ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
DEPARTAMENTO DE ENGENHARIA MECATRÔNICA E DE SISTEMAS MECÂNICOS

Anderson Nishihara

CONTROLE DE ROBÔ MÓVEL PIONEER 3-AT COM IPHONE/IPOD TOUCH

São Paulo
2010

Anderson Nishihara

CONTROLE DE ROBÔ MÓVEL PIONEER 3-AT COM IPHONE/IPOD TOUCH

Monografia apresentada à Escola Politécnica da Universidade de São Paulo relacionado à disciplina PMR2550 – Projeto de Conclusão de Curso II.

Curso de Graduação: Engenharia Mecatrônica

Orientador: Prof. Dr. Jun Okamoto Jr.

São Paulo

2010

FICHA CATALOGRÁFICA

Nishihara, Anderson
Controle de robô móvel Pioneer 3-AT com iPhone/iPod Touch / A. Nishihara. -- São Paulo, 2010.
69 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Robôs 2. Redes de computadores I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

Aos meus pais, irmãos e à todos aqueles que me apoiaram durante meus trabalhos e realizações.

Anderson Nishihara

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Jun Okamoto Jr., pela orientação ao longo do desenvolvimento deste trabalho de formatura e de outros trabalhos ao longo da graduação.

Aos membros do Laboratório de Percepção Avançada da Escola Politécnica da Universidade de São Paulo que me auxiliou durante a graduação.

À Susanna, minha namorada, pelo carinho e apoio nos momentos mais difíceis.

Aos professores da Escola Politécnica, especialmente aos professores do Departamento de Engenharia Mecatrônica, que contribuíram para a minha formação.

RESUMO

O presente trabalho tem como objetivo o controle remoto de um robô móvel Pioneer 3-AT através de uma rede wireless Wi-Fi de computador. O controle será feito utilizando-se do acelerômetro e tela sensível a múltiplos toques do “iPhone”: enquanto a tela múltiplo toques será utilizado como meio de entrada para controle de velocidade, o acelerômetro será utilizado para obter ângulos de inclinação do dispositivo para simular um volante de automóvel ao controlar o robô móvel.

O desenvolvimento do software será realizado estudando-se primeiramente as ferramentas necessárias e suas características. Ao fim deste processo, será feito o levantamento dos requisitos do projeto e será descrito cada aspecto do projeto para a solução do problema proposto. Ao fim, será realizado testes relevantes no controle remoto do robô móvel. O produto final do projeto é um aplicativo protótipo que será executado em um “iPhone” ou “iPod Touch”

ABSTRACT

The main objective of the present work is controlling a Pioneer 3-AT mobile robot remotely using a Wi-Fi wireless computer network. The control will be done using the accelerometer and the multiple touch screen of the “iPhone”: while the multiple touch screen will be used as means to control the velocity, the accelerometer will be used to obtain the tilt angles of the device to simulate and automobile steering wheel to control the mobile robot. Software development is primarily done by studying the necessary tools and features. At the end of this process, the project requirements will be evaluated and every aspect of the project to archive the objective will be described. In the end, relevant tests will be performed to evaluate the work. The final product is a prototype application that will be run on an “iPhone” or “iPod Touch”.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Robôs móveis são utilizados em diversos campos como na indústria, em operações militares, vigilância e como produtos de consumo, como entretenimento e trabalhos de limpeza. O controle do robô, geralmente, é feito por computadores embarcados ou em computadores remotos com boa capacidade de processamento e dispositivos grandes que não permitem a mobilidade.

O “iPhone” e o “iPod Touch” são dispositivos com capacidades diversas lançado em meados de 2007. Enquanto o “iPhone” possui a função de telefonia, o “iPod Touch” conserva todas as outras características do “iPhone”, exceto alguns recursos como o GPS. A grande atração destes dispositivos é a facilidade de manuseio e sua interface inovadora, o qual aceita gestos múltiplos em sua tela sensível a toque. Há diversas aplicações disponíveis para este celular, como leitores de cartão de visita, jogos e aplicativos para acessar o home banking de uma instituição financeira. Há programas para controle de robôs comerciais (ISPYKEE, 2010), robôs de 6 pernas (CRUNCHGEAR, 2010) e quadcópteros (PARROT, 2010). Além disso, há pesquisa acadêmica estudando novos modos de interação com um robô humanóide utilizando o “iPhone” (SUGIURA, FERNANDO, *et al.*, 2009).

Aliando a mobilidade, a conectividade e a capacidade de processamento, o “iPhone” e “iPod” Touch tornam-se dispositivos alternativos capacitados para o controle de um robô móvel.

O presente projeto tem como objetivo fornecer um meio alternativo para o controle de robô móvel, aliando mobilidade e facilidade de uso. Para isto, será construído um programa para o “iPhone”/”iPod Touch” que utilizará recursos como o acelerômetro e a tela multi-toques para controlar o robô móvel “Pioneer 3-AT” através de uma rede wireless. Enquanto a tela multi-toques será utilizada como meio de entrada de dados e controle de velocidade, o acelerômetro será utilizado para obter ângulos de inclinação do dispositivo para simular um volante de automóvel.

No capítulo 2, será discutido os conceitos básicos necessários para o desenvolvimento e entendimento do projeto proposto, como protocolos de comunicação e orientação a objetos.

Durante o capítulo 3, será descrito as várias ferramentas utilizadas na concepção e implementação do trabalho proposto.

O capítulo 4 será iniciado com a discussão sobre o problema a ser resolvido, levantando requisitos necessários para controlar o robô móvel, seguido da modelagem da arquitetura necessário para implementar os vários algoritmos propostos para resolver cada aspecto do *software.*

No capítulo 5 será discutido o que poderá ser realizado na mesma linha de trabalho.

Finalmente, no capítulo 6 serão feitas as considerações finais sobre o trabalho proposto.

2 CONCEITOS BÁSICOS

2.1 Protocolo de comunicação

Protocolo de comunicação (na ciência da computação) é um conjunto de normas pré-estabelecidas descritas formalmente para que seja possível a conexão e troca de mensagens digitais entre dois dispositivos. Um protocolo descreve tanto a sintaxe, a semântica quanto a sincronização da comunicação e pode ser implementada tanto por *hardware* quanto por *software*.

Em geral, os protocolos de comunicação possuem as seguintes descrições:

- Formato do dado de transmissão. *Bitstrings* são divididos em área e cada área possui informações relevantes ao protocolo. Conceitualmente o *bitstring* é dividido em *header* e *data*. A mensagem é armazenado da área do *data* enquanto o header contém informações relevantes ao protocolo. Como as transmissões são limitadas em tamanho, devido ao número de erros que é proporcional ao tamanho do *bitstring* a ser enviado, *bitstrings* maiores que a unidade máxima de transferência (*maximum transfer unit: MTU*) são divididos em pedaços de tamanho apropriado.
- Formato do endereço do dado de transmissão. Para identificar o transmissor e o receptor de um dado, são utilizados os endereços. Os endereços são armazenados na área do *header* do *bistring,* permitindo ao receptor determinar se o *bitstring* recebido são para eles mesmos, se devem ser processados ou se devem ser descartados.

- Mapeamento de endereço. Um protocolo pode necessitar mapear um endereço de um esquema para outro, como por exemplo, traduzir o endereço lógico IP para um endereço de *hardware*.
- *Routing*. Em arquiteturas no qual os sistemas não estão conectados diretamente, sistemas intermediários são utilizados para redirecionar a mensagem até o receptor. A determinação da rota o qual a mensagem deve fazer é chamado de *routing*.
- Detecção de erros é necessário devido a nenhuma rede ser livre de erros. Os erros podem acontecer devido a diversos fatores, como por exemplo, ruídos eletromagnéticos. Geralmente, a detecção de erro é feita calculando o *checksum* do pacote de mensagem. Quando o *checksum* do pacote enviado é diferente do *checksum* do pacote recebido, a mensagem é rejeitada.
- *Acknowledging* da recepção do pacote pode ser utilizado para prevenir que o transmissor envie reenvie os pacotes.
- Perda de informação. Para resolver o problema de alguns pacotes serem perdidos ou sofrerem grandes atrasos, o transmissor espera que uma mensagem de *acknowledging* do pacote seja enviada em um determinado intervalo de tempo e, caso este intervalo de tempo seja ultrapassado, o pacote é retransmitido.
- Direção do fluxo da informação. Quando a transmissão só pode ocorrer em um sentido para cada instante, é necessário o controle da direção do fluxo de informações. Para ganhar o controle da linha, o transmissor deve esperar até que a linha fica sem uso e enviar uma mensagem indicando a intenção de envio de mensagem. O Receptor responde com um *acknowledging* e espera a transmissão da mensagem. O transmissor começa a transmitir a mensagem apenas após o recebimento do *acknowledging*.
- Controle de seqüenciamento. Alguns *bitstrings*, por serem muito longos, são divididos em vários pacotes. Os pacotes podem ser recebidos fora de ordem ou perdidos durante uma transmissão.
- Controle de fluxo pode ser necessário quando o transmissor envia mensagens mais rápido que o processamento de dados pelo receptor,

nesses casos, preparativos devem ser feitos para diminuir a velocidade de envio de mensagens pelo transmissor.

O envio do dado é apenas uma parte do problema. O dado recebido deve ser avaliado no contexto da conversa, portanto, o protocolo deve especificar regras de descrição do contexto e explanar se o formato do dado encaixa-se no contexto ou não. Esse tipo de regra expressa a sintaxe da comunicação. E, finalmente, para expressar a semântica da comunicação, existem regras para determinar se o dado recebido é significativo ou não no contexto. Geralmente, essas duas regras (semântica e sintaxe) e outras especificações formais são representados na forma de máquinas de estado.

2.2 Modelo de arquitetura de redes em camadas TCP/IP

O modelo de arquitetura em camadas “TCP/IP” descreve como um dado passa da camada de aplicativos até a camada física da rede. Quando um dado é enviado, cada camada trata toda a informação recebida da camada superior como dado, adiciona informação de controle (*header*) ao dado e passa a informação para a camada inferior. Quando o dado é recebido, é realizado um processo inverso: cada camada recebe o dado, remove a informação de controle (*header*) e passa a informação para a camada superior, até alcançar a camada do aplicativo (PARZIALE, BRITT, *et al.*, 2006).

O modelo possui 4 camadas:

- Camada de aplicativo. É a camada que define para a interface de usuário os processos para comunicação e transferência de dados na rede. Além disso, especifica o formato de transferência de dado independente de uma arquitetura, encripta e desencripta dados, comprime e descomprime os dados e gerencia as sessões de conexão.
- Camada de transporte. Na arquitetura TCP/IP, há dois tipos de protocolos na camada de transporte principais. O *Transmission Control Protocol* (TCP) garante a transmissão da informação enquanto o *User Datagram Protocolo* (UDP) transporta a informação sem garantia da integridade dos dados em prol da velocidade de transmissão.

- Camada de rede. O *Internet Protocol* (IP) é o principal protocolo de uma rede em camadas TCP/IP. Todos as comunicações das camadas superiores e inferiores devem ser transportados através do IP.
- Camada de acesso a rede e *hardware*. É a camada no qual é feito a detecção e correção de erros de transmissão de pacotes de dados e define os meios físicos de envio de dados, assim como as características mecânica, ópticas e elétricas.

A figura abaixo ilustra como um dado é tratado no envio de informação através da arquitetura:

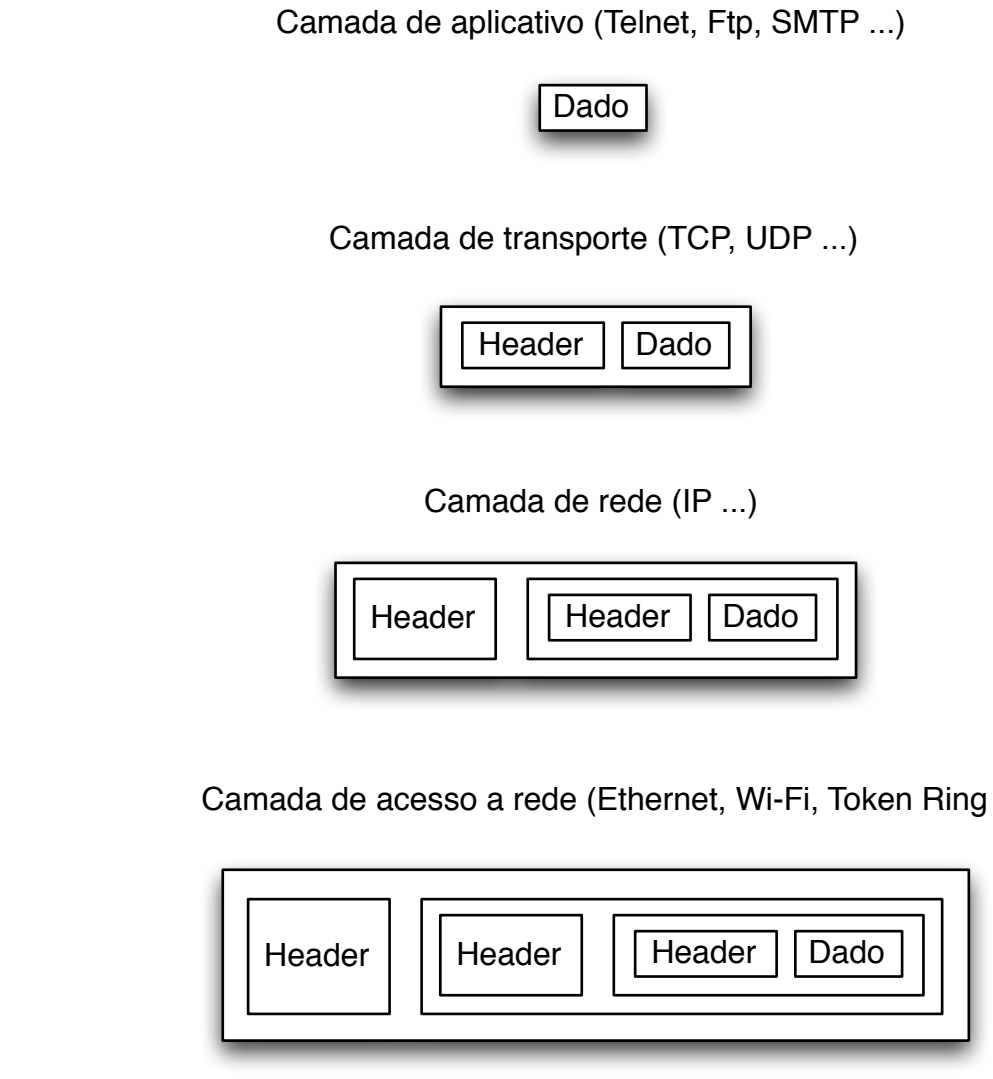

Figure 1 - Encapsulamento de dado ao longo das camadas de rede

2.3 Verificação de erros em pacotes de dados

Em uma transmissão de dados em rede, pode haver a presença de ruídos. Por isso, deve-se utilizar algum método de detecção de erros.

Normalmente, utiliza-se um *checksum* para verificar erros. *Checksum* é um dado de tamanho fixo gerado a partir de um bloco de dados com a finalidade de detectar erros acidentais na transmissão ou armazenamento do dado. A integridade do dado pode ser verificado recalculando o *checksum* do bloco de dado e comparando o resultado com o *checksum* armazenado. Caso o *checksum* calculado e armazenado sejam iguais, é provável que o dado não foi alterado (intencionalmente ou não).

Um bom algoritmo de *checksum* é um algoritmo que irá mostrar um resultado diferente com grande probabilidade caso o dado esteja corrompido.

2.4 Orientação a objetos

Segundo (PRESSMAN, 2006), a orientação a `objetos`[1] é uma abordagem, voltada ao desenvolvimento de software, no qual é proposto enfoque no relacionamento entre o homem, a natureza e as entidades feitas pelo homem. Deste modo, o domínio do problema é caracterizado como um conjunto de `objetos` que possuem `atributos` e comportamentos específicos.

Do ponto de vista da engenharia de software, objetos são abstrações de dados o qual encapsulam dados e operações sobre estes dados. Este ponto de vista reforça a localidade da informação e a independência de representação. Do ponto de vista formal, um `objeto` é uma máquina de estado com um conjunto finito de estados e um conjunto finito de funções de estado. As funções de estado mapeiam estados anteriores e entradas para novos estados e saídas (VLIET, 2008).

Um `objeto` não encapsula apenas seu estado, o comportamento também é encapsulado, ou seja, a maneira pela qual o `objeto` age sobre outros `objetos` e outros `objetos` agem sobre o `objeto`. O comportamento de um `objeto` é descrito em termos de `serviços` (chamados também de `método` ou `operação`) fornecidos

[1] Termos Específicos da Orientação a Objetos são apresentados em `Courier New`

pelo objeto. Estes serviços são chamados através de mensagens enviadas pelo objeto que chama este serviço no objeto no qual o estado é mudado (VLIET, 2008)

Na programação orientada a objetos, os objetos são definidos por classes. Cada classe determina o comportamento (definido nos métodos, serviços ou operações) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. O conceito de classes de objetos inerentemente leva a formação de uma biblioteca de classes e objetos reusáveis, o que torna atrativo o uso do conceito de objetos no desenvolvimento de *softwares.*

Do ponto de vista da programação orientada a objetos, três características tornam os sistemas orientados a objetos únicos (PRESSMAN, 2006):

- Encapsulamento. A implementação interna dos detalhes do dado e operações são escondidos do mundo externo (ocultação de informação). Deste modo, a propagação de efeitos colaterais devido a mudanças são reduzidas.
- Polimorfismo. É a capacidade de definir propriedades similares através de métodos com mesmo nome, porém, diferentes parâmetros.
- Herança. É a capacidade de definir uma classe com todos os atributos e métodos da classe do qual é herdada (classe base). Descendentes da classe base podem escolher em reter a implementação fornecida pela classe base ou podem sobrescrever a implementação. Qualquer mudança feita na classe base é imediatamente herdada para a classe descendente. Quando a classe base não possui implementação, ela é chamada de interface.

2.5 Framework

Frameworks modelam um domínio específico ou um importante aspecto disso. Eles representam o domínio como um projeto abstrato constituída de classes abstratas (ou interfaces). O projeto abstrato é mais do que um

conjunto de classes, pois este define como as instâncias das classes permitem a colaboração com cada um em tempo de execução. Efetivamente, age como um esqueleto que determina como objetos do *framework* relacionam-se entre si (RIEHLER, 2000).

Um *framework* é formado por um conjunto de implementações na forma de classes abstratas e concretas. Implementações abstratas são classes abstratas que implementam partes da abstração de um framework (expressas por uma interface ou uma classe abstrata), mas deixam decisões de implementação para subclasses. Isto é feito utilizando o princípio de "*Design by Primitives"*. O "*Design by Primitives"* baseia a implementação da classe sobre um pequeno conjunto de operações primitivas que são deixadas em aberto para implementação por subclasses. Subclasses concretas implementam estas operações de modo que possam ser instanciados e utilizados sem criar mais subclasses (RIEHLER, 2000).

2.6 API

Uma API (*Application Programming Interface*) é uma interface implementada por software que permite a interação com outros *softwares.* É uma abstração que descreve a interface para interação provendo um conjunto de funções a serem utilizados por componentes de um sistema de *software.* O *software* que fornece as funções descritas por uma API é chamado de implementação da API.

