Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica

Luis Felipe Wolf Batista

Implementação, Controle e Monitoração de uma Plataforma Móvel Utilizando Linux Embarcado

São Carlos
2013

Luis Felipe Wolf Batista

Implementação, Controle e Monitoração de uma Plataforma Móvel Utilizando Linux Embarcado

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos como parte dos requisitos para a obtenção do título de Engenheiro de Computação.

Área de concentração: Sistemas Embarcados

ORIENTADOR: Prof. Dr. Evandro Luís L. Rodrigues

São Carlos
2013

Batista, Luis Felipe Wolf
B333i Implementação, controle e monitoração de uma plataforma móvel utilizando Linux embarcado. / Luis Felipe Wolf Batista; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2013.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

1. Microcontroladores. 2. Sistemas Embarcados. 3. Linux Embarcado. 4. BeagleBone. 5. Arduino. 6. Robô Móvel. 7. Rede sem fio. I. Título.

FOLHA DE APROVAÇÃO

Nome: Luis Felipe Wolf Batista

Título: "Implementação, controle e monitoração de uma plataforma móvel utilizando Linux embarcado"

Trabalho de Conclusão de Curso defendido em 12/11/2013.

Comissão Julgadora:	**Resultado:**
Prof. Associado Evandro Luís Linhari Rodrigues Orientador - SEL/EESC/USP	APROVADO.
Prof. Dr. Júlio Cézar Estrella SSC/ICMC/USP	APROVADO.
Prof. Dr. Valdir Grassi Júnior SEL/EESC/USP	Aprovado

Coordenador pela EESC/USP do Curso de Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Onde quer que você esteja, você sempre estará lá.

Desconhecido

À minha família,
por todo apoio, confiança e incentivo

Agradecimentos

Aos meus pais, Edilson e Lucia, que sempre me incentivaram e apoiaram.
À minha irmã, Luciana, que sempre foi um exemplo a ser seguido.
À minha namorada, Carolina, pelo carinho, paciência e compreensão.
Ao Prof. Evandro, que me orientou, aconselhou e motivou ao longo do projeto.
Aos meus colegas, Juliano e André, pelo companheirismo e ajuda.
Aos amigos que fiz durante a graduação, pelos momentos de alegria.
Aos professores, pelas lições ensinadas, mesmo que, por vezes, da maneira difícil.

Resumo

Com a rápida evolução dos sistemas embarcados, as plataformas que utilizam Linux embarcado têm ganhado bastante espaço no mercado e têm se tornado cada vez mais populares. A utilização de sistemas operacionais completos em ambientes embarcados aumenta significativamente o poder de processamento e a gama de possibilidades que podem ser exploradas. Nesse projeto é desenvolvida uma plataforma móvel que pode ser controlada remotamente por um computador e possui a capacidade de tomar algumas decisões de forma automática e independente, como por exemplo frear evitando a colisão com obstáculos próximos. Com a utilização de um microcontrolador e uma plataforma com Linux Embarcado, o monitoramento e controle da plataforma móvel pode ser feito de maneira relativamente simples. Para alcançar tais objetivos é utilizado um Arduino em conjunto com uma BeagleBone. O computador remoto estabelece uma comunicação com a BeagleBone, a BeagleBone troca dados com o Arduino que, por sua vez, controla os sensores e atuadores permitindo monitorar e controlar a plataforma móvel.

Palavras-chave: Microcontroladores, Sistemas Embarcados, Linux Embarcado, BeagleBone, Arduino, Robô Móvel, Rede sem fio.

Abstract

With the rapid development of embedded systems, platforms that use Embedded Linux has gained enough market space and have become increasingly popular. The use of complete operating systems in embedded environments significantly increases the processing power and the range of possibilities that can be explored. In this project is developed a mobile platform that can be remotely controlled by a computer and has the ability to make some decisions automatically and independently, such as avoiding collisions with nearby obstacles. By using a microcontroller and Embedded Linux platform, the monitoring and control of the mobile platform can be accomplished in a relatively simple manner. To achieve these goals an Arduino is used in conjunction with a BeagleBone. The remote computer establishes communication with the BeagleBone, the BeagleBone exchanges data with Arduino, which controls the sensors and actuators making it possible to monitor and control the mobile platform.

Keywords: Automation, Microcontrollers, Embedded Systems, Embedded Linux, BeagleBone, Arduino, Robotics, Wireless Networks.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas

OS – *Operating System*
ARM – *Advanced RISC Machines*
USB – *Universal Serial Bus*
GPIO – *General Pourpose Input Oputput*
SSH – *Secure Shell*
UDP – *User Datagram Protocol*
TCP – *Transfer Control Protocol*
UART – *Universal Asynchronous Receiver/Transmitter*
PWM – *Pulse-width Modulation*
SD – *Secure Digital*
LCD – *Liquid Crystal Display*
I2C – *Inter-Integrated Circuit*
CAM – *Controller Area Network*
HTTP – *Hypertext Transfer Protocol*
DNS – *Domain Name System*
USB – *Universal Serial Bus*

Sumário

Capítulo 1

Introdução

1.1 Introdução

Antigamente, o *software open source* era visto com um certo ceticismo, mas com o passar do tempo tornou-se evidente que esse modelo de desenvolvimento já está bastante solidificado e tem se mostrado muito eficiente. Atualmente, o mesmo tem acontecido com o *hardware*. Há mais de uma década, houve uma explosão de empresas que publicaram parte do seu código fonte, permitindo assim o desenvolvimento de plataformas de *hardware open source*[12].

Uma pesquisa sobre o impacto do Software Livre e de Código Aberto (SL/CA) na Indústria de Software do Brasil realizada pelo Observatório Econômico da Sociedade Softex em parceria com o Departamento de Política Científica e Tecnológica da Unicamp[1], apontou que os canais de comunicação proporcionados pela Internet levaram a emergência de oportunidades de exploração de economias para a indústria de *software*.

Percebe-se que o mesmo está acontecendo com a indústria de *hardware*. Um ótimo exemplo é o caso do Arduino[5], uma plataforma de prototipagem eletrônica de *hardware* livre que ganhou muita força devido a sua característica de código aberto e sua comunidade *online* extremamente ativa e colaborativa.

O *hardware open source* não está limitado apenas ao escopo de microcontroladores. Com o desenvolvimento da eletrônica e a modernização dos sistemas embarcados, os benefícios do *hardware* aberto têm conquistado um poderoso aliado: o Linux.

Muitos desenvolvedores de sistemas embarcados que historicamente têm baseado suas aplicações em plataformas de microcontroladores estão descobrindo que a utilização de sistemas com Linux abre portas para um conjunto muito rico e diversificado de módulos de *software* e *plug-ins* sem nenhum custo e prontos para serem utilizados[2]. Uma vasta gama de interfaces e pilhas de comunicação como TCP/IP, USB e muitos outros estão disponíveis. Estas pilhas e *drivers* costumam ser amplamente utilizados e, como resultado, são bastante robustos. Na maioria dos casos, os problemas potenciais já foram encontrados por outros desenvolvedores que compartilharam as suas descobertas com o

resto da comunidade.

Outra vantagem do desenvolvimento em Linux embarcado é que o Linux fornece uma camada de abstração subjacente acima do *hardware*[2]. Ao contrário do desenvolvimento baseado na configuração direta de registradores, onde a mudança para uma nova família de microcontroladores significaria reescrever grande parte do *software* já desenvolvido. Uma mudança para um processador diferente em sistemas com Linux não exige redesenvolvimento de *firmware*, em nível de sistema ou aplicativo de *software*.

O rápido avanço do *software* de código aberto criou a necessidade de *hardware open source*, que tem ajudado estudantes, desenvolvedores e entusiastas a criarem projetos inovadores de grande impacto na sociedade de hoje. Em especial, grande parte desses projetos envolvem o desenvolvimento de plataformas robóticas.

A robótica é um ramo educacional e tecnológico que trata de sistemas compostos por partes mecânicas controladas por circuitos eletrônicos e por lógicas de programação. Dessa forma os sistemas mecânicos podem ser controlados manualmente ou automaticamente, permitindo a interação com o mundo físico.

O ato de construir e programar um robô exige a combinação de conhecimentos de diversas áreas, o que dá à robótica um caráter multidisciplinar. Devido a essa característica, a robótica é uma ótima ferramenta de auxílio ao ensino. Mesclando a teoria com a prática ela é capaz de estimular nos alunos conceitos como: capacidade de solucionar problemas, senso crítico, integração de disciplinas, criatividade, trabalho em equipe, autodesenvolvimento e etc.

Neste trabalho é abordado o desenvolvimento de uma plataforma móvel utilizando Linux Embarcado. Essa plataforma móvel pode ser controlada remotamente por um computador.

1.2 Objetivos

Tendo em vista o cenário atual para desenvolvimento em plataformas embarcadas, o principal objetivo deste trabalho é aprofundar o conhecimento necessário para a integração de *software* de alto nível com sistemas eletrônicos de baixo nível. Por isso é proposto o desenvolvimento de uma plataforma móvel, que possa ser explorada didaticamente e que apresente as seguintes características:

- Utilização de *hardware open source*;
- Utilização de Linux embarcado;
- Controlável por um computador remoto, por meio de uma rede;
- Altamente personalizável.

1.3 Justificativa

O desenvolvimento de uma plataforma móvel baseada em *hardware open source* é uma alternativa relativamente barata que permite aprofundar o conhecimento teórico e prático envolvido. O desenvolvimento completo da plataforma móvel exige o estudo e conhecimento do projeto como um todo. Isso envolve desde a utilização de componentes eletrônicos de baixo nível até a programação de interfaces gráficas e protocolos de comunicação através da rede.

1.4 Organização do trabalho

Este trabalho está estruturado em capítulos, sendo eles divididos da seguinte maneira:

- **Fundamentação Teórica:** apresentação dos conceitos relevantes para o correto entendimento do presente trabalho;
- **Desenvolvimento:** apresentação dos materiais utilizados, de como foi feita a montagem da plataforma móvel, da configuração dos componentes e dos métodos utilizados no trabalho;
- **Resultados:** apresentação e discussão dos resultados obtidos;
- **Conclusão:** validação dos objetivos do trabalho e conclusões finais.

Capítulo 2

Fundamentação Teórica

A elaboração deste projeto envolve a utilização de diferentes componentes, cada um com suas funcionalidades e particularidades. Nas seções a seguir será desenvolvido um breve embasamento teórico de alguns dos componentes e tecnologias utilizados ao longo do desenvolvimento do projeto.

2.1 BeagleBone

A BeagleBone[6] é uma plataforma aberta pronta para uso e prototipagem rápida de *hardware*, desenvolvimento de *software* e *firmware*. Lançada nos Estados Unidos em 2008, BeagleBoard.org é uma comunidade *open-source* que fornece aos desenvolvedores e entusiastas os recursos de que precisam para desenvolver rapidamente novos produtos para o mercado e, ao mesmo tempo, reduzir seus riscos[6].

Seguindo o sucesso das ferramentas de desenvolvimento BeagleBoard e BeagleBoard-XM, a BeagleBoard.org criou a BeagleBone, que pode facilitar para uma equipe de desenvolvimento a transição para o mundo *open source*[2]. Para aqueles que já participam da comunidade *open source*, a BeagleBone pode funcionar como uma plataforma de desenvolvimento estável, totalmente disponível e suportada pelo universo diversificado de recursos de código aberto, bem como o sistema de suporte do site BeagleBoard.org.

A comunidade *open source* hospeda os mais recentes desenvolvimentos de *software*, fóruns e *chats* ao vivo e interativos que colaboram para um fácil desenvolvimento de soluções na plataforma. A organização também criou várias plataformas de *hardware* para ajudar a simplificar desenvolvimentos baseados em ARM[19].

Uma característica interessante da BeagleBone é que ela pode ser ligada a uma BeagleBoard ou qualquer computador Linux via USB ou Ethernet e operar como um módulo de expansão para ele. Além disso a BeagleBone permite a expansão em módulos conhecidos como *capes*. Esse conceito é bastante parecido com as *shields* do Arduino, permitindo adicionar periféricos de maneira simples e altamente customizável.

Existem diversas plataformas de *hardware opensource* disponíveis no mercado, como por exemplo a Raspberry Pi, que tem ganhado bastante visibilidade. Algumas das vantagens da Raspberry Pi incluem um alto poder de processamento de imagens e uma comunidade extremamente ativa. Entretanto, existem algumas características da BeagleBone que justificam a sua escolha em detrimento da *Raspberry Pi*. Essas características envolvem a facilidade de expansão com a utilização das *capes* e a grande quantidade de interfaces para comunicação com sensores e outros dispotivos de baixo nível.

Essas características fazem da plataforma um sucesso em aplicações como por exemplo: rede de robôs autônomos sem fio, *kits* de educação eletrônica, dispositivos de jogos, automação de residências entre outras. BeagleBone é uma plataforma de desenvolvimento profissional adequada para engenheiros, designers e desenvolvedores, e também amadores.

O sistema operacional padrão que é distribuído junto com a BeagleBone é conhecido como Ångström[3]. De acordo com o manual da distribuição[14], o sistema operacional foi iniciado por um pequeno grupo de pessoas que trabalharam nos projetos *OpenEmbedded*, *OpenZaurus* e *OpenSimpad* com o intuito de unificar os esforços e disponibilizar uma distribuição estável e amigável para sistemas embarcados como *handhelds*, *set-top boxes*, equipamentos de armazenamento conectados na rede e outros.

Além de executar o kernel Linux, muitos ambientes de desenvolvimento são suportados pela comunidade e desenvolvedores, tais como Android, OpenEmbedded, QNX, Ubuntu, Symbian, Debian, Fedora, Gentoo, FreeBSD e outros.

2.2 Arduino

O Arduino[5] foi projetado com a finalidade de ser de fácil entendimento, de fácil programação e de fácil aplicação, além de ser multi plataforma, podendo ser configurado em ambientes Linux, Mac OS e Windows. Além disso, um grande diferencial deste dispositivo é ser mantido por uma comunidade que trabalha na filosofia *open source*, desenvolvendo e divulgando gratuitamente seus projetos[18].

De acordo com o site oficial[5], o Arduino é uma plataforma de prototipagem eletrônica de *hardware* livre, com suporte de entrada e saída embutido e uma linguagem de programação padrão. O objetivo desse projeto de *hardware* livre é criar ferramentas que são acessíveis, com baixo custo, flexíveis e de fácil utilização por artistas e amadores.

