UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Projeto de Graduação

Desenvolvimento de um sistema de monitoramento doméstico

Matheus Comper Zanetti

VITÓRIA – ES

DEZEMBRO/2018

Matheus Comper Zanetti

Desenvolvimento de um sistema de monitoramento doméstico

Parte manuscrita de Projeto de Graduação do aluno **Matheus Comper Zanetti**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito para obtenção do grau de Engenheiro Eletricista.

Orientador: Prof. D. Sc. Oureste Elias Batista

VITÓRIA – ES

DEZEMBRO/2018

Matheus Comper Zanetti

Desenvolvimento de um sistema de monitoramento doméstico

Parte manuscrita de Projeto de Graduação do aluno **Matheus Comper Zanetti**, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito para obtenção do grau de Engenheiro Eletricista.

Aprovada em 05 de dezembro de 2018

COMISSÃO EXAMINADORA:

Prof. D. Sc. Oureste Elias Batista
Universidade Federal do Espírito Santo
Orientador

M. Sc. Daniel Carletti
Universidade Federal do Espírito Santo
Examinador

M. Sc. Luiz Guilherme Riva Tonini
Universidade Federal do Espírito Santo
Examinador

À minha família, por me incentivar e acreditar em mim. Mãe, seu cuidado e dedicação foi o que, sempre, me deram esperança pra seguir.

Agradecimentos

Agradeço primeiramente à minha mãe Marilda, por todo amor, carinho e apoio que me deu em toda a minha vida. Sou grato por todos os sacrifícios realizados para que eu sempre pudesse alcançar meus objetivos.

Agradeço também a todos os familiares próximos, avós, tios e primos, que em diversas etapas da minha vida foram essenciais para que eu seguisse caminhando, mesmo em frente a diversas dificuldades.

Um agradecimento especial aos amigos e colegas da faculdade, que me ensinaram o valor da amizade e companheirismo, para vencer os desafios aos quais somos postos pela vida.

Aos companheiros de trabalho da Intelliway, Evando, Fred, César, Tiago, Funay e Storck, pela perseverança e aprendizado ao longo dos projetos trabalhando juntos. Sem vocês, a concepção deste trabalho não seria possível.

Finalmente, um agradecimento especial ao meu orientador, Oureste, por toda paciência e boa vontade em ajudar, ao longo do desenvolvimento desse projeto.

A evolução do homem passa, necessariamente, pela busca do conhecimento.

(Sun Tzu)

Resumo

Neste trabalho está proposto o modelo de desenvolvimento para um sistema de monitoramento doméstico, utilizando o microcontrolador Raspberry Pi, sensores de presença ultrassônicos HC-SR04, uma câmera V2 8 Mega-Pixels e o aplicativo Telegram. Toda a infraestrutura de controle, bem como o servidor web, será gerenciado pelo Raspberry Pi, utilizando a linguagem de programação *Python*. Com isso, é criado um sistema que apresenta maior simplicidade e redução de custos, comparado a sistemas de monitoramento tradicionais.

Palavras-chave: Automação Residencial. Raspberry Pi. Python. Telegram. Inteligência Artificial.

Abstract

This work proposes the development model for a home monitoring system, using Raspberry Pi, an ultrasonic presence sensors HC-SR04, a camera and the Telegram application. All the control infrastructure, as well as the web server, will be managed by Raspberry Pi, using the Python programming language. With this, its created a system that is simpler and reduces costs compared to traditional monitoring systems.

Keywords: Residential Automation. Raspberry Pi. Python. Telegram. Artificial Intelligence. Chatbot.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ARM	Advanced RISC Machine
API	Application Programming Interface
FAQ	Frequently Asked Questions
GPIO	General Purpose Input Output
NLTK	Natural Language Toolkit
SoC	System On Chip
USB	Universal Serial Bus

Sumário

1 Introdução

1.1 Apresentação

A domótica é definida como implementação de tecnologias capazes de prover o gerenciamento dos diversos dispositivos presentes em um ambiente residencial (SGARBI; TONIDANDEL, 2006).

O conceito de automação residencial vem há décadas deixando de ser apenas um conceito distante e está se tornando cada vez mais próximo do cotidiano das pessoas.

Com o avanço da tecnologia e expansão do uso da *Web* nos anos 90 surgiram também tecnologias de computação doméstica a baixo custo que inspiraram os pesquisadores da domótica a criarem ferramentas, sistemas, eletrodomésticos inteligentes e aparelhos eletrônicos capazes de interagirem uns com os outros através de computadores.

O avanço da tecnologia tem permitido a criação de sistemas cada vez mais independentes e autônomos, dentre eles destaca-se o surgimento dos chatbots (também conhecido como chatterbots).

No Brasil já existem diversas empresas fornecendo serviços e produtos para residências, porém a maioria apresenta um elevado custo de aquisição, não sendo acessível para uma parcela expressiva da população brasileira.

A internet passou a ser utilizada em inúmeras soluções propostas pelo mercado, pois observou-se nela inúmeras possibilidades. Dentre elas pode-se citar o monitoramento das residências à distância, através de câmeras que enviam suas imagens para um servidor *web* permitindo que o usuário visualize sua casa de qualquer lugar no mundo, desde que haja um computador com acesso à internet.

Existem também sistemas de alarme no mercado que são capazes de enviar alertas ao morador e acionar a polícia automaticamente quando uma residência é invadida. Portanto a automação residencial combinada à internet deixou de ser apenas um meio de aumentar a comodidade de seus moradores e passou a atuar como um instrumento de auxílio na segurança das moradias (PEREIRA et al., 2014).

1.2 Justificativa

A intenção deste trabalho é desenvolver um sistema de monitoramento doméstico simples, que utiliza como principal elemento um microcontrolador Raspberry Pi, sensores de ultrassom e uma câmera de resolução 8 Mega-Pixels. A interação com o usuário final é

feita através de um popular aplicativo de comunicação atualmente, o Telegram.

Nesta proposta, o sistema é capaz de monitorar a movimentação de um ambiente, entrada e saída de pessoas de uma sala, fotografando e monitorando a ação em tempo real. A aplicação apresentada opera mediante comandos de entrada e saída, diretamente pelo Telegram.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo principal deste trabalho é elaborar uma arquitetura para um sistema de monitoramento doméstico, que permita o usuário receber informações de um ambiente em tempo real. Este sistema deve ser capaz de monitorar um ambiente específico, através de sensores ultrassom e, caso seja detectada alguma movimentação, o ambiente deve ser fotografado imediatamente. Esta imagem capturada deve ser enviada para uma caixa de mensagens do usuário, através do aplicativo Telegram, informando o horário da detecção da movimentação, e enviando em seguida a imagem capturada naquele momento.

Antecipando-se a versões futuras com mais funcionalidades, também está disposto um Led conectado ao sistema, para simbolizar sinais de comando como controle de fechaduras ou servomotores, caso o usuário queira aumentar as funcionalidades do dispositivo.