Uma API não é uma interface de usuário e, sim, entre *softwares.* Através de uma API, aplicações comunicam-se sem nenhuma intervenção do usuário ou conhecimento.

Em linguagens orientadas a objetos, uma API geralmente inclui uma descrição de um conjunto de definições de classes com um conjunto de comportamentos associados. Neste contexto, uma API pode ser concebida como a totalidade de todos os métodos publicamente expostos pelas classes (interface da classe), ou seja, a API determina os métodos com os quais o software interage e manipula os objetos derivados das definições das classes.

2.7 Model-View-Control

A arquitetura “Model-View-Control” é um modelo no qual o software é composto basicamente por três tipos de objetos: o modelo (*model*), a visão (*view*) e o controle (*control*). A premissa básica do MVC é separar a lógica do aplicativo da interface do usuário, permitindo desenvolvimento independente, além de facilitar os testes de interface gráfica do usuário e de funcionalidade do aplicativo.

O modelo é utilizado para a manipulação da informação, notificar quando há alteração nos dados e encapsular a funcionalidade do software. O modelo é uma domínio específico de representação do dado no qual o aplicativo utiliza. A lógica do domínio de informação adiciona significado ao dado sem tratamento(como por exemplo, calcular a trajetória de um robô e calcular os dados necessários para enviar o comando de ligar sonar do robô). Quando o modelo muda de estado, ele notifica as visões para que sejam atualizadas.

Apesar de muitos aplicativos utilizar um banco de dados para persistir os dados, a arquitetura MVC não menciona a camada de acesso aos dados, pois este deve ser encapsulado pelo modelo.

A visão apresenta o modelo em uma forma adequada para a interação e visualização, normalmente em um elemento de interface. Várias visões podem existir para um único modelo por diferentes motivos.

O controlador recebe as entradas e respondem interagindo com os objetos de modelo. A lógica do software está no controlador. O controlador recebe entradas do usuário e, a partir das entradas, age sobre o modelo e a visão baseado na entrada.

Um dos benefícios fornecidos pela arquitetura MVC é a simplicidade e facilidade nos testes da interface do usuário. A modificação dos gráficos sem alteração do comportamento da interface é facilitado ao retirar o código da visão. Ao retirar o máximo possível de código da visão, tornando o comportamento independente da interface, torna-se natural a estruturação do código e separação da lógica em camadas.

Softwares desenvolvidos na arquitetura MVC têm duas grandes desvantagens: a primeira é que o modelo precisa comunicar a visão mudanças de estado, a outra desvantagem é que a visão precisa conhecer o modelo. A visão, de fato, é atualizada quando é notificada de mudanças ocorridas no modelo.

2.8 Levantamento de requisitos

O levantamento de requisitos consiste num processo onde são identificados todos os diferentes requisitos que um sistema de *software* deverá satisfazer quando se encontrar funcional. Este processo recorre a diferentes técnicas, algumas delas complementares entre si. O objetivo final é obter todos os requisitos idealizados para o sistema, possivelmente classificados por ordem de importância, descritos o mais claramente possível e devidamente validades pelos interessados (*stakeholders* do sistema).

A máxima prioridade do processo de levantamento de requisitos é alcançado quando os requisitos são descritos com clareza e são abrangentes, tendo em vista a necessidade de transição do conhecimento dos requisitos do sistema para os desenvolvedores que irão implementar e para os utilizadores do sistema, garantindo que todo o conteúdo pretendido esteja identificado antes do processo de implementação iniciar-se, de modo a facilitar a arquitetura e planejamento da implementação da solução, evitando o retrabalho.

Entre as técnicas utilizadas no levantamento de requisitos, as mais conhecidas e aconselhadas são:

- Identificação dos usuários: determinação clara de quem irá utilizar o sistema e de quem o projetou, destacando os objetivos inicias por trás da idéia, de modo a ser possível entender o que esperam que o sistema cumpra;
- Entrevistas com os usuários do sistema: consiste em efetuar entrevistas com os utilizadores e visionários do sistema, tentando obter uma idéias das várias necessidades que o sistema deve satisfazer;
- *Workshop* de requisitos: sessões de grupo com os utilizadores do sistema, promovendo o debate e discussão de idéias sobre o sistema a desenvolver.
- Prototipagem: criação, apresentação e debate de modelos de intereção não funcionais que ajudem a ilustrar como o sistema irá comportar-se, possibilitando o *feedback* mais detalhado dos usuários sobre o sistema;
- Diagrama de Caso de Uso: descreve a funcionalidade proposta para o novo sistema;

- Expansão de Diagrama de Casos de uso: consiste na explicitação de todas as diferentes funcionalidades do sistema, o que permitirá inferir e identificar mais claramente outras necessidades.

2.8.1 Caso de uso

O caso de uso (*use case*) é um ferramenta utilizada para levantar os requisitos necessários de um software. O caso de uso representa uma unidade funcional provida pelo sistema, subsistema, ou classe manifestada por sequências de mensagens intercambiáveis entre os sistemas e um ou mais atores. Cada caso de uso tem uma descrição o qual descreve a funcionalidade que irá ser construída no sistema proposto, definindo o comportamento do sistema.

A idéia básica na modelagem do caso de uso é focar em quem irá utilizar o sistema para identificar com clareza como o sistema deve comportar-se e, o que o sistema deve fazer para que faça algo útil para os usuários do sistema.

O modelo de caso de uso inclui os seguintes componentes (BITTNER e SPENCE, 2002)

- Atores: representam as pessoas ou coisas que interagem como o sistema de alguma forma. Por definição, eles são de fora do sistema. O foco é colocado sobre os atores para que tenha-se certeza que o sistema faça algo útil. Atores possuem um nome e uma pequena descrição, e são associados ao caso de uso com o qual interagem;
- Casos de uso: representam coisas de valor que o sistema executa para os atores. Casos de uso não são funções ou características, e não podem ser decompostos. Casos de uso possuem um nome e uma breve descrição. Eles também possuem descrições detalhadas que são essencialmente estórias sobre como os atores utilizam o sistema para fazer algo que consideram importante, e o que o sistema faz para satisfazer estas necessidades.

O conjunto de todos os atores e casos de uso descrevendo um sistema é conhecido como modelo de casos de uso do sistema.

3 FERRAMENTAS

3.1 Linguagem para a implementação do projeto

A programação para os dispositivos iPod Touch e iPhone é feita com a linguagem Objective-C. O Objective-C é definida como um conjunto de extensões da linguagem ANSI C padrão, e tem como principal característica a orientação a objetos, o que permite o uso de mecanismos como polimorfismo e herança. A maior parte da adições feitas são baseadas no “Smalltalk”, uma das primeiras linguagens de programação orientada a objetos. Como o Objective-C deriva da linguagem C, os programas em C podem ser facilmente adaptados ao Objective-C, além disso, a sintaxe descrita em C pode ser utilizado em Objective-C sem nenhuma modificação, ou seja, o Objective-C incorpora todas as características do C, o que torna possível executar tarefas orientado a objetos e utilizar técnicas de programação procedural (APPLE, 2010).

Comparativamente a outras linguagens orientadas a objetos derivadas do C, Objective-C é uma linguagem dinâmica. O Compilador preserva grande quantidade de informação sobre os objetos para ser utilizados em tempo de execução. Decisões que poderiam ser feitas durante a compilação são adiadas até a execução do programa. Deste modo, os programas em Objective-C garantem grande flexibilidade e poder. Por exemplo, o dinamismo do Objective-C garantem dois grandes benefícios dificilmente encontrados em outras linguagens orientadas a objetos (APPLE, 2010):

- Objective-C suporta um estilo aberto de *dynamic binding*, um estilo que pode acomodar uma arquitetura simples para interfaces de usuário interativa. As mensagens não são necessariamente presas pela classe do receptor e nem pelo nome do método, portanto, um framework pode permitir escolhas do usuário em tempo de execução.
- Possibilidade de construção de ferramentas de desenvolvimento sofisticadas. Uma interface para o sistema em tempo de execução permite o acesso a informações sobre o aplicativo sendo executado, possibilitando desenvolver ferramentas para monitorar, intervir, e revelar a estrutura de baixo e atividade dos aplicativos.

Por baixo do Objective-C, os métodos são representados por um *selector,* uma string que representa o método a ser chamado. Quando uma mensagem é enviada, o *selector* é enviado no tempo de execução do Objective-C e comparado com uma lista de métodos disponíveis, e, finalmente, a implementação do método é chamada. Como o *selector* é um dado de texto, é permitido salvar em um arquivo, transmitir em uma rede ou entre processos, ou manipular de outras formas. A implementação do método é vista em tempo de execução e não durante a compilação. Isto permite que o mesmo *selector* referencie diferentes implementações (APPLE, 2010).

3.2 Arquitetura Cocoa

"Cocoa" é uma *application programming interface* (API) nativa do sistema iOS do "iPhone" e "iPod Touch". Consiste basicamente de duas frameworks:

- "Foundation Kit". É uma biblioteca genérica orientada a objetos que fornece facilidade na manipulação de *strings*, *run loops* e outras funções que não estão diretamente relacionadas à interface gráfica do usuário.
- "Application Kit". Contém código no qual os programas podem criar e interagir diretamente com a interface gráfica do usuário.

Um ponto importante da arquitetura "Cocoa" é a possibilidade de implementar toda a infra-estrutura básica do programa e concentrar-se unicamente no aspecto do conteúdo do objeto, deixando o "Cocoa" gerenciar os aspectos da interface como o redimensionamento, rolagem de tela e outros detalhes de desenho gráfico. Isto é possível devido ao projeto do "Cocoa" seguir estritamente os princípios do "Model-View-Control" (MVC).

3.3 Ambiente de desenvolvimento Xcode

"Xcode" consiste de um conjunto de ferramentas de desenvolvimento para o sistema operacional "Mac OS X" e "iOS" (presente no "iPhone" e "iPod Touch"). O aplicativo principal do conjunto é um *integrated development environment* (IDE) voltado para o desenvolvimento de *software*, tendo como principal linguagem de

programação o “Objective-C” com a API “Cocoa”. Uma IDE é um aplicativo para o auxílio no desenvolvimento de *software* composto de um editor de códigos, compilador e ferramentas para *debugging*. O conjunto do “XCode” também inclui documentação do desenvolvedor da “Apple Inc.”, e o “Interface Builder”, um aplicativo utilizado para construir interface gráfica para o usuário (XCODE, 2010).

Com o advento do *kit* de desenvolvimento de *software* para o sistema “iOS”, “Xcode” passou a gerenciar todos os testes no “iPhone” e “iPod Touch”, empacotando automaticamente os aplicativos com os certificados corretos, e instalando aplicativos no aparelho. Além disso, para auxiliar o desenvolvimento para o dispositivo, o “Xcode” inclui um simulador do aparelho para testes preliminares.

3.4 Redirecionamento de dados de rede Wi-Fi para Serial RS-232

Como o objetivo do trabalho é controlar o robô diretamente através do sistema ARCOS, é necessário que os pacotes de mensagens que trafegam na rede sem fio sejam redirecionados para a porta RS-232 do micro-controlador. É possível fazer isto de duas formas:

- *Hardware* adicional que tem como única função redirecionar dados de rede *Wi-Fi* para a serial RS-232, ligado diretamente na porta RS-232 do micro-controlador do robô móvel;
- *Software* sendo executado em um computador adicional com suporte a conexão em redes *Wi-Fi* e ligado com a porta RS-232 do micro-controlador com o sistema ARCOS.

Há diversos *hardware* e *software* disponíveis. Como software, podemos citar o Serial Por Mapper, com0com e ser2net (utilizado no trabalho).

3.5 Robô móvel Pioneer 3-AT

O robô móvel “Pioneer 3-AT” pertence a plataforma “MobileRobots”. Todos os robôs desta plataforma possuem um corpo de alumínio, um sistema balanceado de movimento, motores DC reversíveis, controle de motor, *encoders* e baterias, todos

estes dispositivos gerenciados por um microcontrolador e um software servidor (MOBILEROBOTS, 2007).

3.5.1 Especificações

3.5.1.1 Características físicas

O robô móvel pesa cerca de 9 kg e é construído sobre uma sólida base de alumínio, conferindo uma boa capacidade de carregar peso adicional de 35 kg.

O robô é constituído das seguintes partes principais:

- Convés
- Botão de parada do motor
- Painel de Controle
- Painel de acesso
- Sonares
- Motores, rodas e *encoders*
- Bateria e sistema de alimentação

3.5.1.1.1 Convés

O “Pioneer 3-AT” possui uma placa superior para o acesso fácil aos componentes internos do robô. O convés é uma simples superfície plana para a montagem de projetos e acessórios, como câmeras e sensores laser. Compartimentos com aberturas em cada lado do convés permite a ligação de cabos dos acessórios aos conectores dos painéis laterais do robô. Uma tampa removível no meio do deck permite o acesso ao interior do robô.

Ao montar os acessórios do robô, deve-se colocar o centro de massa dos acessórios sobre as rodas do robô e, caso necessário, deve-se colocar pesos de contra-balanço.

3.5.1.1.2 Botão de parada do motor

O botão de parada do motor está localizado na parte traseira do convés. É um botão do tipo emergência para a parada emergencial do robô. Ao ser acionado, os motores param e é acionado alto falantes do tipo *buzzer.*

3.5.1.1.3 Painel de controle

No painel de controle é onde pode-se ter o acesso ao micro-controlador com o sistema ARCOS. Consiste de botões de controle e indicados, além de uma porta serial RS-232.

3.5.1.1.4 Painel de acesso

Existe um painel de acesso do lado direito onde pode-se instalar conectores dos acessórios e controles. Neste painel de acesso é onde fica um computador embarcado com sistema Linux, com conectores para monitor externo, teclado, mouse e para rede ethernet, assim como botões de *reset* e interruptor de alimentação.

3.5.1.1.5 Sonares

O sistema ARCOS suporta nativamente até 4 conjuntos de sonares, cada um com até 8 transdutores que fornecem detecção de objetos e informação de distância para sistemas anti-colisões, reconhecimento de características, localização e navegação. A posição dos sonares são fixos e fornecem informação de distância próximo de 360 graus.

3.5.1.1.6 Motores, rodas e encoders

Os motores do “Pioneer 3-AT” são reversíveis de corrente contínua, de alta velocidade e torque, equipados com *encoders* ópticos de alta resolução para o fornecimento de posição e velocidade precisa.

As rodas possuem pneus pneumáticos para o uso em diferentes tipos de terrenos.

3.5.1.1.7 Bateria e sistema de alimentação

O robô pode possuir até 3 baterias de ácido de 7 ampères-hora e 12 V de corrente contínua, fornecendo até 252 watt-hora.

3.5.1.2 Especificações Técnicas

O robô móvel possui as seguintes especificações:

- Processador RISC 32-bits de 44,2368 MHz e com 32K de memória RAM e 128K de memória FLASH.
- 4 portas seriais RS-232 configuráveis de 9,6 a 115,2 kbaud
- 4 conjuntos de sonares de até 8 sonares cada
- 2 conectores digitais de entrada de 8 bits
- Entrada e saída digital de 8 bits, entrada analógica e alimentação DC de 5/12 V
- Porta para conexão de um giroscópio
- Entrada para joystick
- Painel de controle de usuário
- Conector serial *HOST* para o micro-controlador
- Leds bi-coloridos para indicar nível de bateria
- 2 seletores de tensão 5V e 12V
- botões de *reset* e para ligar e desligar motor
- *Buzzer* programável
- Interface para controle de potência do motor com PWM

3.5.1.3 Software Cliente

Todos os robôs da plataforma MobileRobots operam como servidor em um ambiente cliente-servidor: o micro-controlador manipula os detalhes de baixo nível do robô, incluindo a manutenção da velocidade e direção, aquisição de leitura de sensores, como o sonar, e gerenciamento de acessórios extras, como um manipulador robótico. Para completar a arquitetura cliente-servidor é necessário uma conexão com um PC: um software rodando no computador conectado ao micro-controlador do robô através da serial RS-232 que fornece o controle do robô em alto nível, como planejamento de rota, desvio de obstáculos, reconhecimento, navegação, localização, entre outros (MOBILEROBOTS, 2007).

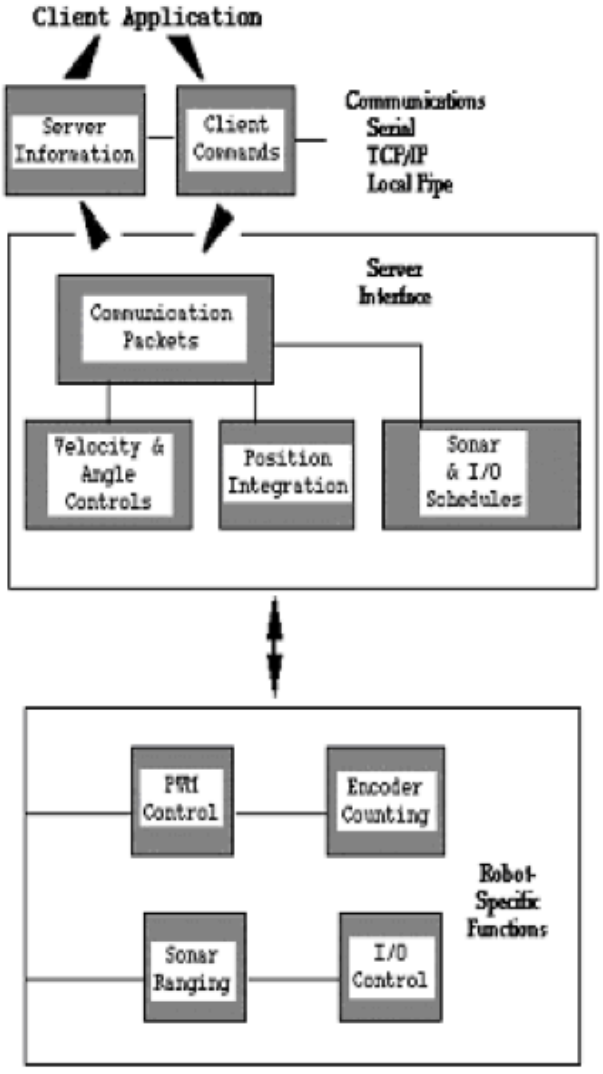

Figura 1 - Arquitetura cliente-servidor (MOBILEROBOTS, 2007)

Um importante benefício da arquitetura cliente-servidor é que o mesmo cliente pode ser executado para conectar-se a diferentes servidores de robôs. Há também a possibilidade de diversos clientes compartilhar a responsabilidade de controlar um único robô, o que permite a experimentação em comunicação distribuída, planejamento e controle.

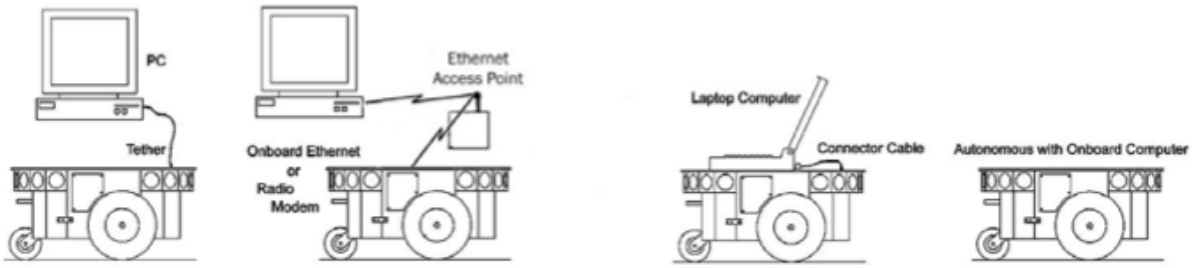

Figura 2 – Alternativas de controle do robô móvel da plataforma MobileRobots (MOBILEROBOTS, 2007)

3.5.1.4 Comunicação do cliente com o servidor

Para o controle do robô móvel, é necessário primeiro que o cliente estabeleça uma conexão com o servidor (ARCOS). A conexão é feita através de uma camada de protocolo próprio e o envio e recebimento de dados através de pacotes descritos a seguir.

