mαcrϕ@ufmg
Universidade Federal de Minas Gerais
Curso de Graduação em Engenharia de Controle e Automação
Grupo de pesquisa MACRO - Mecatrônica, Controle e Robótica

SISTEMA DE AQUISIÇÃO DE DADOS DE UMA UNIDADE DE MEDIÇÃO INERCIAL E INTEGRAÇÃO COM O ROS

Gabriel Mesquita Cangussu
Belo Horizonte, Brasil
2016

Gabriel Mesquita Cangussu

SISTEMA DE AQUISIÇÃO DE DADOS DE UMA UNIDADE DE MEDIÇÃO INERCIAL E INTEGRAÇÃO COM O ROS

Monografia submetida à banca examinadora designada pelo Colegiado Didático do Curso de Graduação em Engenharia de Sistemas da Universidade Federal de Minas Gerais, como parte dos requisitos para aprovação na disciplina Trabalho de Conclusão de Curso II.

Orientador: Bruno Vilhena Adorno

Belo Horizonte, Brasil
2016

À minha família, pelo suporte incondicional.

Agradecimentos

Agradeço a todos os professores que contribuíram para a minha formação, em especial agradeço ao Prof. Bruno Vilhena Adorno por seu apoio e orientação.

Ao grupo de pesquisa MACRO e todos os seus integrantes pela oportunidade proporcionada e pelo suporte a meu trabalho.

Resumo

Este trabalho contextualiza uma arquitetura utilizada em aplicações de robótica que apresenta alta latência. Esta arquitetura era utilizada quando se fazia necessário a utilização de uma unidade de medição inercial (IMU) na aplicação. Essa mesma IMU tinha apenas um sistema de aquisição de dados implementado em MATLAB. Porém, para a diminuição da latência é proposta uma mudança na arquitetura em favor da substituição do MATLAB pelo ROS (*Robot Operating System*). O ROS é um *framework* para desenvolvimento de *software* para robótica o qual permite implementações de sistemas de baixa latência. Para viabilizar o uso da IMU com o ROS é então desenvolvido um sistema de aquisição de dados da IMU que se integra ao ROS. O desenvolvimento desse sistema é conduzido com foco na minimização do tempo de execução do programa, utilizando as linguagens C e C++. Como resultado, na arquitetura com o ROS e implementação em C/C++, obteve-se um sistema com latências mais de 30 vezes menores quando comparadas à versão que utilizava MATLAB.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O MACRO é um grupo de pesquisa da Universidade Federal de Minas Gerais focado em mecatrônica, controle e robótica. No MACRO as pesquisas que envolvem robótica usualmente apresentam todo um desenvolvimento teórico e também uma posterior prova de conceito numa demonstração prática. Para fazer essa demonstração, os pesquisadores do MACRO têm acesso a manipuladores robóticos, bases móveis e diversos periféricos para sensoriamento e controle.

A integração dos componentes necessários para se fazer uma demonstração envolvendo o controle de robôs apresenta diversos desafios. Entre eles está o desenvolvimento de *software* para robótica, o qual tem alta complexidade devido a um escopo que pode abranger desde o desenvolvimento de *drivers* para acesso ao hardware, até o desenvolvimento de inteligência artificial para um robô autônomo. Durante o desenvolvimento de *softwares* nos laboratórios do MACRO, um dos desafios encontrados foi a alta latência das aplicações que utilizam MATLAB em parte do seu sistema. Alta latência nesse contexto é um elevado tempo para amostrar e processar todos os dados necessárias para se controlar um robô. Essa latência elevada não inviabiliza a prova de conceito em si, porém torna lenta a atuação dos robôs que também exibem movimentos não naturais e consequentemente apresentam pouco apelo a possíveis aplicações finais.

Demonstrações responsivas, com baixas latências e movimentos naturais seriam experimentos de alto desempenho. A execução desse tipo de experimento é de interesse do MACRO e tem potencial para promover benefícios sociais e econômicos. Por estes motivos o objetivo principal desse trabalho é aumentar a capacidade de realização de experimentos de robótica de alto desempenho pelo grupo. Para aumentar essa capacidade, será feito um novo sistema de aquisição de dados de uma unidade de medição inercial (IMU), antes só disponível para MATLAB. O sistema novo deverá ser integrado ao ROS (*Robot Operating System*), uma das plataformas utilizadas pelo MACRO para experimentos de alto desempenho. Outros objetivos complementares são a análise dos impactos sociais, econômicos e culturais desse projeto (apêndice A) e a disponibilização de ferramentas de código aberto para a comunidade de desenvolvedores de *software* para robótica.

A arquitetura atual dos experimentos realizados pelo MACRO com a IMU é de grande relevância para esse trabalho. Arquitetura nesse trabalho se refere a "como se integra os diversos componentes necessários para o experimento" e o motivo de sua relevância é que a forma como ela é estruturada e implementada impacta diretamente no desempenho da demonstração. Devido a este impacto no desempenho, será necessário a revisão da arquitetura

de experimentos para se conseguir atender o objetivo principal do projeto.

1.1 Contextualização

O grupo de pesquisa MACRO executa experimentos com robôs para demonstrações de avanços nas teorias da área de robótica. Essas demonstrações geralmente incluem o uso de manipuladores robóticos, bases móveis e sensores. Para processamento dos dados de sensores e controle dos robôs geralmente são utilizados tanto algoritmos implementados em MATLAB quanto serviços para o ROS (*Robot Operating System)*.

O ROS é um projeto de código aberto de um sistema operacional para robôs. Este não é um sistema operacional no sentido tradicional do termo, na verdade o ROS provê uma camada de comunicação estruturada acima do sistema operacional hospedeiro de um cluster heterogêneo de computação (Quigley et al., 2009). De uma maneira mais prática, podemos definir o ROS como um *framework* para desenvolvimento de *software* para robôs, o qual permite a estruturação do sistema em múltiplos processos em dispositivos diferentes. Estes processos são comumente chamados nós ou clientes e eles trocam informações e oferecem serviços através da camada de rede do ROS.

Entre outras funcionalidades, o ROS provê um serviço de troca de mensagens por meio de um esquema *publisher-subscriber*. Nesse esquema um nó se cadastra no serviço como um *publisher* de um dado tópico e outro nó pode receber esses dados se cadastrando como um *subscriber* no mesmo tópico. Toda vez que um dado é publicado em um tópico todos os *subscribers* desse mesmo tópico receberão os dados. Para implementação de um nó compatível com o ROS, existem bibliotecas para C++, Python, Lisp, MATLAB, entre outras linguagens.

Um exemplo de arquitetura utilizada pelo MACRO para seus experimentos está ilustrada na figura 1.1. Esta arquitetura é composta por um manipulador móvel, uma unidade de medição inercial (IMU) e uma câmera com informação de profundidade. O manipulador móvel consiste em uma base móvel iRobot Create com um manipulador AX-18A montado no topo, ambas as partes se comunicam com um computador através de portas seriais. A IMU é uma 3DM-GX2 a qual transmite os dados através de uma rede sem fio IEEE 802.15.4 (2.4 GHz) para uma estação base conectada ao computador através de uma interface serial. E por fim, a câmera é um Microsoft Kinect conectado através de uma interface USB 2.0.