O Arduino é utilizado em vários programas educacionais ao redor do mundo, particularmente por designers e artistas que desejam desenvolver os projetos com facilidade sem a necessidade de um entendimento mais profundo dos detalhes técnicos que estão por trás das suas criações[16].

Arduino é melhor conhecido pelo seu *hardware*, mas também precisa de *software* para programar esse *hardware*. Tanto o *software* como o *hardware* são chamados de "Arduino". A combinação deles permite a criação de projetos que interagem com o mundo físico[16].

2.3 User Datagram Protocol

O *User Datagram Protocol* (UDP) é um protocolo da camada de transporte. Ele é descrito na RFC 768 e permite que a aplicação escreva um datagrama encapsulado em um pacote IPv4 ou IPv6, e enviá-lo ao destino desejado. Entretanto, não há qualquer tipo de garantia que o pacote será entregue ao destinatário final.

O protocolo UDP tem a característica de não implementar a confiabilidade. Caso garantias sejam necessárias, é preciso implementar uma série de estruturas de controle, que podem utilizar, por exemplo, *timeouts*, retransmissões, *acknowlegments*, controle de fluxo, e outras. Cada datagrama UDP tem um tamanho e pode ser considerado como um registro indivisível, diferentemente do TCP, que se trata de um protocolo orientado a fluxos de *bytesi* sem início e sem fim[13].

Podemos dizer que o UDP é um serviço não orientado à conexão, pois não há necessidade de manter um relacionamento longo entre cliente e servidor. Dessa forma, um cliente UDP pode utilizar um *socket* para enviar um datagrama a um servidor e imediatamente enviar outro datagrama a partir do mesmo *socket* para um servidor diferente. De forma parecida, um servidor pode ler datagramas vindos de diversos clientes, usando um único *socket*.

Comparando com o IP, o protocolo UDP inclui poucos serviços adicionais. Entre eles podemos citar a verificação de erros (*Checksum*), o suporte à multiplexação e demultiplexação e também o suporte a *broadcast* e *multicast*. Se o programador achar necessário, outros serviços, como o controle de congestionamento e a entrega confiável de dados, podem ser implementados na camada de aplicação.

Algumas das vantagens do protocolo UDP em relação ao TCP são:

Ausência de estabelecimento de conexão: Para estabelecer a conexão, o TCP utiliza a apresentação de três vias antes de transferir dados. Já o UDP simplesmente os envia, sem preocupar-se com o estabelecimento da conexão, garantindo assim maior velocidade.

Maior controle no nível da aplicação: No caso do protocolo TCP, há mecanismos de controle de congestionamento e entrega confiável de dados que pode, por exemplo, reenviar os segmentos até a recepção ser reconhecida. Já no protocolo UDP isso não acontece pois o datagrama é apenas empacotado e repassado para a camada de rede, permitindo que o programador tenha um melhor controle dos pacotes que estão sendo enviados pela rede em nível de aplicação.

Sobrecarga de cabeçalho inferior: Devido à ausência de mecanismos de controle de confiabilidade de transmissão, o protocolo UDP pode transmitir segmentos com cabeçalhos reduzidos. No caso do TCP o cabeçalho possui um tamanho de 20 *bytes*, enquanto o UDP possui apenas 8 *bytes*.

Ausência de estados de conexão: O TCP utiliza parâmetros de controle de congestionamento, *buffers* de envio e recebimento, números de sequência e reconhecimento para manter o estado da conexão. Como o UDP não mantém estado, ele utiliza menos recursos suportando mais clientes ativos.

Devido a essas características o UDP é uma escolha adequada para fluxos de dados em tempo real principalmente em aplicações sensíveis a atraso na rede. Como existe a possibilidade do corrompimento de parte de seu conteúdo, a verificação de erros necessária fica por conta do programador.

Capítulo 3

Desenvolvimento

3.1 Materiais

Para a realização desse projeto foram utilizados diversos materiais. Nesta Seção é feita uma breve descrição de cada um dos componentes utilizados bem como as suas especificações técnicas principais. Também é importante ressaltar que o Sistema Operacional utilizado durante todo o desenvolvimento do projeto foi o Mac OS X Versão 10.7.5, portanto, parte dos comandos descritos são específicos do ambiente Unix. Obviamente isso não se aplica aos outros comandos que foram executados na BeagleBone com o sistema operacional Ångström instalado.

3.1.1 Arduino Leonardo

A versão do Arduino utilizada nesse projeto é conhecida como Arduino Leonardo[4], que pode ser visto na Figura 3.1.

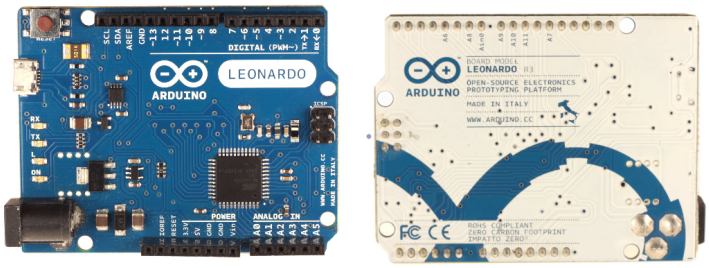

Figura 3.1: Arduino Leonardo.

O Arduino Leonardo é uma placa microcontroladora baseada no chip ATmega32u4. A principal diferença entre o Leonardo e suas versões precedentes é que o ATmega32u4 possui comunicação USB embutida, eliminando a necessidade de um processador secundário[4]. Entretanto, essa característica também possui outras implicações, como, por exemplo, a maneira como deve ser programada a comunicação serial.

A Tabela 3.1 resume as especificações técnicas do Arduino Leonardo.

Tabela 3.1: Características do Arduíno Leonardo

Microcontrolador	ATmega32u4
Tensão de operação	5V
Tensão de entrada (recomendado)	5 V
Tensão de entrada (limites)	7-12 V
Pinos digitais de entrada e saída	20
Canais PWM	7
Canais de entrada analógica	12
Corrente DC por pino de entrada/saída	40 mA
Corrente DC para o pino de 3.3V	50 mA
Memória Flash	32 KB (ATmega32u4)
SRAM	2.5 KB (ATmega32u4)
EEPROM	1 KB (ATmega32u4)
Clock Speed	16 MHz

Esse modelo foi escolhido devido ao baixo preço e a disponibilidade no mercado. Qualquer um dos outros modelos poderiam ter sido utilizados, com uma necessidade apenas de ajustar poucos detalhes da programação.

3.1.2 BeagleBone

A solução de *hardware* utilizada neste projeto para executar uma distribuição de Linux embarcado foi a BeagleBone Rev A6[7] que pode ser vista na Figura 3.2.

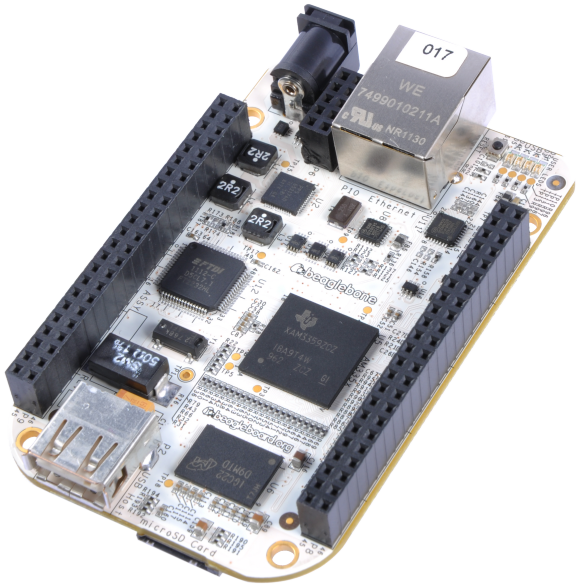

Figura 3.2: BeagleBone Rev A6.

A placa vem de fábrica configurada com uma distribuição de Linux conhecida com Ångström, descrita na Seção 2.1. Para a implementação da plataforma móvel foi mantida a mesma distribuição, sendo feita apenas a atualização do kernel para a última versão disponível.

Para acomodar uma variedade de sensores, controles e outros tipos de interfaces, ela possui dois conectores de 46 pinos para expansão, indicados no manual como P8 e P9. Esses pinos possuem diversas utilidades. De maneira geral podemos citar: 66 pinos de GPIO, sinais de LCD, um barramento de memória paralela, dois barramentos I2C, cinco UARTs, uma porta SPI, uma porta serial, I2S/ AC97-capable bus CAN, 6 PWMs, temporizadores múltiplos, além de 7 conversores analógico/digitais.

Na Figura 3.3 é possível verificar o diagrama de blocos da BeagleBone.

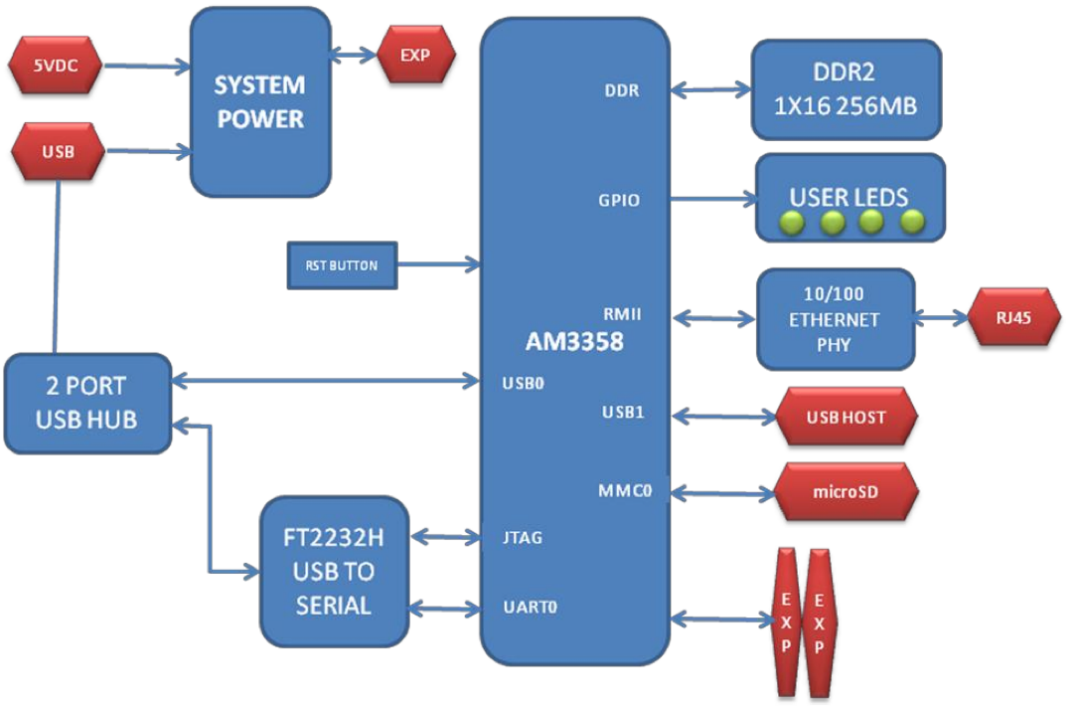

Figura 3.3: Diagrama de Blocos da BeagleBone extraído do *datasheet*[7].

A Tabela 3.2 resume algumas características da BeagleBone.

Tabela 3.2: Características da BeagleBone.

Processador	AM3359 500MHZ-USB 720MHZ-DC	
Memória	256MB DDR2 400MHZ	
Alimentação	USB	5V Conector Externo
Indicadores	Power	4 Led Controlados pelo usuário
Peso	39.68 gramas	

3.1.3 Conversor de Nível Lógico

Para a troca de informações entre o Arduino e a BeagleBone foi escolhida a comunicação serial. Devido ao fato do nível lógico do Arduino ser de 5 volts e da BeagleBone ser de 3.3V não é possível conectar os pinos Rx e Tx diretamente entre as placas. Para contornar esse problema foi utilizado um conversor de nível lógico entre elas, permitindo assim, que fosse estabelecida a comunicação.

O conversor lógico utilizado nesse projeto foi o BOB-08745 projetado e disponibilizado pela SparkFun. Este conversor pode ser visto na Figura 3.4.

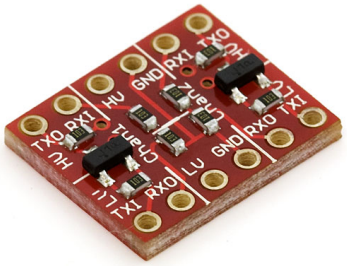

Figura 3.4: Conversor de Nível Lógico BOB-08745.

3.1.4 Adaptador Wiereless

Para adicionar a capacidade de comunicação sem fio na plataforma móvel, foi utilizado um Adaptador Wireless EW-7811Un[15]. Sua escolha foi devido ao chipset utilizado, que já possui *drivers* de fácil instalação no Linux, além de ter um tamanho extremamente reduzido. O adaptador utilizado pode ser visto na Figura 3.5.

Figura 3.5: Adaptador Wireless USB Edimax EW-7811Un.

3.1.5 Plataforma de Locomoção

Para o desenvolvimento da plataforma móvel foi utilizado um carro de controle remoto produzido pela *RadioShack* (modelo 60-4208). Esse carrinho é uma miniatura do Porsche 911 GT1 e pode ser visto na Figura 3.6.

Figura 3.6: Carro de controle remoto (*RadioShack* 60-4208).

Este carrinho foi desmontado e toda a parte eletrônica foi removida, sobrando apenas a estrutura mecânica e o motor de tração traseiro. Dessa forma foi possível aproveitar o sistema de locomoção.

O sistema de locomoção é muito parecido com automóveis de tração traseira, sendo baseado em rodas que seguem a disposição de um quadriciclo.

O par de rodas traseiro é fixo e responsável pela tração. A tração é realizada por um motor com o auxílio da caixa de redução original do carrinho.

O par de rodas dianteiras é orientável e centralizada sendo utilizado para permitir o direcionamento. A direção é controlada por um servo que foi acoplado à plataforma.

3.1.6 Servo

Para controlar a direção da plataforma móvel, foi utilizado um servo do modelo HCAM0149 cujas especificações técnicas mais relevantes estão relacionadas na Tabela 3.3. O servo utilizado possui uma boa capacidade de torque para a aplicação desenvolvida e pode ser visto na Figura 3.7.

Tabela 3.3: Especificações do Motor Servo.

Tensão de operação	4.8 V	6.0 V
Velocidade (60 graus de rotação)	0.19 s	0.16 s
Torque	3.06 Kg-cm	3.57 Kg-cm
Dimensões	41x20x36 mm	

Figura 3.7: Servo HCAM0149.