3.5.1.4.1 Pacote de dados do cliente

O controle do robô pelo software cliente é feito pela porta serial RS-232. Através dela, é possível comunicar-se através de pacotes em protocolos especiais, um pacote de comando do cliente para o servidor e um pacote de informações do servidor para o cliente (SIP – *Server Information Packet*). A tabela abaixo mostra como é composto cada pacote de comando enviado pelo cliente:

Componente	Byte	Valor	Descrição
Header	2	0xFA, 0xFB	O Header do pacote; igual para servidor e cliente
Byte de contagem	1	N	Número de bytes de commandos/argumentos mais os dois bytes do checksum, sem incluir o byte de contagem e os bytes do header
Comando do cliente	1	0 - 255	Número correspondente ao comando do cliente
Tipo de argumento	1	0x3B ou 0x1B ou 0x2B	Tipo de argumento requerido pelo comando: Inteiro positivo, inteiro negativo ou absoluto, ou string
Argumento	n	Informação	Argumento do comando; sempre inteiro de 2 bytes ou string contendo tamanho fixado.
Checksum	2	calculado	*Checksum* calculado

Tabela 1 – Composição do pacote do protocolo de comando do cliente

Os dois protocolos são *streams* de bytes e são constituídos de 5 elementos: um header de dois bytes, um byte de contagem de bytes seguintes, o comando do cliente ou o tipo de SIP, o tipo de comando do cliente e os argumentos ou o SIP e, por fim, um checksum de dois bytes.

Os dois bytes do header são iguais tanto para comandos do cliente e informações do servidor: 250 (0xFA) seguido de 251 (0xFB).

O byte de contagem de bytes seguintes conta todos os bytes, inclusive o checksum, subtraindo dele apenas os dois bytes do header e ele mesmo.

Comando	#	Tipo de argumento	Descrição
Antes da conexão do cliente			
SYNC0	0	-	Estes comandos iniciam a conexão. Devem ser enviados na ordem.
SYNC1	1		
SYNC2	2		
Depois de estabelecer a conexão			
PULSE	0	-	Reseta o *watchdog* do servidor.
OPEN	1	-	Inicia o servidor.
CLOSE	2	-	Fecha a conexão entre o cliente e o servidor.
POLLING	3	string	Muda a sequência de atualização dos sonares.
ENABLE	4	int	Liga e desliga os motores. 1 = habilitado; 2 = desabilitado
SETA	5	int	Define a aceleração de translação, se positivo, ou desaceleração, se negativo; em mm/s2
SETV	6	int	Define a máxima velocidade de translação; em mm/s.

SETO	7	-	Reseta a posição local para a origem 0,0,0.
MOVE	8	int	Translada uma distância em mm para (+) frente ou para (-) trás na velocidade definida em SETV
ROTATE	9	int	Rotaciona no sentido (-) horário ou (+) anti-horário em graus/s na velocidade limitada por SETRV
SETRV	10	int	Define a máxima velocidade de rotação; graus/s.
VEL	11	int	Translada para (+) frente ou para (-) trás na velocidade definida (limitada por SETV)
HEAD	12	int	Rotaciona para a orientação absoluta na velocidade definida por SETRV; ± graus (+ = anti-horário)
DHEAD	13	int	Rotaciona para a orientação relativa a atual.
CONFIG	18	-	Requisita um SIP.
ENCODER	19	int	Requisita um *stream* contínuo ou para os SIPs dos *encoders*.
SETRA	23	int	Define a aceleração de rotação, em graus/s^2.
SONAR	28	int	1 = habilita sonar, 0 = desabilita todos os sonares; ou utilizar os bits 1-3 para especificar um *array* de sonares
STOP	29	-	Para o robô; motor mantém habilitado
VEL2	32	2 bytes	Define velocidades independentes para cada roda; bits de 0 a 7 para a roda da direita, bits de 8 a 15 para a roda da esquerda; em incrementos de 20 mm/s.
SONARCYCLE	48	uint	Muda o ciclo de atualização dos sonares; em milímetros.
HOSTBAUD	50	int	Muda o *baud rate* da serial para 0 = 9600, 1 = 19200, 2 = 38400, 3 = 57600 ou 4 = 115200.
ROTKP	82	int	Modifica o valor proporcional do PID de rotação.
ROTKV	83	int	Modifica o valor derivativo do PID de rotação.
ROTKI	84	int	Modifica o valor integrativo do PID de rotação.
TRANSKP	85	int	Modifica o valor proporcional do PID de translação.
TRANSKV	86	int	Modifica o valor derivativo do PID de translação.
TRANSKI	87	int	Modifica o valor integrativo do PID de translação.
REVCOUNT	88	int	Modifica o contador diferencial do *encoder*.

BATTEST	250	int	Define uma tensão de alimentação da bateria artificialmente; argumento em múltiplos de 10 volts (100 = 10V); 0 para reverter para a tensão real.
RESET	253	-	Força um reset do microcontrolador.
MAINTENANCE	255	-	Entra no modo de manutenção do microcontrolador.

Tabela 2 - Lista de comandos do cliente

3.5.1.4.2 Erros no pacote

Pacotes com contador de byte que excede 204 (pacote com 207 bytes no total) e com erros no *checksum* são ignorados pelo ARCOS. Semelhantemente, o cliente deve ignorar *server information packets* (SIP) com erros (MOBILEROBOTS, 2007).

Devido a natureza da interação cliente-servidor de robôs móveis em tempo real, não é necessário uma interface de comunicação de pacotes com suporte a confirmação de recebimento de pacote (*acknowledgment*), já que a retransmissão de uma informação do servidor ou um pacote de comandos normalmente não tem utilidade devido ao fato de que dados antigos não serem úteis para manter a rápida reação do robô (MOBILEROBOTS, 2007).

Contudo, a interface cliente servidor oferece um meio simples para lidar com pacotes de comandos ignorados: como a maioria dos comandos do cliente alteram o estado das variáveis no servidor, ao examinar esses valores nas respectivas informações que o servidor envia ao cliente, o programa cliente pode detectar os comandos ignorados e reenviá-los até que se consiga atingir o estado correto (MOBILEROBOTS, 2007).

3.5.1.4.3 Conexão cliente-servidor

Antes de exercer qualquer controle sobre o robô, é necessário estabelecer uma comunicação entre o cliente e o robô.

Quando o ARCOS é ligado ou reiniciado, o robô fica em um estado de espera, escutando para ver se chega alguma informação através da porta serial. Para conectar-se, o cliente deve enviar três pacotes de sincronização em sequência (“SYNC0”, “SYNC1” e ”SYNC2”) e obter a resposta do servidor (ARCOS).

Quando em modo de espera, o servidor manda uma resposta após cada pacote de sincronização. O cliente deve esperar cada resposta antes de enviar o próximo pacote de sincronização.

Uma vez estabelecida a conexão com o sistema ARCOS, o cliente deve enviar o comando "*OPEN"* ao servidor, que faz com que o microcontrolador execute algumas função de preparação, inicia os seus vários servidores, como os sonares e motores, e começa a transmitir informações do servidor (SIP) ao cliente.

3.5.1.4.4 Pacotes de informação do servidor (SIP)

Uma vez enviado o comando "*OPEN"*, o servidor ARCOS automaticamente e repetidamente envia pacotes de informação através da serial RS-232 para o cliente conectado. O pacote padrão informa o cliente sobre o número de estados de operação e leituras, utilizando uma ordem e tipo de dado pré-definido (MOBILEROBOTS, 2007).

Informação	Dado	Descrição
Header	2 bytes	Na ordem: 0xFA (250), 0xFB (251)
BYTE COUNT	byte	Número de bytes + 2 (checksum), não inclui *header* e *byte* de contagem
TYPE	0x3s	Estado do motor; s = 2 quando motores estão parados ou 3 quando o robô está movimentando-se
XPOS	int	Coordenadas dadas pelo *encoder* e do giroscópio opcional, em milímetros
YPOS		
THPOS	int	Orientação em unidades de ângulo (0,001534 radianos por unidade de ângulo)
L VEL	int	Velocidade das rodas em milímetros por segundo
R VEL		
BATTERY	byte	Tensão da bateria em volts, em múltiplos de 10 (101 = 10,1 volts)
STALL AND BUMPERS	uint	Travamento do motor e indicador dos *bumpers*. Bit 0 indica o travamento do motor esquerdo, se estiver travado é 1. Bits 1 a 7 correspondem ao primeiro *bumper* de entrada digital (acessório). Bit 8 corresponde ao motor direito e bits 9 a 15 correspondem ao segundo *bumper*.
CONTROL	int	Posição angular do servo em graus.
FLAGS	uint	Bit 0 indica o estado do motor; bit 1 ao 4 indicam estado dos sonares; bits 5 e 6 indicam se o motor está parado; bits 7 e 8 indicam o estado dos sensores infravermelho de borda; bit 9 indica o estado do botão de *fire* do joystick; bit 10 indica se o modo de auto-carregamento está acionado.
COMPASS	byte	Valor da do acessório bússola eletrônica em 2

		unidades de grau.
SONAR COUNT	byte	Número de novas leituras de sonares incluídas.
NUMBER	byte	Se o contador de sonar > 1, é o número de identificação do sonar.
RANGE	uint	Leitura do sonar em milímetros.
... Resto das leituras dos sonares ...		
GRIP_STATE	byte	Estado da garra manipuladora (acessório).
ANPORT	byte	Número da porta analógica selecionada (de 1 a 5).
ANALOG	byte	Leitura da entrada analógica (0 a 255 = 0 a 5 V) na porta selecionada.
DIGIN	byte	Entrada digital codificada em bytes.
DIGIOUT	byte	Saída digital codificada em bytes.
BATTERYX10	int	Tensão da bateria real em unidades de 0,1 V.
CHARGESTATE	byte	Estado do sistema de carregamento automático da bateria; Se igual a -1, desconhecido; se 0, não está carregando; se 1, carregado; se 2, sobrecarregado.

Tabela 3 - Composição do pacote de informação do servidor (SIP)

3.6 “iPhone” e “iPod Touch”

O “iPhone” é uma linha de *smartphones* com capacidade de acesso à internet e multimídia, projetado pela Apple Inc. O primeiro “iPhone” foi introduzido no dia 9 de janeiro de 2007. Já o “iPod Touch” é uma linha de dispositivos multimídia com as mesmas capacidades do “iPhone”, com a única diferença de não possuir a função de telefone e GPS.

Um “iPhone” pode ser operado como uma câmera fotográfica digital, e um player multimídia portátil, além de poder ser usado como um cliente de internet, com *e-mail*, *web browsing,* e conectividade Wi-Fi. A interface de usuário é adaptada à tela multi-toques, inclusive o teclado virtual, no lugar de um teclado físico. Em meados de 2008, foi disponibilizado uma *framework* para o desenvolvimento de aplicativos, habilitando oficialmente a capacidade do dispositivo executar *software* de empresas não relacionadas à Apple Inc. Estes aplicativos têm funções diversas, como jogos, navegação GPS e aplicativos para consulta de dados.

3.6.1 Sensores

Duas importantes inovações foram introduzidas no mundo do *smartphone* com o “iPhone”:

- Tela com suporte a múltiplos toques;
- Acelerômetro

A tela com suporte a múltiplos toques e o acelerômetro são os principais sensores responsáveis pela interação com o dispositivo.

Nos dois próximos capítulos serão discutidos aspectos importantes sobre estes sensores.

3.6.1.1 Acelerômetro

O acelerômetro presente nos modelos de “iPhone” e “iPod Touch” possuem três eixos:

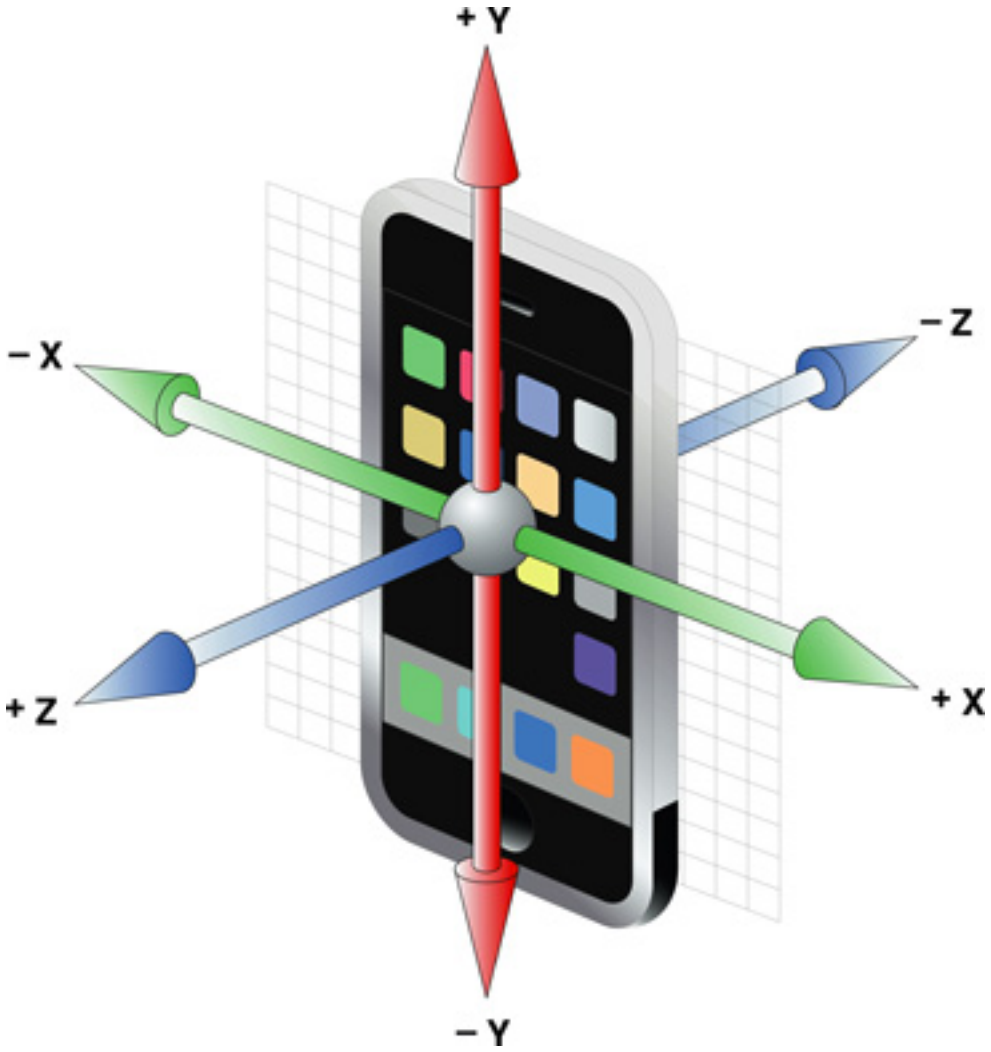

Figura 3 - Eixos do acelerômetro do “iPhone” (APPLE, 2010)

O acelerômetro do dispositivo é utilizado para diversas aplicações, como identificação da orientação e vibração.

Assim como a maioria dos acelerômetros presentes no mercado, o princípio de funcionamento do acelerômetro baseia-se na segunda lei de Newton. O Acelerômetro, baseado no movimento do “iPhone” ou “iPod Touch”, mede os deslocamentos gerados por um elemento em relação a outro, podendo assim, estimar a aceleração do objeto.

O acelerômetro do dispositivo mede as aceleração em termos de aceleração de gravidade.

3.6.1.2 Tela multi-toques

A tela de toques do “iPhone” é de cristal líquido com vidro resistente a riscos. A tela de toques utilizada é do tipo capacitiva e foi projetado para reconhecer toques com contato direto com a pele e múltiplos toques. Nos primeiros modelos do “iPhone”, a resolução da tela era de 320 x 480, enquanto nos últimos modelos (“iPhone” 4), a resolução chega a 640 x 960.

A tela sensível a toques do “iPhone” é uma tela que inclui uma camada de material capacitivo como as outras telas sensível a toques capacitivo. Porém, o arranjo dos capacitores da tela do “iPhone” é que o diferencia das outras telas. Os capacitores são dispostos de acordo com a coordenada do sistema, fazendo com que o circuito possa detectar mudanças em cada ponto ao longo da grade de capacitores, ou seja, cada ponto na grade de capacitores gera sinal independentemente quando tocado e envia o sinal para o processador.

A tela do “iPhone” pode detectar o toque através de dois meios: capacitância mútua ou auto-capacitância. A capacitância mútua utiliza-se de um circuito capacitor que requer duas camadas distintas do material. Uma camada possui as linhas de condução, onde passa a corrente, e a outra, as linhas sensoras, os quais são responsáveis por detectar a corrente nos nós. Já a auto-capacitância utiliza uma camada de eletrodos individuais que são conectados com um sensor sensível a capacitância.

3.6.2 Diferenças no desenvolvimento de software para "iPhone"

Existem diversos aspectos o que torna o desenvolvimento de um *software* para o "iPhone" ou "iPod Touch" diferente do desenvolvimento de *software* para um PC (MARK e LAMARCHE, 2009):

- Apenas um aplicativo é executado por vez. Com exceção do sistema operacional, apenas um aplicativo pode ser executado por vez no "iPhone" e "iPod Touch". Enquanto um aplicativo está sendo executado, não há como outros aplicativos executarem alguma ação. Apesar de a última geração do "iPhone" possuir habilidade de executar mais de um aplicativo por vez, esta característica ainda é predominante;
- Os aplicativos possuem apenas uma janela por vez. Ao contrário dos sistemas operacionais para PC, onde muitos aplicativos são executados ao mesmo tempo, cada um com habilidade de criar e controlar várias janelas, os aplicativos do "iPhone" e "iPod Touch" possuem apenas uma janela, do tamanho da tela do dispositivo.
- Acesso limitado. Enquanto programas de um computador podem ter acesso a todos os arquivos do usuário que está executando o aplicativo, o acesso aos arquivos no "iPhone" é limitado. Só é possível ler e gravar no sistema de arquivos, os arquivos criados pelo aplicativo. Esta área de acesso do aplicativo é chamado de "sandbox do aplicativo", e é onde o aplicativo pode armazenar documentos, arquivos de preferências e todos os outros tipos de dados necessários. Além disso, o programa não têm acesso a algumas portas de conexão de rede ou qualquer coisa que necessite de privilégios de administrador.
- Tempo de resposta limitado. Quando um aplicativo é executado, este deve abrir, preparar e apresentar na tela os dados o mais rápido possível. Se algum processo do "iPhone" e "iPod Touch" levar mais de 5 segundos para ser executado, poderá ser morto pelo sistema sem nenhum aviso.
- Tamanho de tela limitada. Apesar da tela do "iPhone" ser maior que a maioria dos smartphones do mercado, comparado com as telas de PC

é extremamente limitado, sendo necessário a adaptação para pequenas resoluções.