Nessa arquitetura as informações são geridas pelo ROS e MATLAB. O Kinect publica informações de posição e orientação do robô em tópicos no serviço de mensagens do ROS. O MATLAB, inscrito nos mesmos tópicos em que o Kinect publica dados, recebe as informações de posição e orientação do robô através do ROS. Já o manipulador AX-18A, a base Create e a IMU se comunicam com o MATLAB através de bibliotecas que tem acesso às portas seriais. O MATLAB obtém e processa os dados oriundos do ROS, da odometria da base, da odometria do manipulador e da IMU em ciclos. Ao final de um ciclo o MATLAB envia um comando de controle para o manipulador móvel. O tempo de latência do sistema é o período entre comandos de controle, ou seja, é o tempo de execução de um ciclo.

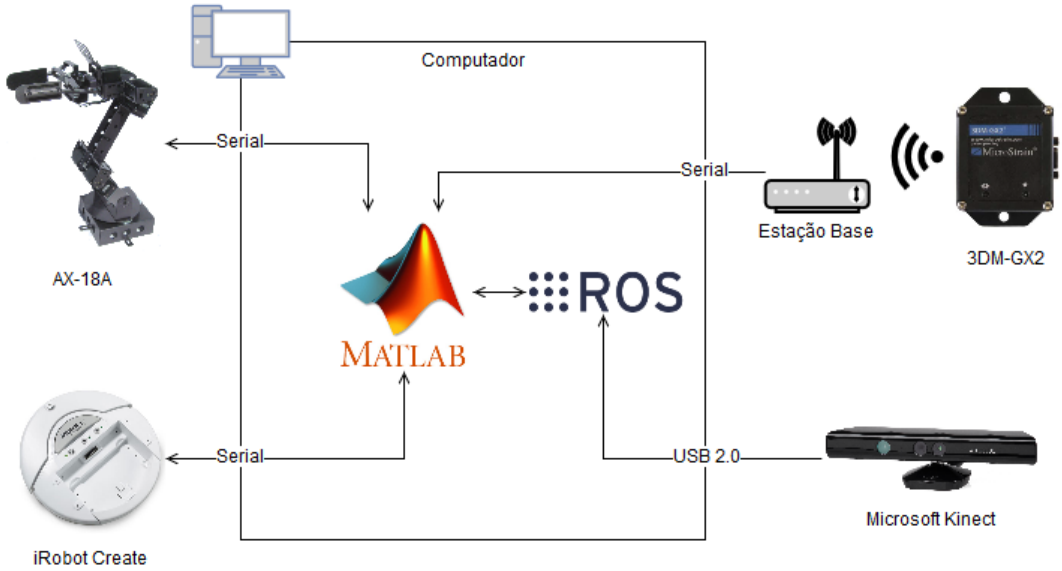

Figura 1.1: Arquitetura considerada neste trabalho

1.2 Problemas Identificados

A arquitetura descrita na seção anterior apresenta mais de 200 milissegundos de latência média durante experimentos, o que é bastante elevado para aplicações de robótica. Devido a isso o manipulador móvel apresenta lentidão e movimentos não naturais, além de haver risco de desestabilização do sistema.

Esses problemas podem reduzir o apelo das técnicas pesquisadas para aplicações que exigem alto desempenho, pois é necessário demonstrar que uma técnica é funcional num contexto de alto desempenho para que só então ela seja considerada adequada para tal tipo de aplicação. Aplicações que exigem a direta interação do robô com humanos é um exemplo de tipo de aplicação em que alto desempenho é essencial, pois a interação com robôs lentos, irresponsivos e até possivelmente instáveis é perigosa e frustrante para o usuário final.

1.3 Escopo

O objetivo deste projeto é revisar uma das arquiteturas atuais utilizadas pelo MACRO em seus experimentos e propor mudanças para redução da latência. A arquitetura que será revisada é a apresentada na figura 1.1. Para a redução da latência dessa arquitetura o MATLAB deverá ser removido do sistema. Por conta dessa remoção, um novo sistema de aquisição de dados da IMU deverá ser desenvolvido, pois até então só existia um sistema dependente do MATLAB. Além da baixa latência, é desejável que a nova arquitetura permita a rápida prototipação de provas de conceito e facilite a integração de outras soluções, como as já desenvolvidas pelo próprio MACRO, as de terceiros e as distribuídas com código aberto.

A fim de demonstrar a realização do objetivo declarado, esse projeto vai identificar os pontos de maior latência da arquitetura atual, especificar mudanças na arquitetura para correção dos pontos de maior latência e desenvolver ferramentas a fim de possibilitar a utilização da nova arquitetura pelos membros do MACRO com os dispositivos já disponíveis em seus laboratórios.

Capítulo 2

Revisão, Especificação e Implementação

Este capítulo revisa a arquitetura atual de aquisição de dados da IMU. Identifica seus principais problemas, inclusive os que podem causar alta latência nos sistemas. Baseado nos problemas identificados é então proposta uma nova arquitetura para melhorar as dificuldades da arquitetura anterior. Por fim, é feita uma implementação de um novo sistema de aquisição de dados da IMU seguindo a nova arquitetura. Essa implementação possibilita a utilização da nova arquitetura pelos membros do MACRO.

2.1 Problemas na Arquitetura Atual

A arquitetura considerada (figura 1.1) apresenta grande dependência do MATLAB para comunicação com os dispositivos utilizados pelo MACRO. O MATLAB é um ambiente ideal para prototipagem rápida de algoritmos, porém tem desempenho abaixo do ideal quando utilizado para comunicação com processos e dispositivos externos. Na arquitetura atual o MATLAB é responsável por se comunicar com três dispositivos externos; a IMU 3DM-GX2, o manipulador AX-18 e a base móvel Create; e um processo externo, o cliente ROS do Microsoft Kinect. Um dos motivos para utilização do MATLAB é a dependência da IMU nesta plataforma. Não existem ferramentas em outras plataformas para aquisição de dados do modelo de IMU disponível no MACRO.

Uma arquitetura centralizada no MATLAB é inapropriada. O MATLAB foi criado primariamente para computação numérica e não oferece um conjunto específico de serviços, ferramentas e padrões para concepção e manutenção de sistemas para robótica, os quais necessitam primordialmente de conectividade de baixa latência entre suas diversas partes. Quanto à latência, a própria natureza dinâmica da linguagem utilizada no MATLAB contribui para aumentar o tempo de execução de bibliotecas escritas para o mesmo. Além disso, à medida que os sistemas crescem a tendência é utilizar mais dispositivos e processos externos. Isso tudo contribui para uma maior latência num sistema dependente do MATLAB.

De acordo com os próprios membros do MACRO, o desempenho é significantemente superior quando utilizado apenas códigos em C ou C++ num projeto. A latência nesses projetos geralmente fica no mínimo uma ordem de magnitude menor quando comparado ao projeto que utiliza MATLAB. Devido a isso, um requisito para uma nova arquitetura é a integração eficiente com programas escritos em outras linguagens, especialmente linguagens compiladas de maior eficiência como C e C++.

