UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
FACULDADE DE ENGENHARIA DA COMPUTAÇÃO E TELECOMUNICAÇÕES

Uso de Reconhecedor e Sintetizador de Voz Embarcados para Controle de Equipamentos Eletrônicos via Luz Infravermelha

Belém – Brasil
Junho/2015

Uso de Reconhecedor e Sintetizador de Voz Embarcados para Controle de Equipamentos Eletrônicos via Luz Infravermelha

Cassio Trindade Batista
cassio.batista.13@gmail.com
201106840003

Gabriel Peixoto de Carvalho
gaburiero.c@gmail.com
201106840010

Pedro Henrique C. F. Soares
pedrofigueiredoc@gmail.com
201106840007

Thiago Barros Coelho
tbarroscoelho@gmail.com
201106840041

Projeto apresentado à disciplina Projetos de Hardware e Interfaceamento como requisito de avaliação. Professores: Jeferson Breno N. Leite e Adalbery Rodrigues Castro.

Belém – Brasil
Junho/2015

Sumário

Lista de Figuras

1 Introdução

A interface homem-máquina encontra-se cada vez mais amigável. O que antes era portado somente por empresas e pessoas com poder financeiro diferenciado e acima da média, em termos de tecnologia, é hoje muito mais acessível e simples para usuários domésticos sem profundo conhecimento no assunto. Devido a abrangência de computadores pessoais e embarcados e da *Internet*, novas oportunidades e expectativas em termos de trabalho, estudos e até lazer são criadas a fim de melhorar ainda mais essa comunicação, de modo que a máquina se aproxime mais de ações típicas do ser humano, como pensar e falar.

Acredita-se que a síntese e o reconhecimento automático de voz (do inglês *text-to-speech* e *automatic speech recognition*, respectivamente, TTS e ASR) [**?**, **?**] tornam a *interface* citada acima muito mais prática e natural, de forma que a comunicação de fato se assemelha àquela estabelecida entre duas pessoas. O ASR refere-se ao sistema que, tomando o sinal de fala digitalizado como entrada, é capaz de gerar o texto transcrito na saída. Já um sistema TTS realiza a função contrária, na qual um sinal analógico de voz é sintetizado de acordo com o texto posto na entrada. Dentre as inúmeras aplicações que utilizam os sistemas que envolvem processamento de fala, pode-se destacar a automação residencial que visa ajudar pessoas que tenham dificuldades no controle de alguns equipamentos eletrônicos, dando ênfase à acessibilidade alcançada através das tecnologias assistivas.

Tecnologia Assistiva (TA) é um campo da engenharia biomédica dedicada a aumentar a independência e mobilidade de pessoas com deficiência, englobando metodologias, práticas e serviços que objetivam promover sua autonomia, qualidade de vida e inclusão social [**?**, **?**]. Tal tecnologia busca reduzir a necessidade vivenciada por pessoas que precisam de soluções que não as deixem à margem da utilização de dispositivos eletrônicos. Em outras palavras, para diminuir a exclusão digital imposta pela incapacidade de manipular certos equipamentos, a acessibilidade é vista como elemento fundamental para elevar a autoestima e o grau de independência dessas pessoas. Além disso, as mesmas soluções apresentadas podem ser úteis também para os não portadoras de necessidades especiais, já que o controle de equipamentos se torna mais prático e confortável.

Nesse sentido, este trabalho busca preparar um servidor local portátil de reconhecimento e síntese de voz em Português Brasileiro (PT_BR) baseado na plataforma BeagleBone Black de modo que, quando acessado pelo dispositivo que agirá como controle remoto — no caso, um *smartphone* com sistema operacional Android —, seja capaz de acessar as funções mais básicas de um aparelho televisivo. Vale ressaltar que todas as APIs, IDEs, *softwares*, bibliotecas e pacotes utilizados para criação dos sistemas e dos recursos utilizados possuem licença *open source* e são encontrados disponíveis livremente na *Internet*.

2 Objetivos

O objetivo principal consiste em criar um protótipo portável, baseado em um microcomputador embarcado, que seja capaz de controlar um aparelho de televisão através do envio remoto de sinais, conforme mostrado na Figura 1. O sistema será configurado como um servidor que disponibiliza um serviço genérico de reconhecimento de fala, de modo que o aparelho de TV mencionado possa ser remotamente controlado através da voz do usuário; e um serviço de síntese de fala, provendo *feedback* das ações de acordo com o entendimento do sistema de ASR. Além disso, a informação a ser enviada para a TV deve ser armazenada em um banco de dados, também configurado no mesmo servidor.

2.1 Reconhecimento e Síntese de Voz

Para que o reconhecimento automático de voz seja possível, o *software* Julius deverá ser instalado no servidor. Julius é um *software* capaz de processar e decodificar áudio em aproximadamente tempo real para tarefas de ditado de até 60 mil palavras. Este também será o principal programa do sistema, o qual receberá em seu código nativo todos os outros módulos.

Para que o Julius possa realizar o reconhecimento em Português Brasileiro, serão necessários basicamente dois recursos: um modelo acústico e um dicionário fonético. Modelos acústicos genéricos para

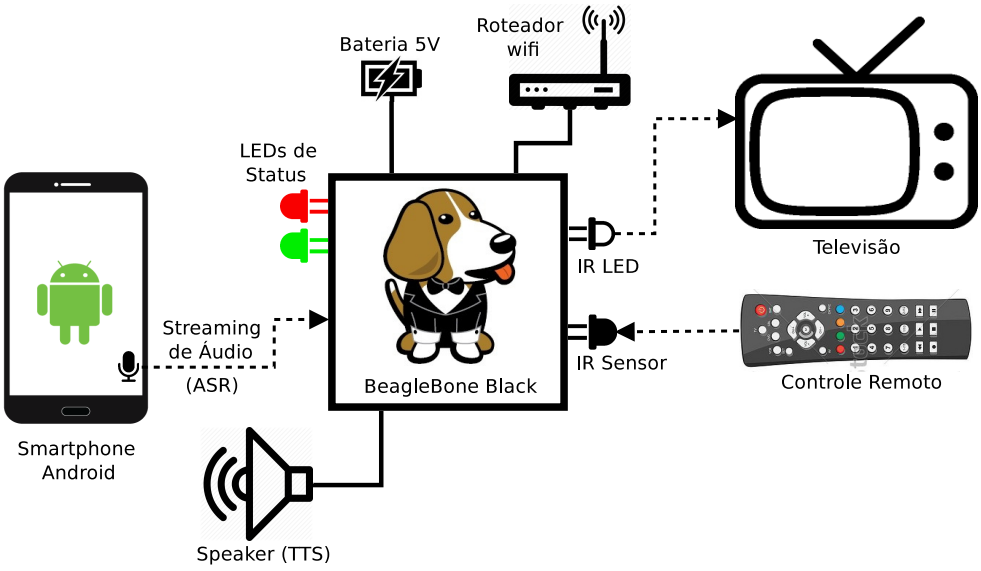

Figura 1: Esquemático do Projeto.

PT_BR podem ser encontrados na página do Grupo FalaBrasil [**?**], bem como o *software* que cria o dicionário fonético (conversor grafema-fonema ou G2P) [**?**]. Entretanto, embora a taxa de acerto dos modelos seja satisfatória, ainda há casos nos quais a acurácia do modelo não é suficiente. Nesse caso, é possível melhorá-la através da criação ou treino de modelos específicos para a aplicação. O processo de treino é realizado pelo *software* HTK (*Hidden Markov Models Toolkit*), o qual é capaz de extrair segmentos de fala de um arquivo de áudio e assinalar uma referência à ele. Tal referência é retirada do dicionário fonético, previamente criado com o *software* G2P.

O *software* eSpeak é a principal referência em síntese de voz em ambientes Linux. Graças à disponibilização de uma API no site oficial do desenvolvedor, o sistema, além de conseguir "ouvir e entender", também será capaz de "falar". O *download* dos modelos para PT_BR é feito juntamente com o das bibliotecas necessárias. Como a BeagleBone Black (BBB) não possui saída de áudio nativa, um alto-falante USB será utilizado.

2.2 Controle Remoto de TV e Servidor LAMP

Os aparelhos televisivos atuais, assim como a grande maioria dos dispositivos eletrônicos domésticos, possuem a tecnologia de controle remoto baseada em luz infravermelha. Pode-se observar que na extremidade superior dos controles remotos, há pelo menos um LED infravermelho (IR LED) capaz de emitir luz e, dessa forma, transmitir uma informação modulada para o circuito localizado na parte frontal da TV. Esse circuito possui um sensor infravermelho (IR sensor), o qual, atuando como o receptor da comunicação, é capaz de receber os bits transmitidos e repassá-los para o processador do circuito, o qual executará a tarefa relacionada à decodificação dos bits (desligar a TV, mostrar o menu, etc).

Nesse sentido, um módulo receptor será adicionado à BBB para que esta seja capaz de "hackear", através de um sensor infravermelho, informações dos controles remotos, dado que o acesso aos *datasheets* de diversos aparelhos não é trivial. O sistema de gerenciamento de banco de dados MySQL será instalado e configurado no servidor para armazenar o código, o qual é relacionado a ações como "aumentar volume" ou "desligar" da TV principal e dos demais aparelhos que vierem a ser cadastrados no banco. Toda a informação de saída a ser enviada para a TV através dos IR LEDs deve ser oriunda do banco de dados.

3 Justificativa

No censo realizado em 2010 pelo Instituto Brasileiro de Geografia e Estatística (IBGE), aproximadamente 23,9% dos brasileiros declararam ter alguma deficiência [**?**]. Esse número é realmente impressionante, pois revela que cerca de um quarto de uma população de 190 milhões de habitantes é portadora de necessidades especiais. Segundo os dados, 6,9% (13,3 mi) dos brasileiros apresentam algum grau de deficiência motora, enquanto 18,8% (35,7 mi) afirmam serem cegas ou terem alguma dificuldade para enxergar.