1.3.2 Objetivos Específicos

Os objetivos específicos são:

- Desenvolvimento do chatbot em Python;
- Integração da Câmera com o Raspberry;
- Integração do Led com o Raspberry;
- Integração do Telegram com o Raspberry;
- Estudo de tecnologias de Inteligência Artificial.

2 Fundamentação Teórica

2.1 Automação Residencial (Domótica)

O termo "Domótica" é a junção da palavra latina *"Domus"* (casa) com *"Robótica"*, vem sendo bastante utilizada atualmente, seja pelos aspectos de segurança, ou mesmo de conforto. Esse conceito surgiu na França, onde foram construídos os primeiros edifícios automatizados, nos anos 80., em que pretendia-se controlar a iluminação, climatização, a segurança, interligando esses elementos (TAKIUCHI; MELO; TONIDANDEL, 2004).

O conceito de automação residencial tem por finalidade remover a interação humana tanto quanto possível de atividades domésticas rotineiras, seja ela em tarefas simples, como abrir ou fechar cortinas, ou seja ela em atividades mais complexas, como controlar a temperatura de um forno elétrico ou monitorar a presença de pessoas em determinados ambientes.

Tradicionalmente, são definidas dois tipos de arquitetura em domótica:

- ABA (Arquitetura Baseada em Automação) - conhecida como domótica estática, trata a automação de residências a partir de dispositivos, como controles remotos, sensores de movimento, dispositivos biométricos;
- ABC (Automação Baseada em Comportamento) - conhecida como domótica inteligente, que faz uso de técnicas de I.A. (Inteligência Artificial) para se adaptar ao comportamento humano.

Neste trabalho, será tratada da arquitetura ABC.

2.2 Sistemas Embarcados

Um sistema embarcado tem a função de implementar uma capacidade computacional dentro de um circuito integrado, por exemplo.

Os sistemas computacionais embarcados possuem um baixo custo tecnológico e por essa característica estão cada vez mais presentes no dia a dia da população em dispositivos como celulares, sistemas de controle de automóveis e eletrodomésticos, por exemplo.

Existem diversas arquiteturas para sistemas embarcados, cada uma com suas características que definem para quais projetos elas são melhores aplicáveis. Arquiteturas de 8 bits são mais utilizadas para atividades mais simples, como controle de eletrodomésticos como um liquidificador.

O surgimento dos sistemas embarcados começou no final da década de 60, com as novas necessidades do mercado e associada com os avanços tecnológicos da microeletrônica, grandes empresas do ramo perceberam um mercado promissor nessa tecnologia, por sua vez, otimizando os componentes eletrônicos para esse segmento.

Atualmente, os sistemas embarcados são a categoria de sistemas processados de maior utilização no mundo, superando em número os computadores, notebooks, servidores e semelhantes.

2.3 Raspberry Pi

O Raspberry Pi é um microcontrolador que possui o tamanho pouco maior de um cartão de crédito, foi desenvolvido pelo engenheiro britânico Eben Upton e sua equipe, então foi criada a Fundação Raspberry Pi. O intuito desse projeto é fornecer um meio de tecnologia de baixo custo, sendo prático e acessível para que pessoas de diferentes idades pudessem aprender e desenvolver programas de modo mais fácil e intuitivo (RASPBERRY. . . ,).

O objetivo inicial do projeto era desenvolver e comercializar um computador de placa única (SoC), de tamanho reduzido e de fácil aquisição pelos interessados no assunto. Apesar do tamanho enxuto e das características pouco convencionais, o Raspberry Pi é um computador como outro qualquer. Dependendo de sua finalidade, pode até ser utilizado como um computador de uso pessoal, tendo os mesmos recursos de computador convencional, como navegadores de internet, reprodução de conteúdo multimídia, etc.

Os primeiros conceitos do Raspberry Pi surgiram em 2006 por Upton e sua equipe, mas só em agosto de 2011 que os primeiros componentes começaram a ser distribuídos para o público em geral.

Para este projeto, foi utilizado o modelo com mais recursos disponíveis atualmente, o Raspberry Pi Modelo B+. Este modelo foi escolhido para a elaboração do projeto devido às inúmeras características, incluindo grande poder de processamento, velocidade na conexão com internet e ótimo custo-benefício.

A alimentação para o modelo B do kit Raspberry Pi deve ser de 5 V com uma fonte que forneça até o limite de 700 mA através de uma entrada micro USB. Tais especificações são facilmente encontradas em carregadores de smartphones.

2.4 Chatbot

O termo chatbot vem do inglês em que *chat* significa “conversador” e bot é uma abreviação para *robot* que significa robô. Alguns autores (FERREIRA, 2008) definem chatbots como sistemas computacionais que simulam o comportamento humano em

conversas, e que são capazes de analisar, interpretar e responder perguntas.

Atualmente, existem duas classificações para um chatbot: os de utilidade e os dirigidos a conteúdo. Os chatbots de utilidade são capazes de realizar alguma tarefa quando são solicitados. Já os chatbots dirigidos à conteúdo, possuem o objetivo de seguir um roteiro e prover conteúdo aos seus usuários. A assistente da Apple, a Siri e a assistente da Microsoft, Cortana, podem desempenhar ambas funções, respondendo sobre condições climáticas ou ativando o alarme do celular quando solicitadas, por exemplo.

Um chatbot dirigido a conteúdo normalmente tem o foco nas conversas com o usuário. Por outro lado, um chatbot dirigido a conteúdo é muito utilizado em tarefas rotineiras, como ajudar os usuários a realizar tarefas como comprar bilhetes, obter informações específicas sobre um produto, etc.

Para este trabalho, o chatbot desenvolvido possui as funções características de um bot dirigido a conteúdo.

2.5 Telegram

O Telegram é um dos aplicativos para troca de mensagens com a maior quantidade de usuários ativos no mundo atualmente. Ele apresenta funções semelhantes às dos demais nomes do gênero, permitindo envio e recebimento de conteúdos em texto, vídeo, áudio e imagem por meio de um pacote de dados ou de uma conexão *Wi-Fi*.

Um fator de destaque da plataforma é o fato do sistema possuir uma API pública, "Telegram API", o que permite o desenvolvimento de novos clientes e sistemas em diversos tipos de dispositivos.

Além do recurso de envio de imagens citado acima, uma outra funcionalidade destaca o Telegram de seus concorrentes, e por isso foi a plataforma de escolha como ponto de comunicação com o usuário neste trabalho: é possível utilizar o programa em vários terminais diferentes ao mesmo tempo, como smartphones, computadores ou tablets, bastando que os dispositivos estejam com uma conexão ativa à internet.

2.6 Python

O Python é uma linguagem de programação de alto nível interpretada e orientada a objetos. Foi lançada por Guido van Rossum em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation (PYTHON. . . ,).

Com esta tecnologia é possível escrever desde pequenos scripts aritméticos até sistemas Web completos. Devido seu ambiente de código aberto, o Python possui uma

forte comunidade de desenvolvedores entusiastas, que diariamente criam uma vasta gama de bibliotecas, capazes de executar diversas funções (PILGRIM, 2004).