2.2 Nova Arquitetura

A nova arquitetura deve utilizar de um *framework* específico para desenvolvimento de *software* para robótica. Em desenvolvimento de *software,* um *framework* é um conjunto de ferramentas, serviços e padrões que provê uma infraestrutura para o desenvolvedor. Essa infraestrutura oferece funcionalidades para permitir que desenvolvedores dediquem seu tempo a atender requisitos de seu projeto ao invés de implementar funcionalidades mais básicas essenciais. Nesse caso é esperado um *framework* que provenha funcionalidades para facilitar a integração de múltiplos dispositivos e processos independentes.

Utilizar um *framework* já estabelecido permitirá utilizar de ferramentas desenvolvidas por outros desenvolvedores. Dado a complexidade e multidisciplinaridade do desenvolvimento de *software* para robótica, muitos desenvolvedores escolhem por utilizar de *frameworks* bem estabelecidos e do desenvolvimento em código aberto para conseguirem suporte da comunidade de desenvolvedores. Ao utilizar um *framework* popular os pesquisadores do MACRO terão acesso a diversos desses trabalhos. Além disso diversas empresas de sensores, manipuladores e bases móveis disponibilizam pacotes de programas para os *frameworks* mais populares.

A escolha mais sensata de *framework* para o MACRO é o ROS. A maior vantagem do ROS é que ele já é utilizado por alguns membros do MACRO, o que facilita a curva de aprendizado para adesão de uma nova arquitetura utilizando o ROS como peça principal. Além disso o ROS é um *framework* maduro, está em desenvolvimento desde 2007; versátil, está sendo utilizado em desde robôs humanoides a veículos aéreos não tripulados; popular e sem custo.

O ROS também é ideal para uma arquitetura de baixa latência. Este *framework* não adiciona custo significativo para a comunicação entre seus diversos processos (nós), pois as conexões estabelecidas são ponto-a-ponto, o ROS agindo apenas como um serviço para estabelecer a conexão entre os nós e padronizando as mensagens. Então, por exemplo, dado um nó A que publica mensagens no tópico T, quando o nó B se inscreve no tópico T o ROS não faz um roteamento das mensagens de A para B pelo seu processo, ao invés disso ele provê a B as informações necessárias para que este possa estabelecer uma conexão direta com A, salvando assim o tempo de propagação da mensagem pelo processo mestre do ROS.

A figura 2.1 ilustra de maneira simplificada as mensagens trocadas para o anúncio e inscrição de um tópico. Primeiramente o nó A anuncia o seu tópico para o ROS, nessa mensagem o nó A também envia as informações de rede necessárias para que outro processo encontre-o. Quando o nó B requisita ao ROS a inscrição no tópico de A (2), o ROS responde com as informações de rede recebidas de A (3). Então B negocia a conexão com A (4) e a conexão ponto-a-ponto é estabelecida entre A e B (5).

Outra possibilidade para reduzir a latência utilizando o ROS é fazer implementações em linguagens mais eficientes. O ROS oficialmente suporta bibliotecas para implementação de nós em C++, Python e Lisp. Logo a biblioteca em C++ pode ser utilizada para implementação de componentes de desempenho crítico, enquanto outras partes do programa podem ser implementadas em linguagens de maior nível de abstração como Python. Também é possível utilizar MATLAB, pois a *Mathworks* distribui uma *toolbox* para implementação de clientes ROS. Outras linguagens também têm bibliotecas não oficiais para implementações de clientes ROS.

A nova arquitetura proposta está ilustrada na figura 2.2. Nessa arquitetura se utiliza do

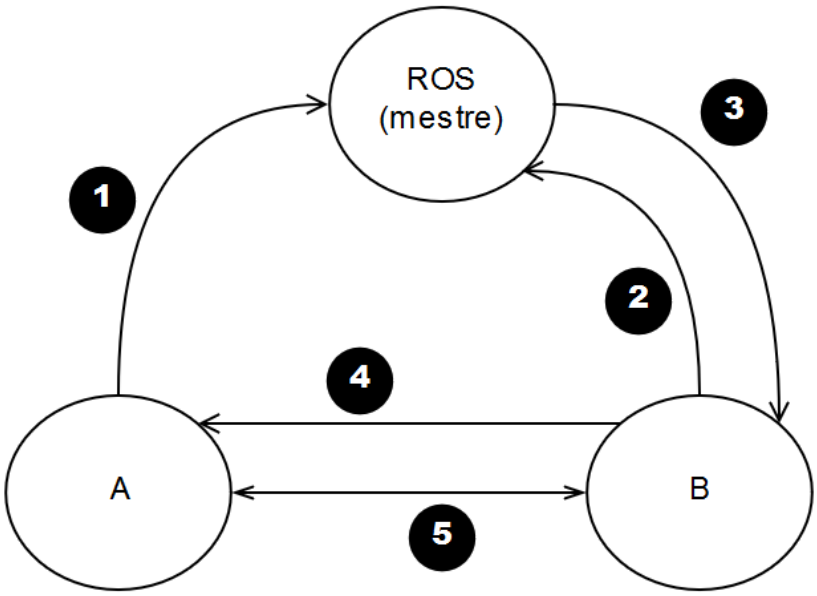

Figura 2.1: Processo simplificado para anúncio e inscrição de um tópico.

framework do ROS para realizar toda a comunicação entre o computador e dispositivos. Idealmente todos os dispositivos já teriam uma implementação eficiente de nó para ROS pronta para ser utilizada pelos pesquisadores. Estes somente escolheriam a linguagem ideal para seu projeto e implementaria o seu nó próprio que consumiria as informações e serviços disponibilizados pelos nós dos dispositivos.

Para ter todos os dispositivos da arquitetura com um cliente ROS disponível seria necessário a implementação somente de um nó para a IMU 3DM-GX2 (sem fio). Todos os outros dispositivos, o AX-18, iRobot Create e Microsoft Kinect, têm implementações para ROS distribuídas livremente como listado a seguir.

AX-18 *dynamixel_motors*[1]
dynamixel_workbench[2]

Create *create_autonomy*[3]
turtlebot_create[4]
roomba_robot[5]

Kinect *freenect_stack*[6]
openni_kinect[7]
kinect_aux[8].