Em [**?**], uma entrevista foi realizada com brasileiros portadores de necessidades especiais para se determinar quais as características que esses cidadãos gostariam de adicionar, se pudessem, em controles remotos convencionais, visando minimizar suas dificuldades em utilizá-los. A sugestão de um *design* mais limpo foi unânime, enquanto os deficientes visuais, em particular, apontaram a necessidade de algum tipo de *feedback* quando os botões fossem pressionados e a implementação de alguma referência reconhecível pelo tato nos botões. Já os deficientes motores sugeriram um dispositivo menor, que não escorregasse facilmente das mãos, contendo um ângulo de controle mais abrangente e com botões maiores para aqueles que não conseguem ter controle absoluto sobre as mãos e/ou dedos. O estudo também mostrou que as referências sentidas pelo tato não foram incorporadas no *design* a fim de se manter o baixo custo.

Essa pesquisa tem como finalidade apresentar uma solução para diminuir a exclusão digital vivenciada especialmente por pessoas com necessidade motora ou visual, as quais estão à margem da utilização de dispositivos eletrônicos por conta da ausência de soluções que os adaptem às suas necessidades. A tecnologia de reconhecimento de fala torna acessível a utilização de qualquer dispositivo eletrônico por usuários incapazes de realizar movimentos específicos com membros superiores, como segurar um controle físico e apertar botões ou digitar, por exemplo. Além disso, os portadores de necessidades visuais também poderão ser ajudados, já que nem todos os controles possuem referências hápticas. A síntese de fala também se torna muito importante no contexto da dificuldade visual, já que, nesse sentido, um *feedback* por fala será muito mais útil do que via texto.

Em [**?**] é sugerido que as interfaces multimodais (métodos alternativos de controle como voz, escrita, gestos, etc.) associadas ao controle de equipamentos provêem uma experiência muito mais prazerosa e prática ao usuário do que os métodos convencionais (botões), apesar de diminuir a usabilidade e a rapidez na execução da tarefa. Isso prova que o sistema proposto, apesar de ter os portadores de necessidades especiais como público alvo, também pode ser utilizado por qualquer pessoa que deseja controlar o ambiente ao redor com mais praticidade e conforto. A ação de restringir soluções aos deficientes, mesmo que seja de boa intenção, ainda é uma forma de segregação.

O Ato de Americanos com Deficiência (ADA) [**?**] é um documento que regula os direitos dos cidadãos com deficiência nos EUA, além de prover a base legal dos fundos públicos para compra dos recursos que estes necessitam. Algumas categorias de TA foram criadas com base nas diretrizes gerais da ADA, das quais podemos ressaltar três como justificativa do trabalho:

a) Recursos de acessibilidade ao computador: Equipamentos de entrada e saída (síntese de voz, Braille), auxílios alternativos de acesso (ponteiras de cabeça, de luz), teclados modificados, *softwares* especiais (reconhecimento de voz, etc.), que permitem as pessoas com deficiência a usarem o computador.

b) Sistemas de controle de ambiente: Sistemas eletrônicos que permitem as pessoas com limitações moto-locomotoras controlar remotamente aparelhos eletro-eletrônicos, sistemas de segurança, entre outros, localizados em seu quarto, sala, escritório, casa e arredores.

c) Auxílios para cegos ou com visão subnormal: Auxílios para grupos específicos que inclui lupas e lentes, Braille para equipamentos com síntese de voz, grandes telas de impressão, sistema de TV com aumento para leitura de documentos, publicações etc.

A decisão de criar um servidor próprio de síntese e reconhecimento de voz baseia-se principalmente na possibilidade de usufruir de tais recursos de forma *offline*, ou seja, sem a necessidade de conexão com a Internet. O fato de as comunicações ocorrerem em LAN diminui muito o tempo de espera do usuário, já

que o áudio não precisa ser transmitido a servidores de voz distantes. Ainda sobre o sistema ASR, pode-se também citar a vantagem de limitar o vocabulário de palavras utilizados através da implementação de uma gramática, já que serviços *online* de reconhecimento (como o disponibilizado pelo Google, por exemplo), trabalham com a inteira modelagem das palavras do idioma, impossibilitando a criação de um contexto reduzido especificamente para a aplicação.

4 Revisão Teórica

Como o iOS e o Android foram lançados, respectivamente, em 2007 e 2008, e a ascensão dos *smartphones* é relativamente recente, a ideia de tornar acessível o controle de equipamentos eletrônicos somente começou a revelar resultados concretos a partir de 2010. Em [**?**], o decodificador PocketSphinx foi embarcado em um *smartphone* Android para que este pudesse controlar aparelhos domésticos através da interface de voz. O resultado do reconhecedor era enviado para placa SparkFun IOIO Board, a qual, estando fisicamente conectada ao controle de uma TV, acionava um determinado comando. A justificativa do trabalho era ajudar pessoas afetadas com tetraplegia a serem mais independentes; os testes avaliaram o desempenho de decodificadores embarcados e distribuídos. Já em [**?**], um sistema inteligente de segurança foi implementado em uma BeagleBone Black, a qual utilizava a biblioteca OpenCV para contar quantos indivíduos encontravam-se próximo à entrada do ambiente a ser vigiado. Um módulo de reconhecimento de fala, também basedo no PocketSphinx, foi utilizado para receber comandos, enquanto o *software* eSpeak foi encarregado de prover as respostas dadas ao usuário via voz sintetizada. Um aparelho celular qualquer poderia se comunicar com o sistema via SMS graças à adição de um módulo GSM à BBB. A desvantagem é que, apesar de o controle não ter de ser executado necessariamente por um *smartphone*, os comandos de voz não poderiam ser dados diretamente do aparelho móvel, e sim pessoal e diretamente ao microcomputador, o qual possuía um microfone conectado à porta USB.

4.1 Reconhecimento e Síntese de Voz

Os fundamentos do reconhecimento automático de voz, assim como os da síntese de voz, são descritos com bastante detalhes em [**?**]. A arquitetura mais geral e aceita na literatura é mostrada na Figura 2. Vale salientar que, ao invés do uso de modelos mais gerais, que descrevem a maior parte de uma linguagem, serão utilizadas gramáticas livres de contexto, as quais limitam o vocabulário utilizado a apenas um conjunto de sentenças possíveis, escolhidas pelo desenvolvedor do sistema. A construção do dicionário fonético para PT_BR dar-se-á através do *software* descrito em [**?**]; o tutorial para o treino do modelo acústico encontra-se disponível na página do projeto Voxforge [**?**], bem como no capítulo 3 do livro do HTK [**?**, p. 22–42] (lembrando que o modelo acústico utilizado também está disponível na página do FalaBrasil); a gramática reconhecida pelo Julius é criada manualmente de acordo com o descrito na página oficial [**?**]. Instruções de configuração e utilização do Julius encontram-se na documentação oficial [**?**].

Como saída analógica do sistema TTS, a voz sintetizada deve ser reproduzida por um dispositivo externo à BeagleBone, já que esta não possui alto-falantes próprios. Como visto em [**?**], o dispositivo primário de saída de áudio é o HDMI, o qual pode ser desabilitado mediante modificações em parâmetros do *kernel*. Feito isso, o USB, que é o dispositivo secundário, se torna o principal, fazendo com que a solução mais simples seja plugar um alto-falante (*speaker*) na porta USB. Na página oficial do eSpeak, um arquivo de cabeçalho (*header*) permite a utilização de uma API em C/C++, a qual facilita o acesso aos módulos do *software* que permitem que a BBB "fale" [**?**].

4.2 Comparação entre Plataformas

A escolha da plataforma foi fundamental para a esquematização do projeto. Arduino, Raspberry Pi e BeagleBone Black foram as três principais opções a serem escolhidas. Diversos tutoriais de comparação entre as plataformas foram consultados e estão disponíveis na Internet [**?**, **?**, **?**]. O Arduino, apesar de ser uma ferramenta flexível e com grande capacidade de interfaceamento com uma vasta quantidade de

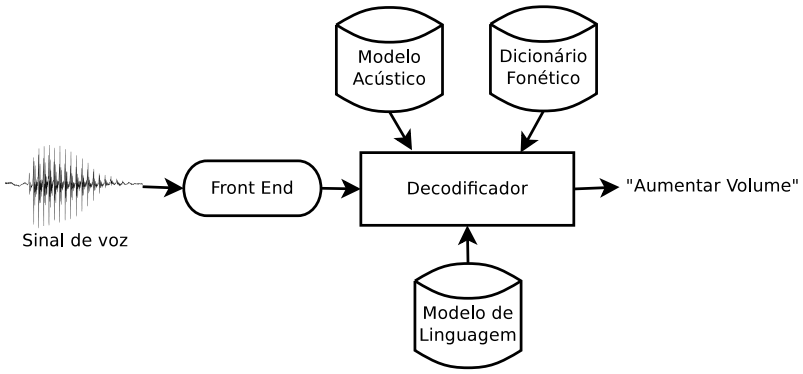

Figura 2: Esquemático de um sistema ASR

dispositivos, é uma plataforma simples, recomendada para projetos de menor porte. O microcontrolador, que pode ser programado em C, torna-se muito limitado quando o projeto requer um servidor estável e relativamente potente. Já o Raspberry Pi, por ser bastante completo, enquadra-se no conceito de mini computador, sendo necessário a instalação de um sistema operacional. Todo o seu armazenamento é fornecido por um cartão SD, além de ser possível conectá-lo à Internet através de um conector Ethernet. O Raspberry Pi possui ainda uma interface de saída HDMI e é bastante útil para aplicações gráficas.