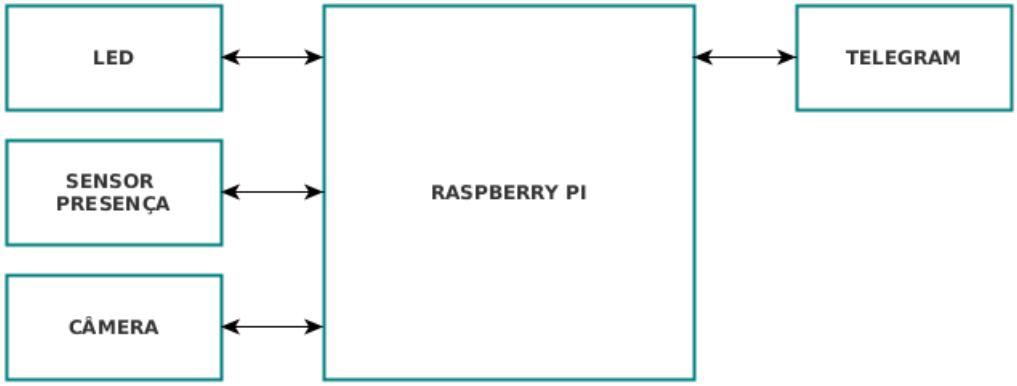
Esta linguagem tem uma forte ligação com esta plataforma Raspberry, derivando dela o termo “Pi” em seu título.

3 Implementação e Desenvolvimento

Neste capítulo serão apresentadas as etapas de desenvolvimento desse projeto. O desenvolvimento inclui um projeto com base eletrônica e um script computacional que permite o monitoramento do ambiente, envio e recebimento de mensagens, captura de imagens e acionamento de Led, indicando dispositivos genéricos.

A arquitetura básica do sistema está sintetizada na Figura 1. No centro, está o Raspberry Pi, que lida com toda a distribuição de informações. A princípio, o sistema está alocado em um ambiente isolado. Caso haja alguma movimentação, o sensor de ultrassom irá detectar a presença, e será enviado um comando para a câmera para a captura de imagem. O usuário é então notificado pelo Telegram, e em seguida é enviada a imagem capturada. Através do chatbot, o usuário pode enviar um comando para o Raspberry Pi acender ou apagar o LED, manifestando a possibilidade de um controle de um mecanismo de fechadura fictício.

Figura 1 – Arquitetura do projeto



Fonte: O autor

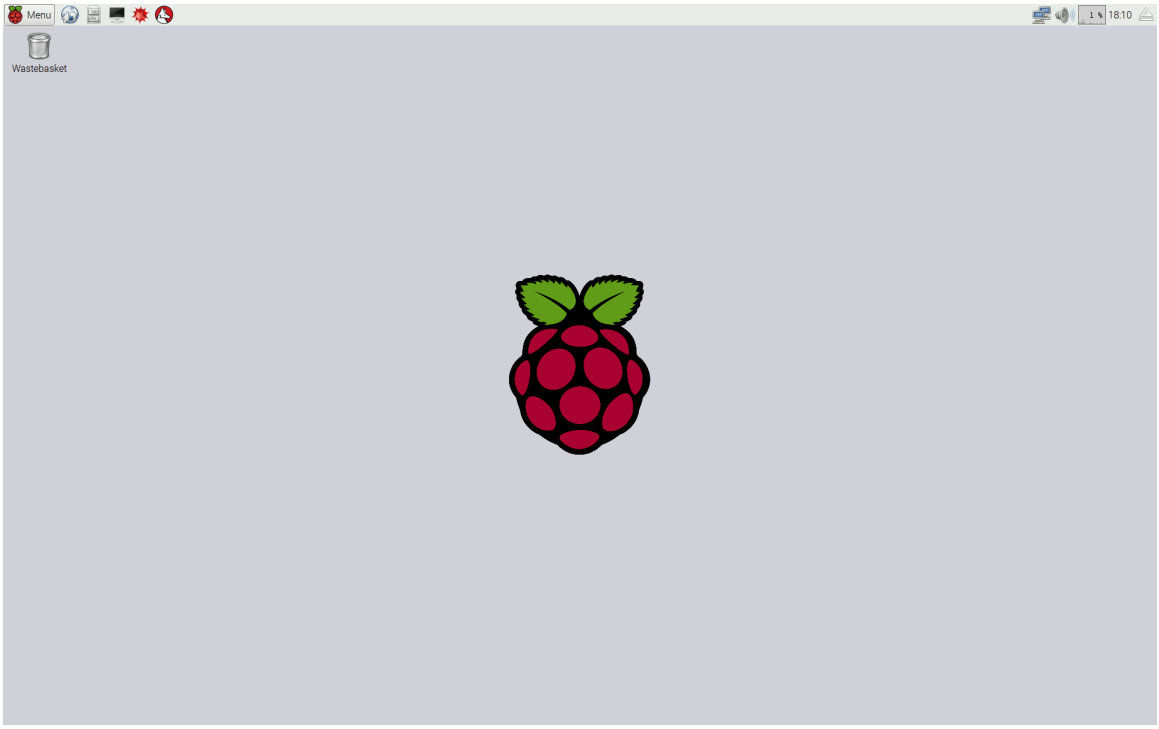
3.1 Configuração do Raspberry Pi

O primeiro passo para o desenvolvimento do projeto é configurar o ambiente de desenvolvimento do Raspberry Pi, para conectá-lo aos sensores e poder monitorar o status das mensagens do Telegram.

Para isso, foi utilizado uma distribuição Linux específica, Raspbian, uma variante do Debian baseada em ARM, otimizada para o conjunto de instruções ARMv6 do hardware

do Raspberry Pi. O sistema operacional foi escolhido por apresentar um grande suporte de outros usuários, frequentes atualizações para a correção de bugs e pela simplicidade de instalação. A tela de trabalho inicial do Raspbian é exibida na Figura 2.