- Recursos de sistema limitado. Comparativamente com um computador pessoal, os recursos de sistema do “iPhone” e “iPod Touch” são bem limitados. É necessário administrar efetivamente a memória *ram* do dispositivo e, geralmente, apenas metade da memória *ram* do dispositivo está livre para uso de um *software* devido aos outros processos do sistema.

4 PROJETO DO SOFTWARE

4.1 Estrutura do problema

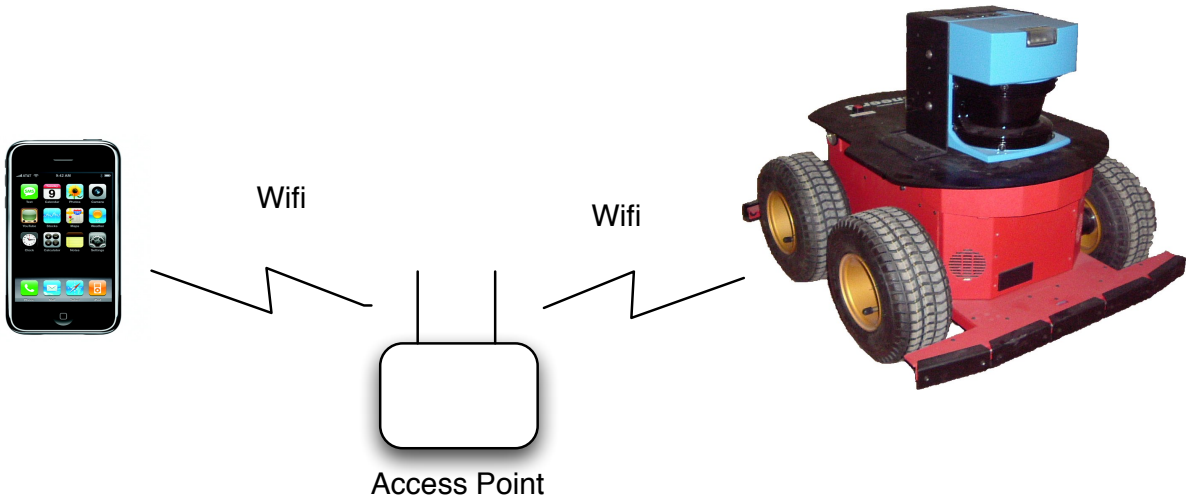

Figura 4 - estrutura do problema

A arquitetura do sistema do robô móvel permite a comunicação com o servidor ARCOS apenas pela porta serial RS-232 para que seja possível controlar o robô remotamente através de uma rede de computadores wireless, é necessário que haja um modo de conectar-se através do protocolo TCP/IP no micro-controlador com o servidor ARCOS. Há dois meios possíveis: através de um programa rodando em um computador embarcado com suporte a redes Wi-Fi ou através de um hardware extra. Caso utilize um hardware extra, não é necessário nenhuma configuração adicional, contudo, caso seja escolhido utilizar um computador embarcado, é

necessário um programa para direcionar os dados que chegam através do protocolo TCP/IP para a porta serial do servidor ARCOS.

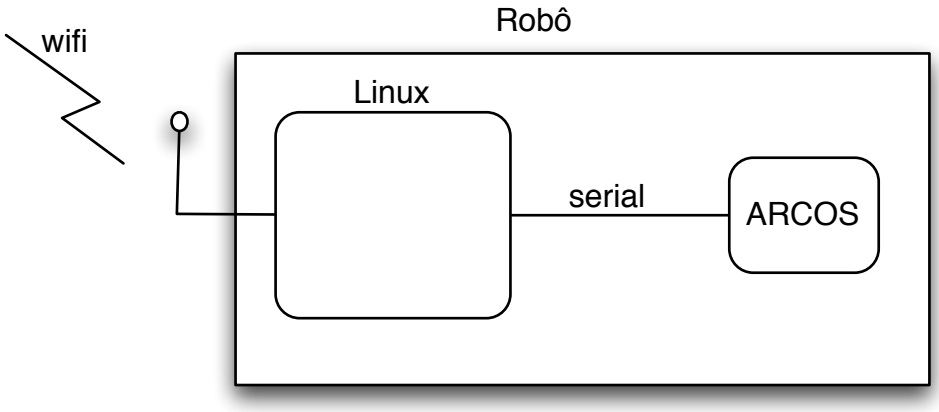

Figura 5 - Estrutura do robô utilizada

O robô móvel utilizado neste trabalho possui um computador embarcado extra com o sistema Linux, portanto, não é necessário utlizar um dispositivo extra. Foi necessário apenas utilizar um software para redirecionar os pacotes vindos de uma rede wireless através do protocolo TCP/IP para a porta serial RS-232 conectada co micro-controlador com o servidor ARCOS. O software utilizado foi o *ser2net*.

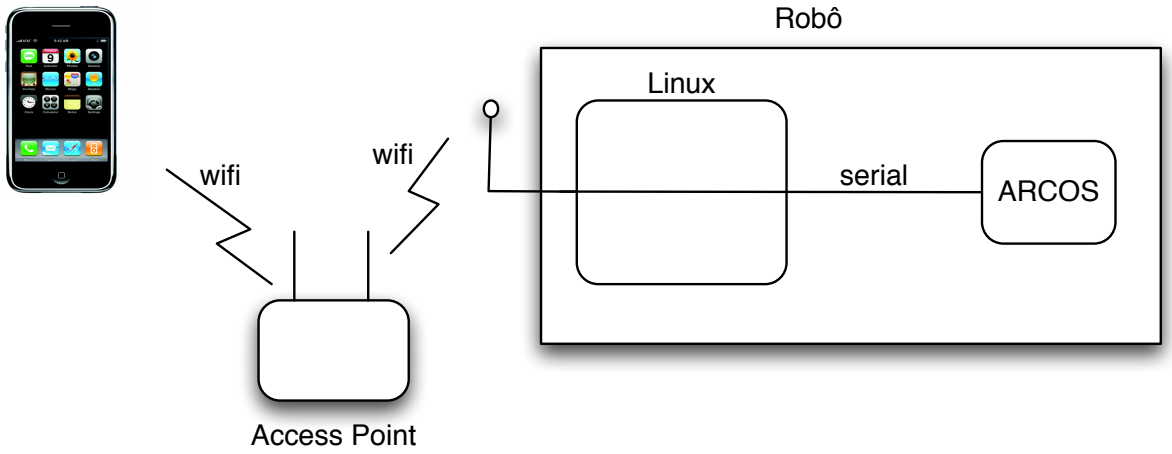

Figura 6 - Estado desejado da estrutura do problema

A figura acima ilustra o estado desejado do problema: essencialmente, o software faz o papel de um *bypass* de dados da rede Wi-Fi para a porta serial RS-232 para que seja possível comunicar-se com o ARCOS através do “iPhone” ou “iPod Touch”.

4.2 Análise de requisitos

Para uma melhor análise de requisitos, dividimos os requisitos em quatro partes: requisitos do sistema, requisitos do computador embarcado, requisitos do *software* para “iPhone” ou “iPod Touch” e requisitos de interface.

4.2.1 Requisitos do sistema

Os requisitos do sistema diz respeito aos requisitos que devem ser satisfeitos pelo sistema de controle proposto. O sistema de controle deve satisfazer as seguintes funcionalidades:

- Conectar remotamente ao robô móvel através de uma rede Wi-Fi;
- Definir a velocidade máxima permitida do robô;
- Ligar e desligar motores;
- Ler dados de sensores do robô, como sonares;
- Controlar a velocidade de cada roda.

Estas funcionalidades básicas devem ser garantidas pelo sistema para que seja possível o controle remoto do robô móvel.

4.2.2 Requisitos do computador embarcado

Estes requisitos referem-se ao computador embarcado que está conectado ao micro-controlador com o servidor ARCOS através da porta serial RS-232. Os requisitos do computador embarcado são:

- Possuir suporte a redes wireless Wi-Fi;
- Possuir algum sistema operacional do tipo Unix;
- Deve executar um programa para o direcionamento dos dados do protocolo TCP/IP, vindos de uma conexão wireless Wi-Fi, para a porta serial RS-232, ou seja, deve fazer um *bypass* entre a conexão wireless Wi-Fi e a porta serial RS-232.

4.2.3 Requisitos do software para "iPhone" ou "iPod Touch"

São os requisitos de software do aplicativo que será executado no dispositivo "iPhone"/"iPod Touch". Os requisitos são:

- Capacidade de conectar-se a um computador remoto através da rede wireless Wi-Fi;
- Deve ser capaz de criar pacotes de comandos para o servidor ARCOS e enviá-los através da rede wireless;
- Deve ser capaz de decodificar e interpretar os pacotes de informação (SIP) do servidor ARCOS;
- Possuir a habilidade de interpretar gestos na tela multi-toques e gestos com a utilização do acelerômetro.

4.2.4 Requisitos de interface

Os requisitos de interface referem-se aos requisitos necessários da interface do *software* executado no "iPhone" ou "iPod Touch", através da interface que o usuário irá interagir com o sistema de controle remoto do robô móvel. Os requisitos de interface são:

- Visualização de dados de sensores, como o nível de bateria;
- Edição de campo referentes a configuração inicial do robô (IP, porta e velocidade máxima);
- Área de interação para gestos multi-toques;
- Visualização de velocidade do robô móvel;

4.2.5 Diagrama de casos de uso do sistema de controle

Para que seja possível formalizar os requisitos do sistema de controle e suas funcionalidades, é necessário o uso do diagrama de casos de uso.

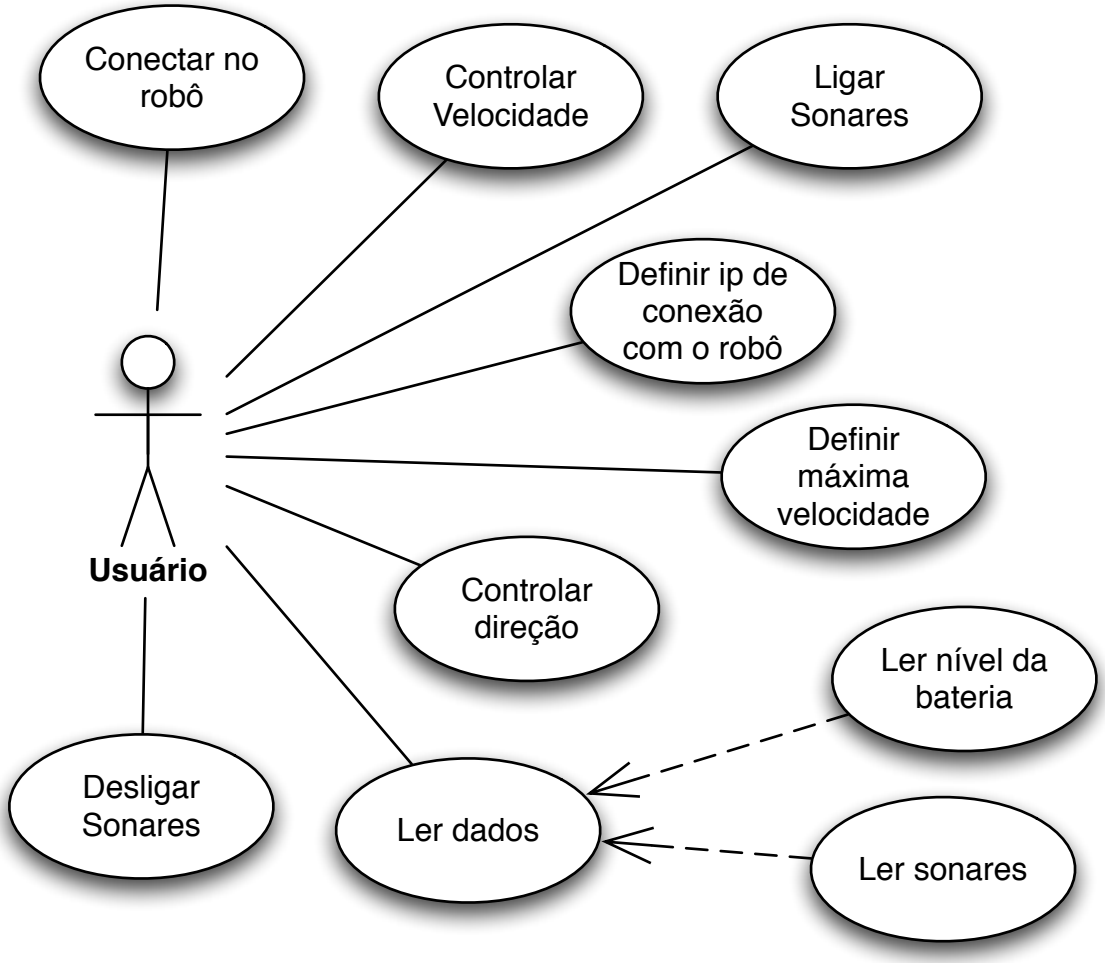

Figura 7 - Diagrama de casos de uso

O usuário, através do programa de “iPhone” e “iPod Touch”, deve conseguir, primeiramente, definir o endereço de IP (*Internet Protocol*) do robô móvel no qual quer-se conectar. Após definido o IP, deve-se definir a velocidade máxima permitida do robô e então, conectar-se ao robô móvel. Ao conectar-se ao robô, o usuário deve ser capaz de, através de uma interface gráfica básica, controlar a velocidade e a direção no qual o robô irá mover-se. Durante o controle do robô móvel, o usuário pode ligar ou desligar os sonares e ler alguns dados relevantes sobre o robô, como a leitura do nível de energia da bateria e dados sobre os sonares.

4.3 Modelagem segundo a arquitetura MVC

Devido ao uso do Objective-C como linguagem de programação e do framework Cocoa, é necessário que o software seja implementado na arquitetura MVC (*Model-view-controller*).

A separação da lógica de negócios e da lógica de apresentação foi feita da seguinte forma:

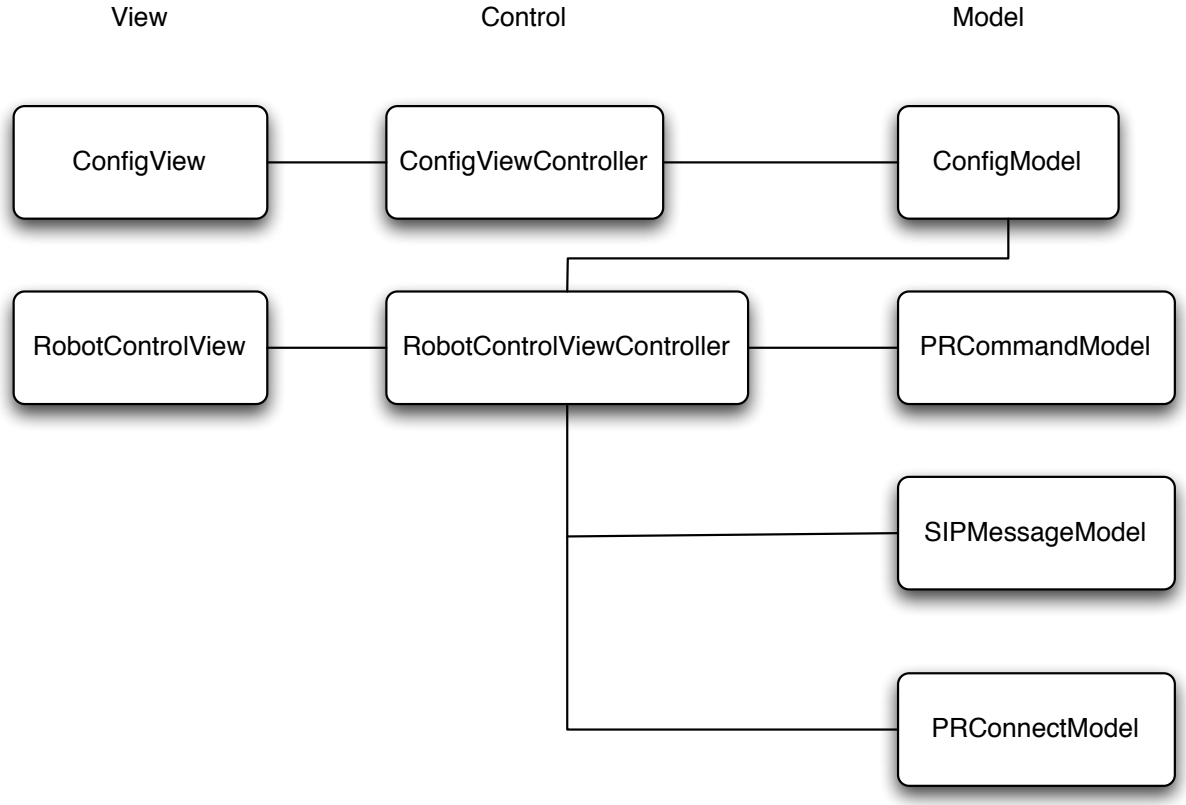

Figura 8 - Modelagem em MVC

Foi utilizada a seguinte convenção no diagrama acima: a última palavra iniciada em caixa alta especifica o tipo do `objeto` e as linhas indicam a relação entre os `objetos`.

Cada elemento do diagrama de modelagem em MVC representa um `objeto`. O primeiro `objeto`, ConfigView, é uma visão para apresentar os dados de configuração a serem inseridos no *software*. Entre os dados, estão o IP (*Internet Protocol*) e a porta a conectar e a velocidade máxima permitida do robô móvel Pioneer 3-AT. O controlador da ConfigView (ConfigViewController) faz a intermediação entre a visão e o modelo, alocando os dados de entrada da ConfigView no ConfigModel, o qual guardará estes dados para o acesso posterior pelo RobotControlViewController, além de ter como função trocar a visão ativa do *software* para RobotControlView.

RobotControlView é a visão no qual será apresentado a interface para controlar os movimentos do robô, além de ler os dados de nível de bateria e dados dos sonares. RobotControlViewController é o controlador responsável pela lógica de

controle do robô móvel. Este controlador que fará com que o software, primeiramente, estabeleça conexão com o robô através do objeto PRConnect em nível de protocolo de rede TCP/IP, e depois, a nível de protocolo do sistema ARCOS, possibilitando a leitura dos dados recebidos através do ConfigModel e o envio de dados de comando. Além disso, o controlador é responsável em processar os dados de pacote do servidor (SIP – *Server information packet*) e apresentar estes dados na visão RobotControllerView, processar as entradas através do acelerômetro e da tela multi-toques interpretando-os e requisitando os pacotes de dados de comando ao PRCommandModel. PRCommandModel manipula e monta os pacotes de dados que serão enviados para controlar o robô móvel. SIPMessageModel é responsável por armazenar os dados recebidos do servidor ARCOS, traduzindo o pacote de dados recebidos em uma forma com o qual RobotControlViewController possa interpretar as informações de estado do robô.

4.4 Conexão entre “iPhone” ou “iPod Touch” com o robô móvel

Como o objetivo do presente trabalho é o controle remoto do robô móvel através de uma rede de computadores wireless Wi-Fi, é necessário implementar métodos para estabelecer uma conexão sem fio.