3.1.7 Motor

Para proporcionar a tração necessária para movimentar a plataforma móvel, foi utilizado o motor original do carrinho.

O motor é do modelo FC-130RA 14150. Trata-se de um motor de corrente contínua com escovas que opera com valores de tensão de até 9 Volts.

O motor é acoplado próximo ao eixo traseiro. Com o auxílio da caixa de redução é possível controlar o sentido e a velocidade da plataforma móvel.

3.1.8 Controlador do Motor

O para controlar a velocidade e a direção dos motores, foi utilizado o circuito DRI0002, que pode ser visto na Figura 3.8.

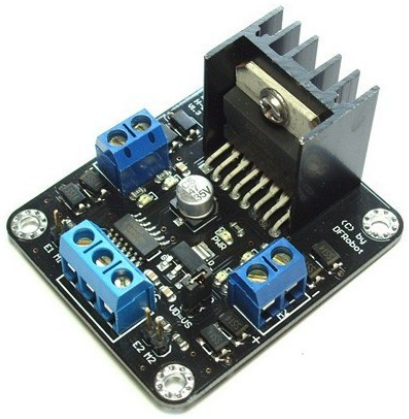

Figura 3.8: Controlador do Motor DRI0002.

Trata-se de um dispositivo baseado no componente L298N que permite o controle de dois motores bidirecionais de corrente contínua.

A Tabela 3.4 resume as especificações técnicas do componente.

Tabela 3.4: Especificações do DRI0002.

Tensão de operação da parte lógica	6 - 12 V
Tensão de operação da parte dos motores	4.8 - 46 V
Consumo de corrente da parte lógica	36 mA
Corrente máxima de operação da parte dos motores	2 A
Dissipassão de potência máxima	25 W
Dimensões	47x53mm

3.1.9 Sensor de Corrente

Para a realização desse projeto, foi utilizado um sensor de corrente com o intuito de monitorar a utilização da bateria, permitindo gerenciar melhor a sua utilização.

O Sensor de corrente utilizado foi o RB-Dfr-149, baseado no componente ACS758 e mostrado na Figura 3.9. A Tabela 3.5 mostra as especificações do sensor de corrente.

Figura 3.9: Sensor de Corrente RB-Dfr-149.

Tabela 3.5: Características do Sensor de Corrente RB-Dfr-149.

Tensão de operação	5 V
Tensão de pico	3000V (AC), 500V(DC)
Faixa de corrente	-50 até 50 A
Sensibilidade	40 mV/A
Temperatura de Operação	-40 até 150 graus C
Dimensões	34x34mm

3.1.10 Sensor de Distância

Para ser possível incluir a funcionalidade de frear os motores da plataforma móvel evitando colisões com objetos, foi utilizado um sensor de distância ultra sônico. O sensor de distância utilizado foi o HCSR04[10] que pode ser visto na Figura 3.10. A Tabela 3.6 mostra as características principais do sensor ultra sônico.

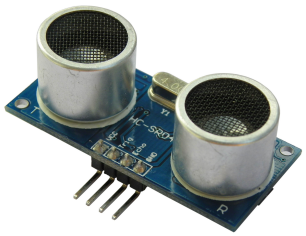

Figura 3.10: Sensor Ultrassonico HCSR04.

Tabela 3.6: Características do Sensor Ultrassonico HC-SR04.

Tensão de operação	DC 5 V
Corrente de operação	15 mA
Frequência de operação	40 KHz
Alcance máximo	4 m
Alcance mínimo	2 cm
Angulo de medida	15 graus
Dimensões	45*20*15mm

3.1.11 Fonte de Alimentação de Bancada

Para realizar os testes com a plataforma móvel, foi adaptada uma fonte de alimentação a partir de um gabinete de computador antigo, com o objetivo de poder fornecer energia para os componentes utilizados. Dessa forma, a fonte pôde ser utilizada para gerar tensões de 5 e 12 Volts, que foram utilizadas em alguns momentos ao longo do desenvolvimento do projeto. A Fonte desenvolvida pode ser vista na Figura 3.11.

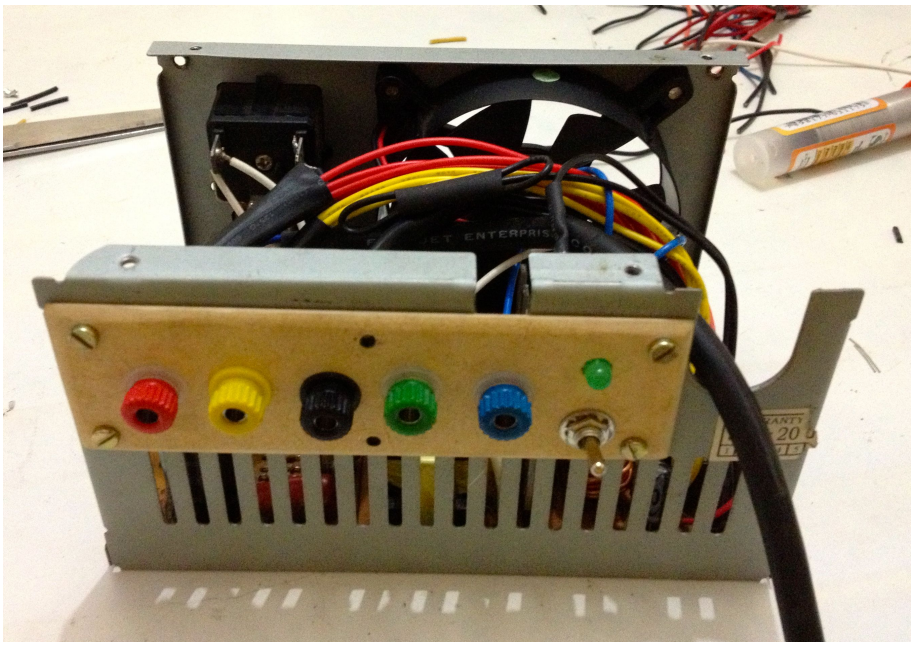

Figura 3.11: Fonte de alimentação adaptada.

3.2 Montagem

Para a realização desse projeto, foi utilizado um carrinho de controle remoto antigo, retirando toda a eletrônica original e deixando apenas os motores e a caixa de redução. Dessa forma, foi possível aproveitar toda a estrutura mecânica e implementar a lógica de controle de maneira conveniente para o desenvolvimento da plataforma móvel. A Figura 3.12 mostra de forma simplificada a organização escolhida para os componentes utilizados no projeto. As setas indicam o sentido de comunicação entre cada componente. O computador apresenta uma interface que permite que um usuário controle remotamente a plataforma móvel. Isso é feito com o intermédio da BeagleBone, que estabelece uma comunicação pela rede com o computador e repassa os dados para o Arduino. O Arduino, por sua vez, é responsável pela comunicação direta com os sensores e os atuadores.

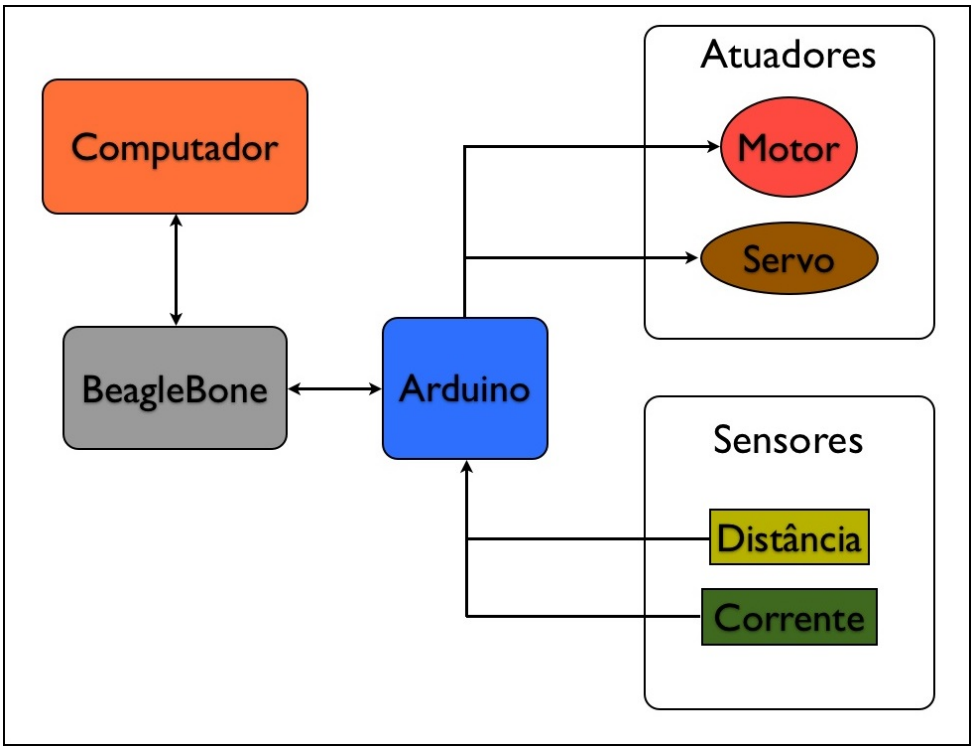

Figura 3.12: Diagrama de Blocos dos Componentes.

Essa abordagem, permite que funcionalidades que dependem de uma precisão de tempo mais refinada possam ser implementados sem dificuldades diretamente no Arduino. Já as funcionalidades um pouco mais tolerantes em relação ao tempo de resposta, podem ser implementas na BeagleBone com toda a flexibilidade que um sistema operacional completo oferece.

A Figura 3.13 ilustra com mais detalhes como os componentes foram conectados entre si. É importante ressaltar que a tensão de alimentação de todos os componentes é de 5 Volts, enquanto o circuito controlador do motor e o próprio motor funcionam com uma tensão na faixa de 6 a 12 Volts.

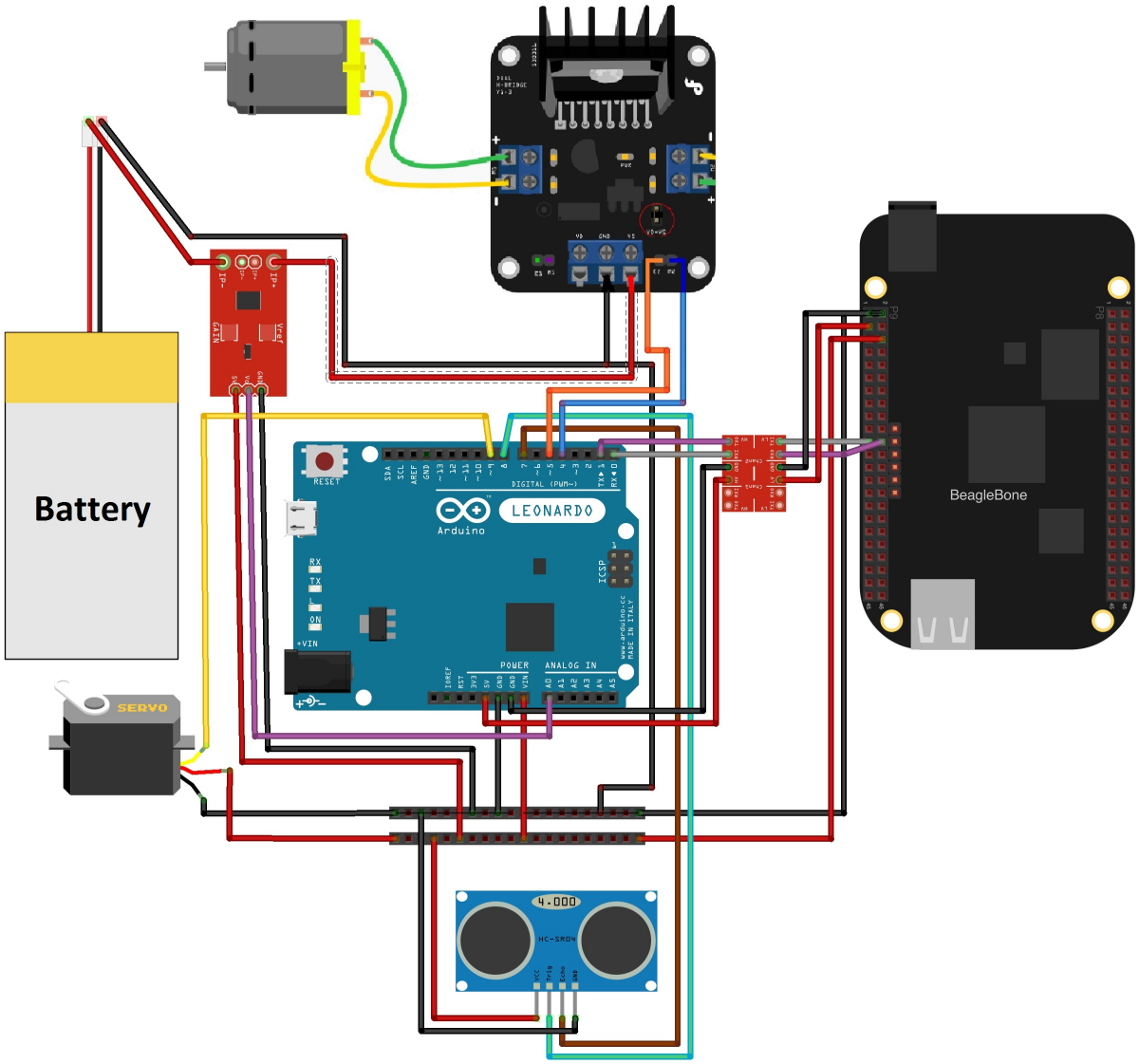

Figura 3.13: Diagrama das conexões dos componentes.

É possível perceber também que o sensor de corrente monitora apenas o consumo dos motores e do próprio circuito controlador dos motores, que estão sendo alimentados pela bateria recarregável.

Com todos os componentes eletrônicos conectados e funcionando, foi necessário fazer alguns ajustes para encaixá-los na plataforma móvel. Para que isso fosse possível, foram montados suportes de madeira que permitiram fixar os componentes de maneira segura, devido a característica isolante do material.

Outra adaptação realizada posteriormente foi o posicionamento do servo para controlar a direção da plataforma. O mesmo foi fixado na parte frontal do carrinho, aproveitando o sistema mecânico já existente para direcionar as rodas.

A Figura 3.14 mostra o protótipo real em sua fase de desenvolvimento. Nessa etapa alguns dos componentes ainda estão temporariamente instalados apenas para a realização

de testes.