[1]Disponível em http://wiki.ros.org/dynamixel_motor
[2]Disponível em http://wiki.ros.org/dynamixel_workbench
[3]Disponível em http://wiki.ros.org/create_autonomy
[4]Disponível em http://wiki.ros.org/turtlebot_create
[5]Disponível em http://wiki.ros.org/roomba_robot
[6]Disponível em http://wiki.ros.org/freenect_stack
[7]Disponível em http://wiki.ros.org/openni_kinect
[8]Disponível em http://wiki.ros.org/kinect_aux

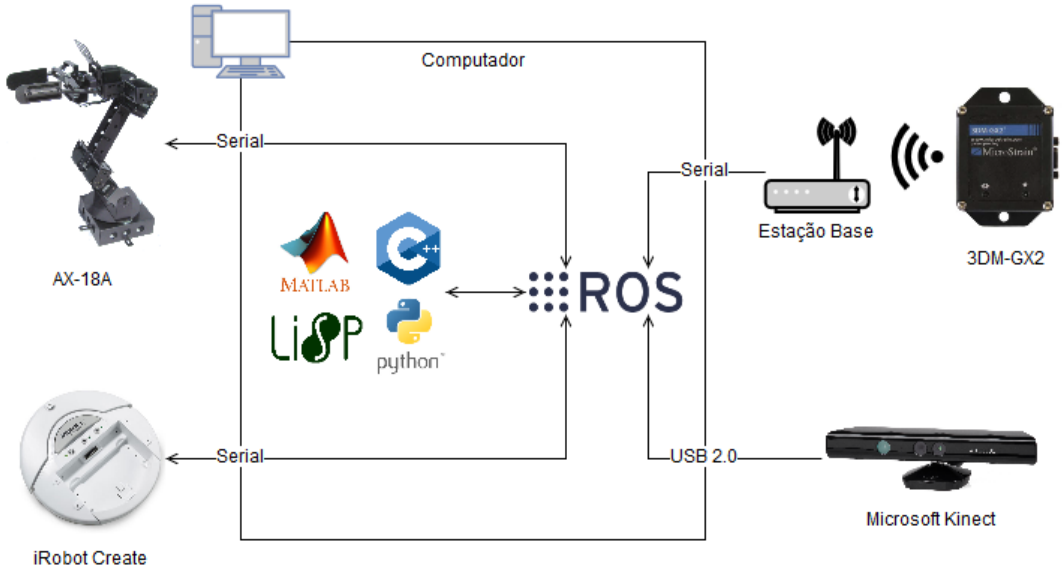

Figura 2.2: Nova arquitetura proposta.

2.3 Implementação para IMU

A versão sem fio da IMU 3DM-GX2 não tem um cliente ROS livremente disponível. Para que seja possível aos membros do MACRO a utilização da nova arquitetura proposta neste trabalho, será feita para a IMU uma implementação de um cliente em C++ utilizando a biblioteca oficial do ROS (*roscpp*).

Os requisitos principais desse cliente ROS são:

- Disponibilizar dados da IMU no tópico `/imu/raw_data`
- Disponibilizar dados da IMU no formato padrão definido pelo ROS (sensor_msgs/Imu)
- Utilizar a biblioteca *roscpp* para maior eficiência

O desenvolvimento do programa foi dividido em duas grandes partes. A primeira é uma biblioteca para processar os pacotes originados da 3DM-GX2 e recebidos por uma interface serial da estação base. E a segunda, que é o uso da biblioteca desenvolvida na primeira parte e a *roscpp* para implementação do nó para ROS.

2.3.1 Protocolo da IMU 3DM-GX2

Na versão sem fio da IMU 3DM-GX2, a troca de mensagens é feita através de uma estação base conectada ao computador através de uma interface serial (MicroStrain, 2010). O uso da estação base é transparente, então por simplicidade as trocas de mensagens serão referidas como trocas entre o PC e a IMU. Outra diferença da versão sem fio é que os comandos devem ser encapsulados num pacote antes de serem enviados para IMU e desempacotados ao serem recebidos no computador. As tabelas 2.1 e 2.2 mostram a estrutura desses pacotes.

Essa IMU tem diversos comandos disponíveis, a maioria deles consistem de apenas um byte e alguns outros precisam de mais argumentos. Para cada comando é esperada uma resposta de tamanho e formato conhecido da IMU. Como exemplo veja a tabela 2.3 e 2.4 para

Nome	**Bytes**	**Valor**
Início do pacote (SOP)	1	0xAA
Flag de entrega	1	0x0B
Tipo de dado	1	0x00
Endereço da IMU	2	Endereço da IMU que receberá este pacote
Tamanho do comando	1	Tamanho do comando + 1
Byte extra sem uso	1	0x00
Comando	1-100	Comando que a IMU executará
Checksum	2	Soma de todos os bytes exceto o SOP

Tabela 2.1: Estrutura dos pacotes de comando (do PC para a IMU)

Nome	**Bytes**	**Valor**
Início do pacote (SOP)	1	0xAA
Flag de entrega	1	0x07
Tipo de dado	1	0x00
Endereço da IMU	2	Endereço da IMU que enviou essa resposta
Tamanho da resposta	1	Tamanho da resposta + 1
Byte extra sem uso	1	0x00
Resposta	1-101	Resposta enviada pela IMU
LQI	1	Qualidade da conexão
RSSI	1	*Received signal strength indication*
Checksum	2	Soma de todos os bytes exceto SOP, LQI e RSSI

Tabela 2.2: Estrutura dos pacotes de dados (da IMU para o PC)

Nome	Bytes	Valor
Comando	1	0xCC

(a) Comando 0xCC

Nome	Bytes	Valor
Comando	1	0xCC
Bytes 1-4	4	Aceleração X em g (float)
Bytes 5-8	4	Aceleração Y em g (float)
Bytes 9-12	4	Aceleração Z em g (float)
Bytes 13-16	4	Velocidade angular X em rad/s (float)
Bytes 17-20	4	Velocidade angular Y rad/s (float)
Bytes 21-24	4	Velocidade angular Z rad/s (float)
Bytes 25-28	4	Magnetômetro X em G (float)
Bytes 29-32	4	Magnetômetro Y em G (float)
Bytes 33-36	4	Magnetômetro Z em G (float)
Bytes 37-40	4	Matriz de orientação (0;0) (float)
Bytes 41-44	4	Matriz de orientação (0;1) (float)
Bytes 45-48	4	Matriz de orientação (0;2) (float)
Bytes 49-52	4	Matriz de orientação (1;0) (float)
Bytes 53-56	4	Matriz de orientação (1;1) (float)
Bytes 57-60	4	Matriz de orientação (1;2) (float)
Bytes 61-64	4	Matriz de orientação (2;0) (float)
Bytes 65-68	4	Matriz de orientação (2;1) (float)
Bytes 69-72	4	Matriz de orientação (2;2) (float)
Timer	4	*Timestamp* do momento de execução do comando
Checksum	2	Soma de todos os bytes precedentes

(b) Resposta para comando 0xCC

Tabela 2.3: Exemplo de comando de um byte (constante). Comando e resposta esperada para obtenção da aceleração, taxa de variação angular, vetor do magnetômetro e matriz de orientação.

um exemplo de comando constante (1 byte) e um exemplo de comando variável, respectivamente. Os comandos podem ser usados em um modo de *polling* ou em um modo contínuo. No modo de *polling,* para cada comando enviado pelo PC a IMU executará aquele comando uma única vez. Já no modo contínuo a IMU irá executar o comando requisitado de maneira contínua, sem mais nenhuma interferência do PC sendo necessária. O modo contínuo pode ser interrompido com o comando 0xFA.

Notas: Todas as quantidades devem ser transmitidas e serão recebidas na ordenação *big endian*.

Pontos flutuantes (float) são 32-bit no formato IEEE-754.

A soma do *checksum* tem *rollover* de 65535 para 0.