Figura 2 – Área de Trabalho do Raspbian



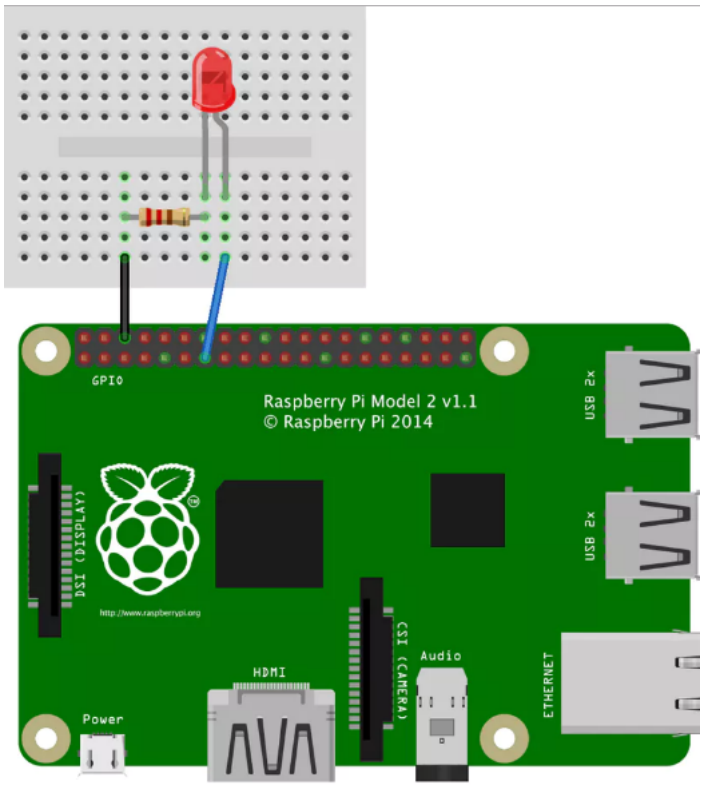
Fonte: O autor

Após a configuração inicial do ambiente, foi feita a conexão do Led com as portas de GPIO do Raspberry. Nesta interface, é possível conectar dispositivos externos como sensores, atuadores, cabos para conexão serial, etc. A configuração do GPIO é composta de 26 pinos distribuídos entre Terra, Entrada/Saída, sendo todas digitais, tensão de 3,3 Volts, tensão de 5 Volts, PWM, RX, TX, MOSI, MISO, SCK, SDA e SCL.

Para conectar o Led ao Raspberry Pi é necessário um resistor de 221 Ohm, para manter um valor de corrente adequado. Um terminal do Led é conectado diretamente ao pino de GPIO, que irá fazer o controle de acionamento. A Figura 3 apresenta um esquema de montagem sugerido.

Para configurar o Raspberry Pi inicialmente foram instaladas as bibliotecas WiringPi3, que permite ao usuário gerenciar com facilidade os pinos de entrada e saída da Raspberry Pi, Python-pip que permite o sistema interpretar arquivos na linguagem Python e Telepot, que permite enviar e receber mensagens para o Telegram via API.

Figura 3 – Conexão do LED com portas GPIO



Fonte: O autor

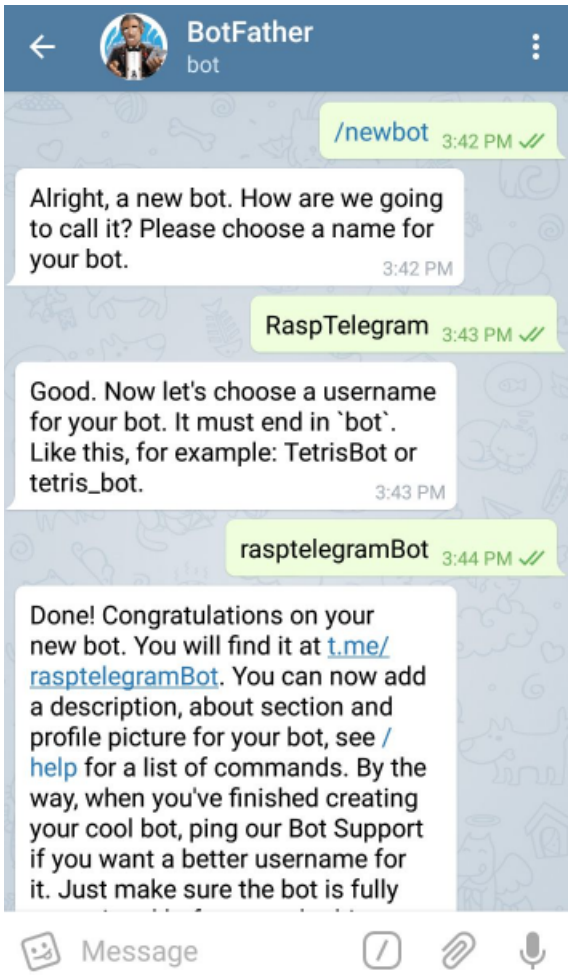
3.2 Configuração do Telegram

Para começar a trocar mensagens de texto simples a partir da linha de comando via terminal, é necessário conectar o Raspberry Pi à sua conta do Telegram e associá-la ao número do seu celular.

Inicialmente, é necessário ter um dispositivo móvel com o aplicativo instalado. Após isso, é preciso usar um bot do Telegram para criar o chatbot que será utilizado como canal de comunicação final.

O processo de gerar um chatbot privado é feito através do próprio Telegram, através de um bot chamado BotFather, como ilustrado por meio da Figura 4.

Figura 4 – Etapa de criação de chatbot interno do Telegram



Fonte: O autor

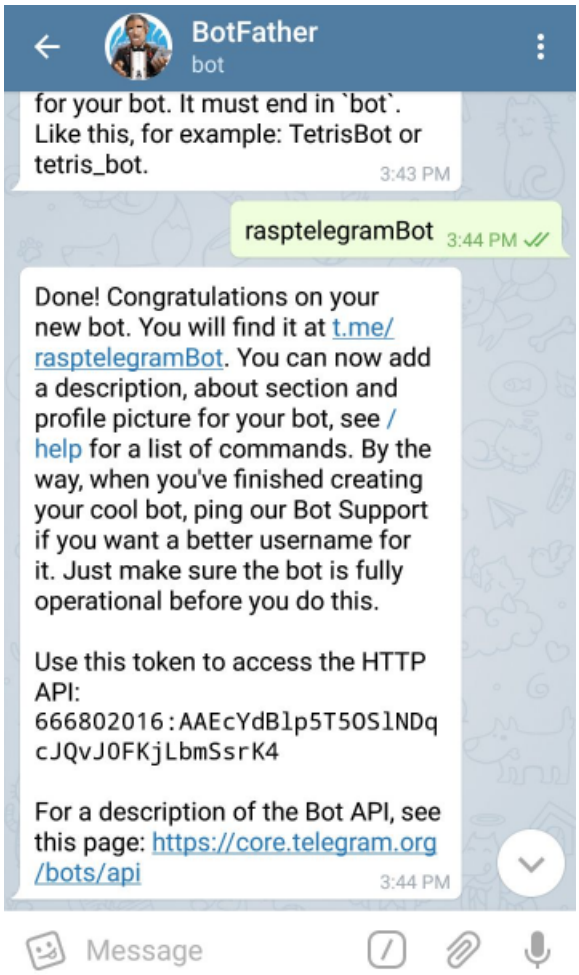
Após iniciar a conversa com o BotFather o usuário precisa enviar alguns comandos básicos, como o nome do chatbot e username de identificação. Ao final do processo, o chatbot já está ativo no sistema do Telegram, e é informado uma chave de acesso específico. Este token funciona como um identificador para o chatbot, pois é através dele que o Raspberry Pi irá enviar os dados de mensagem, controle e texto, via API, para o aplicativo. Um exemplo do token gerado é visto na Figura 5.

3.3 Configuração do Servidor Python

Uma vez definida a arquitetura de software, bem como as tecnologias utilizadas, a etapa de implementação consiste em codificar o software. Essa etapa contém o ambiente de desenvolvimento com ferramentas de codificar.