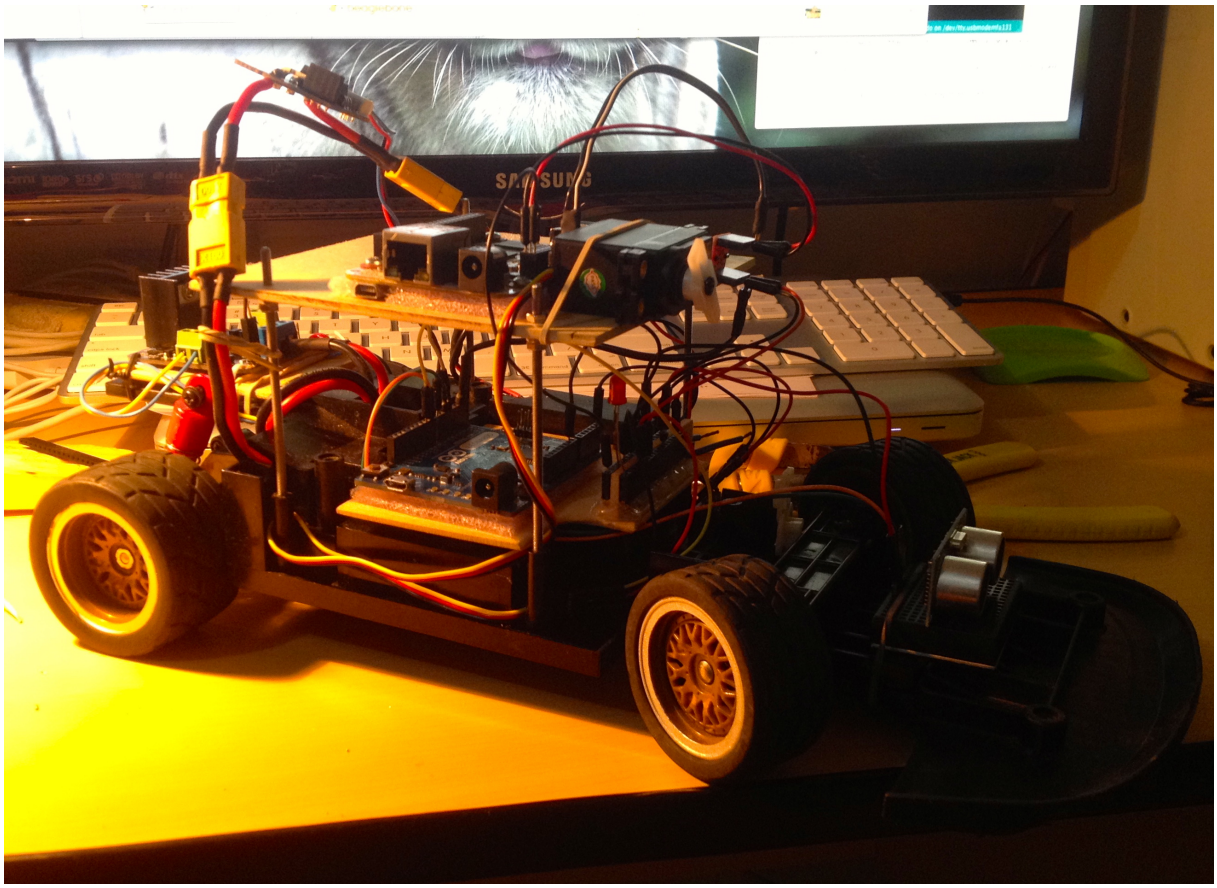

Figura 3.14: Imagem do protótipo com os componentes temporariamente instalados.

3.3 Configuração

Grande parte dos componentes utilizados neste projeto não necessitam de configurações, como, por exemplo, é o caso do sensor de distância, onde é necessário compreender funcionamento do sinal e controle e interpretar o sinal de resposta. Entretanto isso não é valido para todos os componentes. Na seção seguinte são descritas as principais etapas para a configuração da BeagleBone para que a mesma consiga desempenhar as suas funcionalidades corretamente.

3.3.1 BeagleBone

Uma das tarefas mais trabalhosa ao longo do desenvolvimento do projeto foi a configuração da BeagleBone. Nesta seção descrevemos com detalhes todas as etapas necessárias para a gravação da imagem utilizada.

Atualização da imagem

O primeiro passo antes de configurar a BeagleBone foi atualizar a versão do Linux. Para isso é necessário gravar a imagem no cartão *micro SD* que é utilizado pela BeagleBone para inicializar o sistema operacional. Isso pode ser feito realizando os seguintes passos:

1. Instalar o *brew*.

   O *brew* é uma ferramenta de gerenciamento de pacotes que neste caso é utilizado para a instalação do *xz*. O *xz* é a ferramenta utilizada para a descompactação da imagem que será gravada no cartão SD.

2. Instalar o *xz*.

```
brew install xz
```

3. Fazer o *download* da imagem no formato *.img.xz*.

```
wget 'URL para download da imagem'
```

4. Ejetar o cartão de memória para realizar a gravação.

```
diskutil unmountDisk /Volumes/BEAGLE_BONE
```

5. Descobrir o identificador do sistema para o cartão SD.

```
  diskutil list
```

6. Gravar a imagem no cartão SD

```
  sudo xz -dkc Angstrom-Cloud9-IDE-GNOME-eglibc-ipk-v2012.12-
      beaglebone-2013.06.20.img.xz > /dev/disk1
```

Depois desses passos, o cartão *micro SD* pode ser inserido novamente na BeagleBone. Se tudo ocorreu como o esperado, a placa deve inicializar o sistema assim que for fornecida a energia.

Login Remoto

Depois de finalizado o processo de gravação da imagem já é possível realizar o *login* remoto na BeagleBone através do comando *ssh*. Caso já tenha sido realizado o *login* remoto na BeagleBone a partir do seu computador, uma mensagem de erro irá aparecer:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

```
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
d0:f2:9c:ad:65:11:c5:7a:fc:ac:4d:a6:81:72:a7:4f.
Please contact your system administrator.
Add correct host key in /Users/luisfelipewb/.ssh/known_hosts to get rid of
    this message.
Offending key in /Users/luisfelipewb/.ssh/known_hosts:30
RSA host key for 10.0.0.89 has changed and you have requested strict
    checking.
Host key verification failed.
```

Isso ocorre pois fica gravado no arquivo *'.ssh/known-hosts'* uma chave de autenticação gerada automaticamente pela BeagleBone. Quando o cartão de memória é formatado e um novo sistema operacional é instalado, essa chave muda sendo necessário apagá-la dos registros do computador para poder realizar o *login* remoto novamente. Isso pode ser feito com a utilização do seguinte comando:

```
ssh-keygen -R "hostname"
```

Para facilitar o processo de *login* é possível gerar uma chave publica e copiá-la para a máquina remota para não precisar mais digitar a senha toda vez que for fazer o *login*. Isso pode ser feito com os seguintes comandos:

```
ssh-keygen
scp .ssh/id_rsa.pub root@10.0.0.89:/home/root/.ssh/authorized_keys
```

Configuração do Python

Depois de estabelecer uma conexão com a BeagleBone é importante realizar uma atualização da lista dos pacotes disponíveis utilizando o gerenciador de pacotes *opkg*. Com a lista atualizada podemos instalar as bibliotecas necessárias para a execução do código em Python:

```
opkg update
opkg install python-pip python-setuptools python-smbus
pip install Adafruit_BBIO
opkg install python-pyserial
opkg install python-pyserial
```

Uma das bibliotecas instaladas com os comandos acima é disponibilizada pela Adafruit: Trata-se de uma biblioteca que visa simplificar a utilização das portas de entrada e saída da BeagleBone oferecendo uma interface mais simplificada para o desenvolvimento de aplicações utilizando a linguagem Python.

No *site* da Adafruit pode ser encontrado um tutorial com detalhes sobre a utilização da interface UART2 na BeagleBone[8].

É necessário também habilitar a multiplexação correta dos pinos no sistema operacional para permitir a utilização do UART2 pelo programa mencionado. Isso é feito por meio do seguinte comando:

```
echo BB-UART2 > /sys/devices/bone_capemgr.7/slots
```

Para verificar se a multiplexação foi feita adequadamente, podemos analisar o arquivo *'slots'* com o comando abaixo:

```
cat /sys/devices/bone_capemgr.7/slots
```

Verificar as mensagens do sistema também permite acompanhar se a multiplexação ocorreu da forma esperada:

```
dmesg | grep UART
```

Um arquivo de *scritp* foi desenvolvido para automatizar essa funcionalidade e pode ser visto no Apêndice C. Esse arquivo de *script* deve ser executado durante a inicialização do sistema de forma que toda vez que a BeagleBone for reiniciada ela já estará pronta pra iniciar a comunicação através da interface UART2.

O código em Python desenvolvido para funcionar na BeagleBone encontra-se no Apêndice B.

Configuração do adaptador wireless

Para esse projeto foi utilizado um adaptador de rede sem fio do modelo EW-7811Un[15].

Com o dispositivo conectado na porta USB podemos executar o comando *lsusb*:

```
root@beaglebone:~# lsusb
Bus 001 Device 002: ID 7392:7811 Edimax Technology Co., Ltd EW-7811Un
    802.11n Wireless Adapter [Realtek RTL8188CUS]
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Para identificar os erros encontrados na utilização do módulo podemos verificar as mensagens do sistema:

```
root@beaglebone:~# dmesg | grep rtl
[    9.276164] rtl8192cu 1-1:1.0: usb_probe_interface
[    9.276202] rtl8192cu 1-1:1.0: usb_probe_interface - got id
[    9.276439] rtl8192cu: Chip version 0x10
[    9.372609] rtl8192cu: MAC address: 80:1f:02:45:c3:f8
[    9.372652] rtl8192cu: Board Type 0
[    9.372809] rtlwifi: rx_max_size 15360, rx_urb_num 8, in_ep 1
[    9.373049] rtl8192cu: Loading firmware rtlwifi/rtl8192cufw.bin
[    9.376574] usbcore: registered new interface driver rtl8192cu
[    9.389741] rtlwifi: Firmware rtlwifi/rtl8192cufw.bin not available
```

Com esse comando é possível verificar que o *chipset* utilizado pelo dispositivo é o RTL8188CUS e que o *firmware* necessário não encontra-se instalado. Para realizar a instalação do módulo é necessário executar os seguintes comandos:

```
root@beaglebone:~# opkg list 'linux-firmware-rt*'
root@beaglebone:~# opkg install linux-firmware-rtl8192cu
root@beaglebone:~# reboot
```

Depois de instalar e reiniciar o sistema, as mensagens do sistema podem ser verificadas para conferir a inicialização do módulo:

```
root@beaglebone:~# dmesg | grep rtl
[    7.785345] rtl8192cu 1-1:1.0: usb_probe_interface
[    7.785384] rtl8192cu 1-1:1.0: usb_probe_interface - got id
[    7.785683] rtl8192cu: Chip version 0x10
[    7.874954] rtl8192cu: MAC address: 80:1f:02:45:c3:f8
[    7.875047] rtl8192cu: Board Type 0
[    7.876083] rtlwifi: rx_max_size 15360, rx_urb_num 8, in_ep 1
[    7.876327] rtl8192cu: Loading firmware rtlwifi/rtl8192cufw.bin
[    7.879887] usbcore: registered new interface driver rtl8192cu
[    8.071952] ieee80211 phy0: Selected rate control algorithm 'rtl_rc'
[    8.078647] rtlwifi: wireless switch is on
[   34.082761] rtl8192cu: MAC auto ON okay!
[   34.182255] rtl8192cu: Tx queue select: 0x05
```

Ao executar o comando *ifconfig*, é possível encontrar a interface de rede sem fio *wlan0*:

```
root@beaglebone:~# ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 80:1F:02:45:C3:F8
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Para obter mais informações sobre as redes disponíveis através do comando *iwconfig* é necessário instalar o pacote que inclui essa ferramenta:

```
root@beaglebone:~# opkg install wireless-tools
```

O próximo passo necessário é configurar o gerenciador de conexões, no casso do Ängstron é o *connman*. Para configurar a rede sem fio é necessário criar o arquivo */var/lib/connman/wifi.config* com as configurações da sua rede sem fio. Abaixo é mostrado um arquivo de exemplo. Vale a pena reforçar que o arquivo de configuração precisa terminar com uma quebra de linha para que ele funcione corretamente.

```
[service_home]
Type = wifi
Name = Dlink
```

```
4 Security = wpa
5 Passphrase = mypassword
```

Para que as configurações tenham efeito, é necessário reiniciar o gerenciador de conexões:

```
1 root@beaglebone:~# systemctl restart connman.service
```

Após a realização desses passos o adaptador deve estar funcionando e devidamente conectado à rede.

3.4 Implementação

Com todos os componentes do projeto instalados e devidamente configurados, a próxima etapa é a implementação da interface de controle, das lógicas de controle dos atuadores, da monitoração dos sensores e da comunicação com a plataforma móvel. Nas Seções a seguir esses tópicos são abordados com detalhes.

3.4.1 Interface de Controle

Para permitir o controle da plataforma móvel de maneira intuitiva foi implementada uma interface gráfica. Essa interface permite enviar mensagens de controle do computador para a BeagleBone para que os motores sejam acionados.

A interface gráfica projetada pode ser vista na Figura 3.15. Ela pode ser acionada através do *mouse* e também do teclado, uma vez que os botões estão associados às respectivas teclas do teclado.

Figura 3.15: Interface gráfica para o controlador.

Para melhorar a usabilidade algumas informações foram incluídas na interface gráfica. Essas informações são: o número de IP utilizado pela BeagleBone, a porta utilizada para a comunicação e também o último comando de direção que foi enviado.

É importante destacar que o campo *'Last Command'* é atualizado apenas quando o comando foi efetivamente recebido pela BeagleBone. Isso é feito utilizando uma mensagem de reconhecimento respondida pela BeagleBone toda vez que algum comando é enviado.

A interface gráfica foi desenvolvida para funcionar no sistema operacional Mac OS X. Isso tornou necessário a utilização da linguagem Objective C para programação do aplicativo.

Para que fosse possível desenvolver a aplicação utilizando a linguagem Python e incluir a interface gráfica foi utilizado o PyObjC[17].

O projeto PyObjC tem como objetivo prover uma ponte de ligação entre as linguagens de programação Python e Objective-C. Essa ponte tem a intenção de permitir que programas em Python possam utilizar o poder de desenvolvimento oferecido por ferramentas de desenvolvimento baseadas em Objective-C.