Nome	Bytes	Valor
Comando	1	0xC4
Byte 1	1	0xC1
Byte 2	1	0x29
Comando contínuo	1	Byte do comando que será executado no modo contínuo

(a) Comando 0xC4

Nome	Bytes	Valor
Comando	1	0xC4
Comando contínuo	1	Byte do comando que será executado no modo contínuo
Timer	4	*Timestamp* do momento de execução do comando
Checksum	2	Soma de todos os bytes precedentes

(b) Resposta para comando 0xC4

Tabela 2.4: Exemplo de comando de múltiplos bytes (variável). Comando e resposta esperada para ativar o modo contínuo da 3DM-GX2.

prefixo	Descrição
CMD_	*Macro* do primeiro byte do comando.
SIZE_	*Macro* do tamanho total do comando em bytes.
init_	Função para inicializar uma sequência de bytes como um comando.
parse_	Função para retirar dados de uma resposta.

Tabela 2.5: Descrição dos prefixos utilizados na biblioteca.

2.3.2 Biblioteca para Montagem e Análise de Pacotes

A biblioteca de criação e análise de pacotes para a 3DM-GX2 foi criada para a montagem de pacotes a serem enviados e para a análise dos pacotes recebidos através da interface serial. Essa biblioteca foi feita em C sem utilização de alocação dinâmica de memória para tentar alcançar o menor tempo de execução possível.

O principal tipo de dados da biblioteca é uma simples sequência de bytes (`uint8_t[]` ou `unsigned char[]`) e as principais funções operam sobre estes bytes para criar pacotes, inicializar comandos ou retirar dados de respostas (veja tabela 2.5). Também existem *macros* úteis para alocar estaticamente o tamanho de uma sequência de bytes e para ajudar a empacotar comandos e desempacotar respostas. O motivo para trabalhar com sequências de bytes é fazer o mínimo de transformações nos dados, desde a sua alocação até o envio pela interface serial e também no processamento das mensagens recebidas.

O fluxo usual de como utilizar a biblioteca para montar um pacote de comando é dado a seguir com um exemplo prático da montagem de um pacote para o comando que ativa o modo contínuo na 3DM-GX2 (tabela 2.4):

1. Declarar sequência de bytes do tamanho necessário para o pacote de comando. A *macro* `I3DMGX2_CMDP_SIZE()` calcula o tamanho do pacote necessário com base no tamanho do comando. Já as *macros* com prefixo `SIZE_*` definem os tamanhos para cada um dos comandos existentes.

```
uint8_t cmdpack[I3DMGX2_CMDP_SIZE(SIZE_SET_CONTINUOUS)];
```

2. Inicializar o comando do pacote. Utilize as funções com prefixo `init_*` ou, caso seja um comando de um byte, utilizar as macros com prefixo `CMD_*`. Para calcular a posição de memória do comando de dentro de um pacote utilize a *macro* `I3DMGX2_PAYLOAD_PTR()`.

```
init_set_continuous_mode(I3DMGX2_PAYLOAD_PTR(cmdpack),
                         CMD_ACC_ANGR_MAGV_ORIENT);
```

3. Inicializar outras variáveis do pacote de comando. A função `i3dmgx2_init_cmdp()` realiza esta tarefa.

```
i3dmgx2_init_cmdp(cmdpack, 95, SIZE_SET_CONTINUOUS);
```

4. Ao final o pacote de comando está pronto para ser enviado à IMU de endereço 95. Isso pode ser feito utilizando alguma API para acesso à porta serial da estação base. Como referência é dado um exemplo utilizando a API nativa de sistemas Unix (`write()`) e a função de conveniência distribuída juntamente com essa biblioteca `i3dmgx2_open_port()`, a qual cria um *file descriptor* para acesso à porta serial passada por parâmetro.

```
int fd = i3dmgx2_open_port("/dev/ttyUSB0");
write(fd, cmdpack, sizeof(cmdpack));
```

Seguindo o exemplo anterior, o fluxo para extrair os dados da resposta enviada pela IMU é dado a seguir:

1. Receba os dados enviados pela IMU utilizando alguma API para acesso à porta serial da estação base. Como referência é dado um exemplo utilizando a API nativa de sistemas Unix (`read()`).

```
uint8_t buffer[255];
int nbytes = read(fd, buffer, sizeof(buffer));
```

2. Procurar no *buffer* de dados recebidos pela interface serial por um pacote recebido. Para isso utilize a função `i3dmgx2_parse_buffer()`. Esta função retorna 0 caso encontre um pacote no *buffer*.

```
uint8_t *pack;
size_t pack_size;
int result;
result = i3dmgx2_parse_buffer(buffer, nbytes,
                              &pack, &pack_size);
```

3. Extraia as informações da resposta com as funções com prefixo `parse_*`. Para descobrir de qual comando é a resposta do pacote utilize a *macro* `I3DMGX2_CMD()`. Para calcular a posição de memória da resposta de dentro de um pacote utilize a *macro* `I3DMGX2_PAYLOAD_PTR()`. As funções com prefixo `parse_*` retornam um valor inteiro diferente de zero para indincar um erro.

```
if (result == 0) {
  if (I3DMGX2_CMD(pack) == CMD_SET_CONTINUOUS) {
    struct set_continuous_resp record;
    int error = parse_set_continuous(
                    I3DMGX2_PAYLOAD_PTR(pack),
                    I3DMGX2_PAYLOAD_LEN(pack),
                    &record);
  }
}
```

Ao final, as informações estarão disponíveis na estrutura `record`. Para mais detalhes sobre cada função, veja os arquivos de cabeçalho da biblioteca.

2.3.3 Cliente ROS

O cliente ROS é o responsável por obter continuamente dados da IMU de orientação, aceleração e velocidade angular e publicar estas informações no tópico `/imu/raw_data`. Para maior facilidade de integração com outras ferramentas, a mensagem publicada deve seguir a especificação padrão do ROS para mensagens de IMU (sensor_msgs/IMU). Os itens principais desta especificação são a orientação como um quatérnio unitário, um vetor de 3 dimensões com as velocidades angulares em rad/s e um vetor de 3 dimensões com as acelerações lineares em m/s^2.

A figura 2.3 mostra o fluxograma do cliente desde sua inicialização até a publicação de mensagens no ROS. Após iniciada a conexão com a IMU ela é configurada no modo contínuo com um comando que proverá os dados necessários para formação da mensagem a ser publicada. Depois disso o programa deve passar a receber dados continuamente pela porta serial, caso isso não ocorra existe um *timeout* para alertar o usuário que algo está errado. Ao receber os dados, estes são colocados em um montante (*buffer*) o qual pode conter dados de uma mensagem parcialmente recebida anteriormente. Após isso o *buffer* é analisado e todas suas mensagens válidas são processadas, publicadas no tópico do nó e removidas do montante. Ao final de todas as mensagens processadas é fechado um ciclo com o programa voltando ao processo de receber dados pela interface serial. O cliente permanecerá nesse ciclo publicando todas as mensagens recebidas da IMU.