Para utilizar a API do Telegram, é necessário instalar um pacote Python chamado

Figura 5 – Token gerado pelo Telegram



Fonte: o autor

Telepot . Isso pode ser feito via comando no terminal do Raspbian:

sudo apt-get install python-pip

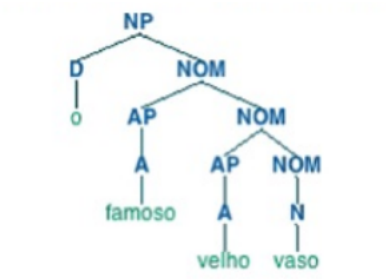
sudo pip install telepot

O componente mais importante de um chatbot consiste na sua engine. A engine é responsável por transformar linguagem natural em uma ação entendível por máquinas. As engines de chatbots geralmente são desenvolvidas utilizando-se vários modelos de Processamento de Linguagem Natural e Aprendizado de Máquina para prover níveis aceitáveis de precisão (KAR; HALDAR, 2016).

Uma das ferramentas mais utilizadas para processamento de linguagem natural é o NLTK (NATURAL. . . ,), que foi desenvolvido em Python e apresenta uma gama muito grande de recursos, como: classificação, tokenização, stemming, tagging, parsing e raciocínio semântico.

Todas essas funções são utilizadas para análise dos textos, que extrai essas informações e as organiza em estruturas hereditárias, similares a árvores genealógicas. Essas árvores apresentam as classes de cada palavra, ou seja, se uma palavra é uma preposição, artigo, verbo, tempo verbal e consegue identificar até nomes próprios. A Figura 6 exibe um modelo de classificação semântica via NLTK. Por meio da identificação de algumas dessas categorias, em combinação com outras técnicas, é possível identificar a semântica das frases, semelhante aos exemplos de frases citados anteriormente.

Figura 6 – Exemplo de análise sintática do NLTK



Fonte: O autor

A implementação do servidor do chatbot também foi feita via Python. Enquanto o sistema está ativo, o chatbot fica em constante aguardo por mensagens de comando do usuário e por algum acionamento do sensor de presença.

Caso o sensor de presença identifique alguma movimentação no ambiente, ele irá acionar a câmera, para efetuar um disparo, e enviar a mensagem ao chatbot no Telegram. A Figura 7 mostra a implementação desta verificação.

Figura 7 – Função de checagem do estado do sensor

```
if self.config['motion']['enable']:
    # Checa se o sensor está ativo
    if not self.isMotionRunning():
        message.reply_text('Erro: Câmera está ligada mas há erro de software desconhecido!')
        return
    message.reply_text('Todos os dispositivos estão funcionando corretamente.')
else:
    message.reply_text('Câmera ligada!')
```

Fonte: O autor

Caso o usuário deseje saber o estado do sistema, se a câmera está funcionando, ou deseja ver em tempo real como está o cômodo monitorado, é possível realizar uma série de ações, via chatbot.

Para isso, é necessário programar cada comando pretendido, para que o sistema responda adequadamente a cada informação fornecida pelo usuário. Neste projeto, foram definidos os comandos através da Tabela 1.

Tabela 1 – Tabela de comandos do Telegram

Comando	Resposta via Python	Ação
/start	"Iniciando monitoramento"	Habilita o funcionamento do chatbot
/arm	"Habilitando dispositivo de captura"	Prepara a câmera para tirar uma foto, se necessário
/disarm	"Desabilitando dispositivo de captura"	Desliga a câmera
/kill	"Finalizando conexão"	Encerra a conexão com o Telegram e desativa os dispositivos
/status	"Todos os dispositivos estão funcionando corretamente"	Verifica o status da câmera e a conexão com o Telegram
/capture	"Captura em progresso"	Tira uma foto com a câmera e envia pelo Telegram
/ledon	"Ativando Led"	Acende o Led
/ledoff	"Desativando Led"	Apaga o Led

Fonte: O autor

A estrutura para organização dos comandos foi feita via NLTK, como visto na Figura 8.

Figura 8 – Função de checagem do estado do sensor

```
def performCommand(self, message):
    cmd = message.text.lower().rstrip()
    if cmd == '/start':
        # ignora comando default start do chatbot
        return
    if cmd == '/arm':
        self.commandArm(message)
    elif cmd == '/disarm':
        self.commandDisarm(message)
    elif cmd == 'kill':
        self.commandKill(message)
    elif cmd == '/status':
        self.commandStatus(message)
    elif cmd == '/capture':
        # caso o software esteja rodando, envia o comando para disparar a câmera
        stopStart = self.isMotionRunning()
        if stopStart:
            self.commandDisarm(message)
        self.commandCapture(message)
        if stopStart:
            self.commandArm(message)
    else:
        self.logger.warn('Comando desconhecido: "%s"' % message.text)
```

Fonte: O autor

Além das ações padrão para o funcionamento do sistema, o controle de usuários também é feito via script Python. Isso é necessário para que o bot responda apenas a um usuário específico. A identificação para o projeto é vista na Figura 9.

Figura 9 – Função de checagem do estado do sensor

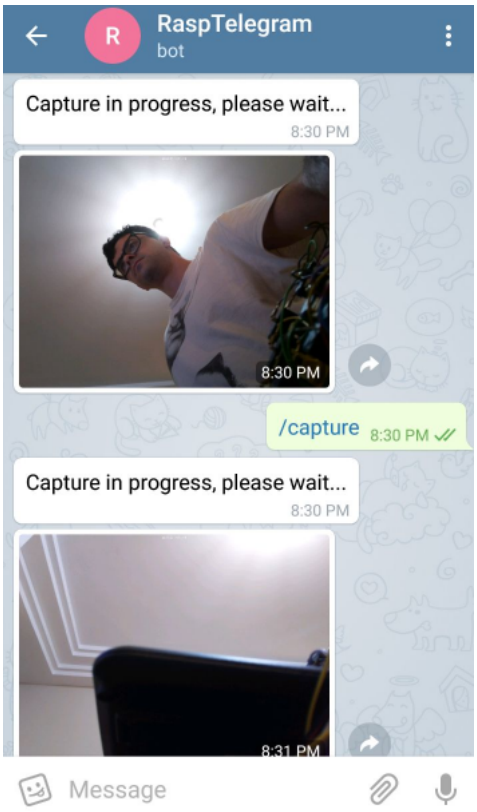
```
"//": "Configurações relacionadas ao Telegram",
"telegram": {
    "//": "Token de Autorização ao Chatbot RaspTelegram",
    "token": "666802016:AAEcYdBlp5T5OSlNDqcJQvJ0FKjLbmSsrK4",

    "//": "ID do usuario",
    "owner_ids": [ 200819131 ]
},
```

Fonte: O autor

4 Resultados

O presente trabalho visou a criação de um sistema de monitoramento doméstico, utilizando Raspberry Pi, sensores de presença ultrassônicos HC-SR04, uma câmera e o aplicativo Telegram. Após a conclusão da etapa de validação do chatbot, foi possível testar todas as funcionalidades do mesmo de maneira satisfatória, como exibido na Figura 10. Os critérios para a seleção dessas ferramentas foram basicamente, custo, eficácia, simplicidade e o quanto determinada ferramenta permitiria a exploração e implementação de técnicas que estão na fronteira atual da tecnologia.