Essa utilização do Objective-C em códigos escritos em Python pode ser vista no trecho de código abaixo:

```
1     @objc.IBAction
2     def dirRight_(self, sender):
3         send('d')
4         self.updateDisplay()
```

Esse trecho de código corresponde à implementação da lógica do botão 'd' da interface gráfica. Nele podemos ver a primeira linha escrita utilizando Objective-C e o resto do código em Python.

3.4.2 Sensores e Atuadores

O controle dos sensores e atuadores é implementado no microcontrolador Arduino, apresentado na Seção 2.2. Isso é feito pelo microcontrolador devido a facilidade de programação de componentes que necessitam de uma precisão de tempo alta para funcionarem adequadamente.

A Figura 3.16 mostra de maneira simplificada como os componentes estão conectados com o Arduino e quais as ligações são necessárias para realizar o controle de cada uma delas.

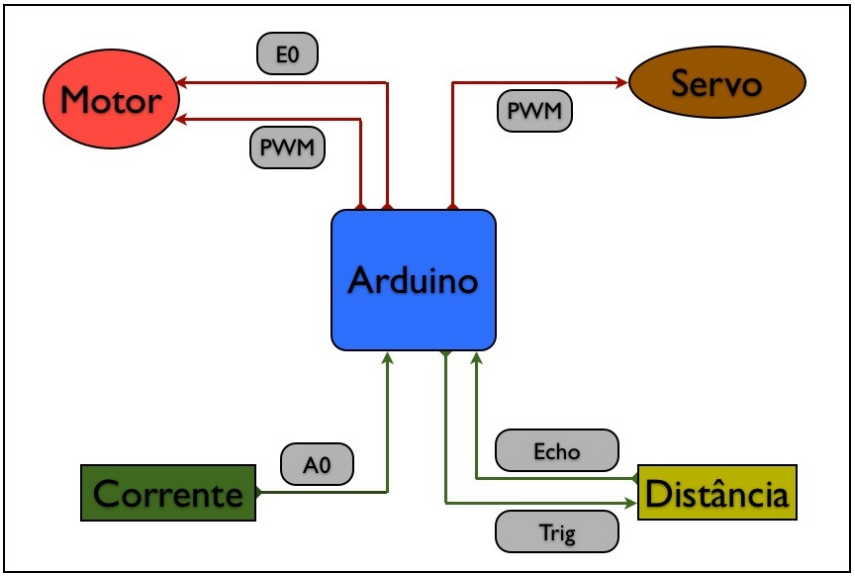

Figura 3.16: Controle dos sensores e atuadores a partir do Arduino.

A programação do Arduino é dividida em duas funções principais: *setup* e *loop*. A implementação dessas funções é obrigatória para o funcionamento do programa.

A função *setup* serve para inicializar a placa e o programa. Ela é executada apenas uma vez quando a placa é ligada ou reiniciada. Nessa função são inicializadas as variáveis e também é descrito qual *hardware* será utilizado.

A função *loop* é executada indefinidamente. Ao terminar a execução da última linha desta função, o programa volta novamente para a primeira linha da função *loop* e continua a executar até que a placa seja desligada ou o botão de *reset* seja pressionado.

Na Figura 3.17 é mostrado o fluxo de execução do código desenvolvido para o Arduino. Este código pode ser encontrado no apêndice A.

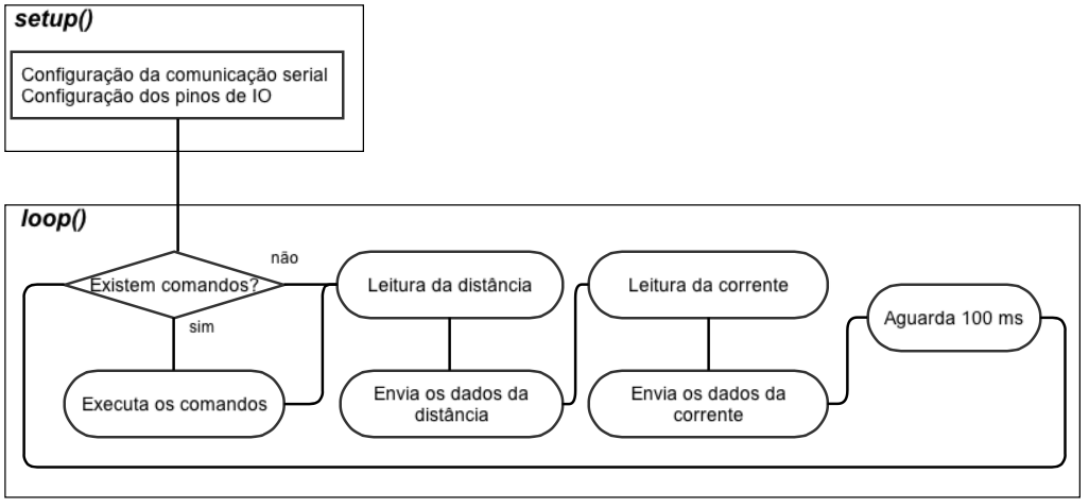

Figura 3.17: Fluxograma do código do Arduino.

Motor

O controle do motor é feito com o auxílio do circuito DRI0002 descrito na Seção 3.1.8. Esse circuito é baseado no componente L298N. Para controlar o motor, o Arduino utiliza dois sinais: M1 e E1. O primeiro, denominado M1 é um sinal digital que indica o sentido de rotação do motor. Quando M1 é igual a 1 a plataforma vai para frente, quando é igual a 0, para trás. Para controlar a velocidade é utilizado o segundo sinal de controle, denominado E1. Este sinal é do tipo PWM e pode ser controlado com a variação do *duty-cycle*. Para variar o *duty-cycle* o Arduino utiliza uma variável que pode assumir valores de 0 até 255, sendo 255 correspondente à velocidade máxima do motor.

O trecho de código abaixo mostra um exemplo das funções utilizadas para controlar o carro para frente com velocidade máxima.

```
1        digitalWrite(M1,1);
2        analogWrite(E1, 255);
```

O código completo de controle dos motores pode ser encontrado no Apêndice A.

A lógica de controle implementada utiliza os caracteres 'w' e 's' para aumentar e diminuir a velocidade do motor. Além disso, pode ser utilizado o caractere 'x' para fazer com que o motor pare.

Servo

O servo é controlado com um sinal de PWM, que faz com que o motor rotacione até a posição desejada. O Arduino disponibiliza a biblioteca *Servo.h* que facilita a utilização, sendo necessário apenas inicializar um servo especificando qual o pino está conectado e utilizar a função correta passando o ângulo desejado como parâmetro para que o servo rotacione. As três linhas de código abaixo mostram um exemplo das funções utilizadas para declarar um servo e rotacioná-lo para a posição de 90 graus.

```
1 Servo myservo;
2 myservo.attach(9);
3 myservo.write(pos);
```

A lógica de controle implementada utiliza os caracteres 'a' e 'd' para girar para a esquerda e para a direita, respectivamente. De maneira análoga ao funcionamento dos comandos de aceleração, o caractere 'x' também faz com que o servo retorne para a posição central.

Embora o servo permita uma amplitude de rotação de até 180 graus, o movimento permitido pelo sistema mecânico do carrinho é menor. Para garantir o funcionamento adequado foi necessário limitar o ângulo máximo de rotação do servo até atingir o esterçamento máximo permitido pelo sistema mecânico.

Sensor de Distância

De acordo com o *datasheet* do fabricante[10], o sonar funciona emitindo ondas em alta frequência e verificando o tempo em que elas demoram para refletir de volta ao sensor.

Após a emissão de um pulso de pelo menos 10 μs no pino *trig*, é emitido uma onda sonora a uma frequência de 40 KHz de forma automática pelo sonar. Um pulso digital no pino *echo* indica o tempo que a onda demora para ser recebida de volta após refletir em algum objeto.

O funcionamento do sensor de distância é ilustrado na Figura 3.18 retirada do *datasheet* do componente[10].

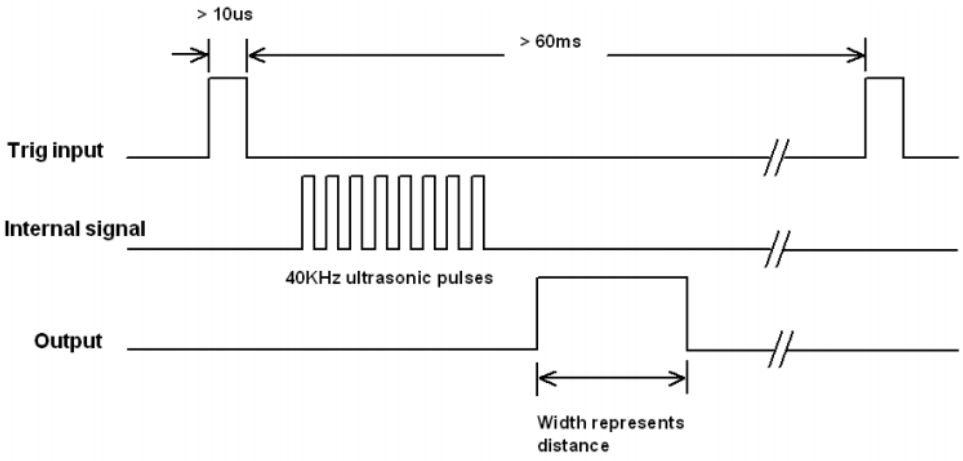

Figura 3.18: Funcionamento do sensor de distância.

Como a velocidade do som no ar é conhecida precisamos apenas resolver a equação a seguir para determinar a distância do objeto. HLT é o tempo indicado pelo pulso no pino *echo* e Vsom é a velocidade do som no ar.

$$\frac{HLT * Vsom}{2}$$

A seguir mostramos a função implementada para realizar a leitura do tempo de resposta do pino *echo*, da conversão da medida de tempo para distância, da lógica implementada para interromper a aceleração do motor caso a distância lida seja muito pequena e, por fim, o envio da informação de distância para a BeagleBone.

```
1 void readDistance(){
2     /*Sequencia de leitura*/
3     digitalWrite(trigPin, LOW);
4     delayMicroseconds(2);
5
6     digitalWrite(trigPin, HIGH);
7     delayMicroseconds(50);
```

```
8
9       digitalWrite(trigPin, LOW);
10      duration = pulseIn(echoPin, HIGH,10000);
11
12      /*Calcula distancia baseado na velocidade do som*/
13      distance = duration/58.2;
14
15      /*Indicar erro de leitura*/
16      if (distance >= maximumRange || distance <= minimumRange){
17          Serial.println("-1");
18          digitalWrite(LEDPin, HIGH);
19      }
20      /*Para ao chegar perto de um objeto*/
21      if (distance <= stopDistance){
22          Serial.println(distance);
23          direction=95;
24          speed=0;
25          digitalWrite(LEDPin, HIGH);
26          direcao.write(direction);
27          digitalWrite(M1,HIGH);
28          analogWrite(E1, speed);
29      }
30      /*Imprime distancia*/
31      else {
32          Serial.print("[cm]: ");
33          Serial.println(distance);
34          digitalWrite(LEDPin, LOW);
35      }
36  }
```

Sensor de Corrente

Para medir a corrente foi utilizado o Sensor de Corrente RB-Dfr-149, baseado no componente ACS758 e descrito na Seção 3.1.9.

De acordo com o manual do componente[9], o valor da corrente pode ser aferido analisando o valor da tensão de saída, que foi conectada no pino analógico *A0* do Arduino.

No Arduino, o pino *A0* é um conversor analógico digital de 10 bits de maneira que a tensão lida é mapeada em um intervalo de 0 até 1024 de acordo com a tensão de referência fornecida.

O trecho do código utilizado responsável por medir o consumo de corrente é mostrado a seguir:

```
1  void readCurrent()
2  {
3      total= total - readings[index];
4      readings[index] = analogRead(currentSensor); //Raw data reading
```

```
5       readings[index] = (readings[index]-520)*5/1024/0.04;
6       total= total + readings[index];
7       index = index + 1;
8       if (index >= numReadings)
9           index = 0;
10      average = total/numReadings;
11      currentValue= average;
12
13      Serial.print("Corrente: ");
14      Serial.println(currentValue);
15  }
```

Observando o código podemos perceber que foi utilizado um algoritmo para suavizar os resultados aferidos para minimizar a oscilação devido a ruídos. Isso é realizado através do cálculo da média dos últimos valores aferidos que são armazenados em um vetor. O valor utilizado para o número de leituras no cálculo da média foi igual a cinco para que a velocidade de resposta não seja muito lenta.

3.4.3 Comunicação

Para compreender como é feita a troca de informações entre cada um dos componentes, primeiramente precisamos entender que há basicamente dois tipos de informações que são trocadas: comandos de direção e dados dos sensores.

Os comandos de direção são comandos assíncronos que são gerados pelo usuário a partir da interface de controle toda vez que ele deseja modificar a direção ou a velocidade

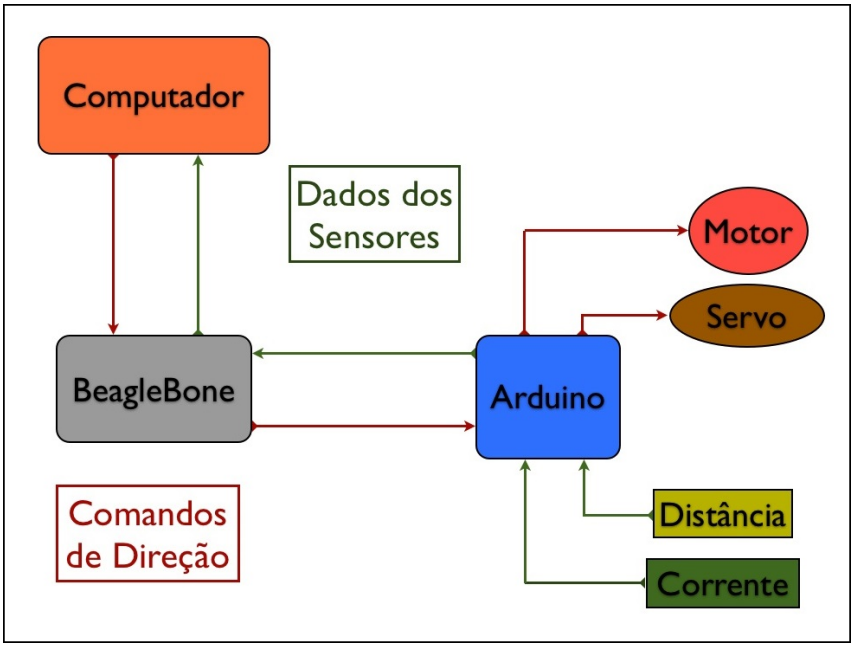

Figura 3.19: Etapas de comunicação de acordo com o tipo de dado

da plataforma móvel.