A mensagem padrão do ROS para uma IMU utiliza um quatérnio unitário para representar sua orientação. A IMU 3DM-GX2 calcula internamente sua orientação utilizando uma matriz de rotação e é nesse formato também que a orientação é enviada nas suas mensagens. Então, para a formação da mensagem no formato padrão do ROS a partir da mensagem original da IMU, é necessário converter uma matriz de rotação para um quatérnio unitário. Para obter um quatérnio unitário de uma matriz de rotação é utilizado o Algoritmo de Shepperd (Shepperd, 1978) conforme apresentado por Landis Markley (Markley, 2008). Seja a matriz de rotação

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

e o quatérnio unitário dado por

$$\mathbf{q} = q_1 i + q_2 j + q_3 k + q_4,$$

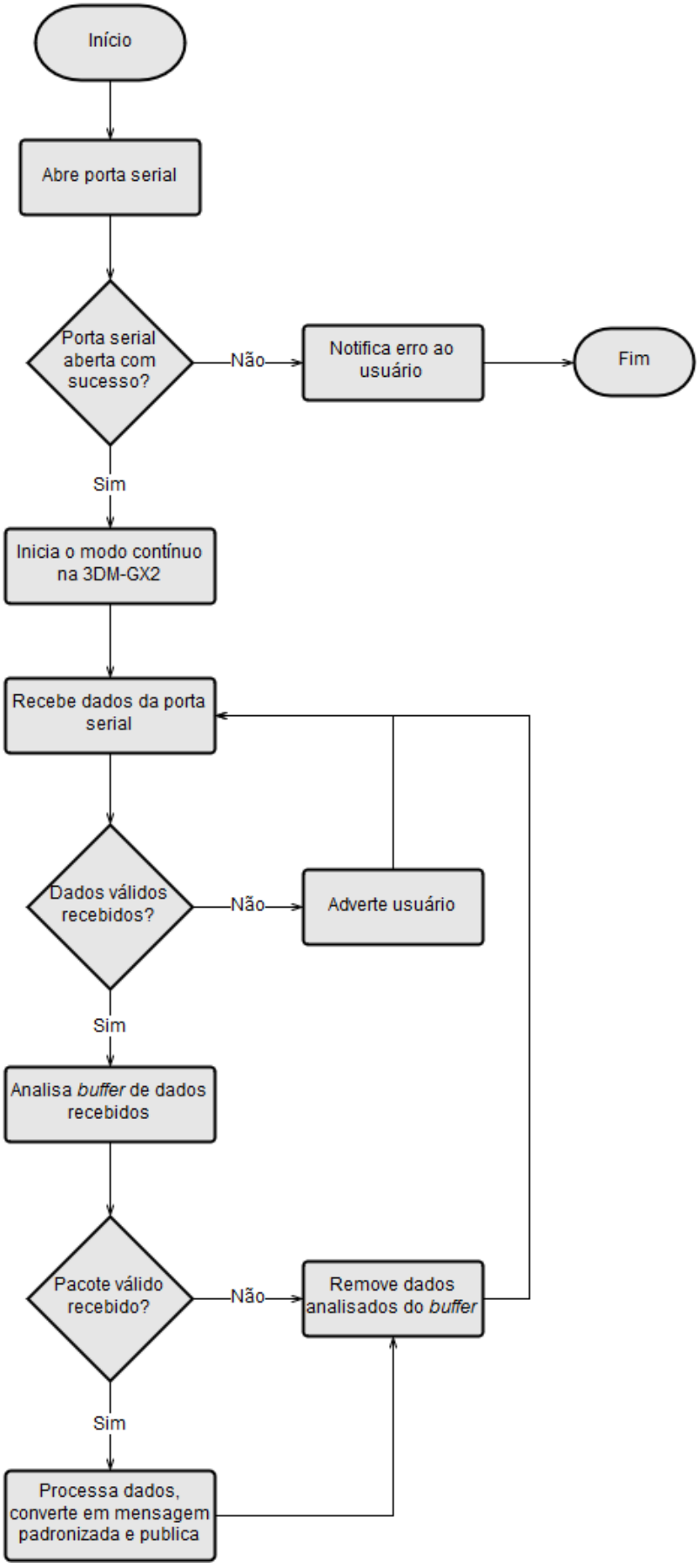

Figura 2.3: Fluxograma do funcionamento do nó da IMU para o ROS.

então os componentes do quatérnio podem ser calculados com as seguintes equações:

$$q_i = \pm\frac{1}{2}\left(1 + A_{ii} - A_{jj} - A_{kk}\right)^{1/2},$$

$$q_j = \left(A_{ij} + A_{ji}\right)/4q_i,$$
$$q_k = \left(A_{jk} + A_{ki}\right)/4q_i,$$
$$q_4 = \left(A_{jk} - A_{kj}\right)/4q_i.$$

Onde $\{i, j, k\}$ é uma permutação cíclica de $\{1, 2, 3\}$.

Para implementação desse cliente ROS foi utilizada a API Unix para leitura e escrita em portas seriais, a biblioteca para criação e análise de pacotes da 3DM-GX2 desenvolvida nesse projeto e a biblioteca de implementação de nós para ROS *roscpp*. As funções `read()` e `write()` da API dos sistemas Unix são utilizadas respectivamente para leitura e escrita na porta serial. A biblioteca da 3DM-GX2 foi utilizada para estabelecer a conexão inicialmente com a IMU e extrair os dados das mensagens recebidas. A *roscpp* é utilizada para anunciar o tópico `/imu/raw_data` e também para publicar as mensagens neste tópico.

O programa está organizado como um pacote ROS. Pacotes ROS são conjuntos de nós, bibliotecas, definições de mensagens e qualquer outra utilidade para estender as funcionalidades do ROS. Esse pacote foi nomeado de `i3dmgx2` e ele conta apenas com o nó `i3dmgx2_node` que é o cliente ROS descrito nesta seção. Esse pacote foi criado com a ferramenta `catkin_create_pkg` e pode ser compilado com `catkin_make` em um *workspace* `catkin` de pacotes ROS. Após compilado, para rodar o `i3dmgx2_node` é necessário passar dois argumentos, o primeiro é a porta em que a estação base está conectada e o segundo é o endereço da IMU que se deseja monitorar. Por exemplo, o comando para conectar na porta `/dev/ttyUSB0` e contactar a IMU de endereço 95 seria

```
rosrun i3dmgx2 i3dmgx2_node /dev/ttyUSB0 95.
```

Após isso o nó começa automaticamente a publicar as mensagens da IMU no tópico `/imu/raw_data`.

2.3.4 Desempenho

Os testes realizados nesta seção foram realizados com Ubuntu 14.04 (64-bit) e ROS Indigo em um computador com processador Intel Core i5-2450M e 6 GB de memória RAM. Os tempos de execução para a implementação para ROS foram calculados através da função `gettimeofday` disponível em sistemas Unix. No MATLAB os tempos foram calculados com as funções `tic` e `toc` nativas. A IMU estava configurada com uma taxa de saída de dados igual a 100 Hz.