Tabela 1: Comparação entre as três principais plataformas

	Arduino UNO	BeagleBone Black	Raspberry Pi
Chip	-	TI AM3359	BCM2835
CPU	ATMega328	ARM Cortex-A8	ARM1176JZ-F
CPU Freq.	16 MHz	1 GHz	700 MHz
GPU	-	PowerVR SGX530	Dual Core VideoCore IV
Memória. RAM	2 kB SRAM	512 MB DDR3	512 MB SDRAM
Armaz. Flash	32 kB	2 GB eMMC, MicroSD	SD, MMC, SDIO card slot
Pinos	14	65	8
Video	-	mini HDMI	HDMI
Sistema Op.	-	Linux	Linux
Amperagem	42 mA	210-460 mA	150-350 mA
Voltagem	7-12 V	5 V	5 V
USB	-	1 Host, 1 Mini Client	2 Hosts, 1 Micro Power
Ethernet	-	1 10/100 Mbps	1 10/100 Mbps

A BeagleBone Black é comparável ao Raspberry Pi. Entretanto, por ter mais pinos (GPIO) e um processador mais poderoso, a plataforma é uma escolha óbvia para projetos mais elaborados. Além de possuir diversas opções de conexão, a BBB une a flexibilidade de interfaceamento do Arduino com a capacidade de processamento rápido do Raspberry Pi. Apesar da desvantagem no preço, não restaram muitas dúvidas no momento da escolha dessa plataforma para o projeto. Uma comparação entre os principais parâmetros dos três equipamentos é dada na Tabela 1.

4.3 Protocolos de Comunicação via Luz Infravermelha

O funcionamento de controles remotos, com ênfase nos baseados em luz infravermelha para televisores, é explicado de de forma clara e detalhada em diversos tutoriais para "curiosos" disponíveis na Internet, como os da revista Mundo Estranho [**?**] e do blog *How Stuff Works?* da UOL [**?**]. Atualmente, a maioria

dos aparelhos eletrônicos recebe informação através de sensores infravermelhos localizados em painéis frontais. Entretanto, devido à interferência que surge com a vasta transferência de informação via IR, a comunicação entre o controle remoto e a televisão ocorre geralmente em 4 passos: um comando *start* inicia a transferência, seguido dos bits do comando específico (como aumentar o volume, por exemplo) e do endereço do aparelho. Por fim, um comando de *stop* encerra o envio de bits. Dessa forma, a chance de a informação ser reconhecida por mais de um aparelho é baixa (salvo o caso de serem dois equipamentos do mesmo tipo e da mesma empresa).

Uma das grandes dificuldades relacionadas ao controle de equipamentos eletrônicos é que os 4 passos acima são apenas uma forma genérica de descrever a comunicação, ou seja, cada empresa praticamente segue o seu próprio padrão para estabelecer a comunicação entre os dispositivos: o número, a ordem e o significado dos bits é variado, a modulação e codificação usada são diferentes e a frequência dos pulsos pode oscilar entre 32 kHz e 42 kHz, chegando a 50 kHz em determinados aparelhos mais modernos. Além disso, tão rara quanto o seguimento de uma comunicação IR padronizada é a disponibilização de documentação pelos fabricantes, que também é bastante escassa.

Neste trabalho, serão detalhados protocolos de duas fabricantes: RC-6 e sua versão pioneira RC-5, ambos da Philips; e o protocolo utilizado pela Samsung.

4.3.1 O Protocolo da Philips: RC-5 e RC-6

A Philips utiliza seu próprio protocolo de comunicação, chamado RC-5. A última documentação foi liberada em 1992, quando ainda não existiam muitos dos equipamentos eletrônicos atuais, como *home theaters* ou DVDs.

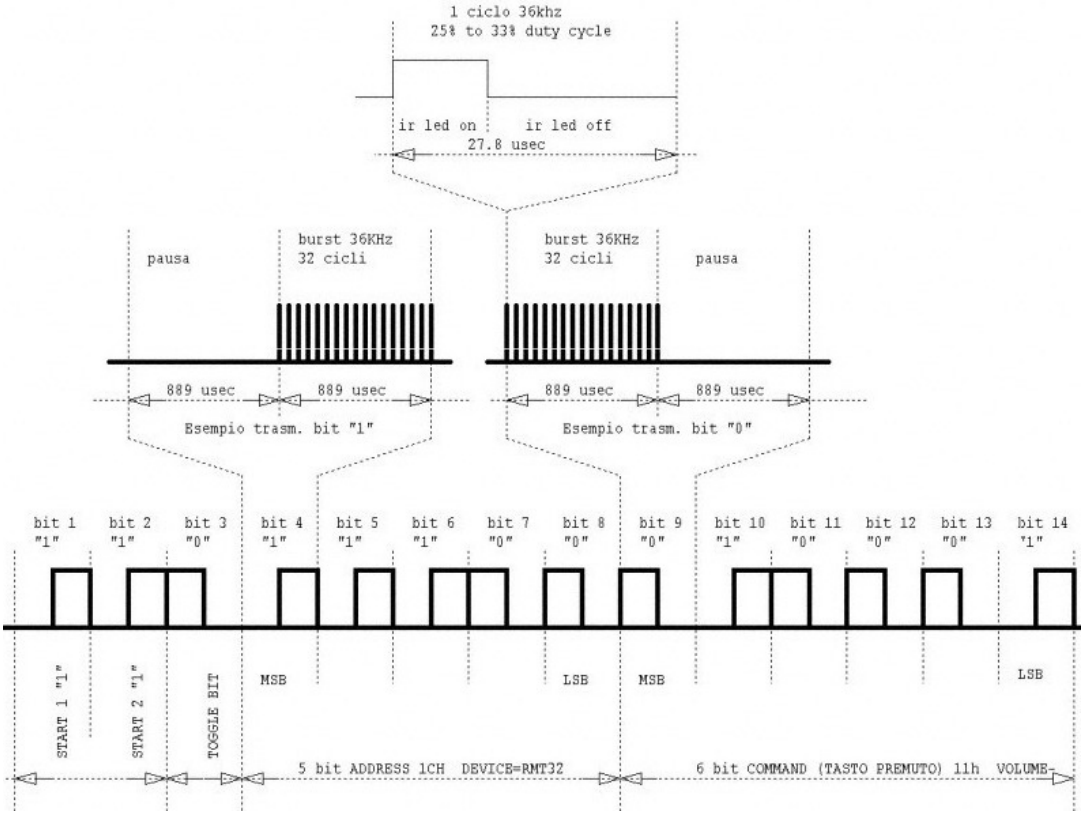

Figura 3: Esquemático do protocolo RC-5 da Philips.

Nesse padrão, os bits são codificados de acordo com o código de Manchester, onde cada bit, transmitido dentro de um período fixo, é representado por uma transição *high-to-low* (0) ou *low-to-high* (1). Esses padrões de dados são obtidos através de uma operação do tipo XOR (OU-Exclusivo) realizada entre o clock do dispositivo e o dado propriamente dito [**?**].

A Figura 3 mostra a informação referente ao comando "diminuir volume" contida num vetor de 14 bits. Os 11 últimos bits definem o endereço do aparelho de destino e o comando em si. Pode-se observar que qualquer bit é representado por duas partes, sendo uma metade em nível baixo e a outra, em nível alto. Cada nível ocorre num intervalo de 32 períodos. O protocolo utiliza a modulação por largura de pulsos (*pulse width modulation* ou PWM). O nível alto é gerado por um PWM de *duty cycle* que varia entre 25% e 33% do período do pulso. Essa percentagem define o tempo em que o IR LED permanece aceso, ou seja, no nível alto, o IR LED permanece ligado por período de $0,25 \times 1/36000\ s$ e desligado por $0,75 \times 1/36000\ s$, sendo o processo repetido 32 vezes. A economia de energia é mostrada como justificativa para explicar o motivo de o led piscar ao invés de se manter aceso por 100% do período do pulso.

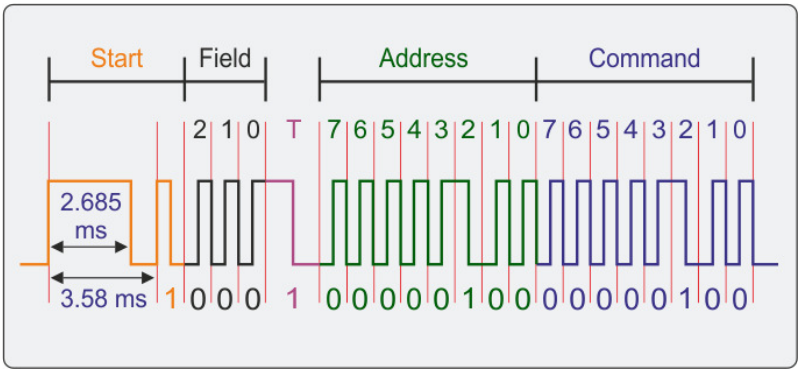

Figura 4: Esquemático do protocolo RC-6 da Philips.

Os aparelhos mais novos já implementam a versão atualizada desse protocolo, chamado de RC-6. Embora a Philips não tenha disponibilizado qualquer documentação sobre este protocolo, há fóruns na Internet que, através da aplicação de engenharia reversa, conseguem descrever o padrão utilizado na construção do sinal. Pela Figura 4, é fácil notar que a quantidade de bits carregada pelo sinal (em comparação com o RC-5) aumentou de 14 para 22. O código Manchester também aparece invertido, já que valor lógico alto passa agora a ser representado pela transição *high-to-low* (1), enquanto o valor baixo é definido por uma mudança *low-to-high* (0). O primeiro bit de *start* tem uma duração maior, para garantir o ganho do AGC (*Automatic Gain Controller*) no circuito receptor; já o segundo, também de *start*, é sempre mantido em valor alto para sincronização. O bit de *toggle*, o qual muda de estado caso uma tecla deixe de ser pressionada, também possui uma duração mais longa do que os outros bits comuns. O período de um bit passa de 32 para 16 pulsos, exceto para os casos especiais (AGC e *toggle*). Por fim, os últimos 16 bits representam o endereço do aparelho e o referido comando a ser transmitido, respectivamente.