Os dados dos sensores são aferidos pelo Arduino e enviados ao computador de maneira síncrona permitindo que o usuário monitore a plataforma móvel em tempo real.

A Figura 3.19 mostra as etapas de comunicação entre todos os componentes.

Após compreender de maneira genérica as etapas de comunicação envolvidas, as seções a seguir descrevem com mais detalhes o processo para estabelecer o controle e monitoramento da plataforma móvel abordando primeiro a comunicação entre o computador e a BeagleBone e depois entre a BeagleBone e o Arduino.

Comunicação entre o computador e a BeagleBone

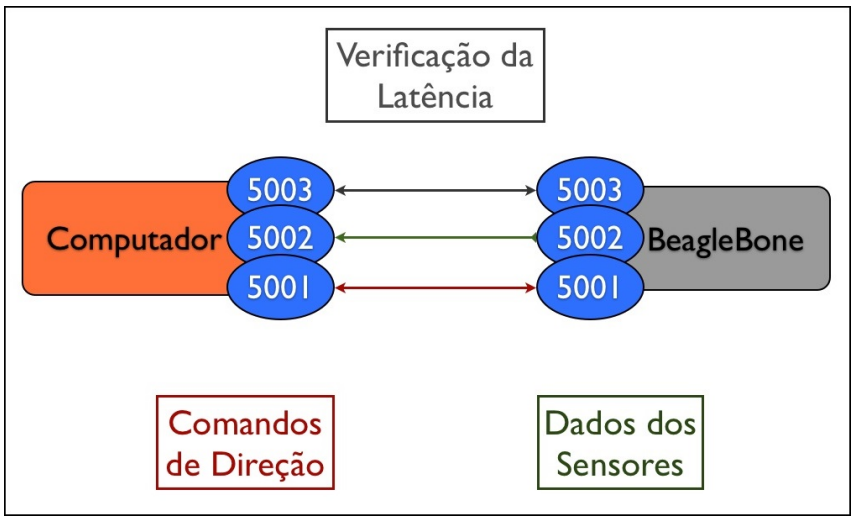

Figura 3.20: Comunicação entre o computador e a BeagleBone.

Foi implementado um protocolo de comunicação entre o computador e a BeagleBone que utiliza o envio de datagramas utilizando três portas de comunicação separadas. Essas portas foram escolhidas arbitrariamente, tomando o cuidado para não utilizar portas reservadas para outros protocolos já existentes. A Figura 3.20 ilustra este protocolo.

A porta 5001 é utilizada para enviar dados referentes ao comando de direção e velocidade da plataforma móvel. Ao clicar em algum botão da interface gráfica o respectivo comando é gerado e enviado à BeagleBone. A BeagleBone, por sua vez, responde ao computador uma mensagem com o mesmo caractere. Essa resposta ao computador caracteriza um mecanismo de confirmar o recebimento da mensagem.

A porta 5002 apresenta um fluxo unidirecional de dados. É através dessa porta que a BeagleBone envia periodicamente ao computador as informações registradas pelos sensores. Essas informações são enviadas na forma *strings* contendo os respectivos valores.

A porta 5003 é utilizada para a verificação da latência da rede. Periodicamente, o computador envia à BeagleBone um determinado número de caracteres. A BeagleBone possui uma rotina programada para responder imediatamente aos caracteres recebidos na

porta 5003. Dessa maneira o cliente pode implementar uma lógica para verificar se todos os pacotes enviados estão sendo recebidos pela BeagleBone, calculando assim a perda de pacotes e verificando o tempo que esses pacotes demoram para ser respondidos.

A implementação desse protocolo de comunicação pode ser vista nos Apêndices B, D, E e F.

Comunicação entre a BeagleBone e o Arduino

Para realizar a comunicação entre a BeagleBone e o Arduino foi utilizado o protocolo serial RS232 com *baudrate* de 9600. Para estabelecer a comunicação é necessário conectar os terminais transmissores e receptores de maneira cruzada, como mostra a Figura 3.21.

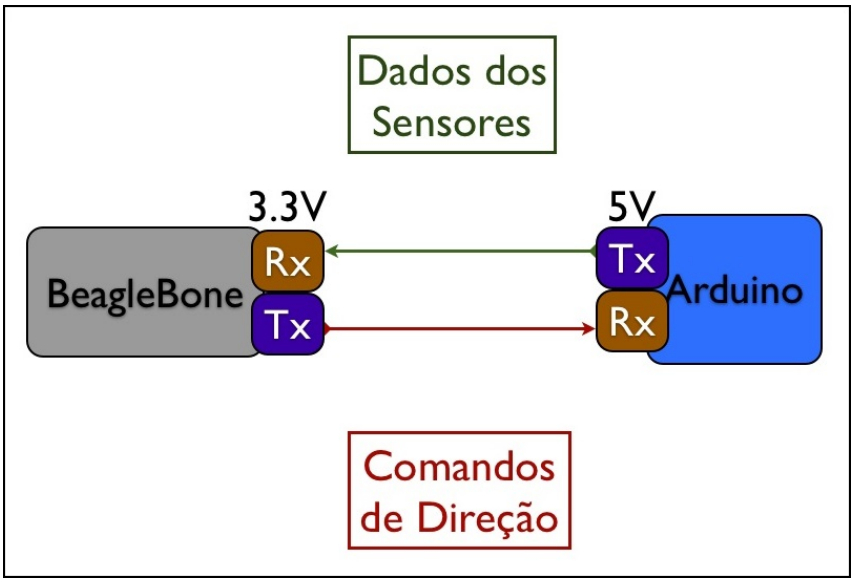

Figura 3.21: Comunicação entre a BeagleBone e o Arduino.

Devido ao fato do Arduino possuir um nível lógico de 5V e a BeagleBone um nível lógico de 3.3V, foi necessário a utilização do conversor de nível lógico descrito na Seção 3.1.3. As conexões necessárias para realizar a comunicação utilizando o conversor de nível lógico podem ser vistas na Figura 3.22.

O Arduino envia uma sequência periódica contendo as informações dos sensores e recebe da BeagleBone mensagens de comando para modificar o estado dos atuadores. Essas mensagens podem ser os caracteres 'w', 'a', 's', 'd' ou 'x'.

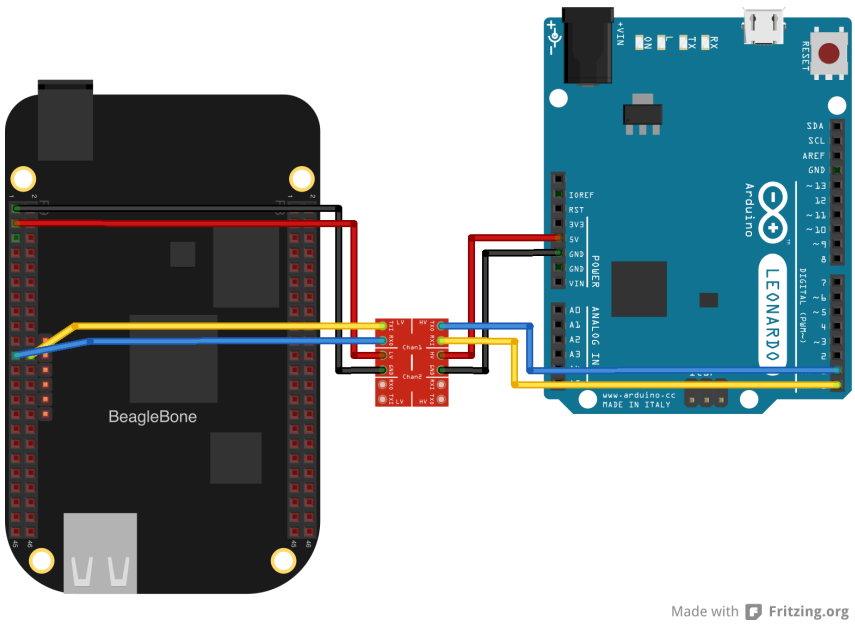

Figura 3.22: Conexões para comunicação serial entre o Arduino e a BeagleBone.

Capítulo 4

Resultados

Após a conclusão da montagem e configuração da plataforma móvel, foi observado o comportamento e desempenho de algumas funcionalidades. Nas seções abaixo são mostrados e discutidos os resultados obtidos.

4.1 Consumo de Energia

A fonte de energia montada e utilizada durante os testes (Seção 3.1.11) não apresentou um desempenho satisfatório. À medida que mais componentes eram utilizados foi possível verificar que a tensão não se mantinha em 5 Volts como era esperado. Quanto maior a corrente consumida a tensão de saída diminuía.

Esse fator atrapalhou, em alguns momentos, os testes realizados, uma vez que quando a tensão alcançava valores abaixo de 4,85 Volts os componentes mostravam instabilidade. Principalmente a BeagleBone que era a primeira a parar de funcionar.

Para contornar essa dificuldade foram utilizadas , simultaneamente, fontes de alimentação alternativas, como a porta USB do computador e carregadores de celular para alimentar a BeagleBone e o Arduino. Para que isso fosse possível o principal cuidado foi realizar as ligações de terra comum para garantir o funcionamento adequado da plataforma móvel.

4.2 Protocolo de Comunicação

O protocolo de comunicação implementado permitiu o controle da plataforma robótica e a monitoração dos dados dos sensores. A função de verificação da latência de rede mostrou uma perda de pacotes muito baixa, (inferior a 1%) em condições normais.

É importante destacar que a verificação da latência da rede depende de diversos parâmetros, e a implementação realizada nesse projeto pode ser otimizada, diminuindo a sobrecarga da rede e obtendo valores mais precisos.

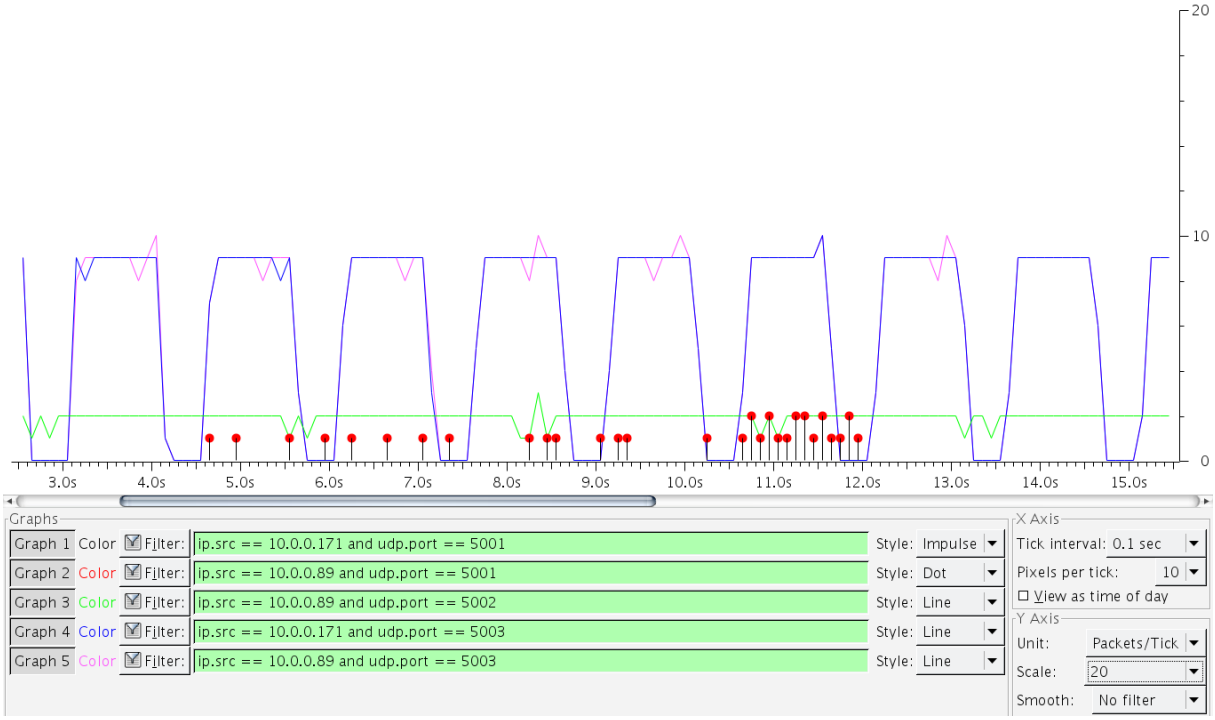

Figura 4.1: Número de pacotes transmitidos pelo protocolo de comunicação em função do tempo.

Para observar o comportamento do protocolo de comunicação entre o computador remoto e a BeagleBone através da rede foi utilizada a ferramenta Wireshark[11], que permite a monitoração e visualização dos pacotes que trafegam pela rede.

A Figura 4.1 mostra uma gráfico com os pacotes transmitidos pelo protocolo de comunicação descrito na Seção 3.4.3 na página 52. O gráfico apresentado mostra o número de pacotes enviados em função do tempo.

Os pacotes utilizados para o envio de comandos à plataforma móvel utilizaram a porta 5001. Eles estão representados pelas barras verticais em preto e os pontos em vermelho, significando, respectivamente, o envio de comandos e a resposta de confirmação.

Os pacotes com destino à porta 5002 são mostrados em verde. Estes pacotes representam o fluxo de dados dos sensores enviados da BeagleBone para o computador.

O fluxo de dados gerado para verificar a latência da rede (porta 5003) é representado no gráfico pelas curvas azul e rosa. A curva em azul mostra os pacotes periodicamente enviados do computador para a BealgeBone. A resposta desses pacotes é mostrada em rosa.

Devido ao fato da comunicação entre o computador remoto e a BeagleBone basear-se em uma rede sem fio, esta apresenta maior vulnerabilidade à falhas e lentidão, devido a essa alta vulnerabilidade a latência da rede foi observada com maior cuidado também em situações onde a rede encontra-se com um fluxo muito grande de pacotes.