O cliente ROS implementado apresentou um desempenho muito superior quando comparado com a implementação em MATLAB. Os tempos médios, máximos e mínimos para receber e processar uma mensagem entre essas implementações estão exibidos na tabela 2.6. O tempo de execução do cliente ROS corresponde ao recebimento de uma mensagem e sua publicação. Isso inclui a análise dos dados recebidos pela interface serial (*parsing* dos dados) e a conversão da mensagem nativa da IMU para a mensagem padrão do ROS (sensor_msgs/IMU). Já para a implementação em MATLAB o tempo corresponde somente ao recebimento da mensagem via porta serial e *parsing* dos dados, já que não é necessário

Mensagens	Implementação	max.	médio	min.
1600	ROS	1,16 ms	0,99 ms	0,20 ms
	MATLAB	78,43 ms	33,12 ms	24,74 ms

Tabela 2.6: Testes realizados para mensurar o tempo de execução a partir do recebimento da mensagem da IMU até a chamada para publicação da mensagem.

nenhuma conversão de mensagens. O tempo do cliente ROS foi consistentemente mensurado abaixo de 1,16 ms, enquanto que o tempo médio do MATLAB é um pouco superior a 33 milissegundos, o que torna essa implementação mais de 30 vezes mais lenta quando comparada ao tempo médio do ROS. Ainda no MATLAB, no terceiro experimento foi mensurado um tempo máximo muito superior à média, esse comportamento é indesejável pois pode causar uma degradação do desempenho de sistemas robóticos. Tal comportamento não foi detectado na implementação para ROS.

Com um tempo de execução máximo de 1,16 ms, o cliente ROS poderia publicar mais de 862 mensagens por segundo caso a IMU seja capaz de enviar mensagens com essa frequência. Como, de acordo com o *datasheet* da IMU[9], esta é capaz de transmitir até 100 mensagens por segundo, o desempenho alcançado foi considerado mais que satisfatório.

2.3.5 Verificação

Dados adquiridos com o ROS e MATLAB foram comparados para verificar a qualidade das informações processadas com a nova implementação. Para aquisição dos dados a IMU foi colocada em três posições:

1. Eixo X da IMU alinhado com vetor estático da aceleração da gravidade (figuras 2.4 e 2.5);

2. Eixo Y da IMU alinhado com vetor estático da aceleração da gravidade (figura 2.6);

3. Eixo Z da IMU alinhado com vetor estático negativo da aceleração da gravidade (figura 2.7).

O posicionamento da IMU nestas posições foi feito com auxilio do RVIZ, uma ferramenta do ROS para fazer visualizações. A figura 2.8 mostra uma captura de tela dessa ferramenta para a IMU na 2ª posição da lista acima.

Para cada posição foi coletado os dados de aceleração linear e velocidade angular da IMU. As figuras 2.4, 2.6 e 2.7 mostram a aceleração linear para cada eixo coletado por cada programa. Já as informações de velocidade angular foram muito similares entre as posições, o que era esperado já que a IMU estava em repouso. Por conta disso, é necessário apenas um gráfico para mostrar o comportamento da velocidade angular entre as implementações (figura 2.5), note que mesmo a IMU estando em repouso os sinais não têm média em zero, isso ocorre devido a imperfeições do giroscópio da IMU, o qual gera um pequeno sinal mesmo em completo repouso (*gyroscope bias*). Com uma análise qualitativa dos gráficos de todas essas figuras, pode-se concluir que a nova implementação feita para ROS está processando

[9]MicroStrain, 3DM-GX2 Datasheet. Disponível em http://files.microstrain.com/3dm-gx2_datasheet_v1.pdf

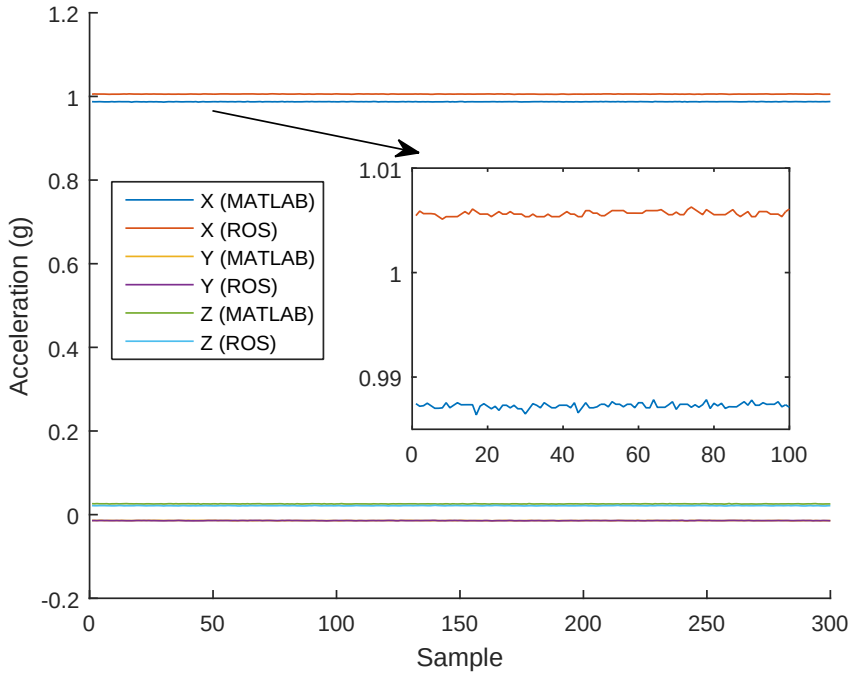

Figura 2.4: Dados de aceleração linear para IMU com eixo X alinhado com vetor estático da aceleração da gravidade.

os dados corretamente, pois os gráficos dos dados do ROS apresentam forma muito similar à dos gráficos dos dados do MATLAB. Algumas pequenas diferenças são criadas apenas pela dificuldade de colocar a IMU na exata mesma posição entre cada experimento para coleta de dados.

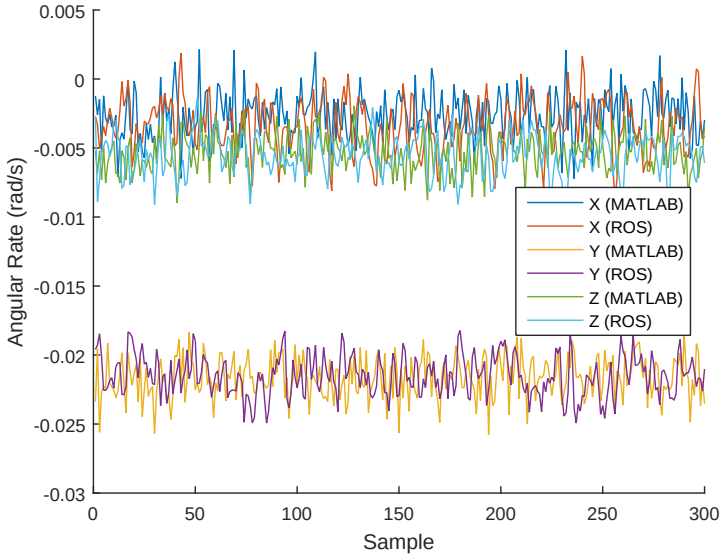

Figura 2.5: Dados de velocidade angular para IMU com eixo X alinhado com vetor estático da aceleração da gravidade.