No lado do “iPhone”/”iPod Touch”, PRConnect é o `objeto` responsável por transmitir e receber dados através de um *stream socket*, o qual utiliza o protocolo TCP e um conjunto de *endpoints* (compostos pelo IP e pela porta). Este objeto possui quatro funções básicas: conectar-se ao PC embarcado do robô móvel, dado o IP e a porta, enviar e receber dados e, desconectar-se do PC do Pioneer 3-AT.

No lado do robô, ao estabelecer uma conexão TCP/IP no PC embarcado, caso a porta de conexão seja a porta associada à serial RS-232 (função provida pelo software de bypass de dados da rede wireless para a serial), todos os dados recebidos pela rede Wi-Fi são enviados para a porta serial e, consequentemente, para o servidor ARCOS, assim como todos os dados transmitidos do servidor ARCOS para o PC embarcado é enviado ao “iPhone”/”iPod Touch” através da rede Wi-Fi.

Para que o dado faça todo o percurso através dos diferentes meios de comunicação, ocorre o fenômeno do encapsulamento de dados. Ao ser transmitido

pelo “iPhone” na rede Wi-Fi, o dado é adicionado a um novo pacote de dados, ou seja, ocorre o encapsulamento do dado ao descer uma camada de rede (representado por um protocolo). Este novo pacote, por sua vez, ao descer outra camada da rede também é encapsulado, formando um novo pacote, e assim sucessivamente até a última camada. No PC embarcado, ao receber o pacote de informação, ocorre o caminho inverso, ou seja, o pacote é desencapsulado até alcançar o protocolo mais alto no nível de camadas, tendo acesso ao dado enviado pelo “iPhone”. Como o dado é retransmitido para a porta serial, este é encapsulado novamente, no entanto, pelo protocolo da serial RS-232. O servidor ARCOS, por sua vez, recebe o pacote encapsulado pelo protocolo físico RS-232 e desencapsula, podendo utilizar o dado enviado pelo “iPhone”. No caminho inverso, ocorre o mesmo fenômeno.

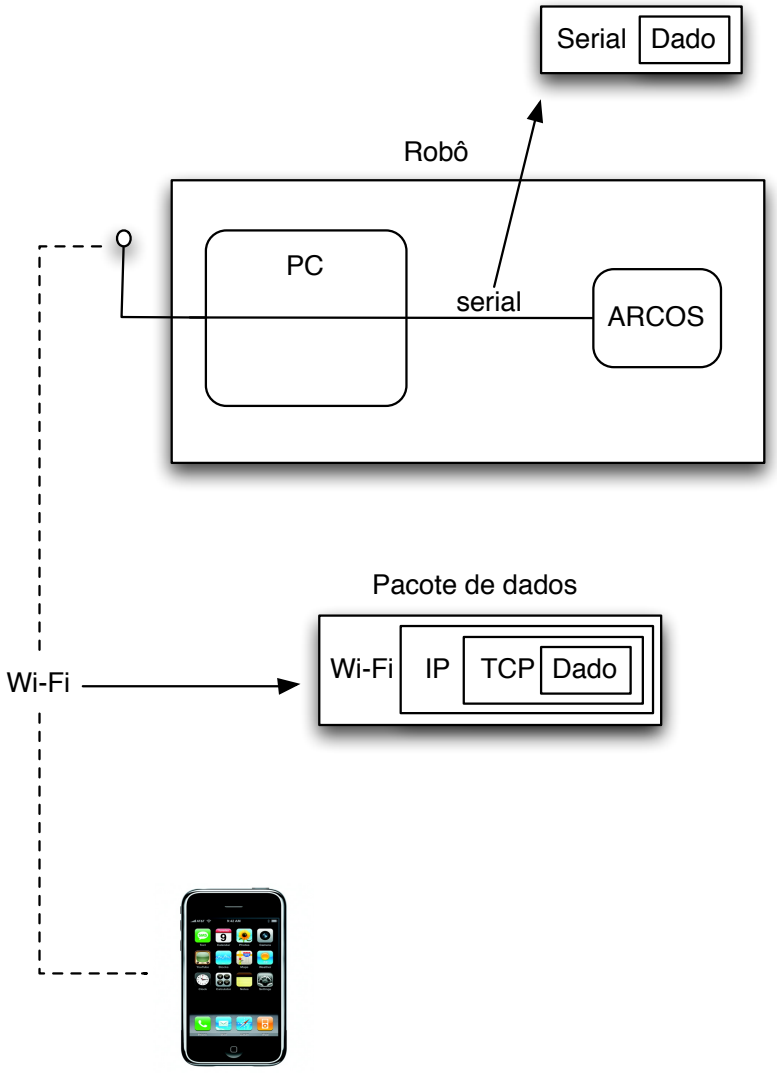

Figura 9 - Encapsulamento de dados ao longo da transmissão

4.5 Comandos de controle

O `objeto` PRCommandModel da arquitetura MVC (*Model-View-Control*) é o modelo com capacidade de gerar os pacotes de dados a serem enviados ao robô móvel, para serem interpretados pelo servidor ARCOS. A implementação abrange os seguintes comandos da tabela de comandos:

- SYNC0;
- SYNC1;
- SYNC2;
- OPEN;
- Ligar e desligar motor;
- Definir a velocidade máxima;
- Definir velocidade de cada motor;
- Para o robô;
- Ligar e desligar sonares;
- Desconectar.

A geração dos pacotes de comando é iniciado com a configuração dos dois bytes do *Header*. Em seguida, é selecionado o byte de comando do cliente e, a partir disto, é selecionado o byte correspondente ao tipo de argumento do comando e os dados do argumento. É calculado quantos bytes o pacote possui, sem contar o *Header* e o byte de contagem; é calculado o *CheckSum* e, finalmente, concatena-se os bytes na ordem estabelecida pelo protocolo do servidor ARCOS.

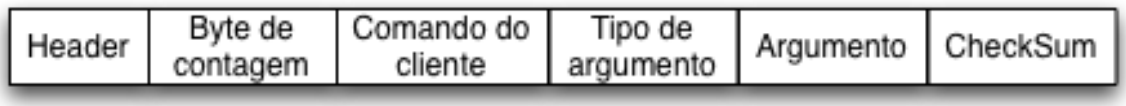

Figura 10 - Pacote de dados no protocolo do servidor ARCOS

No diagrama abaixo, temos um fluxograma representando o algoritmo de geração de pacotes e um exemplo de geração de pacote:

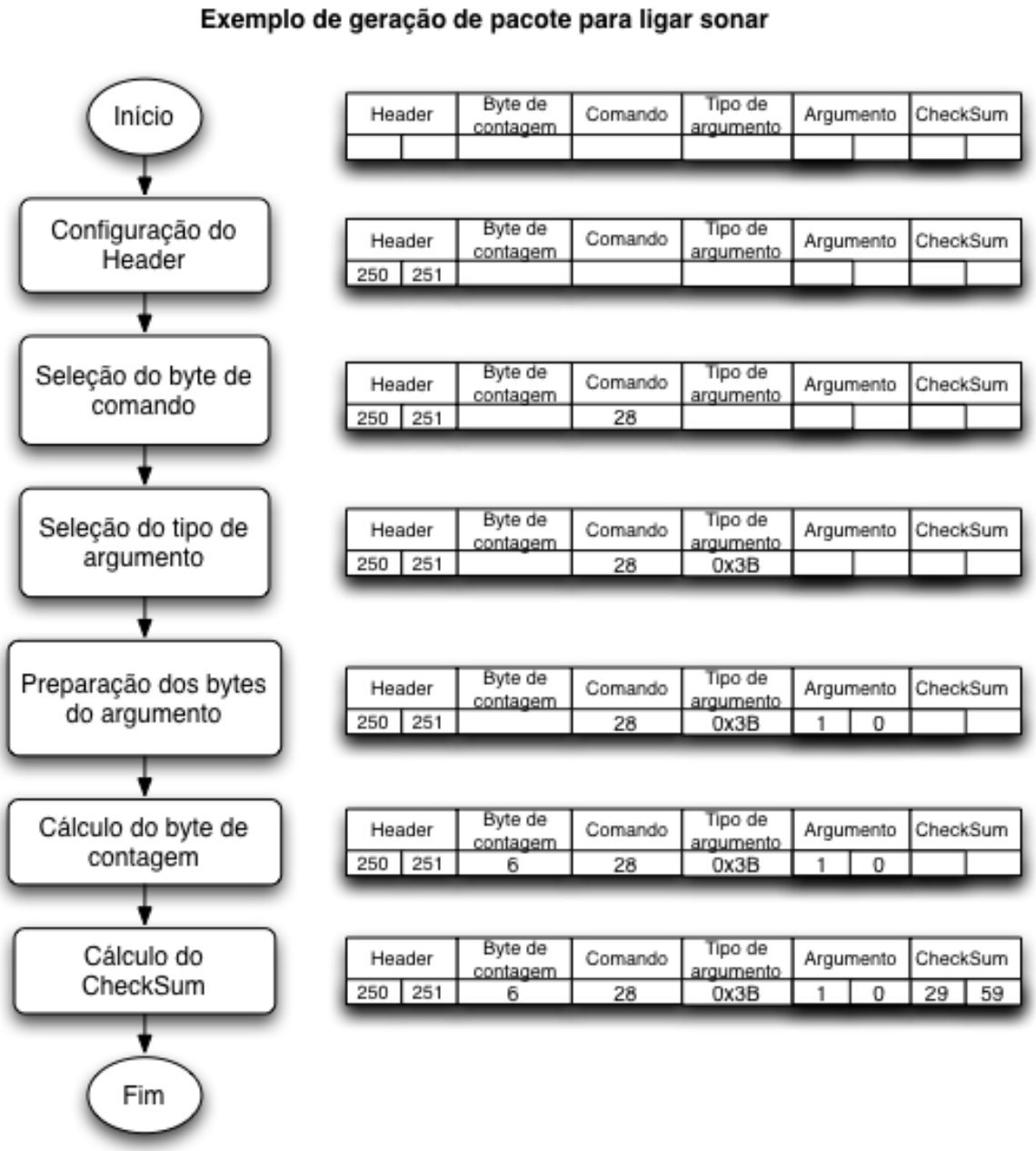

Figura 11 - Fluxograma de geração de pacotes e exemplo de geração de pacote para ligar sonar

4.6 Leitura dos SIP

Para controlar o robô móvel, é necessário a leitura de alguns dados básicos de sensores, como o nível de tensão da bateria. A leitura destes dados é realizado lendo-se os pacotes de informação do servidor (SIP).

A implementação da conexão TCP/IP armazena dados em um buffer e só permite a leitura dos dados recebidos no esvaziamento do buffer (quanto este fica cheio). Por isto, um pacote de dados no protocolo do servidor pode estar divido em um ou mais pacotes do protocolo TCP/IP, portanto, é necessário implementar uma máquina de estados para verificar qual parte do pacote de informação do servidor (SIP) foi lido.

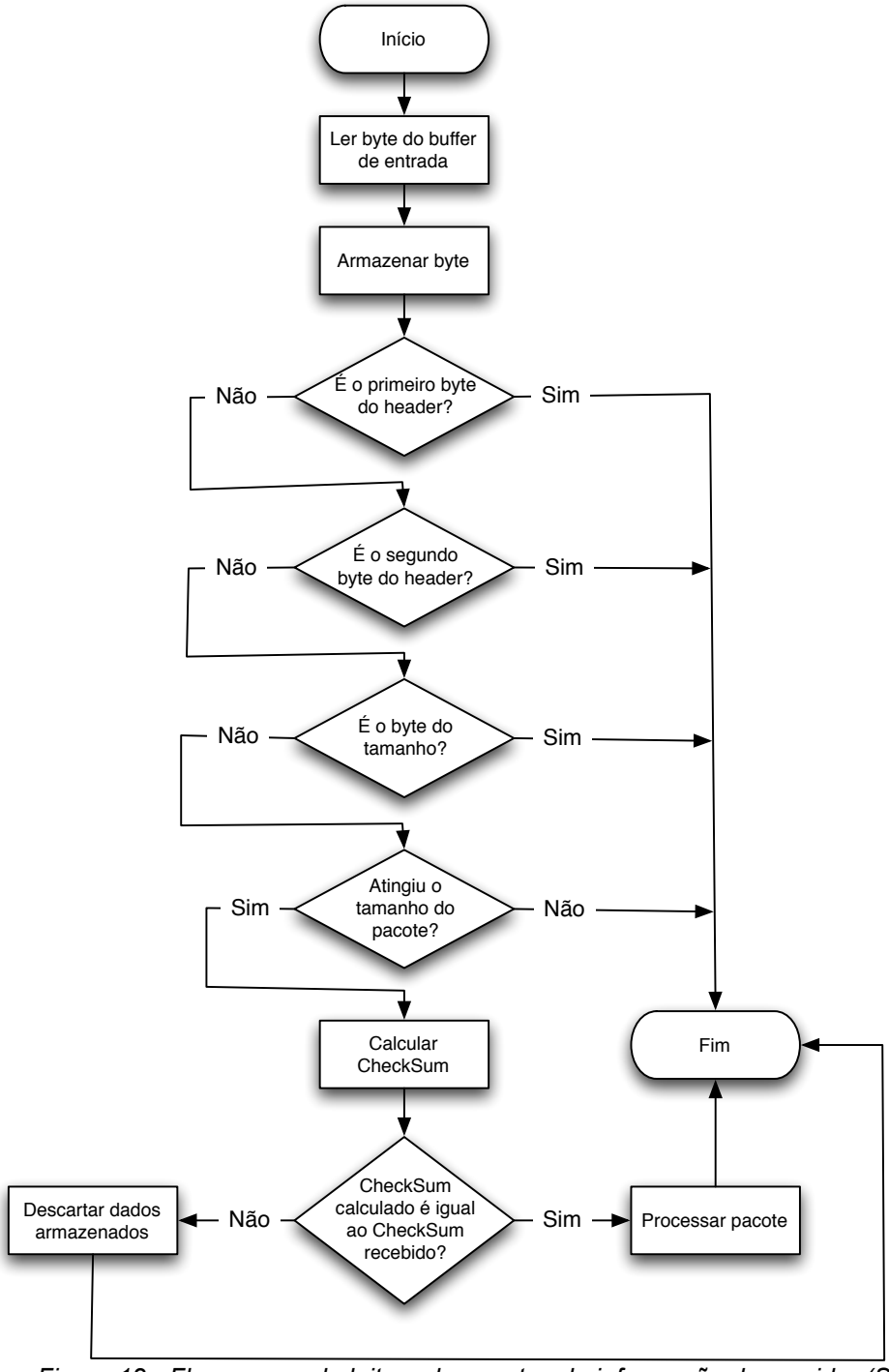

Figura 12 - Fluxograma de leitura de pacotes de informação do servidor (SIP)

Primeiramente, armazena-se o byte lido do buffer. Após armazenar, verifica-se se o byte faz parte do Header, caso positivo, o ciclo é reiniciado, caso contrário, verifica-se se o byte é o byte de tamanho. Se for o byte do tamanho, faz-se a leitura do byte do buffer sucessivamente até atingir o tamanho do pacote lido (byte de tamanho). Ao atingir o tamanho do pacote, calcula-se o CheckSum e compara-se com o CheckSum recebido pelo pacote. Caso o CheckSum calculado e o recebido no pacote sejam iguais, o pacote não possui erro e, portanto, é processado, atualizando os dados de sensores. Caso contrário, como o pacote possui erro, é descartado.

4.7 Controle dos movimentos do robô

O robô móvel Pioneer 3-AT não possui um sistema de direção como o de um automóvel. Ao invés de utilizar rodas que mudam de direção, é utilizado um sistema a diferença de velocidades em cada roda determinar a direção de movimento.

O presente trabalho tem como um dos objetivos utilizar os acelerômetros do “iPhone” ou “iPod Touch” para simular o esterçamento de um volante de carro. Devido a diferença de sistemas de direção, é necessário, primeiramente, modelar o sistema de direção de um automóvel. Ou seja, para que o robô simule o esterçamento, devemos definir as velocidades nas rodas em cada lado.

Na figura abaixo, temos o modelo utilizado na modelagem:

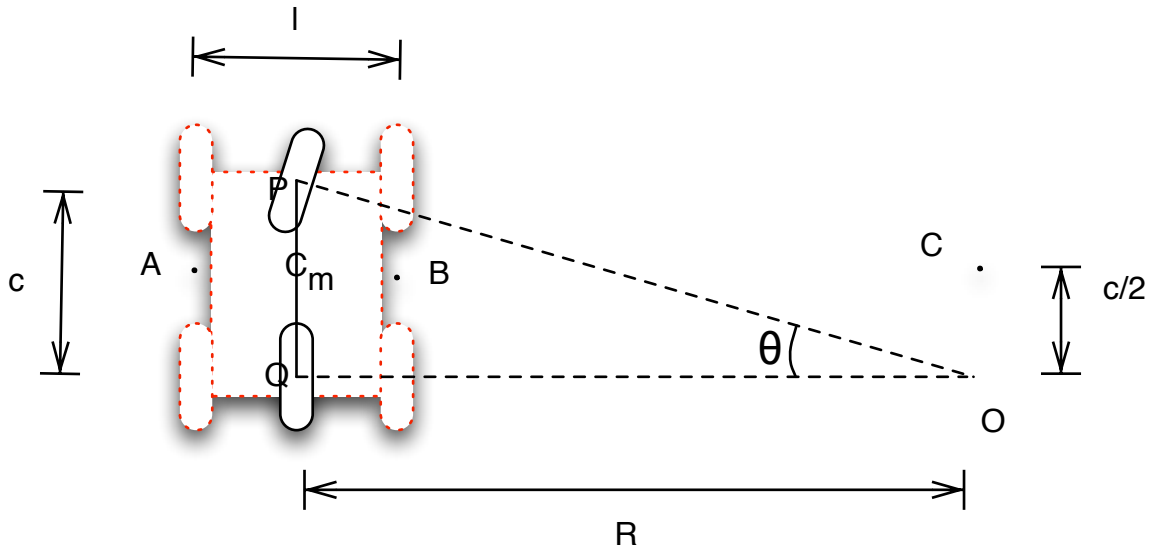

Figura 13 - modelagem de curvas

As linhas pontilhadas vermelhas representam o robô e as linhas pretas inteiras, o modelo utilizado para simular o esterçamento do volante. Foram feitas as seguintes simplificações no modelo: o modelo possui apenas duas rodas, a distribuição de massa é constante, o centro da curva (C) não muda de posição, ou seja, não há deslocamento do centro de curva.

Dado a velocidade do centro de massa (V_{Cm}) e o ângulo de esterçamento θ, Queremos a velocidade nos pontos A e B (V_A e V_B), localizados na mesma reta que liga C_m e C.

Do triângulo PQO, temos:

$$\tan\theta = \frac{c}{R} \Leftrightarrow R = \frac{c}{\tan\theta} \qquad (1)$$

Da aceleração centrípeta, temos que:

$$a_c = \frac{V_{Cm}^{\ 2}}{R} = \frac{V_A^{\ 2}}{R + \frac{l}{2}} = \frac{V_B^{\ 2}}{R - \frac{l}{2}} \qquad (2)$$

Substituindo (1) em (2) e, colocando V_A e V_B em função de V:

$$V_B = \pm V_{Cm}\sqrt{1 - \frac{l.\tan\theta}{2c}} \qquad (3)$$

$$V_A = \pm V_{Cm}\sqrt{1 + \frac{l.\tan\theta}{2c}} \qquad (4)$$

Como estamos interessados apenas nas velocidades no mesmo sentido de V_{Cm}:

$$V_B = +V_{Cm}\sqrt{1 - \frac{l.\tan\theta}{2c}} \qquad (5)$$

$$V_A = +V_{Cm}\sqrt{1 + \frac{l.\tan\theta}{2c}} \qquad (6)$$

Da equação (5), percebemos que temos uma limitação quanto a θ:

$$1 - \frac{l.\tan\theta}{2c} \geq 0 \Leftrightarrow \theta \leq \arctan\left(\frac{2c}{l}\right) \qquad (7)$$

Disto:

$$0 \leq \theta \leq \arctan\left(\frac{2c}{l}\right) \qquad (8)$$

A partir das equações (5) e (6), e com as limitações dadas por (8), podemos calcular as velocidades V_A e V_B.