Para verificar o comportamento do protocolo de comunicação nessas situações foi

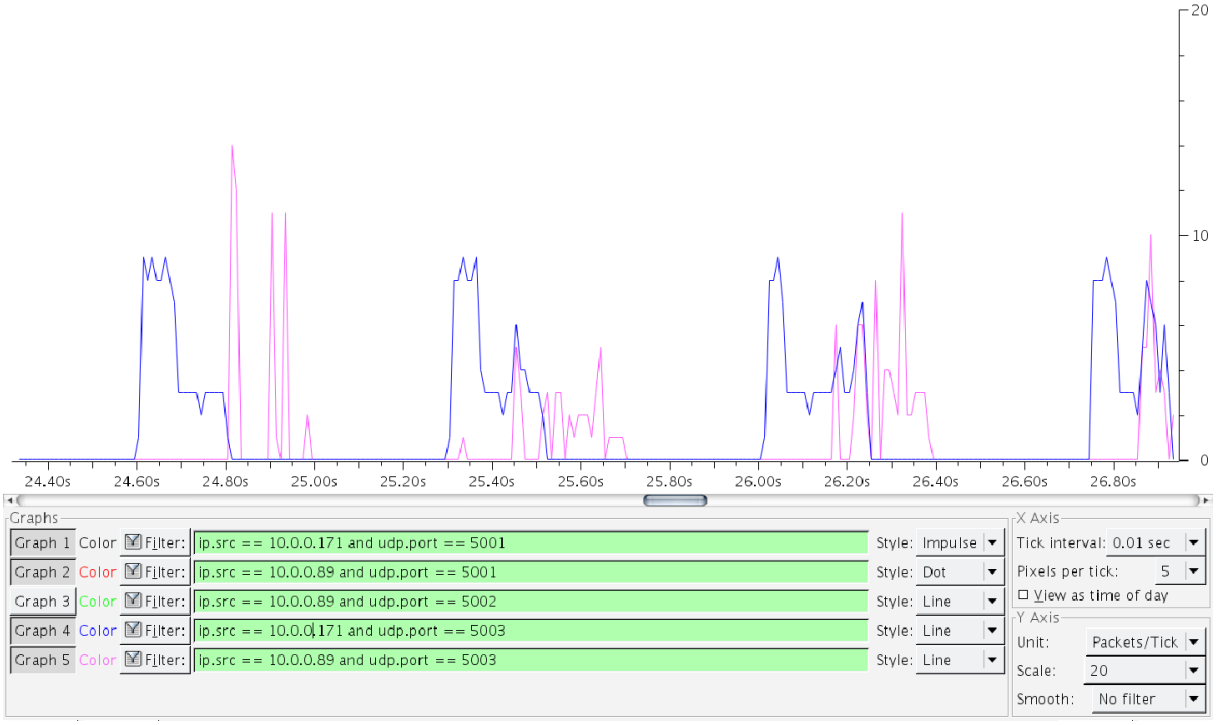

Figura 4.2: Atraso gerado devido a sobrecarga da rede.

gerado um tráfego intenso de pacotes com o comando *ping* com a opção de inundação. Essa opção envia novos pacotes mesmo antes de receber a resposta do pacote anterior gerando um tráfego bastante intenso na rede. O comando utilizado pode ser visto a seguir:

```
1 sudo ping -f 10.0.0.89
```

Ao analisar o tráfego da rede nessas condições, foi possível constatar o atraso gerado e a perda de pacotes. O atraso fica evidente na Figura 4.2, chegando a valores da ordem de 0.2 segundos. Esse valor encontrado para os atrasos é limitado pelo tempo de *timeout* utilizado na implementação do protocolo.

A Figura 4.3 destaca a perda de pacotes, que pode ser verificada observando o intervalo de tempo em que a rede estava sobrecarregada: Entre 60 e 90 segundos. O número de pacotes respondidos sofre uma queda significativa. Nesse intervalo a verificação da latência da rede implementada no protocolo de comunicação mostrou uma perda de pacotes em torno de 58%.

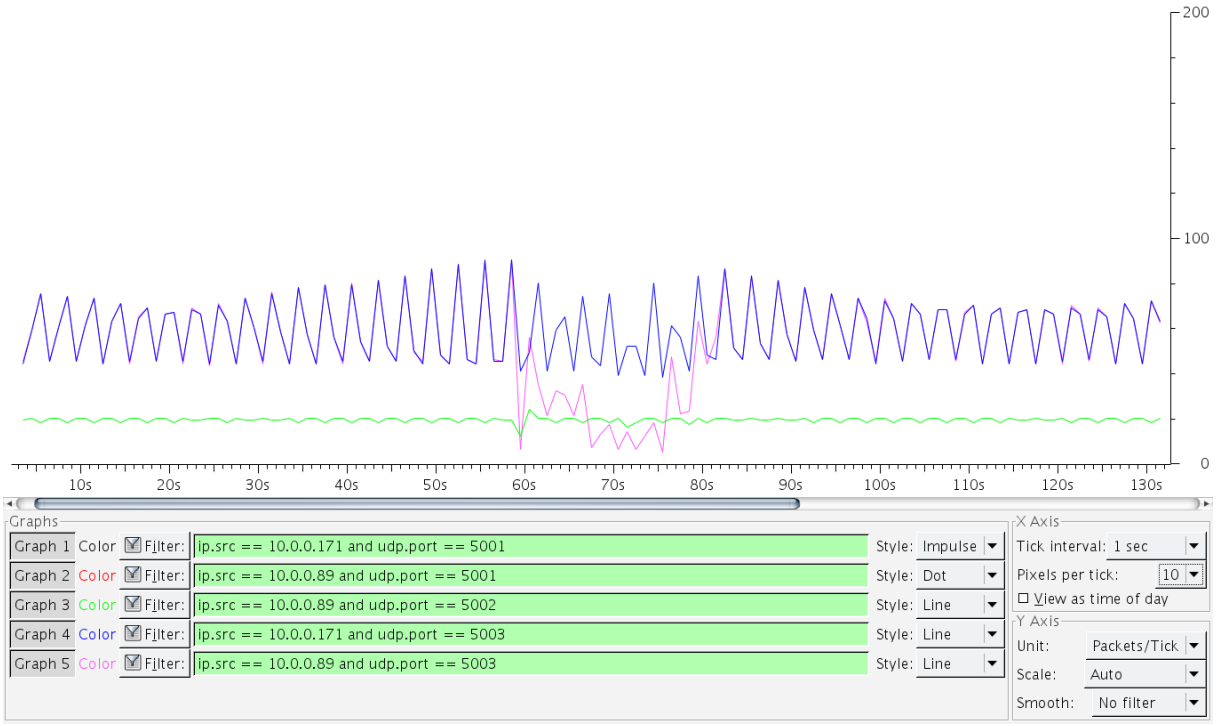

Figura 4.3: Perda de pacotes devido a sobrecarga da rede.

4.3 Sensor de Distância

O sensor de distância mostrou-se funcional e permite medir a distância dos objetos próximos com precisão de até 0,5 centímetros. Dessa forma foi possível incluir a funcionalidade de parar os motores quando um objeto encontra-se muito perto. Essa funcionalidade tem o objetivo de evitar colisões frontais prevenindo danos à plataforma móvel.

De acordo com as especificações encontradas no manual do fabricante[10], o sensor apresenta um alcance mínimo de 2 centímetros e máximo de 4 metros, com um ângulo de medida máximo de 15 graus. Essas informações estão ilustradas na imagem da esquerda na Figura 4.4.

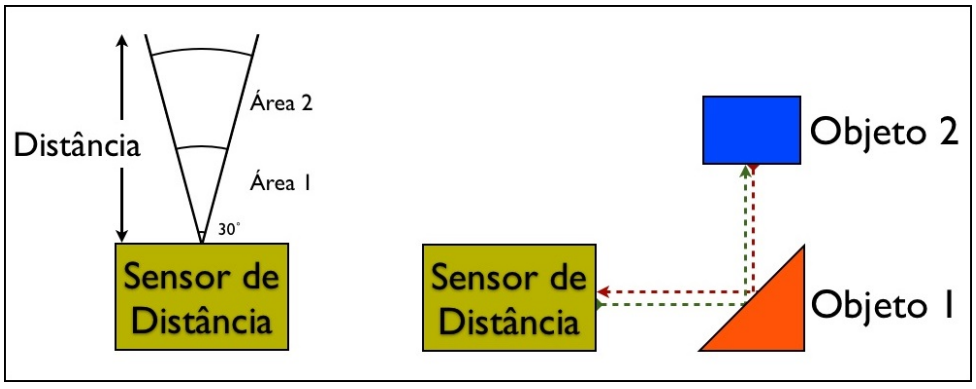

Figura 4.4: Problemas encontrados na utilização do sensor de distância.

Este ângulo, na medida realizada pelo dispositivo, acaba introduzindo erros quando o objeto começa a se distanciar do sonar. Isso ocorre devido ao fato que a área abrangida pelo sonar começa a aumentar, fazendo com que a probabilidade da influência de demais elementos presentes no ambiente seja maior. Para distâncias inferiores a àrea é menor de maneira que o erro introduzido na medida da distância acaba sendo bastante pequeno.

Outro problema identificado na utilização desse tipo de sonar foi em situações onde a onda sonora emitida reflete em mais de um objeto antes de retornar ao sensor. Esta situação é ilustrada na imagem da direita na Figura 4.4. Nela podemos ver que a onda emitida incide no Objeto 1 com um ângulo de 45 graus, é redirecionada para o Objeto 2. Nesta situação a distância informada pelo sensor é referente à soma da distância entre o sensor e o Objeto 1 com a distância entre o Objeto 1 e o Objeto 2.

Devido a problemas como os discutidos nessa Seção é importante utilizar o sensor ultra sônico com bastante cuidado, pois suas medidas podem sem influenciadas por diversos fatores.

4.4 Sensor de Corrente

A utilização do sensor de corrente foi interessante por questões didáticas, entretanto suas características de operação tornou a utilização pouco precisa.

Se levarmos em consideração que o conversor analógico do pino *A0* do Arduino Leonardo é de 10 bits e que a tensão de referência é de 5 Volts podemos calcular a precisão do conversor analógico. O resultado encontrado é uma precisão de 4,88mV.

Como visto na Seção 3.1.9 o sensor de corrente pode ser utilizado para medir correntes no intervalo de -50 até 50 amperes, com uma sensibilidade de 40mV/A. Sabendo dessas informações é possível calcular a precisão do conversor analógico em relação a corrente aferida pelo sensor. O valor encontrado é uma precisão de 0,122A.

Como o consumo de corrente do motor não passa de 2,4 A, mesmo em situações extremas é possível perceber que a precisão do sensor é muito baixa para esse projeto.

Devido a essas características, seria interessante a utilização de sensores mais apropriados para a faixa de consumo de corrente desse projeto. Permitindo assim alcançar resultados mais satisfatórios.

Capítulo 5

Conclusão

Acionar o funcionamento de um motor com apenas um clique em um botão na tela de um computador parece uma funcionalidade bastante simples, mas dependendo de como é feita a implementação, pode ser bastante complexo. Para implementar essa funcionalidade, foi necessário projetar uma interface de controle, construir a plataforma móvel e implementar todas etapas de integração dos componentes do sistema.

Durante o desenvolvimento do projeto foi possível avaliar a eficiência de algumas das escolhas feitas para realizar a implementação de cada etapa necessária para concluir o projeto. Algumas dessas escolhas e suas implicações são apresentadas e discutidas a seguir.

A utilização de sensores de distância ultra sônicos é uma alternativa bastante barata e didática. Porém, para projetos que necessitam de uma qualidade maior e que não são tolerantes às limitações observadas é indicado a utilização de sensores baseados em outras tecnologias, como por exemplo sensores a laser. Essas alternativas podem encarecer muito o custo do projeto mas, dependendo da aplicação, são mudanças necessárias.

O protocolo de comunicação entre a BeagleBone e o computador remoto utilizando rede sem fio mostrou-se possível e bastante flexível. A utilização de outras tecnologias como uma alternativa para estabelecer a comunicação poderia trazer vantagens relacionadas com a distância máxima de alcance e velocidade de comunicação. Entretanto, foi dado preferência pela utilização da rede sem fio devido à possibilidade de expandir o projeto facilmente para que a plataforma móvel seja controlada pela internet.

A utilização de plataformas de *hardware* e *software open source* mostrou-se bastante produtiva devido à comunidade ativa que compartilha e disponibiliza soluções para a maioria dos problemas encontrados. O contato com a distribuição do Linux utilizada permitiu conhecer melhor algumas ferramentas diferentes bem como o funcionamento básico do *kernel*, comum a todas as distribuições.

O uso de Linux Embarcado é uma solução apropriada por se tratar de uma alternativa altamente personalizável e com e a facilidade para a expansão das funcionalidades oferecidas pela plataforma móvel. A utilização de uma interface web para monitoração e

telemetria da plataforma móvel seria uma funcionalidade interessante. Com a utilização de Linux Embarcado essa funcionalidade pode ser facilmente implementada.

Foi possível perceber também que existe um grande número de alternativas diferentes para solucionar os mesmos problemas. Esse leque de possibilidades ressalta a importância de elaboração de estudos comparativos entre as alternativas para descobrir quais as vantagens e desvantagens de cada uma delas, permitindo tomar decisões de projeto mais coerentes com o objetivo de alcançar uma solução com alta qualidade e eficiência.

Depois de realizar essas considerações pode-se afirmar que foi possível alcançar o objetivo proposto, aprofundando os conhecimentos necessários para a implementação de um sistema que depende da integração entre o *software* e o *hardware*. Realizar essa integração de tecnologias diferentes e ajudar o desenvolvimento de novas e melhores tecnologias faz parte do papel de um engenheiro de computação. A realização de projetos como esse caracteriza uma ótima maneira de aprofundar-se nesse universo de conhecimento de uma maneira prática, funcional e didática.

Trabalhos Futuros

O projeto desenvolvido mostrou-se funcional e atingiu o objetivo proposto, mas há diversas melhorias que podem sem implementadas. A seguir são citadas algumas delas:

- Analisar e comparar o atraso de comunicação gerado pela comunicação serial entre a BeagleBone e o Arduino;
- Modificar a instalação elétrica para que a plataforma funcione com o uso de baterias;
- Transferir o controle de todos os sensores e atuadores para a BeagleBone, eliminando a necessidade da utilização do Arduino;
- Melhorar os protocolos de comunicação, incluindo mais redundâncias para garantir o envio dos comandos de controle;
- Aprimorar a interface gráfica para permitir um controle mais intuitivo da plataforma;
- Incluir a utilização de outros sensores para permitir a inclusão de mais funcionalidades autônomas na plataforma móvel;
- Desenvolvimento de uma interface gráfica que permita uma maior portabilidade para outras plataformas;
- Utilização de uma interface web para monitoração e telemetria da plataforma móvel;
- Modularização dos componentes da plataforma facilitando o a utilização da plataforma como um kit didático para estudantes.

