porém com este mesmo módulo de acionamento da imagem é possível fazer o acionamento de dois Motores DC independentes, indicando o acionamento de dois vidros elétricos.

II. Descrição de software:

O Software do projeto tem como estrutura básica o seguinte diagrama de estado:

Imagem 3: Diagrama de Blocos Software do Projeto.

O software interage com algumas bibliotecas já prontas do Python. As bibliotecas mais importantes utilizadas são “judy”, “RPi.GPIO” e “pygame”.

A biblioteca Judy é a que faz toda a função de gerenciar o reconhecimento de voz do programa, ela utiliza os recursos de conversão TTS (Text-to-Speech) e SST (Speech-to-Text) para poder comunicar com o programa. A bliblioteca Judy armazena o áudio dito pelo usuário, utiliza a ferramenta de SST para converter para uma string e entra na função “handle” criada no programa do projeto caso a palavra dita seja “Judy”.

A biblioteca RPi.GPIO é a biblioteca que permite fazer a mudança de nível lógico dos pinos do Raspberry Pi. No programa são determinados quais pinos são de entrada e quais são de saída pelo comando GPIO.setup e a forma que será padronizado a numeração dos pinos pelo comando GPIO.setmode podendo ser BOARD ou BCM. Caso seja BOARD, o número dos pinos é de acordo com a ordem física deles na placa. Caso seja BCM, o número dos pinos é de acordo com a ordem em GPIO.

A biblioteca pygame é a biblioteca responsável para fazer do Raspberry Pi um reprodutor de áudio. Para iniciar o reprodutor de áudio, usa-se o comando pygame.mixer.init. Para carregar um áudio em buffer, usa-se o comando pygame.mixer.music.load(diretório_arquivo). Para comandos como play, pause, stop entre outros, utiliza-se o comando pygame.mixer.music.comando.

O programa desenvolvido, após declarar todas as bibliotecas e pinos utilizados, é declarado onde a biblioteca Judy irá buscar o vocabulário determinado pelo desenvolvedor utilizando o comando vin. O comando vout é para determinar onde o programa irá buscar os áudios de beep alto e baixo mostrados no diagrama.

A função handle é onde está toda a lógica de comparação de strings. Nela a biblioteca Judy já entregou a string da conversão de voz e ela é comparada com todos os comandos pré-determinados pelo desenvolvedor.

O programa foi desenvolvido em Python e implementado na plataforma Raspberry Pi 3, os comentários estão em vermelhos determinados pela # e foi desenvolvido da seguinte maneira:

Implementação (código)

```
import judy
import time
import RPi.GPIO as GPIO
import pygame
import threading
#Declaração das bibliotecas

GPIO.setmode(GPIO.BOARD)
#GPIO setado como BOARD
GPIO.setup(7, GPIO.OUT)
#Pino do relé de iluminação
GPIO.setup(11, GPIO.OUT)
#Pino 1 de acionamento motor
GPIO.setup(13, GPIO.OUT)
#Pino 2 de acionamento motor
GPIO.setup(15,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
#Pino 1 de botão fim de curso
GPIO.setup(16,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
#Pino 2 de botão fim de curso
```

```
vin = judy.VoiceIn(adcdev='plughw:1,0',
                   lm='/home/pi/Documents/vocabulary/7106.lm',
                   dict='/home/pi/Documents/vocabulary/7106.dic')
              #Declaração de vocabulário
              #utilizado pela Judy
vout = judy.VoiceOut(device='plughw:0,0',
                     resources='/home/pi/Documents/audio')
#Declaração dos beeps
pygame.mixer.init()
#Iniciando reprodutor de áudio
pygame.mixer.music.load("Dont call me white – Nofx.mp3")
#Carregando música em buffer

def handle(phrase):

   print 'Heard:', phrase
    if phrase == 'TURN LIGHTS ON':
        time.sleep(1)
        vout.say("LIGHTS ON")
        GPIO.output(7, GPIO.HIGH)
#Caso esse comando seja
#acionado, aciona a bobina
#do relé, ligando o farol do
#carro


    if phrase == 'TURN LIGHTS OFF':
        time.sleep(1)
        vout.say("LIGHTS OFF")
        GPIO.output(7, GPIO.LOW)
#Caso esse comando seja
#acionado, desliga a bobina
#do relé, desligando o farol do
#carro

    if phrase == 'CLOSE GLASS':
        time.sleep(1)
        if GPIO.input(16) == 1:
            vout.say("GLASS ALREADY CLOSE")

        else:
            vout.say("CLOSE THE GLASS")
            GPIO.output(11, GPIO.LOW)
            GPIO.output(13, GPIO.HIGH)
#Caso esse comando seja
#acionado, analisa se o
#botão fim de curso está
#pressionado, caso esteja não
#muda em nada, caso o botão
#não esteja acionado, o motor
#do vidro elétrico é acionado e
#entra em um loop até o botão
#fim de curso ser pressionado

            while GPIO.input(16) == 0:
                time.sleep(1)
        GPIO.output(13, GPIO.LOW)
#indicando que o vidro chegou
#fim do curso e para o motor

    if phrase == 'OPEN GLASS':
        time.sleep(1)
        if GPIO.input(15) == 1:
            vout.say("GLASS ALREADY OPEN")
        else:
            vout.say("OPEN THE GLASS")
            GPIO.output(11, GPIO.HIGH)
            GPIO.output(13, GPIO.LOW)
#Caso esse comando seja
#acionado, analisa se o
#botão fim de curso está
#pressionado, caso esteja não
#muda em nada, caso o botão
#não esteja acionado, o motor
#do vidro elétrico é acionado e
#entra em um loop até o botão
#fim de curso ser pressionado

            while GPIO.input(15) == 0:
                time.sleep(1)
            GPIO.output(11, GPIO.LOW)
#indicando que o vidro chegou
#fim do curso e para o motor


    if phrase == 'PLAY MUSIC':
        time.sleep(1)
        vout.say("MUSIC ON")
        pygame.mixer.music.play()
#Caso esse comando seja
#acionado, o programa inicia
#a música carregada em buffer

    if phrase == 'STOP MUSIC':
        time.sleep(1)
        vout.say("MUSIC OFF")
        pygame.mixer.music.stop()
#Caso esse comando seja
#acionado, o programa para
#a música que estava iniciada

    if phrase == 'PAUSE MUSIC':
        time.sleep(1)
        vout.say("MUSIC PAUSED")
        pygame.mixer.music.pause()
#Caso esse comando seja
#acionado, o programa pausa
#a música que estava iniciada

    if phrase == 'UNPAUSED MUSIC':
        time.sleep(1)
        vout.say("MUSIC PLAY")
        pygame.mixer.music.unpause()
#Caso esse comando seja
#acionado, o programa volta a
```

```
#tocar a música pausada
#anteriormente


   if phrase == 'HIGHER MUSIC':
      time.sleep(1)
      volume = pygame.mixer.music.get_volume()
      print volume
      time.sleep(1)

      if volume == 1.0:
         time.sleep(1)
         vout.say("ALREADY IN THE HIGHEST")

      else:
         vout.say("CHANGE VOLUME")
         pygame.mixer.music.set_volume (volume+0.3)
#Caso esse comando seja
#acionado, o programa analisa
#qual volume que está tocando
#a música, caso a música esteja
#em volume máximo (1), não
#acontece nada, caso não seja
#volume máximo, incrementa o
#o volume em 0.3

  if phrase == 'LOWER MUSIC':
      time.sleep(1)
      volume = pygame.mixer.music.get_volume()
      print volume
      time.sleep(1)

      if volume == 0.0:
         time.sleep(1)
         vout.say("ALREADY IN THE LOWER")

      else:
         vout.say("CHANGE VOLUME")
         pygame.mixer.music.set_volume (volume-0.3)

judy.listen(vin, vout, handle)

#Caso esse comando seja
#acionado, o programa analisa
#qual volume que está tocando
#a música, caso a música esteja
#em volume mínimo (0), não
#acontece nada, caso não seja
#volume mínimo, decrementa o
#o volume em 0.3
```

Implementação (código)- fim

III. RESULTADOS

Inicialmente, na implementação foi utilizada a biblioteca Jasper, que foi instalada na Raspberry Pi seguindo o tutorial do próprio site da Jasper. Pode-se configurar duas vertentes, Speech-To-Text (STT), que converte o comando de voz para texto, que no caso primeiramente utilizamos o Google API (STT), e podemos configurar o Text-To-Speech (TTS), que é exatamente o oposto do STT.