Figura 10 – Demonstração prática do projeto



Fonte: O autor

O Telegram foi inicialmente escolhido por se mostrar uma ferramenta mais aberta do que seus concorrentes, com destaque para sua API clara e bem documentada. Durante o desenvolvimento deste projeto, sua escolha foi confirmada como acertada, visto que a integração foi feita de maneira rápida e eficaz, atendendo inteiramente aos objetivos iniciais.

Com toda a arquitetura implementada pode-se comprovar o bom funcionamento

do sistema. O protótipo básico do projeto é exibido na Figura 11. Entre a captura da imagem e o recebimento da mensagem pelo Telegram é observado um tempo de espera da ordem de 2 a 5 segundos. Esse atraso está relacionado a dois fatores principais: a etapa de gravar a imagem no cartão SD do Raspberry Pi e o tráfego da informação via internet. Apesar de perceptível pelo usuário, este atraso não compromete o uso do sistema.

Figura 11 – Protótipo do projeto

Fonte: O autor

Outra característica importante da utilização do Raspberry Pi como servidor é o baixo consumo de energia elétrica. Foram avaliados o consumo de energia do sistema implantado com o Raspberry Pi e o de um sistema de monitoramento tradicional, e a solução sugerida neste projeto apresenta uma economia significativa. A Tabela 2 apresenta um comparativo do consumo de energia para um sistema tradicional, baseado em soluções comerciais encontradas no mercado, e o implantado neste projeto. Foram considerados 30 dias de uso, durante 24 horas por dia.

Tabela 2 – Comparativo de consumo de energia elétrica

Equipamento	Potência [W]	Dias de Uso	Tempo de Uso [h]	Total [kW/mês]
Sistema Tradicional	200	30	24	144
Raspberry Pi	2,5	30	24	1,8

Fonte: o autor

O código fonte em python pode ser acessado na plataforma GitHub através do link.

5 Conclusões

Com o avanço da tecnologia e o avanço dos estudos na área de Inteligência Computacional, os chatbots estão cada vez mais presentes no dia a dia das pessoas, sendo em forma de Serviço de Atendimento ao Consumidor, em forma de FAQ ou até em formas mais avançadas, como os assistentes pessoais em celulares e dispositivos caseiros, como os produtos Google Home e Apple HomePod.

Devido a tendências de mercado e ao desenvolvimento acelerado das tecnologias de Inteligência Artificial e Machine Learning, o Raspberry Pi e a linguagem Python se tornaram plataformas extremamente populares, existindo uma imensa variedade de ferramentas disponíveis para essas plataformas. Contudo, a dificuldade associada a esse fato é que nem toda a informação existente está organizada de maneira didática e consistente, de onde surgem os problemas intrínsecos ao excesso de informação. Portanto para elaborar este projeto foi necessário definir quais os escopos das necessidades do projeto, pesquisar as ferramentas já existentes, analisar cada uma delas para só então escolher a mais simples e eficaz de ser implementada.

É importante ressaltar que a escolha do cartão SD têm influência direta no desempenho geral do sistema. A velocidade de leitura e gravação é proporcional à categoria de classificação. Desta forma, o ideal é escolher os modelos de classes de especificações mais elevadas. Para trabalhos futuros, sugere-se que seja aumentada a gama de dispositivos conectados, como um buzzer para emissão de alarme sonoro em casos que seja detectada a intrusão do ambiente. Dado o avanço da tecnologia de análise de imagens, estima-se ser possível utilizar mecanismos de reconhecimento facial para abertura de portas ou autorização de entrada.

Sugere-se também, estudar uma maneira de compartilhar vídeos através da API do Telegram, para obter um nível de segurança mais elevado.

Referências

FERREIRA, L. P. Desenvolvimento de um chatbot para auxiliar o ensino de espanhol como lıngua estrangeira. 2008. Citado na página 26.

KAR, R.; HALDAR, R. Applying chatbots to the internet of things: Opportunities and architectural elements. *arXiv preprint arXiv:1611.03799*, 2016. Citado na página 33.

NATURAL Language Toolkit. <http://www.nltk.org/>. Accessed: 2018-11-15. Citado na página 33.

PEREIRA, E. H. H. et al. *Soluções inteligentes e de baixo custo para a automação residencial utilizando smartphones.* Tese (Doutorado) — UNIVERSIDADE DE SÃO PAULO, 2014. Citado na página 23.

PILGRIM, M. *Mergulhando no Python.* [S.l.]: Rio de Janeiro: Alta Books, 2004. Citado na página 28.

PYTHON Foundation. <https://www.python.org/psf/>. Accessed: 2018-11-15. Citado na página 27.

RASPBERRY Pi Foundation. <https://www.raspberrypi.org/about/>. Accessed: 2018-11-15. Citado na página 26.

SGARBI, J. A.; TONIDANDEL, F. Domótica inteligente: Automação residencial baseada em comportamento. In: *Workshop de Teses e Dissertações em Inteligência Artificial, Ribeirão Preto, São Paulo.* [S.l.: s.n.], 2006. Citado na página 23.

TAKIUCHI, M.; MELO, É.; TONIDANDEL, F. Domótica inteligente: automação baseada em comportamento. In: *CBA 2004-XV CONGRESSO BRASILEIRO DE AUTOMÁTICA, Gramado-RS.* [S.l.: s.n.], 2004. Citado na página 25.