4.3.2 O Protocolo da Samsung

Assim como a Philips, a Samsung também utiliza seu próprio padrão para comunicação entre os controles e os dispositivos eletrônicos, o qual não possui uma nomenclatura específica. Felizmente, o documento de *application notes* do microcontrolador S3F80KB [**?**] presente em controles da Samsung encontra-se disponível na Internet. O protocolo define uma sequência de 34 bits, onde ambos os valores 0 e 1 são representados por uma mudança no estado e diferenciados pela distância entre o nível baixo dos pulsos. Antagonicamente ao código Manchester, onde a ordem da transição define o valor do bit, o protocolo da Samsung sempre define os estados como uma transição *high-to-low*, na qual é estabelecida uma duração de 560 μs para o nível alto e duas para o nível baixo: 1690 μs para o bit '1' e 560 μs para o bit '0'.

O nível alto do pulso dos bits normais é caracterizado por um PWM com frequência de 37,9 kHz e *duty cycle* que pode variar entre 33% e 50% do período do pulso. Aproximadamente 21 pulsos na frequência portadora resultam na duração de 560 μs. O primeiro bit, chamado de *leader*, tem duração mais longa para garantir o ganho no circuito receptor: 4,5 ms no nível alto e 4,5 ms no nível baixo; os 16 bits seguintes são divididos em dois blocos exatamente iguais de 8 *custom* bits cada; outras duas sessões de 8 bits definem o bloco de dado, onde o segundo bloco é o complemento dos bits do primeiro, o qual define os dados propriamente ditos; o 34° e último bit é sempre baixo e encerra o comando.

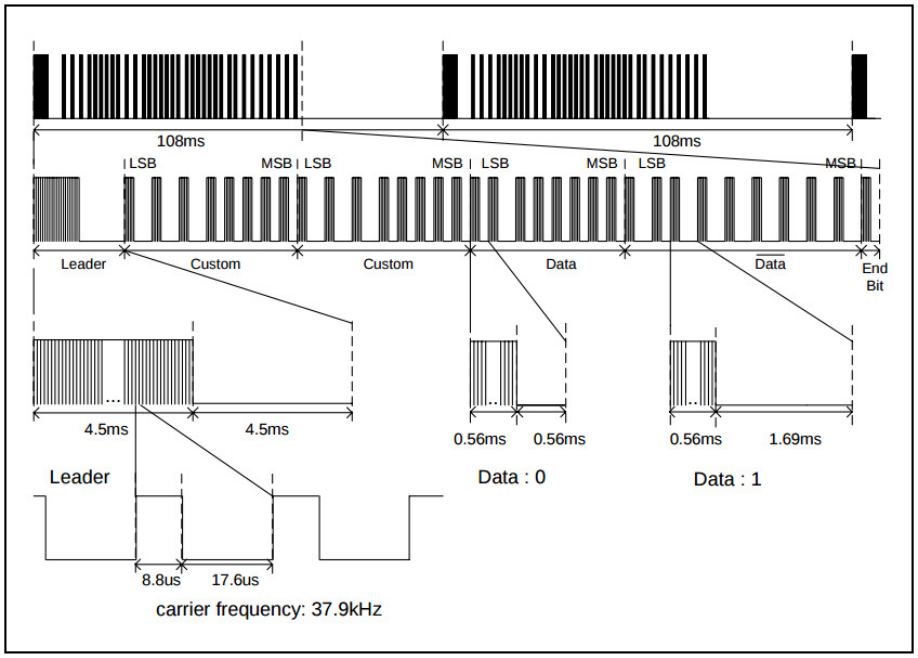

Figura 5: Esquemático do protocolo da Samsung.

4.4 Produtos Relacionados

Existem alguns produtos disponíveis no mercado com a finalidade de tornar o controle de equipamentos eletrônicos mais prático. Um deles é o Harmony Smart Control [**?**], o qual possui em seu "pacote" um aplicativo para *smartphones* iOS e Android (sem versão para tablets, todavia), um *hub* e um controle remoto genérico. Segundo a revisão da CNET, vale a pena pagar US$ 130 por todas as funcionalidades que o sistema apresenta, como usar uma conexão de rádio-frequência entre o *smartphone*/controle com o *hub* (que, infelizmente, ainda não conseguiu se livrar do tão antiquado infravermelho), o que faz com que o usuário não precise apontar o controle para o dispositivo que precisa controlar. Todavia, o *hub* precisa estar numa boa posição para conseguir emitir a informação de forma clara para o aparelho desejado.

IRdroid é outro aplicativo que permite o controle de aparelhos televisivos com o celular [**?**]. Como o próprio nome sugere, funciona apenas em dispositivos Android, desde que seu módulo de *hardware* esteja acoplado na saída de audio *jack* do *smartphone*. Versões mais recentes já possuem o *hardware* que pode ser acessado via *bluetooth*, custando US$ 60 o mais caro. A grande vantagem é que o IRdroid, além de ser baseado na biblioteca LIRC, a qual possui uma vasta quantidade de equipamentos em seu banco de dados, possui código livre e disponível.

Outras diversas soluções são aplicadas apenas à *smart TVs*, onde a informação de controle é transmitida por *wifi*. Nenhuma das aplicações encontradas para TVs convencionais possui suporte à reconhecimento e síntese de voz *offline* em PT_BR.

5 Metodologia

O servidor, por ser o elemento chave na consolidação do projeto, deve ser o módulo a ser prioritariamente configurado, a fim de ser preparado para atender às devidas requisições, bem como executar qualquer tipo de aplicação solicitada. Sendo assim, a instalação do sistema operacional Debian foi tomada como primeiro passo, visto que houveram muitos problemas na instalação do Ubuntu e do Ångström. As dependências a serem instaladas são mostradas na Lista 1.

É importante ressaltar que os sistemas operacionais embarcados são simplificações de sistemas operacionais mais robustos, tendo a maior parte das suas funcionalidades reduzidas para se adequar a uma plataforma de menor porte. Por isso, a preparação deve ocorrer a partir dos pacotes mais básicos, como o GCC, por exemplo. Outros pacotes devem ser instalados de forma gradual, tais quais os requeridos pelo Julius, eSpeak e os necessários para a implementação do servidor LAMP em C.

Listing 1: Pre-instalação de dependências no servidor

```
# general dependences
build-essential alsa-tools alsa-base alsa-utils sox

# eSpeak dependences
libespeak-dev libportaudio2 libportaudio-dev

# Julius dependences
libasound2 libasound2-dev

# LAMP dependences
apache2 libapache2-mod-fastcgi # apache server
mysql-server libapache2-mod-auth-mysql php5-mysql # MySQL
libmysqlclient-dev # C
phpmyadmin # (opcional?)
```

5.1 Entrada de Áudio e Reconhecimento de Voz

Em [**?**], o Julius foi configurado para funcionar em modo servidor através da opção nativa "-adinnet" (A/D *Input from Network*, conversão A/D com entrada pela rede). Isso permite que o Julius receba amostras de áudio via *streaming* através de uma comunicação com um cliente genérico via *socket*. O código foi alterado para que o resultado gerado pelo Julius, também conhecido como sentença, seja retornado ao cliente através desse mesmo *socket*. Além disso, uma aplicação foi construída sobre a plataforma Android exclusivamente para se comunicar com o servidor. Basicamente, as amostras de áudio obtidas pelo microfone do aparelho são enviadas, enquanto são paralelamente analisadas a fim de se detectar o silêncio do fim da fala do usuário. Feito isso, o aplicativo apenas aguarda a sentença a ser enviada pelo servidor.

A construção do dicionário fonético para o PT_BR se dá por meio do *software* `lapsg2p`, o qual recebe uma lista de palavras como entrada e gera suas transcrições fonéticas, conforme visto na lista abaixo, à direita. Já a gramática é utilizada para restringir o vocabulário, de modo a gerar somente uma das senteças listadas, como mostrado na lista abaixo, à esquerda. A construção da gramática no formato do Julius utiliza diretamente o dicionário fonético em seu escopo.

```
<s> aumentar volume </s>
<s> diminuir volume </s>
<s> canal mais </s>
<s> canal menos </s>
<s> ligar televisão </s>
<s> desligar televisão </s>
<s> cadastrar controle </s>
<s> selecionar controle </s>
```

```
aumentar   a u~ m e~ t a X
diminuir   dZ i~ m i~ n u j X
volume     v o l u~ m i
canal      k a n a w
mais       m a j s
menos      m e~ n u s
televisão  t e l e v i z a~ w~
...
```

5.2 Análise do Sinal Infravermelho

Inicialmente, um Arduino UNO fui utilizado para verificar o tempo em que o IR LED permanecia ativo e inativo, armazenando-o em uma matriz de duas colunas, como visto no Apêndice B. O Octave foi utilizado para converter a matriz em um vetor único, no qual os índices ímpares representavam o tempo de duração do modo *burst* do IR LED e as posições pares, o tempo em que o IR LED permanecia *idle*. Sendo assim, o vetor no qual as durações dos níveis altos e baixos alternam-se entre si foi convertido para uma forma de onda quadrada, a qual mostra claramente as características dos protocolos descritas na seção anterior.

5.2.1 Philips

O controle RC2954201/01 da TV Philips 39PFL3008D/78 foi usado como base para a análise do sinal emitido. A Figura 6 mostra a forma de onda quadrada para 4 botões diferentes pressionados de forma não consecutiva. Pode-se notar que os sinais são exatamente iguais até aproximadamente $t = 15ms$, onde começam a ser exibidos os bits referentes ao comando específico. Já na Figura 7, o botão "volume mais" foi pressionado por quatro vezes consecutivas. Percebe-se que os sinais são completamente idênticos, exceto o bit de *toggle*, o qual muda sempre que um botão é solto.