Após o microfone captar o "comando de voz", ele faz a conversão do sinal analógico para o sinal digital. Dessa forma, a Jasper envia o "comando de voz" para a Google API via internet que é responsável por fazer a conversão do áudio para texto, quando concluído, é mandado de volta o texto convertido.

Após ter o áudio convertido para texto, o mesmo é comparado com as strings que foram adicionadas no arquivo "commands.conf", a Jasper faz a comparação do texto convertido com a string.

Conforme o projeto foi sendo implementado, observou-se a necessidade de realizar a troca do Controle de voz Jasper pelo Controle de voz Judy, para que as ferramentas de conversão texto/áudio fizessem a conversão sem utilizar a internet.

Dessa forma, foram configuradas as duas vertentes Speech-To-Text (STT) e Text-To-Speech (TTS), que não utilizam o artefato da internet para fazer a conversão. O STT é feito pelo Pocketsinx e o TTS é realizado pela Pico; toda conversão é feita offline.

Com a implementação do TTS, a RaspBerry Pi se comunica com o usuário, dessa forma, é possível ter uma maior interação entre máquina e usuário.

O software criado é feito em python, importando a biblioteca Judy no início do programa.

O programa faz a comparação do que foi falado no microfone com as strings definidas ("TURN LIGHTS ON", PLAY MUSIC", etc) se a comparação for verdadeira, a RPi utiliza o TTS para se comunicar com o usuário, emitindo a string desejada, logo após é realizado o acionamento do pino de saída.

Foi utilizada a biblioteca pygame, que emula uma central multimídia, dessa forma é possível realizar o acionamento da musica e o pause, além de realizar o controle do volume.

Para melhorar a acurácia do reconhecimento de voz, é criado um vocabulário de palavras onde o programa se atenta somente a essas palavras (arquivos .lm e .dic). As palavras são colocadas em um arquivo .txt que é enviado para uma universidade no EUA, onde é feita a conversão do arquivo .txt para os arquivos .lm e .dic, que são enviados de volta para RPi.

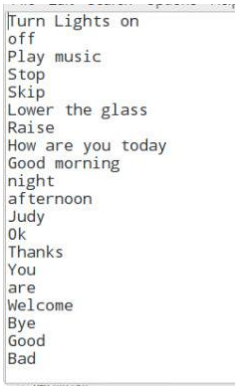

Imagem 5: Arquivo .txt utilizado para confecção dos arquivos .lm e .dic.

O código em python que foi criado para implementação, utiliza os arquivos .lm e .dic para poder limitar o vocabulário da RaspBerry Pi, e possui dois áudios de BIP como saída para indicar que está pronto para ouvir e que entendeu o que foi falado.

Ao final da implementação, o sistema projetado para ser acionado com controle de voz está completo. Realizando o controle (ligar e desligar) dos faróis, controle (subir e descer) vidros elétricos e controle da central de música, podendo dar play, pause, stop, além de controlar o volume.

Para que o sistema fique completo, é necessário realizar a integração da RaspBerry Pi com a central multimídia do automóvel.

IV. CONCLUSÃO

Foi possível realizar a configuração da RaspBerry para atuar como receptor de dados, analisador e atuador de periféricos, utilizando bibliotecas já implementadas como a Jasper/Judy, pygame e Rpi.GPIO. Dessa forma, foi possível realizar o controle de voz com o objetivo de realizar acionamentos de periféricos, como faróis, vidros elétricos e reprodutor de áudio.

A integração entre as bibliotecas é de extrema importância, pois se as mesmas não estão configuradas para trabalhar de forma integrada, não é possível obter êxito.

A implementação do projeto sem utilizar-se as bibliotecas (Jasper/Judy, pygame e Rpi.GPIO) é tecnicamente inviável, pois é de extrema complexidade.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Jasper Project, configuração Jasper, disponível em:
<www.jasperproject.github.io/documentation> Acesso em 30 de março de 2016.

[2] RaspBerry PI, controle de voz utilizando Siri, disponível em:
<www.raspberrypi.org> Acesso em 30 de março de 2016

[3] Mitchell, M., Oldham, J. & Samuel, A., Advanced Linux Programming, Editora: Newriders, 2001.

[4] Source Forge, siri proxy, disponível em:
<https://sourceforge.net/p/siriproxyrpi/wiki/Home/> Acesso em 30 de março de 2016.

[5] Judy Project, configuração Judy, disponivel em:
<https://github.com/nickoala/judy> Acesso em 01 de junho de 2017.