Anexos

ANEXO A – Código Python

```
import importlib
import inotify.adapters
import json
import logging
import logging.handlers
import os
import shlex
import shutil
import signal
import subprocess
import sys
import telegram
import threading
import time
import traceback
from six.moves import range
from telegram.error import NetworkError, Unauthorized

class piCamBot:
    def __init__(self):
        # identificacao para manter posicao da ultima mensagem
        self.update_id = None
        # configuracao do arquivo config
        self.config = None

        self.logger = None
        # checa se h movimentacao e captura imagem
        self.armed = False

        self.bot = None

        self.GPIO = None

    def run(self):
```

```
logFormat = logging.Formatter('%(asctime)s␣-␣%(name)s␣-␣%(
    levelname)s␣-␣%(message)s')
self.logger = logging.getLogger(__name__)
fileHandler = logging.handlers.TimedRotatingFileHandler(filename='
    picam.log', when='D', backupCount=7)
fileHandler.setFormatter(logFormat)
self.logger.addHandler(fileHandler)
stdoutHandler = logging.StreamHandler(sys.stdout)
stdoutHandler.setFormatter(logFormat)
self.logger.addHandler(stdoutHandler)
self.logger.setLevel(logging.INFO)

self.logger.info('Starting')

try:
    self.config = json.load(open('config.json', 'r'))
except Exception as e:
    self.logger.error(str(e))
    self.logger.error(traceback.format_exc())
    self.logger.error("Could␣not␣parse␣config␣file")
    sys.exit(1)
#checa por conflitos em config options
if self.config['pir']['enable'] and self.config['motion']['enable'
    ]:
    self.logger.error('Enabling␣both␣PIR␣and␣motion␣based␣
        capturing␣is␣not␣supported')
    sys.exit(1)


if self.config['buzzer']['enable'] or self.config['pir']['enable'
    ]:
    self.GPIO = importlib.import_module('RPi.GPIO')


signal.signal(signal.SIGHUP, self.signalHandler)
signal.signal(signal.SIGINT, self.signalHandler)
signal.signal(signal.SIGQUIT, self.signalHandler)
signal.signal(signal.SIGTERM, self.signalHandler)
```

```
self.armed = self.config['general']['arm']

self.bot = telegram.Bot(self.config['telegram']['token'])

# checa permissao de acesso da API

self.logger.info('Esperando␣pela␣disponibilidade␣da␣rede␣e␣API.')
timeout = self.config['general']['startup_timeout']
timeout = timeout if timeout > 0 else sys.maxsize
for i in range(timeout):
    try:
        self.logger.info(self.bot.getMe())
        self.logger.info('Acesso␣a␣API␣permitido!')
        break # success
    except NetworkError as e:
        pass
    except Exception as e:

        self.logger.error(str(e))
        self.logger.error(traceback.format_exc())
        raise
    time.sleep(1)

# manda mensagem de inicializacao
for owner_id in self.config['telegram']['owner_ids']:
    try:
        self.bot.sendMessage(chat_id=owner_id, text='Ola,␣estou␣de␣
            volta!')
    except Exception as e:

        self.logger.warn('Nao␣foi␣possivel␣enviar␣mensagem␣para␣%s:
            ␣%s' % (owner_id, str(e)))


try:
    self.update_id = self.bot.getUpdates()[0].update_id
except IndexError:
```

```
        self.update_id = None

    # configura o disparo do buzzer
    if self.config['buzzer']['enable']:
        gpio = self.config['buzzer']['gpio']
        self.GPIO.setmode(self.GPIO.BOARD)
        self.GPIO.setup(gpio, self.GPIO.OUT)

    threads = []

    # inicia thread no Telegram
    telegram_thread = threading.Thread(target=self.
       fetchTelegramUpdates, name="Telegram")
    telegram_thread.daemon = True
    telegram_thread.start()
    threads.append(telegram_thread)

    # cria Thread de imagens
    image_watch_thread = threading.Thread(target=self.
       fetchImageUpdates, name="Image␣watch")
    image_watch_thread.daemon = True
    image_watch_thread.start()
    threads.append(image_watch_thread)


    if self.config['pir']['enable']:
        pir_thread = threading.Thread(target=self.watchPIR, name="PIR")

        pir_thread.daemon = True
        pir_thread.start()
        threads.append(pir_thread)

    while True:
        time.sleep(1)

        for thread in threads:
            if thread.isAlive():
                continue
```

```
            msg = 'Thread␣"%s"␣died,␣terminating␣now.' % thread.name
            self.logger.error(msg)
            for owner_id in self.config['telegram']['owner_ids']:
                try:
                    self.bot.sendMessage(chat_id=owner_id, text=msg)
                except Exception as e:
                    pass
            sys.exit(1)

    def fetchTelegramUpdates(self):
        self.logger.info('Configurando␣telegram␣thread')
        while True:
            try:

                for update in self.bot.getUpdates(offset=self.update_id,
                    timeout=10):

                    if not update.message:
                        continue

                    # necessario chat id para enviar qualquer mensagem
                    chat_id = update.message.chat_id
                    self.update_id = update.update_id + 1
                    message = update.message

                    #
                    if message.from_user.id not in self.config['telegram'][
                        'owner_ids']:
                        self.logger.warn('Recebendo␣mensagem␣de␣usuario␣
                            desconhecido␣"%s":␣"%s"' % (message.from_user,
                            message.text))
                        message.reply_text("Nao␣permitido")
                        continue

                    self.logger.info('Mensagem␣recebida␣do␣usuario␣"%s":␣"%
                        s"' % (message.from_user, message.text))
                    self.performCommand(message)
            except NetworkError as e:
```

```
            time.sleep(1)
        except Exception as e:
            self.logger.warn(str(e))
            self.logger.warn(traceback.format_exc())
            time.sleep(1)

    def performCommand(self, message):
        cmd = message.text.lower().rstrip()
        if cmd == '/start':
            # ignora comando default start do chatbot
            return
        if cmd == '/arm':
            self.commandArm(message)
        elif cmd == '/disarm':
            self.commandDisarm(message)
        elif cmd == 'kill':
            self.commandKill(message)
        elif cmd == '/status':
            self.commandStatus(message)
        elif cmd == '/capture':
            # caso o software esteja rodando, envia o comando para
                disparar a cmera
            stopStart = self.isMotionRunning()
            if stopStart:
                self.commandDisarm(message)
            self.commandCapture(message)
            if stopStart:
                self.commandArm(message)
        else:
            self.logger.warn('Comando␣desconhecido:␣"%s"' % message.text)

    def commandArm(self, message):
        if self.armed:
            message.reply_text('Dispositivo␣de␣captura␣j␣habilitado!')
            return

        if not self.config['motion']['enable'] and not self.config['pir'][
            'enable']:
            message.reply_text('Erro!')
```

```
            return

        message.reply_text('Habilitando␣cmera.')

        if self.config['buzzer']['enable']:
            buzzer_sequence = self.config['buzzer']['seq_arm']
            if len(buzzer_sequence) > 0:
                self.playSequence(buzzer_sequence)

        self.armed = True

        if not self.config['motion']['enable']:

            return

        # incia sensor de movimento
        if self.isMotionRunning():
            message.reply_text('Iniciando␣sensor␣de␣movimento.')
            return

        args = shlex.split(self.config['motion']['cmd'])
        try:
            subprocess.call(args)
        except Exception as e:
            self.logger.warn(str(e))
            self.logger.warn(traceback.format_exc())
            message.reply_text('Erro:␣%s' % str(e))
            return


        for i in range(10):
            if self.isMotionRunning():
                message.reply_text('Sensor␣de␣movimento␣inicializado.')