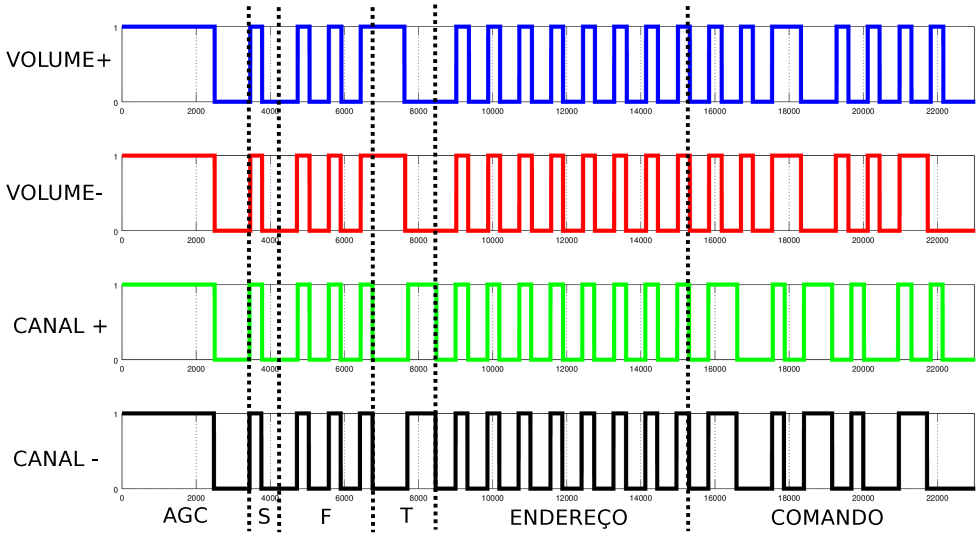

Figura 6: Mudança nos bits de comando após pressionar NÃO consecutivamente 4 botões diferentes.

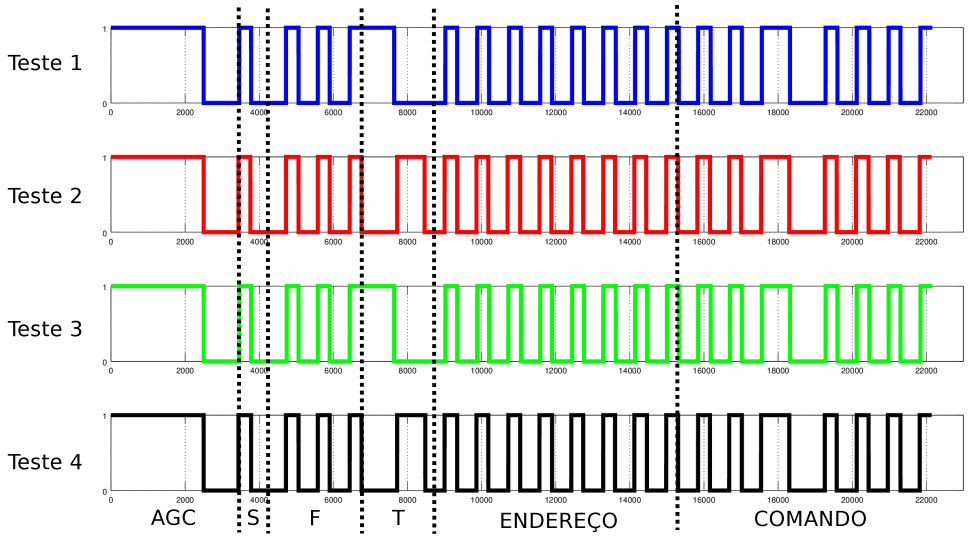

Figura 7: Mudança no bit de *toggle*(T) após pressionar o botão "volume+" por 4 vezes consecutivas.

5.2.2 Samsung

O controle da TV Samsung LT22B300LBMZD também foi usado para análise do sinal emitido à TV. A Figura 8 mostra a forma de onda quadrada obtida após o pressionamento consecutivo de 4 botões diferentes. Nota-se que os sinais são idênticos até o final do segundo bloco *custom*, o qual é imediatamente seguido pelos blocos de dados.

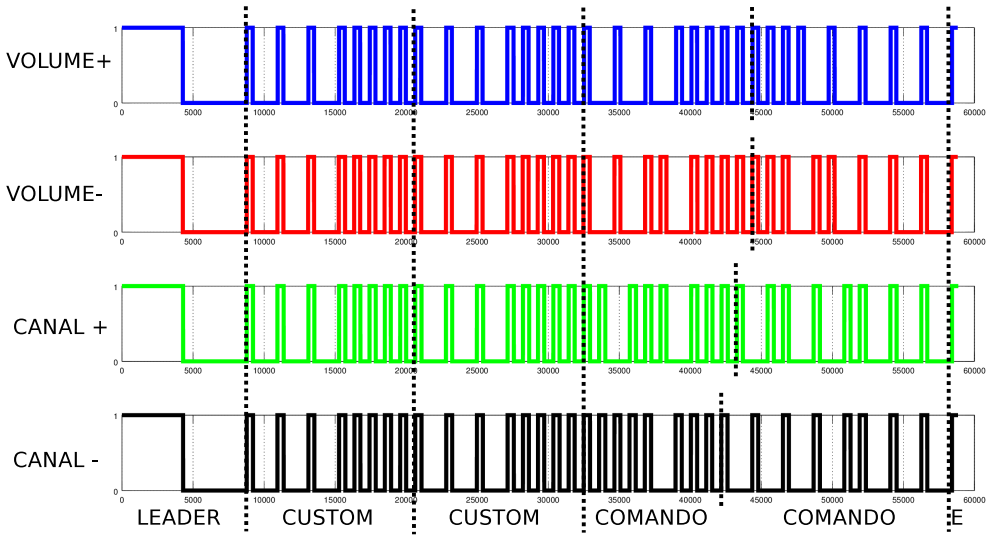

Figura 8: Mudança nos bits de comando após pressionar consecutivamente 4 botões diferentes.

5.3 Envio do Sinal Infravermelho

Nas seções seguintes, será mostrado que a atribuição de valores aos pinos da BeagleBone é feita através da escrita em arquivos por quaisquer linguagens de programação e que, apesar de prática e funcional, o tempo demandado para a execução desta tarefa não atende às exigências dos protocolos, os quais requerem faixas na ordem de microssegundos para modular os bits transmitidos. Em outras palavras, a BBB acabou sendo mais lenta do que o Arduino para emitir sinais à TV.

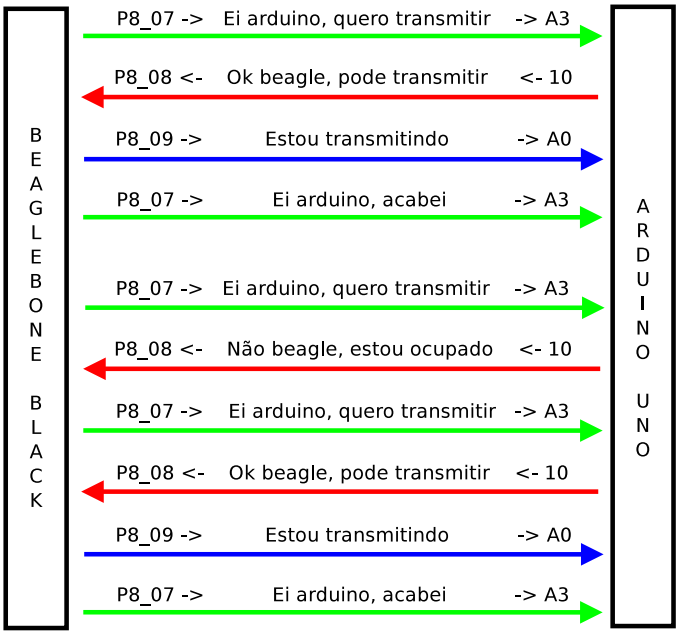

Figura 9: Protocolo de envio do *bitstream* da BeagleBone para o Arduino.

A BeagleBone continua sendo o servidor de voz e, de acordo com o resultado retornado pelo sistema de reconhecimento, é realizada a busca por um vetor de bits no banco de dados. Essa *bitstream* é transmitida bit a bit para o Arduino através de pinos, seguindo um protocolo tal qual o ilustrado na Figura 9. Há dois pinos de controle e um de dados em cada plataforma. Quando a BBB quer transmitir, um sinal alto é colocado no pino de controle. Imediatamente após o Arduino perceber esse sinal, o caminho para o bit de dados é aberto e uma resposta é dada à BBB também na forma de um sinal alto; caso o Arduino ainda esteja processando o bit de dados anterior, a *stream* de dados permanece fechada e um sinal baixo é enviado à BBB pelo canal de controle. Após finalizar o envio do bit, a BeagleBone coloca um sinal baixo no pino de controle. O processo é repetido para todos os bits do vetor.

Em posse do vetor, o Arduino passa cada bit a função que o modula e envia de acordo com o protocolo da TV em questão.

5.4 Banco de Dados MySQL

Como o sistema foi projetado para, futuramente, dar suporte ao controle de diversos aparelhos eletrônicos, a adoção de um banco de dados foi vista como solução para facilitar o acesso a uma maior variedade de dispositivos e, assim, tornar o projeto um sistema de propósito mais geral. Além disso, a segurança dos dados, que serão constantemente manipulados, será mantida, podendo estes ser recuperados apenas olhando para o banco.