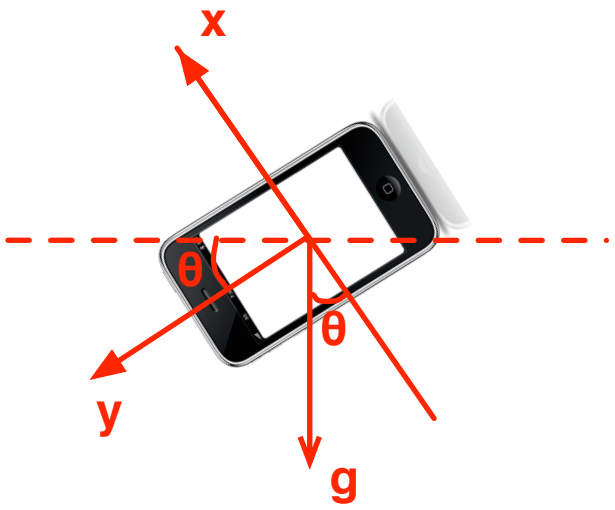

Figura 14 - Cálculo do ângulo θ de inclinação do "iPhone"

Como os valores que o acelerômetro do "iPhone" fornecem são as componentes do vetor gravidade g nas coordenadas x e y, temos:

$$\tan\theta = -\frac{g_y}{g_x} \qquad (9)$$

No "iPhone" ou "iPod Touch", a velocidade a ser calculada para o robô é feito através de um gesto de deslizamento do dedo sobre a tela multi-toques. Ao tocar a tela, o algoritmo armazena o primeiro ponto do toque e calcula a componente da distância entre o ponto de início do toque e o último ponto do deslizamento do dedo paralela à largura do aparelho (d). Dado esta distância, calculamos a V_{Cm} desejada:

$$V_{Cm} = k \times d \qquad (10)$$

Onde: k = fator de conversão da distância para a velocidade do centro de massa desejada.

Substituindo (9) e (10) em (5) e (6), temos:

$$V_B = k \times d \times \sqrt{1 + \frac{l \times g_y}{2 \times c \times g_x}} \qquad (11)$$

$$V_A = k \times d \times \sqrt{1 - \frac{l \times g_y}{2 \times c \times g_x}} \qquad (12)$$

Para $\frac{g_y}{g_x} \leq \frac{2 \times c}{l}$

Onde:

V_B e V_A = velocidade em cada roda do robô móvel;

d = distância entre o ponto do início do deslizamento e o último ponto de deslizamento, paralelo ao eixo x solidário ao “iPhone”;

k = fator de conversão de distância para velocidade do centro de massa;

l = distância entre as rodas laterais;

c = distância entre os eixos das rodas dianteiras e traseiras;

g_x = componente do vetor gravidade no eixo x solidário ao “iPhone”, obtido pelo acelerômetro;

g_y = componente do vetor gravidade no eixo y solidário ao “iPhone”, obtido pelo acelerômetro.

O fluxograma abaixo detalha o algoritmo utilizado para calcular as velocidades de cada roda do robô móvel para simular o esterçamento do volante de um automóvel:

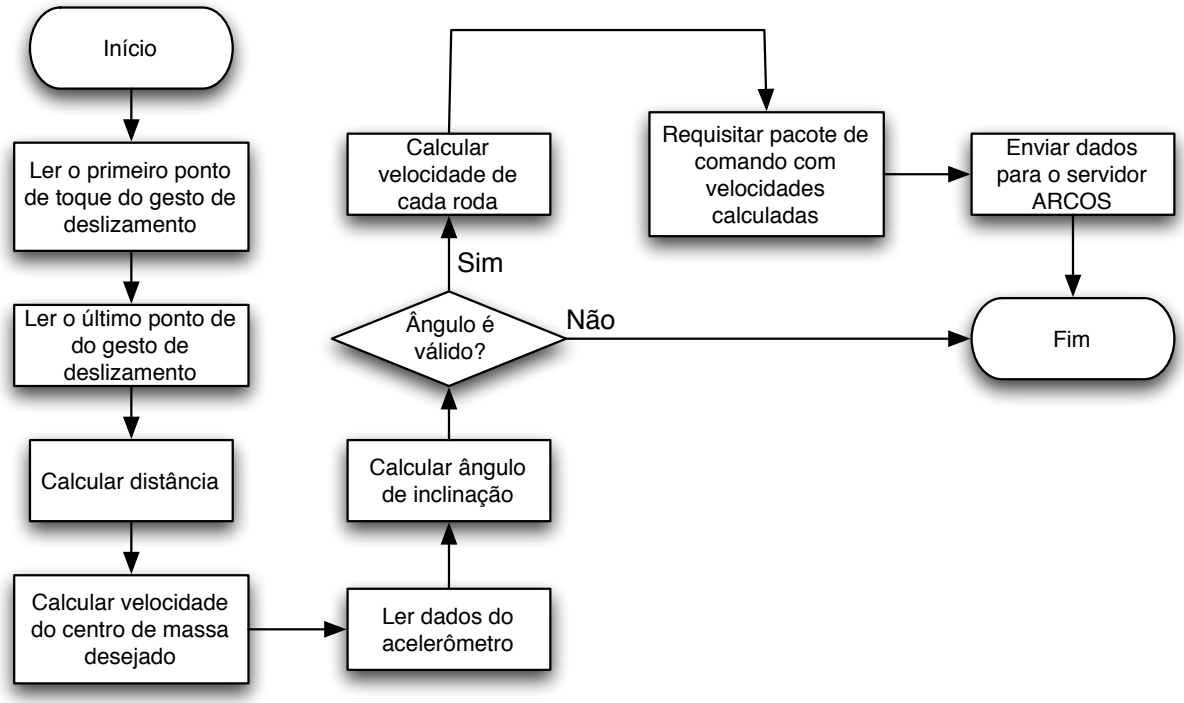

Figura 15 - Fluxograma do algoritmo de curvas

4.8 Diagrama de classes

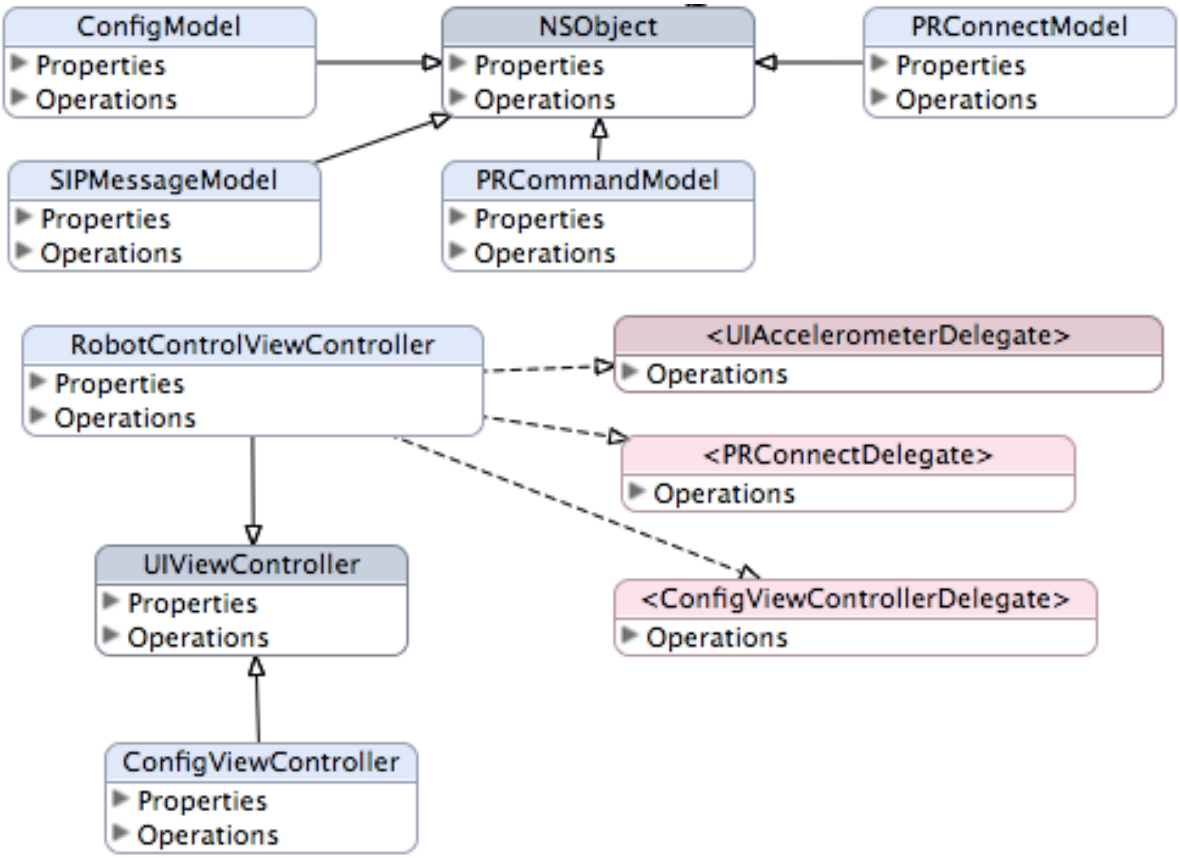

Figura 16 - Visão geral do diagrama de classes

Nota-se pelo diagrama de classes que os objetos de modelo da arquitetura MVC herdam da `classe base` “NSObject”. No “Cocoa”, todas as `classes` herdam da `classe` “NSObject”.

A `classe` “ConfigModel” tem como função encapsular três importantes dados para conexão com o robô: ip (string), velocidade máxima (inteiro) e porta de conexão (inteiro). Através do “ConfigModel”, a `classe` do controlador “RobotControlViewController” irá acessar estes dados previamente definidos pelo controlador “ConfigViewController”.

“PRConnectModel” é a `classe` que encapsula a lógica da conexão TCP/IP. Possui três importantes métodos: “connect”, “disconnect” e “sendData”. O método “connect” recebe dois parâmetros, o ip (*string*) e port (*int*); basicamente, este método estabelece uma conexão TCP/IP dado o endereço de IP e a porta. Uma vez estabelecido a conexão, o método “sendData” encarrega-se de enviar os dados, enquanto “disconnect” termina a conexão.

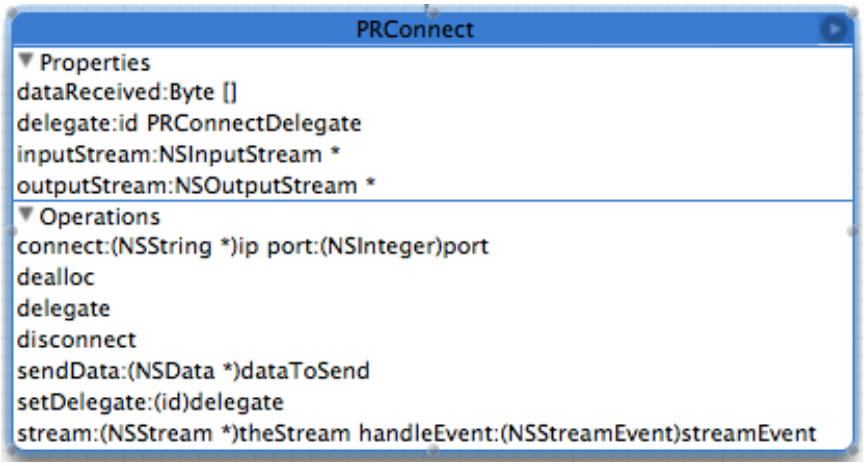

Figura 17 - Classe PRConnect

A `classe` "PRCommandModel" é responsável por gerar os pacotes de comando para controlar o robô móvel, enquanto a `classe` "SIPMessageModel" é a responsável por separar e interpretar os pacotes SIP.

"ConfigViewController" tem como única função manipular os dados iniciais necessários para estabelecer a conexão e controle do robô móvel.

"RobotControlViewController" é a `classe` fundamental para o funcionamento do aplicativo. Esta `classe` é quem chama os métodos da `classe` "PRConnectModel" para estabelecer e terminar conexão, além de transmitir dados para o robô móvel. Uma vez estabelecida a conexão com o robô móvel, "RobotControlViewController" decidirá, através de gestos na tela sensível a toques e leitura de acelerômetros, o que o robô irá realizar. Uma vez decidido a tarefa do robô, é gerado os pacotes de comandos e transmitidos ao robô e, concorrentemente, os dados dos pacotes de informação do servidor são redirecionados ao "SIPMessageModel", o qual interpreta os dados e, em intervalos de tempo pré-definidos, "RobotControlViewController" recebe a notificação que há novos dados a serem exibidos na interface do usuário.

4.9 Diagramas de sequência

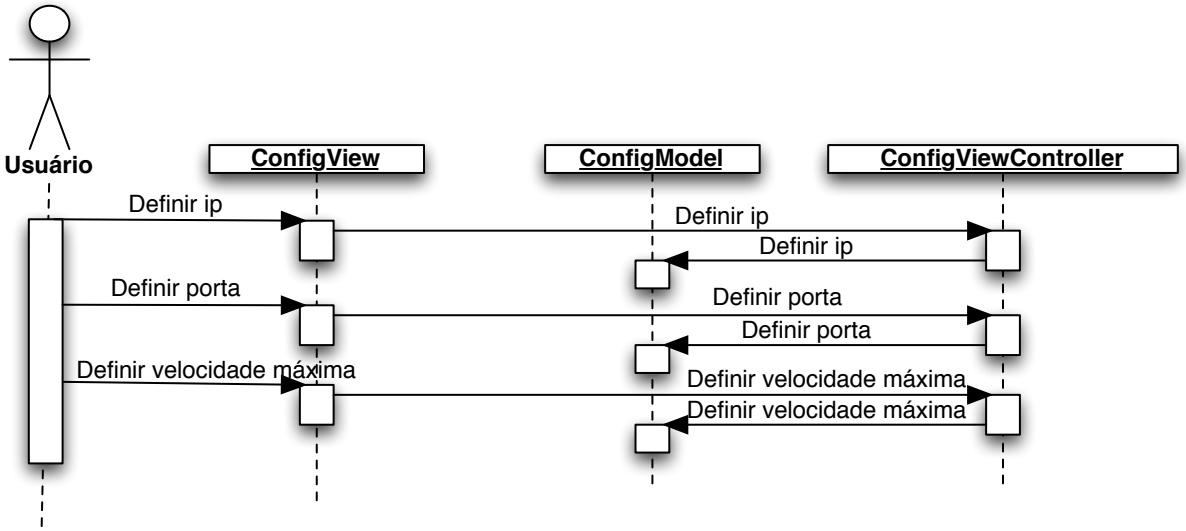

Figura 18 - Diagrama de sequência para definição das configurações iniciais

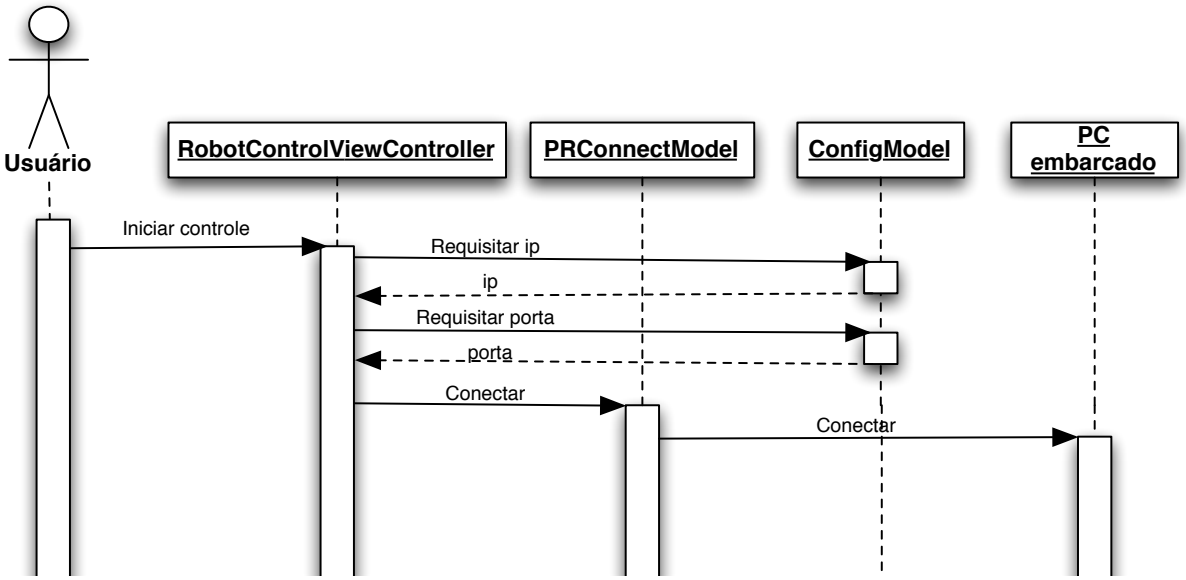

Figura 19 - Diagrama de sequência para conexão com robô remotamente

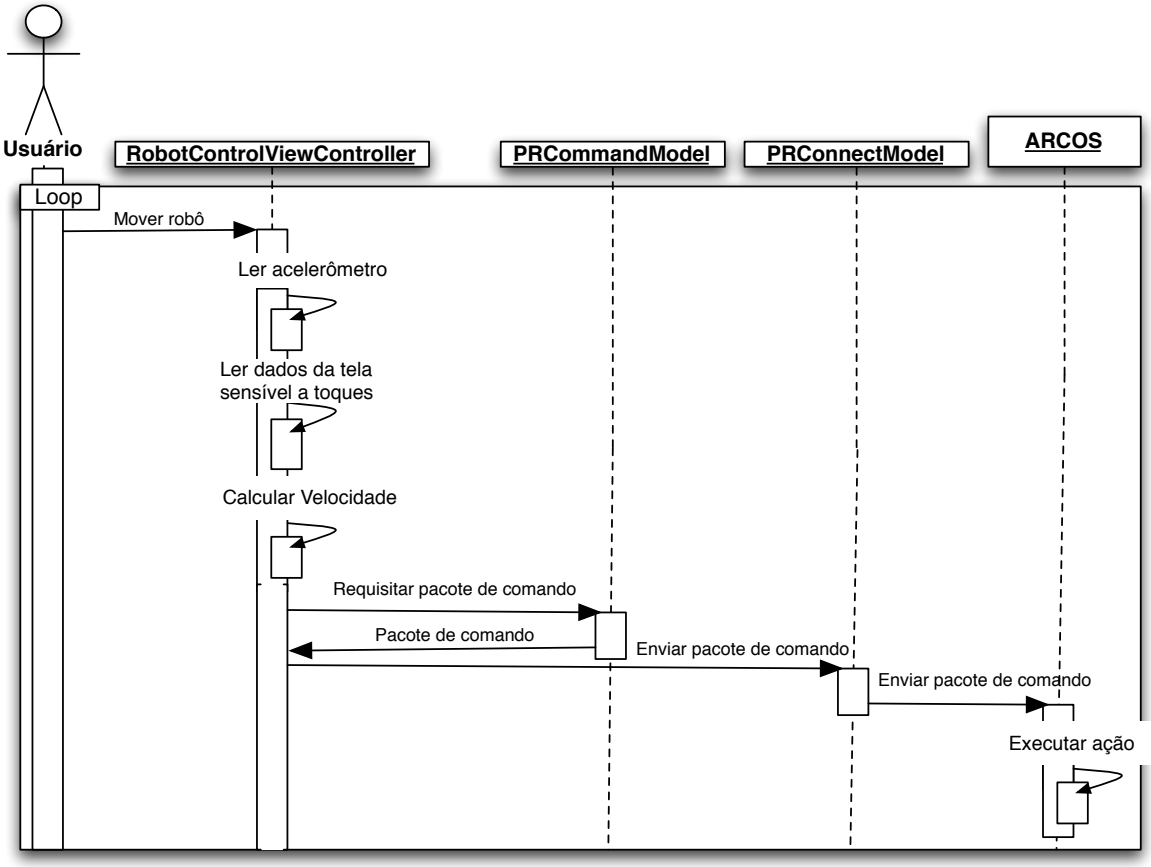

Figura 20 - Diagrama de sequência para envio de comando de movimento para o robô

No diagrama de sequência acima, foi omitido o “PC embarcado”, já que, uma vez estabelecido a conexão TCP/IP, tem como única função direcionar os dados para o servidor ARCOS.