                return
            time.sleep(1)
        message.reply_text('Sensor␣de␣movimento␣no␣est␣funcionando.␣Tente␣
            novamente.')

    def commandDisarm(self, message):
```

```
if not self.armed:
    message.reply_text('Sensor␣de␣movimento␣desabilitado.')
    return

message.reply_text('Desabilitando␣sensor␣de␣movimento.')

if self.config['buzzer']['enable']:
    buzzer_sequence = self.config['buzzer']['seq_disarm']
    if len(buzzer_sequence) > 0:
        self.playSequence(buzzer_sequence)

self.armed = False

if not self.config['motion']['enable']:

    return

pid = self.getMotionPID()
if pid is None:
    message.reply_text('No␣PID␣file␣found.')
    return

if not os.path.exists('/proc/%s' % pid):
    message.reply_text('Removing␣PID␣file.')
    os.remove(self.config['motion']['pid_file'])
    return

try:
    os.kill(pid, signal.SIGTERM)
except OSError:

    pass

for i in range(10):
    if not os.path.exists('/proc/%s' % pid):
        message.reply_text('Sensor␣de␣presena␣parou.')
        return
    time.sleep(1)
```

```
        message.reply_text("Erro.")
        try:
            os.kill(pid, signal.SIGKILL)
        except OSError:

            pass


        for i in range(10):
            if not os.path.exists('/proc/%s' % pid):
                message.reply_text('Sensor␣de␣movimento␣parou.')
                return
            time.sleep(1)
        message.reply_text('Erro.')

    def commandKill(self, message):
        if not self.config['motion']['enable']:
            message.reply_text('Erro.')
            return
        args = shlex.split('killall␣-9␣%s' % self.config['motion']['
            kill_name'])
        try:
            subprocess.call(args)
        except Exception as e:
            self.logger.warn(str(e))
            self.logger.warn(traceback.format_exc())
            message.reply_text('Error::␣%s' % str(e))
            return
        message.reply_text('Kill.')

    def commandStatus(self, message):
        if not self.armed:
            message.reply_text('Dispositivo␣de␣captura␣no␣habilitado.')
            return

        image_dir = self.config['general']['image_dir']
        if not os.path.exists(image_dir):
            message.reply_text('Erro:Imagem␣no␣disponvel!')
            return
```

```
        if self.config['motion']['enable']:
            # Checa se o sensor est ativo
            if not self.isMotionRunning():
                message.reply_text('Erro:␣Cmera␣est␣ligada␣mas␣h␣erro␣de␣
                    software␣desconhecido!')
                return
            message.reply_text('Todos␣os␣dispositivos␣esto␣funcionando␣
                corretamente.')
        else:
            message.reply_text('Cmera␣ligada!')

    def commandCapture(self, message):
        message.reply_text('Capturando␣imagem,␣espere␣um␣instante...')

        if self.config['buzzer']['enable']:
            buzzer_sequence = self.config['buzzer']['seq_capture']
            if len(buzzer_sequence) > 0:
                self.playSequence(buzzer_sequence)

        capture_file = self.config['capture']['file']
        if sys.version_info[0] == 2:
            capture_file = capture_file.encode('utf-8')
        if os.path.exists(capture_file):
            os.remove(capture_file)

        args = shlex.split(self.config['capture']['cmd'])
        try:
            subprocess.call(args)
        except Exception as e:
            self.logger.warn(str(e))
            self.logger.warn(traceback.format_exc())
            message.reply_text('Error:␣Capture␣failed:␣%s' % str(e))
            return

        if not os.path.exists(capture_file):
            message.reply_text('Error:␣Capture␣file␣not␣found:␣"%s"' %
                capture_file)
            return
```

```
        message.reply_photo(photo=open(capture_file, 'rb'))
        if self.config['general']['delete_images']:
            os.remove(capture_file)

    def fetchImageUpdates(self):
        self.logger.info('Setting␣up␣image␣watch␣thread')


        watch_dir = self.config['general']['image_dir']
        #
        if self.config['general']['delete_images']:
            shutil.rmtree(watch_dir, ignore_errors=True)
        if not os.path.exists(watch_dir):
            os.makedirs(watch_dir)
        notify = inotify.adapters.Inotify()
        notify.add_watch(watch_dir.encode('utf-8'))


        for event in notify.event_gen():
            if event is None:
                continue

            (header, type_names, watch_path, filename) = event


            matched_types = ['IN_CLOSE_WRITE', 'IN_MOVED_TO']
            if not any(type in type_names for type in matched_types):
                continue


            if sys.version_info[0] == 3:
                watch_path = watch_path.decode()
                filename = filename.decode()
            filepath = ('%s/%s' % (watch_path, filename))

            if not filename.endswith('.jpg'):
                self.logger.info('New␣non-image␣file:␣"%s"␣-␣ignored' %
                    filepath)
```

```
            continue

        self.logger.info('New␣image␣file:␣"%s"' % filepath)
        if self.armed:
            for owner_id in self.config['telegram']['owner_ids']:
                try:
                    self.bot.sendPhoto(chat_id=owner_id, caption=
                        filepath, photo=open(filepath, 'rb'))
                except Exception as e:

                    self.logger.warn('Could␣not␣send␣image␣to␣user␣%s:␣%
                        s' % (owner_id, str(e)))


        if self.config['general']['delete_images']:
            os.remove(filepath)

    def getMotionPID(self):
        pid_file = self.config['motion']['pid_file']
        if not os.path.exists(pid_file):
            return None
        with open(pid_file, 'r') as f:
            pid = f.read().rstrip()
        return int(pid)

    def isMotionRunning(self):
        pid = self.getMotionPID()
        return os.path.exists('/proc/%s' % pid)

    def watchPIR(self):
        self.logger.info('Setting␣up␣PIR␣watch␣thread')

        if self.config['buzzer']['enable']:
            buzzer_sequence = self.config['buzzer']['seq_motion']

        gpio = self.config['pir']['gpio']
        self.GPIO.setmode(self.GPIO.BOARD)
        self.GPIO.setup(gpio, self.GPIO.IN)
        while True:
```

```
        if not self.armed:

            time.sleep(1)
            continue

        pir = self.GPIO.input(gpio)
        if pir == 0:

            time.sleep(1)
            continue

        self.logger.info('PIR:␣motion␣detected')
        if self.config['buzzer']['enable'] and len(buzzer_sequence) >
            0:
            self.playSequence(buzzer_sequence)
        args = shlex.split(self.config['pir']['capture_cmd'])

        try:
            subprocess.call(args)
        except Exception as e:
            self.logger.warn(str(e))
            self.logger.warn(traceback.format_exc())
            message.reply_text('Error:␣Capture␣failed:␣%s' % str(e))

    def playSequence(self, sequence):
        gpio = self.config['buzzer']['gpio']
        duration = self.config['buzzer']['duration']
        for i in sequence:
            if i == '1':
                self.GPIO.output(gpio, 1)
            elif i == '0':
                self.GPIO.output(gpio, 0)
            else:
                self.logger.warnprint(':␣%s', i)
            time.sleep(duration)
        self.GPIO.output(gpio, 0)

    def signalHandler(self, signal, frame):
        # always disable buzzer
```

```
        if self.config['buzzer']['enable']:
            gpio = self.config['buzzer']['gpio']
            self.GPIO.output(gpio, 0)
            self.GPIO.cleanup()

        msg = 'Caught␣signal␣%d,␣terminating␣now.' % signal
        self.logger.error(msg)
        for owner_id in self.config['telegram']['owner_ids']:
            try:
                self.bot.sendMessage(chat_id=owner_id, text=msg)
            except Exception as e:
                pass
        sys.exit(1)

if __name__ == '__main__':
    bot = piCamBot()
    bot.run()
```