Referências Bibliográficas

[1] O Impacto do Software Livre e de Código Aberto na Idústria de Software do Brasil, 2005. Disponível em: <http://www.mct.gov.br/upd-blob/0008/8690.pdf> Acesso em 24 Out. 2013.

[2] Beaglebone: Placa de desenvolvimento de baixo custo, Junho 2012. Disponível em: <http://www.sabereletronica.com.br/pt-BR/artigos-2/2849-beaglebone-placa-de-desenvolvimento-de-baixo-custo> Acesso em 24 Out. 2013.

[3] angstrom-distribution.org. Site oficial, Junho 2007. Disponível em: <http://www.angstrom-distribution.org> Acesso em 24 Out. 2013.

[4] Arduino.org. Arduino leonardo, Junho 2007. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardLeonardo> Acesso em 24 Out. 2013.

[5] Arduino.org. Site oficial, 2012. Disponível em: <http://www.arduino.cc> Acesso em 24 Out. 2013.

[6] BeagleBoard.org. Site oficial, Setembro 2013. Disponível em: <http://beagleboard.org/Products/BeagleBone> Acesso em 24 Out. 2013.

[7] Gerald Coley. *BeagleBone Rev A6 System Reference Manual*, 2012.

[8] Justin Cooper. Setting up IO Python Library on BeagleBone Black. Disponível em: <http://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/uart> Acesso em 12 Ser. 2013.

[9] Datasheet. *RB-Dfr-149*. DFRobot.

[10] Datasheet. *Ultrasonic Ranging Module HC - SR04*. Elec Freaks.

[11] Wireshark Foundation. Página oficial, Setembro 2013. Disponível em: <http://www.wireshark.org> Acesso em 24 Out. 2013.

[12] Jonathan Feldman. O hardware open source é a próxima grande sacada da TI corporativa?, 2012. Disponível em: <http://informationweek.itweb.com.br/8292/o-hardware-open-source-e-a-proxima-grande-sacada-da-ti-corporativa/> Acesso em 29 Out. 2013.

[13] James F Kurose and Keith W Ross. *Computer Networking*. Pearson Education, 2012.

[14] Merciadri Luca and Koën Kooi. *Ångström Manual*, Junho 2012.

[15] Manual do Fabricante. *150 Mbps Wireless IEEE802.11b/g/n nano USB Adapter*, 2012.

[16] Michael Margolis. *Arduino Cookbook*. O'Reilly Media, 2011.

[17] pythonhosted.org. Página oficial pyobjc, Setembro 2013. Disponível em: <http://pythonhosted.org/pyobjc/> Acesso em 24 Out. 2013.

[18] Roberto Brauer Di Renna, Rodrigo Duque Ramos Brasil, Thiago Elias Bitencourt Cunha, Mathyan Motta Beppu, and Erika Guimarães Pereira da Fonseca. Introdução ao kit de desenvolvimento arduino, Junho 2013. Disponível em: <http://www.telecom.uff.br/pet/petws/downloads/tutoriais/arduino/Tut-Arduino.pdf> Acesso em 24 Out. 2013.

[19] Andrew N. Sloss, Dominic Symes, and Chris Wright. *ARM System Developer's Guide*. Elsevier, San Francisco, CA, 2004.

Apêndices

Apêndice A

Código do Arduino

Código completo utilizado no Arduino Leonardo para realizar o controle dos atuadores, monitorar os dados dos sensores e realizar a comunicação serial com a BeagleBone.

```
#include <Servo.h>

#define E1 5 // Direção Motor1
#define M1 4 // Velocidade Motor1
#define servo 9 // Servo
#define echoPin 7 // Echo Pin
#define trigPin 8 // Trigger Pin
#define LEDPin 13 // Onboard LED
#define currentSensor 0 // Current Sensor

int speed;
int direction = 95;
int maximumRange = 2000; // Maximum range needed
int minimumRange = 0; // Minimum range needed
long duration, distance; // Duration used to calculate distance

Servo direcao;
byte incomingByte;

const int numReadings =5;
float readings[numReadings];      // the readings from the analog input
int index = 0;                    // the index of the current reading
float total = 0;                  // the running total
float average = 0;                // the average
float currentValue = 0;

void setup()
{
    Serial.begin(9600);
    Serial1.begin(9600);
    delay(400);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    pinMode(LEDPin, OUTPUT);
    pinMode(M1, OUTPUT);
    speed=0;

```

```
38      direcao.attach(servo);  // attaches the servo on pin 9 to the servo object
39      direcao.write(95);
40      /*zera o vetor das medidas do sensor de corrente*/
41      for (int thisReading = 0; thisReading < numReadings; thisReading++)
42          readings[thisReading] = 0;
43  }
44
45  void loop()
46  {
47    //Serial1 le do pino 0 e 1
48    //serial le do USB
49      if (Serial1.available() > 0) {
50          comando(Serial1.read());
51      }
52      //tambem faz a leitura pelo USB
53      if (Serial.available() > 0) {
54          comando(Serial.read());
55      }
56
57      //leitura do sensor
58      readDistance();
59      //imprime na UART para a BB
60      Serial1.write('d');
61      Serial1.println(distance);
62
63      //leitura do sensor
64      readCurrent();
65      //imprime na UART para a BB
66      Serial1.write('c');
67      Serial1.println(currentValue);
68
69      delay(100);
70  }
71
72  void readDistance(){
73      /*Sequencia de leitura*/
74      digitalWrite(trigPin, LOW);
75      delayMicroseconds(2);
76
77      digitalWrite(trigPin, HIGH);
78      delayMicroseconds(50);
79
80      digitalWrite(trigPin, LOW);
81      duration = pulseIn(echoPin, HIGH,10000);
82
83      /*Calcula distancia baseado na velocidade do som*/
84      distance = duration/58.2;
85
86      /*Indicar erro de leitura*/
87      if (distance >= maximumRange || distance <= minimumRange){
88          Serial.println("-1");
89          digitalWrite(LEDPin, HIGH);
90      }
91      /*Para ao chegar perto de um objeto*/
92      if (distance <= stopDistance){
93          Serial.println(distance);
94          direction=95;
```

```
95          speed=0;
96          digitalWrite(LEDPin, HIGH);
97          direcao.write(direction);
98          digitalWrite(M1,HIGH);
99          analogWrite(E1, speed);
100     }
101     /*Imprime distancia*/
102     else {
103          Serial.print("[cm]: ");
104          Serial.println(distance);
105          digitalWrite(LEDPin, LOW);
106     }
107 }
108
109 void readCurrent()
110 {
111     total= total - readings[index];
112     readings[index] = analogRead(currentSensor); //Raw data reading
113     readings[index] = (readings[index]-520)*5/1024/0.04;
114     total= total + readings[index];
115     index = index + 1;
116     if (index >= numReadings)
117          index = 0;
118     average = total/numReadings;
119     currentValue= average;
120
121     Serial.print("Corrente: ");
122     Serial.println(currentValue);
123 }
124
125 void comando(byte incomingByte){
126     switch (incomingByte) {
127          case 'a':
128               direction-=60;
129               break;
130          case 'd':
131               direction+=60;
132               break;
133          case 'x':
134               direction=95;
135               speed=0;
136               break;
137          case 'w':
138               speed+=60;
139               break;
140          case 's':
141               speed-=60;
142               break;
143          default:
144               break;
145     }
146
147     if (speed>255) speed=255;
148     if (speed<-255) speed=-255;
149     if (direction>155) direction=155;
150     if (direction<35) direction=35;
151
```

```
152     direcao.write(direction);
153     if (speed>=0) {
154         digitalWrite(M1,HIGH);
155         analogWrite(E1, speed);
156     }
157     else {
158         digitalWrite(M1,LOW);
159         analogWrite(E1, -speed);
160     }
161     delay(1);
162 }
```

Apêndice B

Aplicação servidora em Python

Código em Python utilizado pelo servidor para estabelecer a comunicação entre a BeagleBone e o computador remoto utilizando o protocolo UDP e também entre a BeagleBone e o Arduino de forma serial.

```
import Adafruit_BBIO.UART as UART
import serial
import os
import threading
import socket

#habilita a utilizacao da UART2
os.system("echo BB-UART2 > /sys/devices/bone_capemgr.7/slots")
UART.setup("UART2")
ser = serial.Serial(port = "/dev/ttyO2", baudrate=9600)
ser.close()
ser.open()

#def configureSocket():
UDP_IP = "10.0.0.171"
UDP_PORT = 5005
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("0.0.0.0",UDP_PORT))

if ser.isOpen():
  print "Serial is open!"
  ser.write("Hello World!")

def readSerial():
  while True:
    byte = ser.read()
    if byte == "c" :
      corrente = ""
      byte = ser.read()
      while byte != '\n' :
        corrente = corrente + byte
        byte = ser.read()
      resposta = 'Corrente: %s' % corrente
      print resposta          # terminal
      sock.sendto(resposta, (UDP_IP, UDP_PORT)) # socket
```

```
36        #ser.write('C_B: %s\n' % corrente)     # arduino
37
38      if byte == "d" :
39        distancia = ""
40        byte = ser.read()
41        while byte != '\n' :
42          distancia = distancia + byte
43          byte = ser.read()
44        resposta = 'Distancia: %s' % distancia
45        print resposta            # terminal
46        sock.sendto(resposta, (UDP_IP, UDP_PORT)) # socket
47        #ser.write('D_B: %s\n' % distancia)      # arduino
48
49
50  def recieveCommand():
51    while True:
52      #command = raw_input("Enter a command ")
53      data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
54      print "received message:", data
55      ser.write(data);
56
57
58
59  t_serial = threading.Thread(target=readSerial)
60  t_serial.daemon = True
61  t_serial.start()
62
63  recieveCommand();
```

Apêndice C

Script de inicialização para utilização do UART2

O trecho de código abaixo é utilizado para inicializar o módulo de comunicação UART2 da BeagleBone

```
cat /sys/devices/bone_capemgr.7/slots
echo "Habilitando BB-UART2"
echo BB-UART2 > /sys/devices/bone_capemgr.7/slots
echo "UART2 Habilitada"
echo "Arquivo slots:"
cat /sys/devices/bone_capemgr.7/slots
echo "Mensagens do sistema"
dmesg | grep UART
```

Apêndice D

Código em Python da interface de controle

O código a seguir é responsável pela implementação da interface gráfica utilizada no computador remoto bem como a a comunicação com a BeagleBone através da porta 5001 para enviar os comandos para a plataforma móvel. Trata-se de um código em Python e Objective C.

```
from Cocoa import *
import socket
import os

UDP_IP = "10.0.0.89"
UDP_PORT = 5001
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("0.0.0.0",UDP_PORT))
command = 'x'
response = '#'

def send(char):
    global command
    global response
    command = char
    sock.sendto(command, (UDP_IP, UDP_PORT))
    response, addr = sock.recvfrom(1024)


class ClientController(NSWindowController):

    command = 'x'
    TextField1 = objc.IBOutlet()
    TextField2 = objc.IBOutlet()
    currentTextField = objc.IBOutlet()
    #distanceTextField = objc.IBOutlet()

    def windowDidLoad(self):
        NSWindowController.windowDidLoad(self)

    @objc.IBAction
```

```
32      def dirRight_(self, sender):
33          send('d')
34          self.updateDisplay()
35
36      @objc.IBAction
37      def dirLeft_(self, sender):
38          send('a')
39          self.updateDisplay()
40
41      @objc.IBAction
42      def dirFront_(self, sender):
43          send('w')
44          self.updateDisplay()
45
46      @objc.IBAction
47      def dirBack_(self, sender):
48          send('s')
49          self.updateDisplay()
50
51      @objc.IBAction
52      def stop_(self, sender):
53          send('x')
54          self.updateDisplay()
55
56      @objc.IBAction
57      def loop_(self, sender):
58          time.sleep(1)
59
60      def updateDisplay(self):
61          self.TextField1.setStringValue_(UDP_IP)
62          self.TextField2.setStringValue_(UDP_PORT)
63          self.currentTextField.setStringValue_(command)
64          #self.distanceTextField.setStringValue_(response)
65
66
67  if __name__ == "__main__":
68
69      app = NSApplication.sharedApplication()
70
71      # Initiate the contrller with a XIB
72      viewController = ClientController.alloc().initWithWindowNibName_("Window")
73
74      # Show the window
75      viewController.showWindow_(viewController)
76
77      # Bring app to top
78      NSApp.activateIgnoringOtherApps_(True)
79
80      from PyObjCTools import AppHelper
81
82      AppHelper.runEventLoop()
```

Apêndice E

Código em Python para monitorar a latência

O código em Python abaixo é executado no computador cliente e tem como objetivo verificar a latência de comunicação com a BeagleBone.

```
import threading
import socket
import time
import select
from multiprocessing import Process
BeagleBone_IP="10.0.0.89"
Port_Latency = 5003
sock_ltc = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock_ltc.bind(("0.0.0.0",Port_Latency))
sock_ltc.setblocking(0)
pacotes = 100
def latencyRequest():
        avg = 0
        cont = 0
        for i in range(0, pacotes):
            sock_ltc.sendto('p', (BeagleBone_IP, Port_Latency))
            send_time = time.time()
            ready = select.select([sock_ltc], [], [], 0.002)
            if ready[0]:
                data = sock_ltc.recv(1024)
                if data == 'p':
                    cont += 1
                    recieve_time = time.time()
                    avg += recieve_time - send_time
            time.sleep(0.005)
        avg = avg/pacotes
        print '%0.3f ms ' % (avg*1000.0)
        print 'packet loss: %.1f ' % (pacotes-cont)
while True:
    p1 = Process(target=latencyRequest)
    p1.start()
    p1.join()
    time.sleep(0.5)
```

Apêndice F

Código em Python para monitorar os sensores

O código abaixo é resposável pelo recebimento das informações dos sensores no computador cliente. Sua implementação é em Python.

```
import threading
import socket
import time
import select


BeagleBone_IP="10.0.0.89"

Port_Sensor = 5002
sock_sen = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock_sen.bind(("0.0.0.0",Port_Sensor))
sock_sen.setblocking(0)


def sensorRead():
    print "Sensor"
    while True:
        send_time = time.time()
        ready = select.select([sock_sen], [], [], 0.01)
        if ready[0]:
            data = sock_sen.recv(1024)
            print data



t_sensor = threading.Thread(target=sensorRead)
t_sensor.daemon = True
t_sensor.start()

while True:
    pass
```