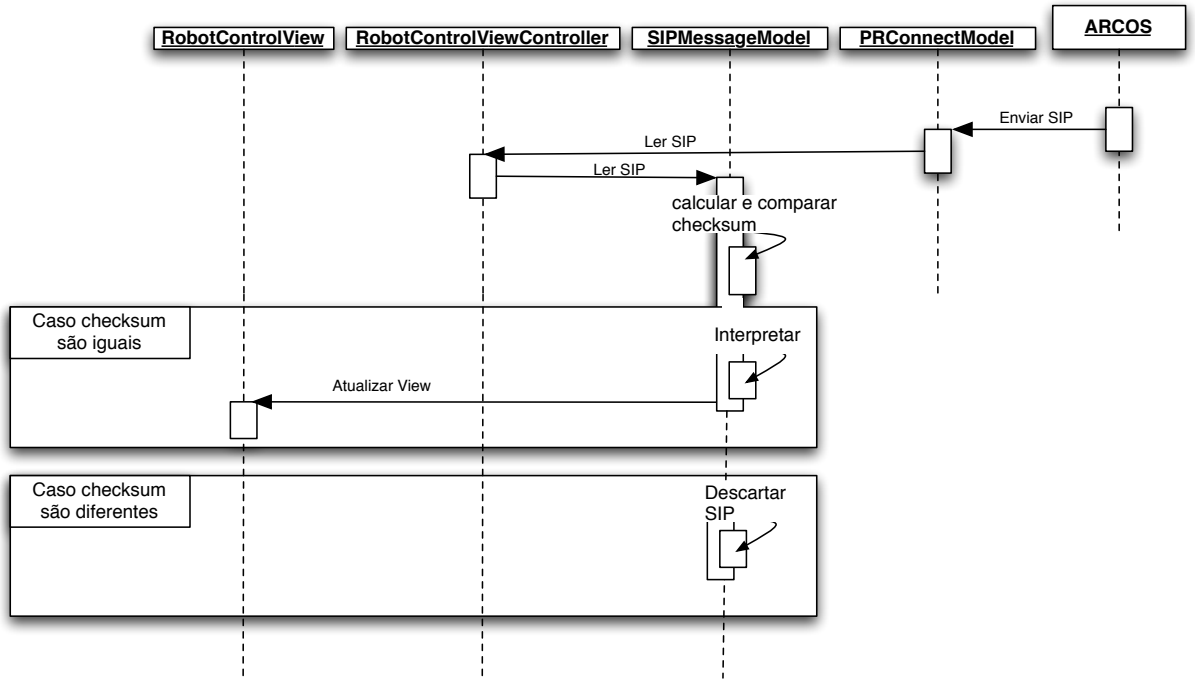

Figura 21 - Diagrama de sequência para leitura de SIP

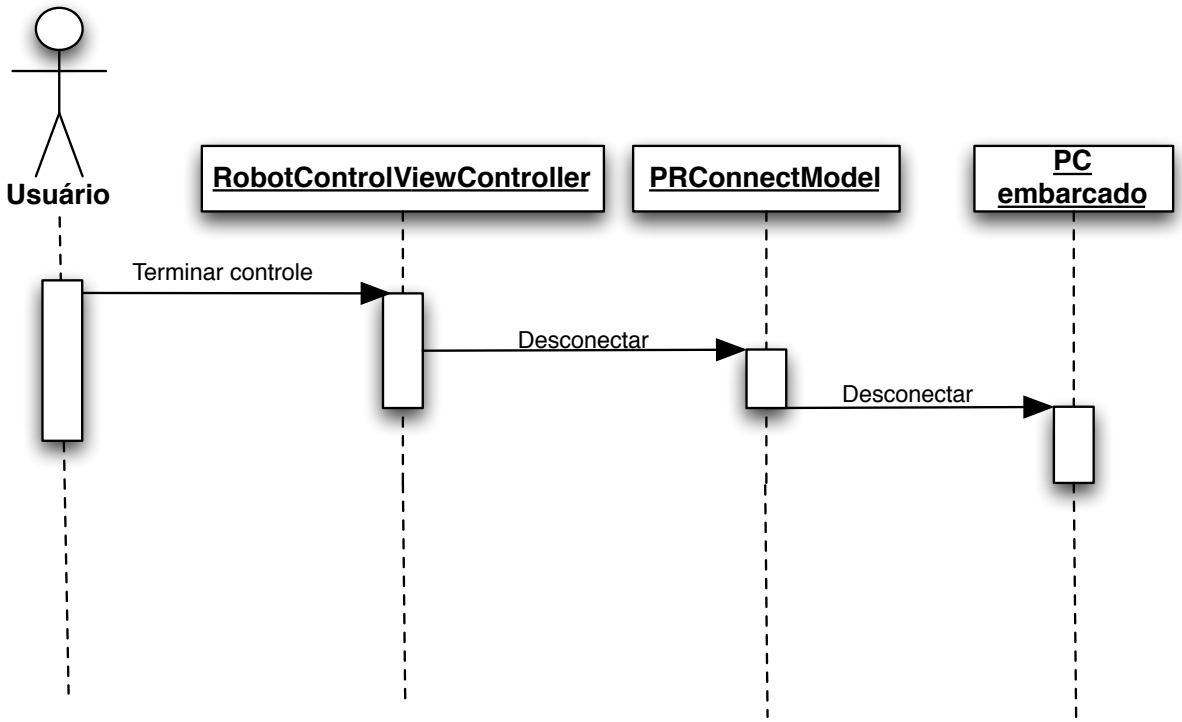

Figura 22 - Diagrama de sequência para desconexão

4.10 Operação do software

A primeira tela ao ser exibida ao iniciar o aplicativo é a tela de operação do robô móvel, contudo, antes de exercer o controle, deve-se definir as configurações iniciais. Para definir as configurações, deve-se tocar a tela onde está localizado um pequeno botão cinza com a letra “i”.

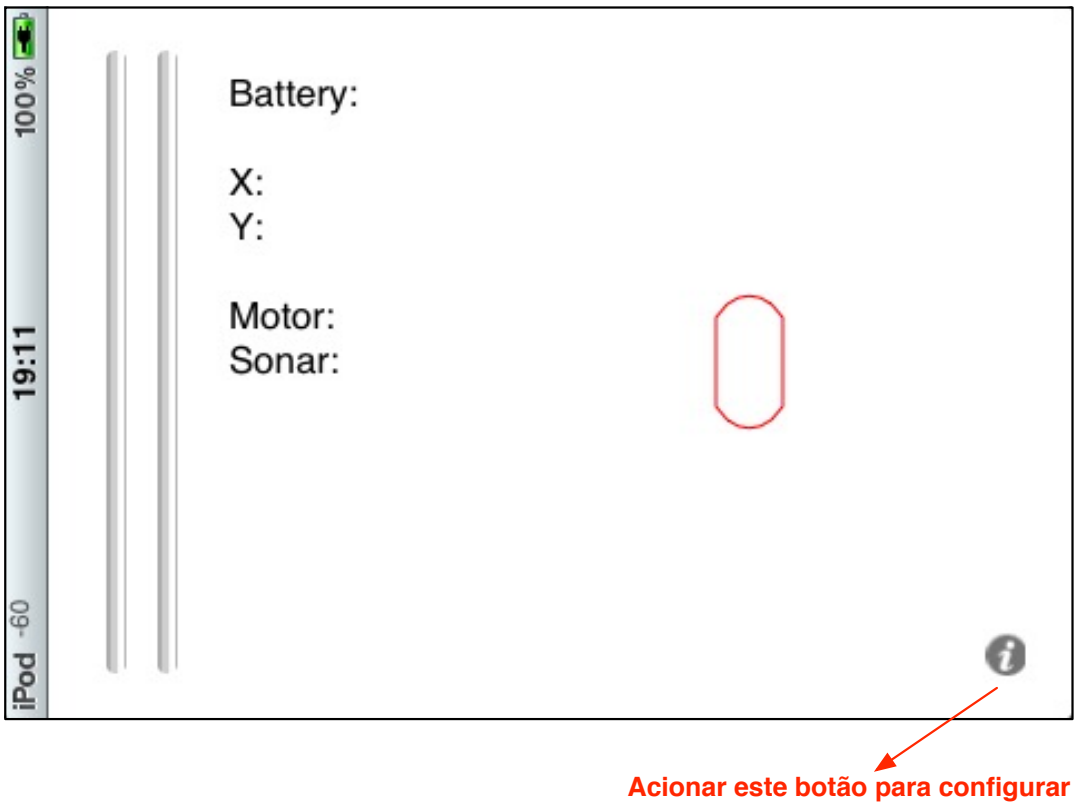

Figura 23 - Tela inicial do aplicativo

Ao tocar no botão “i”, o aplicativo troca de tela para a interface de configuração básica. É necessário definir três parâmetros: o endereço de IP (Internet Procotol) do robô móvel Pioneer 3-AT na rede wireless Wi-Fi, a porta a conectar-se, previamente definido pelo software ser2net, e a máxima velocidade permitida ao controlar o robô.

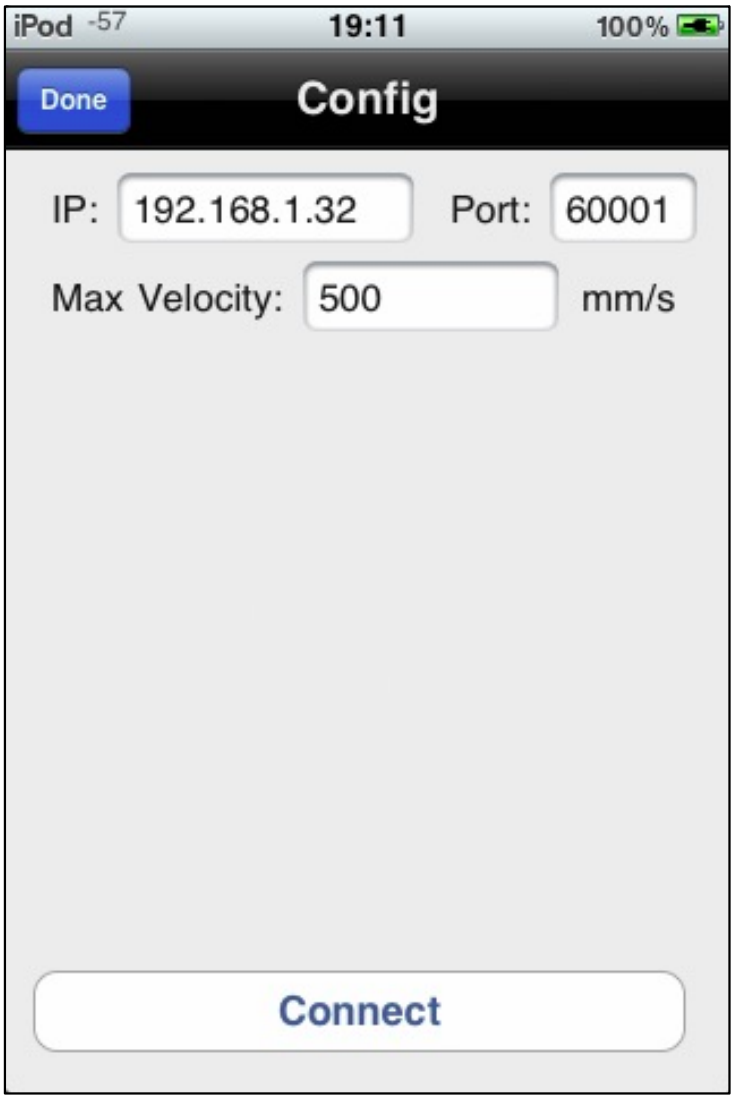

Figura 24 - Tela de configuração

Após definir os parâmetros de configuração, deve-se acionar o botão “Connect” para iniciar a comunicação com o robô, caso os parâmetros estejam corretos, a tela muda para a tela inicial do aplicativo.

Ao conectar-se ao robô, as informações dos sensores são automaticamente atualizados: nível de tensão da bateria (em múltiplos de 10 V), coordenadas X e Y segundo os *encoders*, o estado de movimento (parado ou movendo) e o estado do sonar (ligado ou desligado).

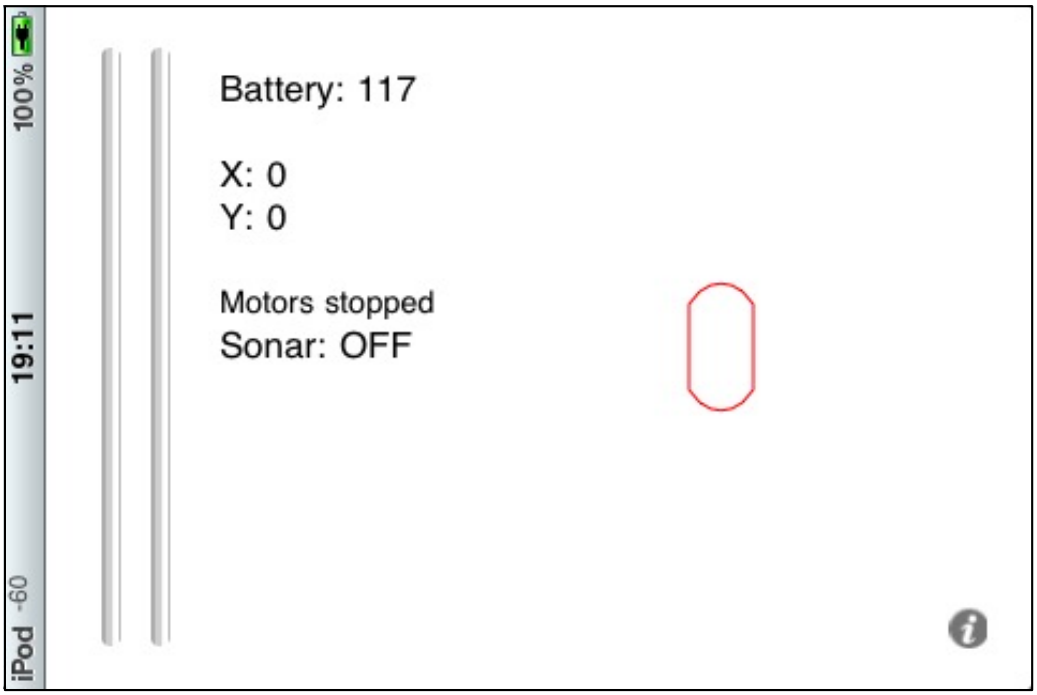

Figura 25 - Tela de comando exibindo leitura de sensores

O desenho em vermelho no lado direito da tela de comando representa o robô móvel, enquanto as barras à esquerda, representam a velocidade das rodas.

Para mover o robô, é necessário iniciar um gesto de deslizamento sobre a tela sensível a toques do “iPhone”. Para virar o robô, enquanto este está movendo-se deve-se simultaneamente inclinar o dispositivo para o lado o qual deseja-se fazer a curva.

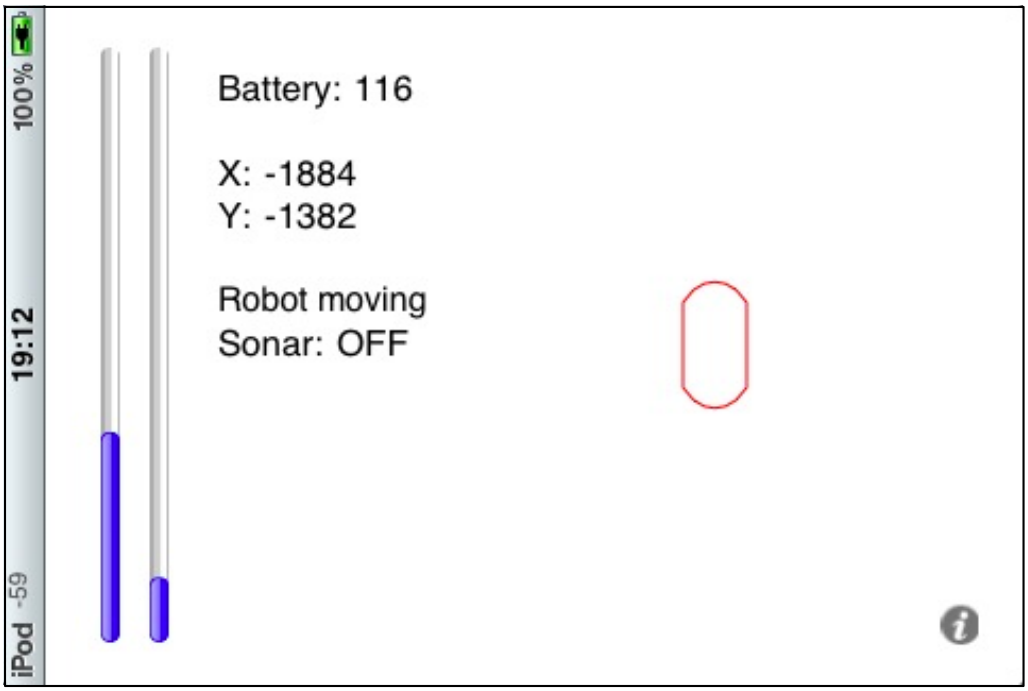

Figura 26 – Velocidades positivas sendo exibidas na tela de comando

O nível das barras à esquerda, o qual representam a velocidade de cada lado do robô (rodas da direita e esquerda), enchem-se conforme a velocidade aumenta, assim como sua cor muda conforme o sentido de movimento do robô (frente ou trás). Ao mover-se para frente, as barras mudam para a cor azul, enquanto movimentos de ré modifica a cor das barras para vermelho.