Inicialmente, apenas a entidade TELEVISÃO foi criada, onde uma tabela intitulada “TV” contém atributos como a marca do aparelho e os comandos a serem transmitidos, como visto na Tabela 2. Com

isso, quaisquer campos podem ser satisfatoriamente acessados através de um simples código SQL (como o mostrado abaixo). Assim, pode-se recuperar os bits de referência para, posteriormente, decodificá-los de acordo com o protocolo do aparelho e, finalmente, construir a informação exata a ser passada para o IR LED.

```
SELECT campo FROM TV WHERE marca='marca_da_tv'
```

O acesso ao banco de dados é inteiramente intermediado pela linguagem C. Para tal, uma biblioteca especial chamada `mysql.h` foi utilizada, na qual as *queries* são realizadas pela passagem de um comando SQL como uma *string* (Vide Apêndice A).

Tabela 2: Exemplo hipotético de uma tabela entitulada 'TV' no banco de dados.

Marca	volume+	volume-	canal+	canal-	ligar
Lg	10111011	11110110	11000010	00010111	01011111
Sony	11110110	01011111	00011111	10111011	11000000
Samsung	00110111	11000010	11010000	11011110	11011100
Philips	00011111	10111011	11000000	11011010	11010111

5.5 GPIO

Como no Arduino, a BBB possui muitos pinos de entrada e saída para diversas funções, algo que o Raspberry Pi carece. Os GPIOs (*general purpose input/output*, ou pinos de entrada e saída de propósito geral) podem ser usados da forma mais conveniente ao desenvolvedor, sendo possível controlar a tensão de saída, receber *feedback* de sensores digitais e analógicos, etc. O processador possui 128 pinos de GPIO divididos em 4 módulos de 32 pinos cada. Porém, o mapeamento entre os pinos da BeagleBone Black e os do processador não é exatamente igual (da mesma forma que o número dos pinos do Arduino UNO não equivale aos do ATmega328, por exemplo). Portanto, é necessário saber como os pinos estão dispostos na placa.

Tabela 3: Mapeamento de Pinos do *Header* P8 da BeagleBone

Head pin	$Pins	Endereço	*offset*	N° GPIO	Nome	Modo7	Pino
1,2					DGND		
3	6	0x818	018	38	GPIO_6	gpio1[6]	R9
4	7	0x81c	01c	39	GPIO_7	gpio1[7]	T9
5	2	0x808	008	34	GPIO_2	gpio1[2]	R8
6	3	0x80c	00c	35	GPIO_3	gpio1[3]	T8
13	9	0x824	024	23	EHRPWM2B	gpio0[23]	T10
19	8	0x820	020	22	EHRPWM2A	gpio0[22]	U10

Como o objetivo não é prototipar a partir do processador, o mapeamento correto dos pinos é crucial para conseguir alguma saída válida dos pinos da placa. As Tabelas 3 e 4 mostram parte desse mapeamento.

Existem 3 formas de se acessar e controlar os pinos de GPIO. A primeira, através da IDE web que vem instalada no Ångström de fábrica, não é viável, visto que o sistema operacional foi trocado pelo Debian. A segunda consiste em iniciar o módulo e atribuir valores através de registradores, tal como explicado no *datasheet* [**?**], mas, por ser mais complexa, despenderia muito tempo e seria pouco portável. A última solução foi introduzida com o *kernel* 3.8.x do Linux embarcado, acessando as interfaces de GPIO através da simples atribuição de valores a arquivos. Tal processo só pode ser executado como *root*, então deve-se entender bem o que está fazendo antes de executar tais comandos.

Os arquivos de configuração do GPIO estão localizados no caminho `/sys/class/gpio`. Ao se modificar o arquivo `export`, o qual inicializa o pino, o diretório `gpiochipXX`, que contém os arquivos necessários para

Tabela 4: Mapeamento de Pinos do *Header* P9 da BeagleBone

Head pin	$Pins	Endereço	*offset*	N° GPIO	Nome	Modo7	Pino
1,2					DGND		
3,4					DC_3.3V		
5,6					VDD_5V		
7,8					SYS_5V		
9					PWR_BUT		
10					SYS_RESETn		A10
11	28	0x870	070	30	UART4_RXD	gpio0[30]	T17
12	30	0x878	078	60	GPIO1_28	gpio0[28]	U18

a configuração do pino, é criado. Apenas os arquivos `direction` e `value` são necessários para a tarefa de piscar um LED, os quais indicam se o pino será de entrada ou saída e o valor que será atribuído ao pino, respectivamente.

```
echo 51 > /sys/class/gpio/export # exporta o pino 51 / GPIO_19 / pino 16
echo out > /sys/class/gpio/gpiochip51/direction # define o pino como saída
for i in `seq 10`; do
    echo 1 > /sys/class/gpio/gpiochip51/value # seta o valor alto no pino
    sleep 1
    echo 0 > /sys/class/gpio/gpiochip51/value # seta o valor baixo no pino
    sleep 1
done
echo 51 > /sys/class/gpio/unexport # libera o pino 51 / GPIO_19 / pino 16
```

Os comandos listados acima são o suficiente para piscar um LED conectado ao pino 51, ou pino 16 do *header* P9.

6 Orçamento

Aqui é apresentado um orçamento aproximado de todos os materiais utilizados com base no preço de sites de venda *online* Amazon [1] e Alibaba [2] e com dados recolhidos por Statista em 2013 sobre o preço médio de *smartphones* com o sistema operacional Android. A Tabela 5 mostra os preços de cada material juntamente com o total aproximado em dólar e real.

Tabela 5: Preço dos materiais utilizados

Produto	USD (U$)	BRL (R$)	IOF (R$)	Total (R$)
1 × BBB	60.00	180.0	11.3	191.3
1 × Smartphone	280.0	850.0	54.2	904.2
1 × IR LED (Tx)	0.10	0.30	0.02	0.32
1 × Photosensor (Rx)	0.14	0.42	0.03	0.45
2 × LEDs (status)	0.12	0.36	0.02	0.38
1 × USB Headset ANDREA	40.00	120.0	7.66	127.66
Total	380.36	1152.08	73.42	1225,5

[1] `http://www.amazon.com`
[2] `http://www.alibaba.com`

7 Dificuldades e Soluções

1. O Ångström é o sistema operacional de fábrica da BeagleBone Black, bastante referenciado em fóruns e candidato principal para ser a base do projeto. Entretanto, o sistema e a nomenclatura de pacotes não correspondiam com os usados nas distribuições Debian, o que impossibilitava a instalação das dependências necessárias para o uso das ferramentas de processamento de voz. Além disso, o Ångström também consumia muito espaço de armazenamento na *flash*, ocupando sozinho 1.6 GB de 2 GB por conta de sua interface gráfica.

   A opção seguinte foi o Ubuntu, pois a similaridade do sistema de pacotes `apt-get` com a distribuição *desktop* tornaria a integração mais fácil. Porém, a instalação da versão 14.04 também ocupava espaço demais na memória *flash*: aproximadamente 1.2 GB. Além do problema de espaço, outra dificuldade foi a conexão com a Internet, já que o acesso convencional ao repositório de pacotes não funcionava.

   A solução veio com a instalação de um terceiro e último sistema operacional, o Debian 7.8 wheezy. Este se mostrou muito mais leve e "enxuto" em sua versão *minimal*, ocupando menos de 500 MB da memória. Além disso, todos os pacotes necessários para conexão com a Internet funcionaram perfeitamente.

2. A BeagleBone Black não possui saída de áudio nativa, tampouco conectores do tipo *audio jack*. No Debian, a saída padrão de áudio é pelo conector HDMI, mas pode ser substituída pelo USB mediante modificações em parâmetros do *kernel*. O modo mais fácil, considerando que redirecionar a saída do eSpeak para um GPIO seria muito trabalhoso, seria conectar um alto-falante USB à BeagleBone Black. Em se tratando de um protótipo, um *headphone* faz o papel de um alto-falante que deve consumir pouca energia e ter um tamanho limitado. Foi-se cogitada a construção de um circuito com um amplificador LM386 para o *speaker*, porém a obtenção de um D/A PCM2707 não custaria menos de US$ 15.

3. A BeagleBone Black não possui conexão *wifi*. Além da dificuldade em atualizar o *kernel* pra receber um *shield/cape*, o mesmo teria de ser conectado na porta USB, a qual já estaria sendo usada pelo alto-falante. Portanto, a solução mais fácil foi conectar um roteador *wifi* à porta Ethernet da BBB através de um cabo UTP.

4. O código criado para o Arduino faz um *dump* da informação captada pelo *photosensor*, verificando o tempo que ele passa ligado e desligado e salvando várias medidas de tempo numa matriz de inteiros, o que deixa o armazenamento dessa informação pouco viável. Após traduzir o código para que seja executado na BBB, o ideal seria converter os valores de tempo para um *array* de bits ("110101010", por exemplo) para que fosse mais fácil salvar no banco de dados. Isso já funciona, porém somente para o protocolo RC-6.

5. A função `mysql_query()` recebe uma *string* contendo o comando SQL para acessar o banco de dados MySQL. Caso se queira editar o valor de um atributo numa determinada tabela, por exemplo, basta passar o valor e o atributo como parâmetros para a função, o que exige que as *strings* sejam manipuladas constantemente a fim de tornar as requisições automáticas. Para que não se utilize mais espaços de memória do que necessário, optou-se pela alocação dinâmica através da função `malloc()`, auxiliada pela função `sprintf()`, a qual é responsável pela junção de *strings*. O código ainda está sob revisão, já que diversas falhas de segmentação vêm ocorrendo devido à erros no gerenciamento de memória. Além disso, o custo computacional provocado pelas diversas *queries* no banco ainda não pode ser previsto. Essa previsão é muito importante visto que, em uma futura utilização de um servidor Apache para acesso remoto de qualquer lugar via cliente PHP, por exemplo, o efeito *snowball* pode surgir como uma consequência negativa.

6. Como explicado na seção 4, um bit é representado por duas partes, sendo uma metade nível baixo e a outra nível alto. Para o RC-6, cada nível ocorre num intervalo de 16 ciclos (ou 32 ciclos para no

caso do protocolo RC-5), sendo o nível alto gerado pelo protocolo por modulação PWM com *duty cycle* mínimo de 25% do período do ciclo, tal como o mostrado na Figura 10.