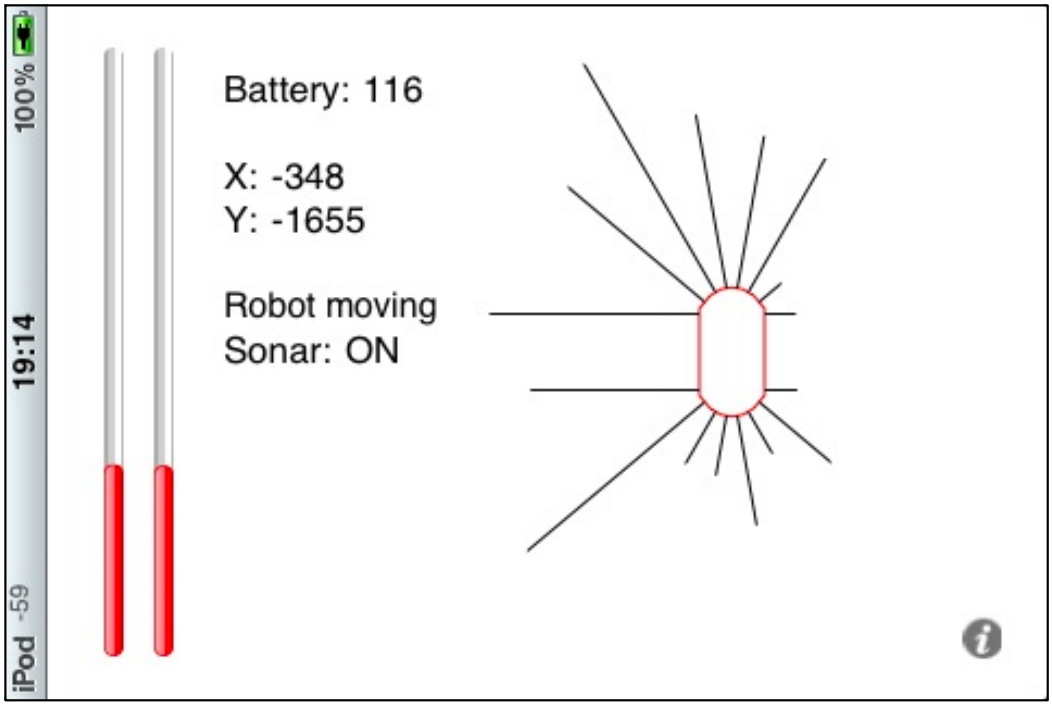

Figura 27 - Tela exibindo leitura dos sonares

Para ligar e desligar os sonares, deve-se balançar o dispositivo, caso o sonar esteja ligado, ao balançar, este é desligado; caso o sonar esteja desligado, ao balançar, este é ligado.

Ao ligar os sonares, as leituras são exibidas na tela na forma de comprimento de retas em torno do desenho representando o robô móvel. Cada uma das retas representam uma leitura de um sonar e são desenhados na tela conforme sua orientação no robô.

Para desconectar-se do robô, há dois modos: acionar novamente o botão “i” e mudar para a tela de configurações ou apertar o botão “Home” do “iPhone” ou “iPod Touch” e terminar o aplicativo.

5 RESULTADOS

Para a realização dos testes, foram considerados dois aspectos importantes para o controle remoto do robô móvel: a comunicação e o controle de velocidade.

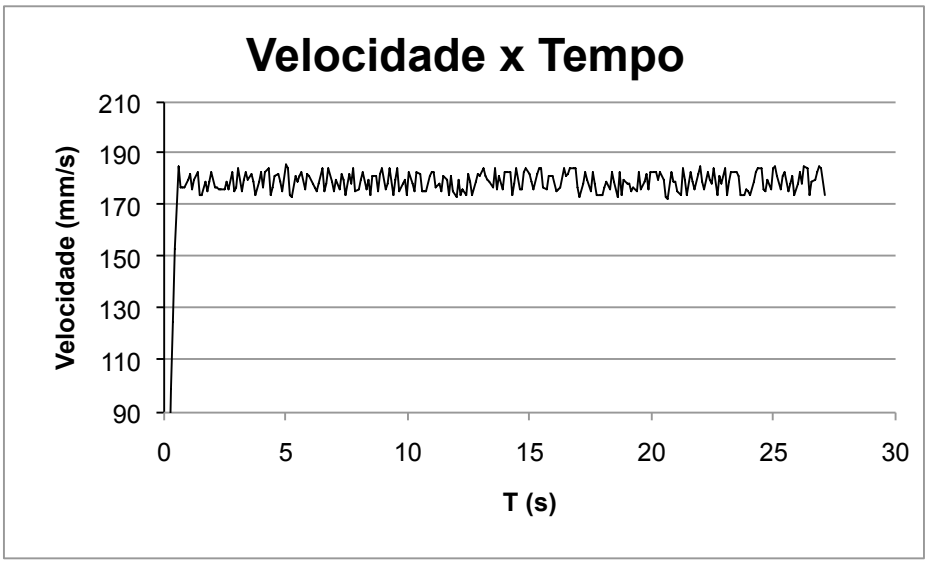

Figura 28 - Oscilação natural da velocidade

No primeiro teste, foi enviado ao robô uma velocidade fixa para movimentação e obteve-se a velocidade real do robô através da leitura dos pacotes de informação. Desta forma, podemos observar a oscilação natural da velocidade do robô. Observa-se pelo gráfico de oscilação natural da velocidade que a velocidade de referência é alcançada rapidamente, demonstrando um pequeno *overshooting*. O desvio padrão calculado foi de apenas 3,5, ou seja, a velocidade pouco varia depois de alcançado o valor de referência.

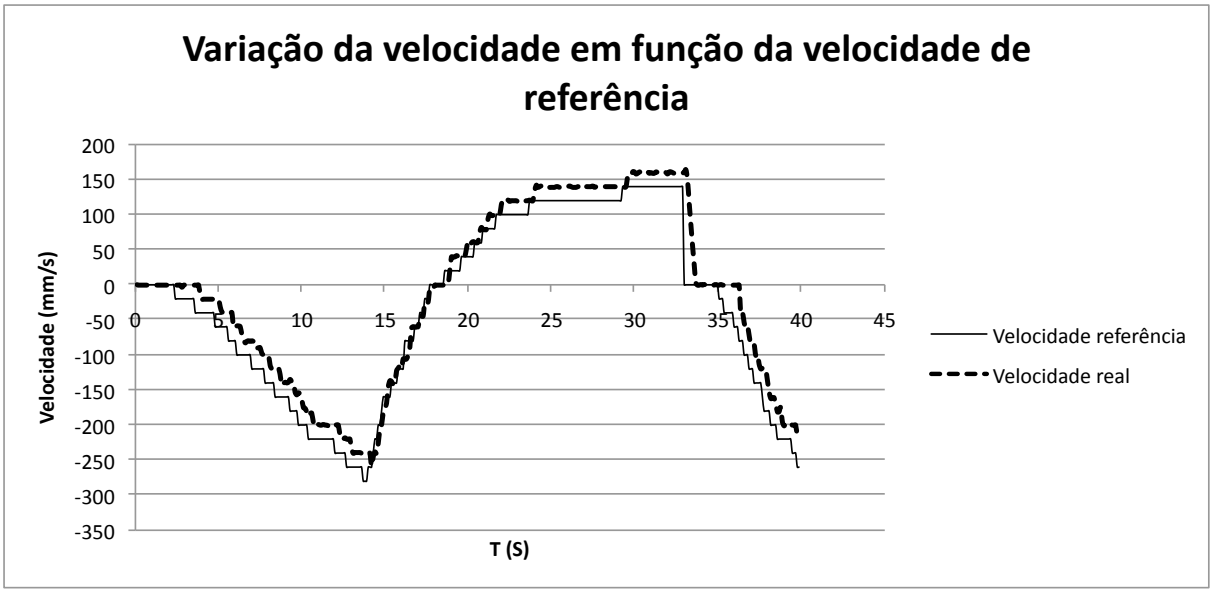

Figura 29 - Variação da velocidade em função da velocidade de referência

No segundo teste, foi verificado como o robô responde dada uma variação da velocidade de referência. Observa-se no gráfico que os maiores atrasos são por volta de 1 segundo. O atraso pode ser explicado por dois motivos: inércia do robô móvel e velocidade de transmissão de dados. Quanto maior a inércia do robô móvel, maior o atraso (devido a necessidade de altas acelerações); uma menor velocidade de transmissão de dados acarreta na mesma consequência devido ao atraso dos dados.

Por último, foi comparado quantos pacotes de dados chegavam danificados. Foi descoberto uma peculiaridade quanto ao número de pacotes de informação do servidor recebido: com o sonar ligado, o número de pacotes de dados danificados cresceu relevantemente em comparação com o caso do sonar desligado. Isto pode ocorrer devido ao aumento relevante do tamanho do pacote de dados. Quanto maior o pacote de dados, maiores são as chances de perda de dados.

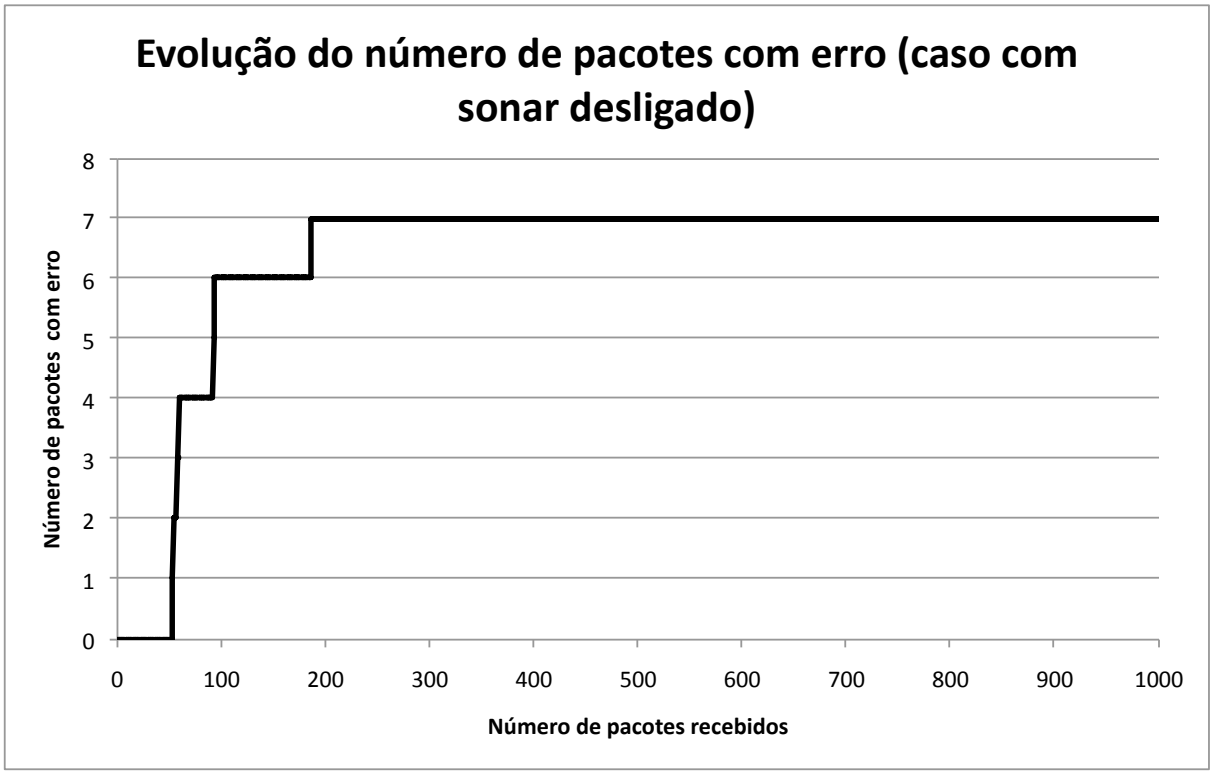

Figura 30 - Evolução do número de SIP com erro no caso do sonar desligado

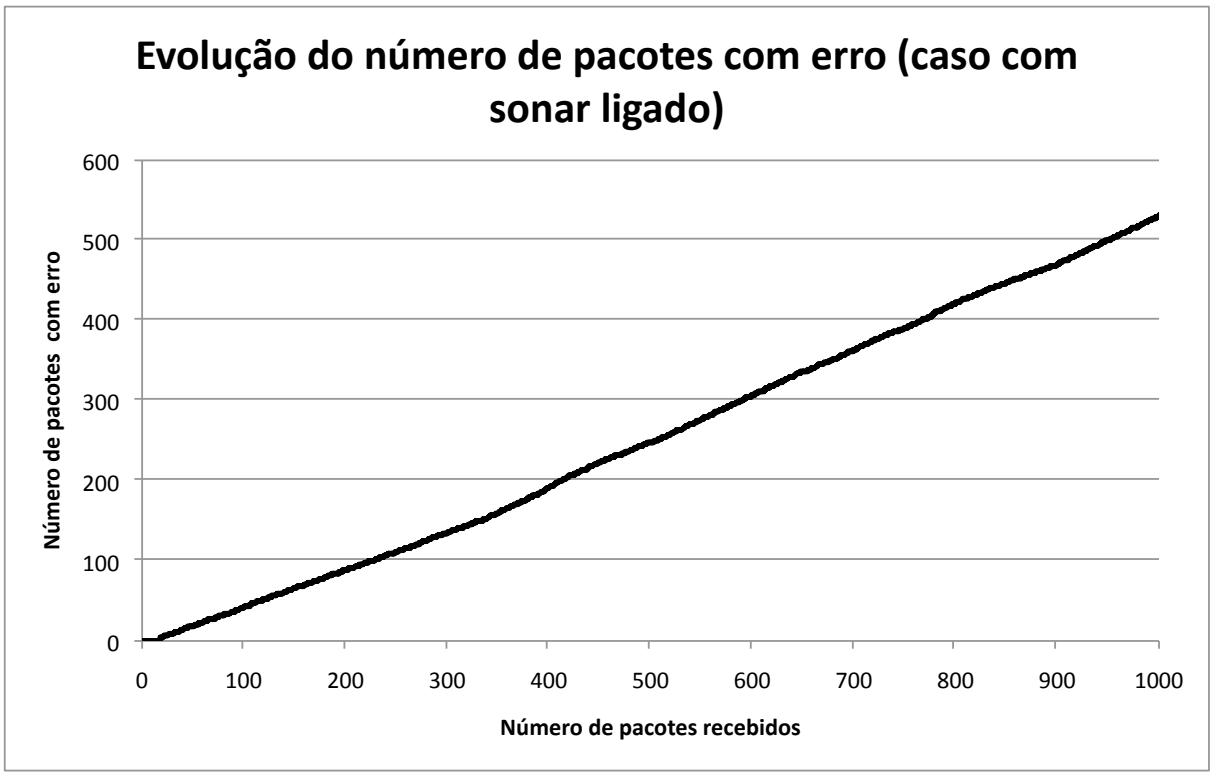

Figura 31 - Evolução do número de SIP com erro no caso do sonar ligado

6 CONSIDERAÇÕES FINAIS

Neste trabalho foi desenvolvido um aplicativo que permite o controle remoto de um robô “Pioneer 3-AT” por “iPhone” e “iPod Touch” através de uma rede de computadores wireless “Wi-Fi”. O controle é feito utilizando-se de dois principais sensores do dispositivo: acelerômetro e tela multi-toques. A tela multi-toques foi utilizada como meio para definir configurações iniciais e em conjunto com o acelerômetro, em um gesto de deslizamento sobre a tela, controlar a velocidade dos motores do robô móvel. Além disso, o acelerômetro foi utilizado para detectar gesto de vibração para ligar e desligar os sonares.

Enquanto o usuário está conectado ao robô móvel, é possível através de uma interface gráfica visualizar a velocidade de cada roda, o nível de tensão da bateria, ler as coordenadas cartesianas X e Y e o estado de movimento do robô móvel.

Durante os testes do *software* desenvolvido foi encontrado uma peculiaridade relacionado a leitura dos sonares. Quando os sonares estavam ligados, a maioria dos pacotes de informação do servidor continham *checksum* diferente do *checksum* calculado, ou seja, estavam corrompidos e, consequentemente, a atualização dos dados dos sensores é prejudicada.

Como trabalho futuro, poderia ser desenvolvido novos meios de interação para controle do robô, como o controle da posição do robô utilizando toques na tela multi-toques através de um mapa, ou o uso da inclinação do “iPhone” ou “iPod Touch” para controlar a velocidade simulando uma esfera sobre uma mesa. Além disso, poderia ser aperfeiçoado a leitura dos dados dos sonares. Finalmente, poderia ser incluso a visualização de uma câmera de vídeo na interface do usuário, permitindo o controle remoto através da internet em uma visão em primeira pessoa.

7 BIBLIOGRAFIA

APPLE. **The Objective-C Programming Language: Introduction to The Objective-C Programming Language**. Disponível em: <**http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html**> Acesso em: 14 abr. 2010.

BITTNER, K.; SPENCE, I. **Use Case Modeling**. Boston: Addison-Wesley Professional, 2002. 368 p. ISBN 978-0201709131.

CRUNCHGEAR. **DIY: Control your Hexapod robot with your iPhone**. Disponível em: <http://www.crunchgear.com/2010/03/08/diy-control-your-hexapod-robot-with-your-iphone/>. Acesso em: 10 mar. 2010.

ISPYKEE. **The free open-source Skypee iPhone App**. Disponível em: <http://ispykee.toyz.org/intro>. Acesso em: 10 mar. 2010.

MARK, D.; LAMARCHE, J. **Beginning iPhone Development:** Exploring the iPhone SDK. Berkeley: Apress, 2009. 508 p. ISBN 978-1-4302-1626-1.

MOBILEROBOTS. **Pioneer 3 Oprations Manual**. 5. ed. 2007. 75 p.

PARZIALE, L.; BRITT, D. T.; DAVIS, C.; FORRESTER, J.; LIU, W. **TCP/IP Tutorial and Technical Overview**. 8. ed. : IBM Redbooks, 2006. 974 p. ISBN 0738494682.

PARROT. **AR.Drone.com USA - Parrot Wi-Fi quadricopter. Augmented Reality games on iPhone, iPod touch and iPad**. Disponível em: <http://ardrone.parrot.com/parrot-ar-drone/usa/>. Acesso em: 23 ago. 2010.

PRESSMAN, R. S. **Engenharia de Software**. 6. ed. Rio de Janeiro: McGraw-Hill, 2006. 720 p. ISBN 8586804576.

RIEHLER, D. **Framework Design: A Role Modeling Approach**. Swiss Federal Institute of Technology Zurich. Zurique, p. 209. 2000.

SUGIURA, Y.; FERNANDO, C. L.; WITHANA, A. I.; KAKESHI, G.; SAKAMOTO, D.; SAKAMOTO, M.; INAMI, M.; IGARASHI, T.; INAKAGE, M. **An Operating Method for a Bipedal Walking Robot for Entertainment**. ACM SIGGRAPH ASIA 2009 Emerging Technologies. Yokohama: SIGGRAPH. 2009. p. 79-79.

VLIET, H. V. **Software Engineering:** Principles and Practice. 3nd Edition. ed. Amsterdam: John Wiley & Sons, 2008. 740 p. ISBN 0-47-003146-8.

XCODE. **XCode – Developer Tools Technology Overview – Apple Developer**. Disponível em: <http://developer.apple.com/technologies/tools/xcode.html> . Acesso em: 2 mar. 2010.