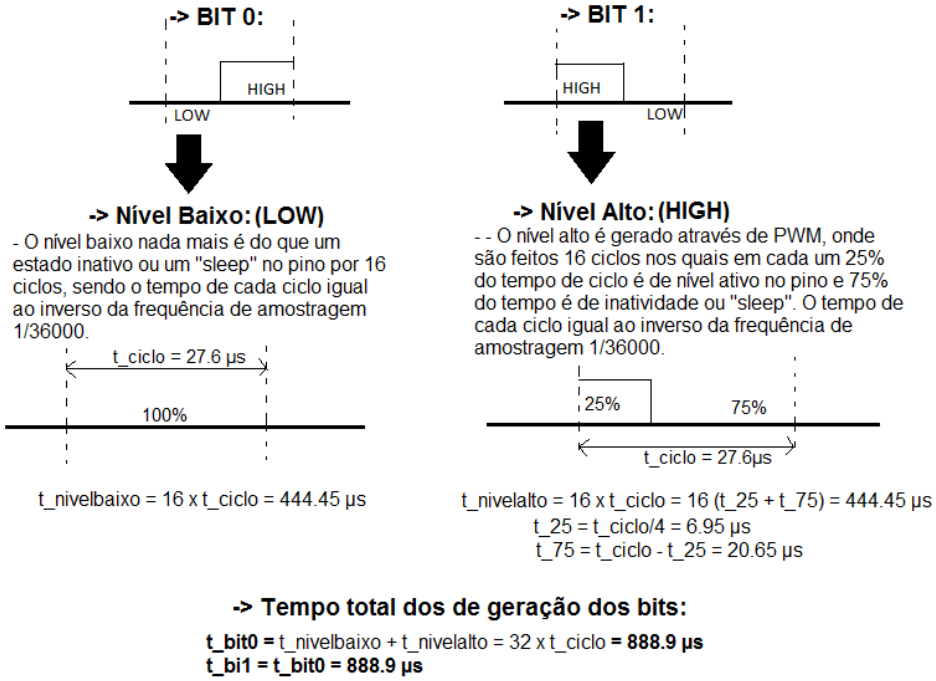

Figura 10: Processo de geração de cada bit

Porém, a BeagleBone Black leva um tempo de 42 μs para escrever HIGH no pino, isto é, para gerar a parte ativa do PWM, a plataforma responde com um tempo maior do que o próprio ciclo, o qual demora 27.6 μs aproximadamente. Levando-se em conta o tempo que a BeagleBone leva para escrever (chamado de t_{set}), de acordo com o esquema mostrado na Figura 11, a duração do bit seria de 111,6 μs.

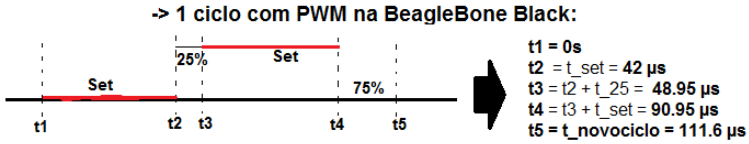

Figura 11: Ciclo com PWM na BeagleBone Black

Logo, este novo tempo de ciclo irá afetar diretamente na aquisição dos dados, pois reduzirá a frequência de amostragem e não será mais possível transmitir os comandos em infravermelho, fazendo uma conta de quanto irá impactar, tem se que:

$$1\ s \rightarrow 36k\ valores \rightarrow 27\ \mu s/valor$$

$$1\ s \rightarrow n\ valores \rightarrow 111.6\ \mu s/valor$$

Resolvendo a regra de 3 inversa, tem se:

$$n \approx 8.96k \rightarrow f_{coding} = 8.96\ kHz$$

$$\frac{f_s}{f_{coding}} = \frac{36k}{8.96k} \approx 4$$

Com isso pode se observar que a frequência de amostragem seria 4 vezes menor que a desejada.

8 Trabalhos Futuros

De acordo com o relato disponível em [**?**]: "Dado que o meu *home theater* é modesto, ele requer que eu consiga manejar 'apenas' 6 controles remotos para a simples tarefa de assistir a um filme". Seria ótimo se houvesse um controle remoto universal que permitisse acesso à *todos* os aparelhos do ambiente residencial, mesmo os que estão em cômodos diferentes. Ter o controle sempre consigo e poder usá-lo por uma tecnologia *hands free* mesmo quando estivesse fora de casa é um sonho de qualquer consumidor.

- Expandir para vários aparelhos, tornando a BeagleBone um servidor centralizado no ambiente doméstico. Isso também acarretará em um número maior e mais complexo de tabelas no banco de dados, contendo os diversos aparelhos e seus comandos;
- Em cada compartimento onde houvesse um aparelho eletrônico a ser controlado, haveria um microcontrolador (a ser avaliado, preferencialmente mais barato que o Arduino) capaz de controlar determinado(s) aparelhos;
- A BeagleBone e todos os outros microcontroladores estariam conectados à mesma rede LAN. Somente a BBB precisaria estar conectada à Internet, de modo que não houvesse limitação de distância para a conexão com o *smartphone*;
- Utilizar de forma eficiente o servidor Apache com o PHP, para que uma página *web* seja criada e por meio dela informações de análise do sistema possam ser guardadas e acessadas remotamente pelo desenvolvedor, para que haja um algoritmo de detecção e análise de problemas mais eficiente ao usuário. Essa nova funcionalidade também poderá disponibilzar um registro das atividades do usuário com o sistema, podendo ficar disponível para monitoramento em *high-level* ou para que os aparelhos possam ser controlados remotamente a uma distância maior do que a limitada pela LAN.

A LAMP

Com a sensível complexidade dos comandos usados para a comunicação com a TV devido ao número de campos e a diferença entre eles, foi-se adotado um banco de ados para armazenar informações sobre os aparelhos, bem como seus comandos. Isso evitará possíveis problemas de escalabilidade quando o sistema evoluir para o controle de diversos equipamentos eletrônicos.

Além disso, visando uma integração maior do sistema implementado no BeagleBone Black®, a adoção de um servidor Apache com PHP será necessária. Essa integração se dá pela criação de uma página web onde serão armazenados dados provenientes das interação do sistema, por exemplo, caso o usuário tenha ligado a TV, mudado de canal ou desligado o ar condicionado, todas essas informação poderão ir para um *log* de uma página web em que pode ser acessado pelo desenvolvedor para investigar a performance do sistema. Além do *log* de comandos executados pelo usuário, podemos guardar erros cometidos pelo Julius, ou eSpeak ou ainda falha no envio do comando para o aparelho eletrônico, em casos mais específicos o usuário poderá até ligar aparelhos remotamente ou o desenvolvedor avaliar o desempenho do sistema e identificando problemas antes de uma visita técnica.

Para integrar todos estes recursos é necessário um sistema operacional que, no caso, é um Linux Debian, configurando assim o servidor LAMP (Linux, Apache HTTP Server, MySQL e PHP). Após a instalação das dependências e subsequente configuração, o acesso e comunicação com o MySQL via códigos em C foi necessário. Para tal, a biblioteca `mysql.h` é usada. No código em C para a comunicação, foram criadas 8 funções específicas, sendo elas mostradas na Lista 2.

Listing 2: Definição das funções

```
#define HOST "localhost"
#define USER "root"
#define PWD  "beagle1234"
#define DB   "ir_db"

/* Get element from table ("vol_mais", "samsung", con) */
char *get_element(char *field, char *branch, MYSQL *con);
/* Insert element into table ("ch_mais", "010101010", "lg", con); */
int set_element(char *field, char *value, char *branch, MYSQL *con);
/* Insert row in the table ("marca", "samsung", con) */
int insert_table(char *field, char *value, MYSQL *con);
/* Grant database access to a specific user ("root", "ir_db", con) */
int grant_access(char *user, char *db, MYSQL *con);
/* Create a table in the database ("TV", con) */
int create_table(char *table, MYSQL *con);
/* Drop a table in the database ("TV", con) */
int drop_table(char *table, MYSQL *con);
/* Create a table in the database ("TV", con) */
int create_db(char *db, MYSQL *con);
/* Drop a table in the database ("TV", con) */
int drop_db(char *db, MYSQL *con);
/* Default exit if mysql_query() returns an error */
void finish_with_error(MYSQL *con);
/* Configure databased used in this work */
int setup();
```

A Lista 3 mostra a função `get_element()`, responsável por fazer a *fetch* no banco de dados. Ela retornará o valor do atributo específico que se deseja obter, recebendo como parâmetros a coluna e a marca, bem como a conexão com o MySQL. Com isso, pode-se selecionar o atributo *field* onde a marca é igual ao *branch* passado à função. Feito isso, a função `mysql_fetch_row()` é chamada para retornar todos os valores que satisfazem a condição exigida pela *query*, o qual, para esse caso especifico, será um valor único. Pode-se notar que o comando passado à consulta é simplemente um *select* filtrado com a cláusula *where*, tal como mostrado abaixo:

```
SELECT field FROM tv WHERE marca='branch'
```

Listing 3: Função `get_element()`

```
/* Get element from table, pass the column and the row ("vol_mais","philips",con) */
char *get_element(char *field, char *branch, MYSQL *con)
{
    char *str_value; // pedro: no malloc?
    char *cmd_arr[] = {"SELECT ", field, " FROM tv WHERE marca='", branch, "'"};

    /* compute size of each string on vector that'll compose the command for calloc */
    uint8_t i, len = 0;
    for(i=0; i<sizeof(cmd_arr)/sizeof(cmd_arr[0]); i++)
        len += strlen(cmd_arr[i]);

    /* allocate space to and compose the string command */
    char *cmd = (char *) calloc ((len+1), sizeof(char));
    for(i=0; i<sizeof(cmd_arr)/sizeof(cmd_arr[0]); i++)
        sprintf(cmd, "%s%s", cmd, cmd_arr[i]);

    if(mysql_query(con, cmd))
        finish_with_error(con);

    /* pedro: comenta aqui em uma linha soh */
    MYSQL_RES *result = mysql_store_result(con);
    if(result == NULL)
        finish_with_error(con);

    /* pedro: funciona sem alloc pra str_value? row[0] Ã© string? */
    MYSQL_ROW row;
    while((row = mysql_fetch_row(result)))
        str_value = (row[0] ? row[0] : "NULL");

    mysql_free_result(result);
    free(cmd);

    return (char *) str_value;
}
```

As demais funções basicamente chamam a função `mysql_query()`, a qual recebe como parâmetros a conexão com o banco e uma *string* contendo o comando SQL. A função `setup()`, mostrada na Lista 5 foi criada para configurar o banco de dados de acordo com as necessidades deste trabalho, utilizando tanto funções nativas da biblioteca usada quanto as criadas manualmente para algumas ações específicas. Essa função, se traduzida literalmente para seu código SQL, resultaria na sequência de comandos descritos na Lista 4.

Listing 4: Código SQL para configuração do MySQL

```
/*
 * 1st: connect to MySQL on bash terminal
 * root@beagle # mysql -u root -p
 * password: beagle1234
 */

DROP database if EXISTS ir_db; --delete database
CREATE database if NOT EXISTS ir_db; --create the previous deleted one
USE ir_db; --switch to the database created

DROP table if EXISTS tv; --delete table
-- create table with fields for the commands used
CREATE table if NOT EXISTS tv (
marca varchar(40) NOT null,
vol_mais varchar(40),
vol_menos varchar(40),
ch_mais varchar(40),
ch_menos varchar(40),
on_off varchar(40),
primary key(marca)
);

-- 35 bits command length
INSERT into tv(marca, vol_mais, vol_menos, ch_mais, ch_menos, on_off) values (
"samsung",
"11110000011100000111000000000111110", -- volume mais
"11110000011100000110100000001011110", -- volume menos
"11110000011100000010010000101101110", -- canal mais
"11110000011100000000010000111101110", -- canal menos
"11110000011100000010000000101111110" -- on/off
);

-- 22 bits command length
INSERT into tv(marca, vol_mais, vol_menos, ch_mais, ch_menos, on_off)
values (
"philips",
"1100010000000000010000", -- volume mais
"1100010000000000010001", -- volume menos
"1100010000000001001100", -- canal mais
"1100010000000001001101", -- canal menos
"1100010000000000001100" -- on/off
);

-- get_element() in C
SELECT vol_mais FROM tv WHERE marca='samsung';
SELECT ch_menos FROM tv WHERE marca='philips';
```

Listing 5: Função `setup()`

```
/* Configure databased used in this work */
int setup()
{
    /* init connection */
    MYSQL *con = mysql_init(NULL);
    if(con == NULL)
        finish_with_error(con);

    /* connect to MySQL (mysql -u root -p [beagle1234]) */
    if(mysql_real_connect(con, HOST, USER, PWD, NULL, 0, NULL, 0) == NULL)
        finish_with_error(con);

    drop_db(DB, con); //drop/delete database
    create_db(DB, con); //create database

    /* change database (USE ir_db) */
    if(mysql_select_db(con, DB))
        finish_with_error(con);

    drop_table("tv", con); //drop/delete table
    create_table("tv", con); //create table
    /* set branch on primary key and values for remaining fields */
    insert_table("marca", "samsung", con);
    set_element("vol_mais", "11110000011100000111000000000111110", "samsung", con);
    set_element("vol_menos","11110000011100000110100000001011110", "samsung", con);
    set_element("ch_mais", "11110000011100000010010001011011110", "samsung", con);
    set_element("ch_menos", "11110000011100000000010001111011110", "samsung", con);
    set_element("on_off", "11110000011100000010000001011111110", "samsung", con);

    /* fetch bistream: just checking out (select) */
    printf("Samsung Vol+ = '%s'\n", get_element("vol_menos", "samsung", con));

    /* close MySQL connection */
    mysql_close(con);

    return EXIT_SUCCESS;
}
```

B Recebimento e Envio de Sinais Infravermelhos

Listing 6: Função `ir_dump()`

```
/* */
void ir_dump(void)
{
    uint16_t highpulse, lowpulse;
    highpulse = lowpulse = 0;

    /* Leitura direta do pino */
    while (IRpin_PIN & (1 << IRpin)) {
        highpulse++;
        delayMicroseconds(RESOLUTION);

        /* Se pulso muito longo, ou nada ocorreu ou o codigo terminou */
        /* Mostra os tempos de pulso no monitor serial */
        if ((highpulse >= MAXPULSE) && (currentpulse != 0)) {
            printPulses(); //mostra no Serial monitor
            currentpulse=0;
            return;
        }
    }
    /* Pulso nao foi muito longo: armazena na matriz */
    pulses[currentpulse][0] = highpulse;

    /* Leitura direta do pino */
    while (! (IRpin_PIN & _BV(IRpin))) {
        lowpulse++;
        delayMicroseconds(RESOLUTION);
        if ((lowpulse >= MAXPULSE) && (currentpulse != 0)) {
            printPulses(); //mostra no Serial monitor
            currentpulse=0;
            return;
        }
    }
    pulses[currentpulse][1] = lowpulse;

    /* Pulso lido corretamente */
    currentpulse++;
}
```

Listing 7: Função `mark()`

```
/* Envia um pulso de 38kHz para o pin por x ms, reproduzindo o comando decodificado (PWM) */
void mark(long microsecs) {
    cli(); // Desativa interrupcoes de background
    while (microsecs > 0) {
        // 38 kHz eh aproximadamente 13 us high e 13 us low
        digitalWrite(IRledPin, HIGH); // digitalWrite dura 3 us
        delayMicroseconds(10);       // espera 10 us
        digitalWrite(IRledPin, LOW); // 3 us
        delayMicroseconds(10);       // espera mais 10 us

        /* subtrai 3+10+3+10 us do pulso */
        microsecs -= 26;
    }
    sei(); // Ativa as interrupcoes
}
```

C Protocolo de Comunicação Duplex entre Arduino e BBB

Como já mencionado, muitos problemas impediram que o envio do sinal IR fosse implementado diretamente na BeagleBone. Assim, a solução adotada foi transmitir os bits de comando, através de pinos, do servidor para o Arduino, o qual possui um código bem funcional para o envio de sinais IR via PWM.

C.1 Rx: UNO

Listing 8: Código do Arduino, o qual recebe um *bitstream* de 34 bits

```
/**/
void receive_from_bbb()
{
    for(i=0; i<34; i++)
        command[i] = 1; // inicializa a bitstream antes de receber um novo

    i=0;
    while(i != 34) {
        transmissionSignal = LOW;
        digitalWrite(transmissionPin, transmissionSignal);
        /* Listening: ei BBB, quer transmistir bit? */
        controlSignal = analogRead(cchannelPin);
        if(controlSignal > 690) { // BBB quer!
            /* Speaking: Ei BBB, pode mandar o bit! */
            transmissionSignal = HIGH;
            digitalWrite(transmissionPin, transmissionSignal);
        }

        /* BBB quer transmitir .:. controlSignal = HIGH */
        /* UNO pronto pra receber .:. transmissionSignal = HIGH */
        while(transmissionSignal == HIGH) {
            /* Listening: ei BBB, terminou de mandar o bit? */
            controlSignal = analogRead(cchannelPin);
            /* Listening: ei BBB, manda o bit no canal de dados */
            dataSignal = analogRead(dchannelPin);
            if(controlSignal<100) { // BBB terminou!
                /* Speaking: Ei BBB, nao manda bit, pois estou ocupado! */
                transmissionSignal = LOW;
                digitalWrite(transmissionPin, transmissionSignal);
                if(dataSignal<100) // se data eh LOW, bit eh 0
                    command[i] = 0;
                i++; // Jah que leu 1 bit, incrementa o contador e quebra o loop
                break;
            }
        }
    }

    /* Imprime comando recebido no Serial Monitor */
    for(i=0; i<34; i++)
        Serial.print(command[i]);
    Serial.println("");
}
```

C.2 Tx: BBB

Listing 9: Código em C, o qual envia um *bitstream* de 34 bits

```
void send2uno(char *stream)
{
    int i, count, limit=0;
    char rx;

    /* activate pins */
    gpio_export(tx_ctrl);
    gpio_export(tx_data);
    gpio_export(rx_ctrl);

    /* pinMode() like function */
    gpio_set_dir(tx_ctrl, OUTPUT_PIN);
    gpio_set_dir(tx_data, OUTPUT_PIN);
    gpio_set_dir(rx_ctrl, INPUT_PIN);

    for(i=0; i<strlen(stream); i++) {
        count = 0;
        /* BBB: Hey arduino, eu quero transmitir */
        gpio_set_value(tx_ctrl, HIGH);
        do {
            usleep(100);
            if(++count==30) // desiste apos 30 tentativas
                limit = 1;
        } while(!limit && (rx = gpio_get_value(rx_ctrl)) == '0');

        /* Tentei 30x e voce negou (rx='0'), entÃ£o desisti */
        if(limit)
            break;

        /* BBB: To enviando */
        if(stream[i]=='0') {
            gpio_set_value(tx_data, LOW);
        } else {
            gpio_set_value(tx_data, HIGH);
        }

        /* BBB: Acabei de enviar */
        gpio_set_value(tx_ctrl, LOW);
        usleep(300);
    }

    /* deactivate pins */
    gpio_unexport(tx_ctrl);
    gpio_unexport(tx_data);
    gpio_unexport(rx_ctrl);
}
```