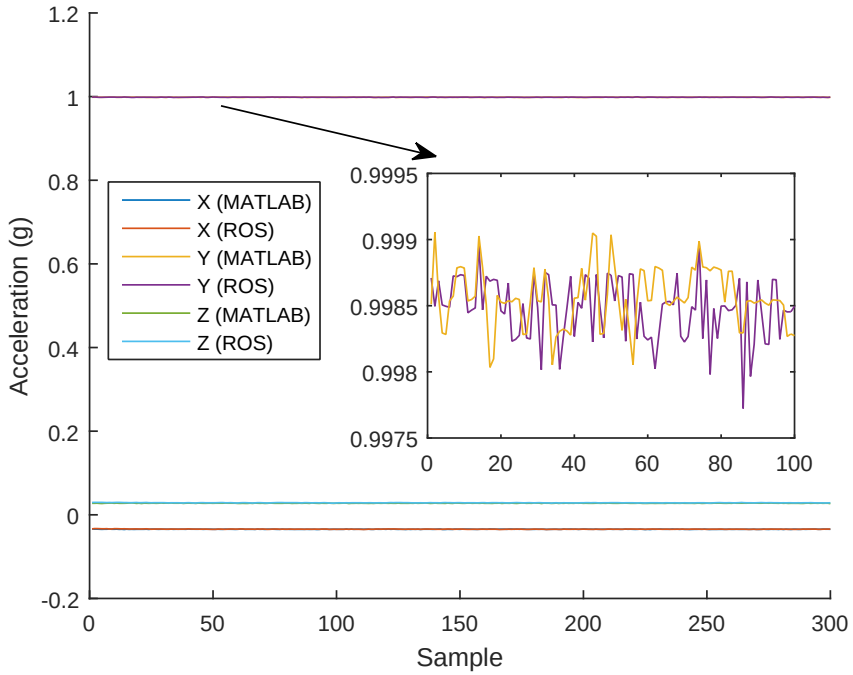

Figura 2.6: Dados de aceleração linear para IMU com eixo Y alinhado com vetor estático da aceleração da gravidade.

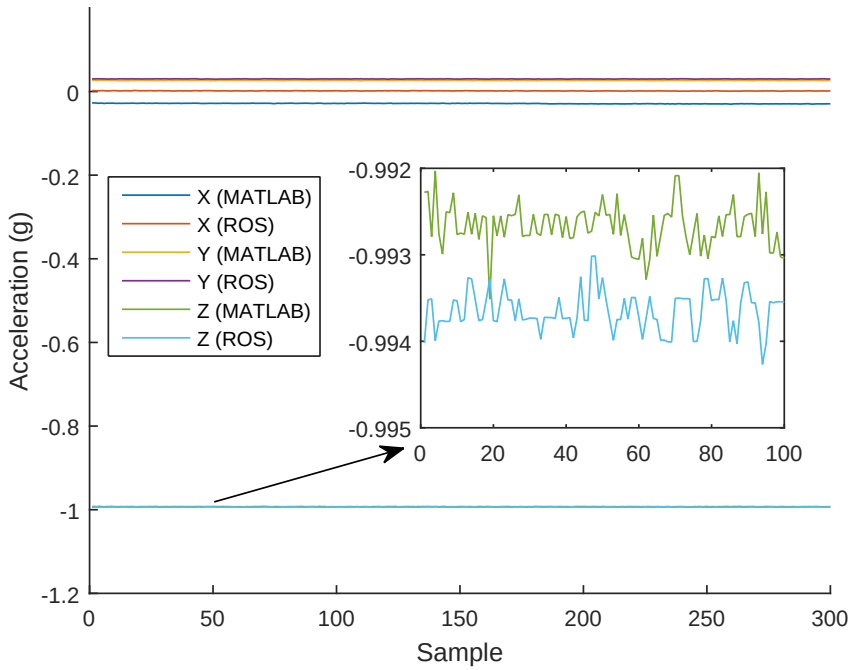

Figura 2.7: Dados de aceleração linear para IMU com eixo Z alinhado com vetor estático negativo da aceleração da gravidade.

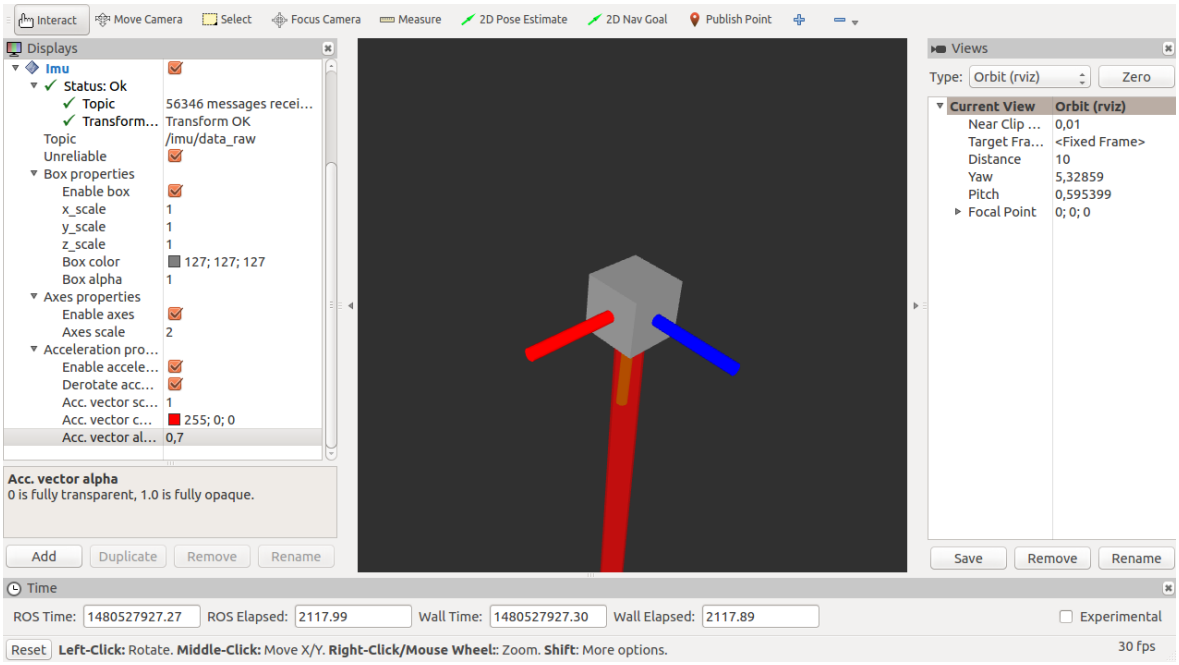

Figura 2.8: Captura da tela do RVIZ com o posicionamento da IMU com seu eixo Y (em verde) alinhado ao vetor estático da aceleração da gravidade (em vermelho transparente).

Capítulo 3

Conclusões

Esse estudo detalhou o problema da alta latência na execução de experimentos na área de robótica no contexto de aquisição de dados de uma unidade de medição inercial. Foram discutidas as consequências causadas pela alta latência no contexto de experimentos, as quais são lentidão, movimentos não naturais e instabilidade. Para contextualizar o problema, foi detalhada uma arquitetura utilizada pelo grupo MACRO em experimentos que utilizavam a IMU sem fio 3DM-GX2. Esse detalhamento listou cada componente, como eles estão integrados, explicou o caminho dos dados na arquitetura e por fim definiu o conceito de latência utilizado por esse trabalho.

Foi proposta uma nova arquitetura para a aquisição de dados da IMU pelos membros do MACRO. Essa arquitetura permite a realização de experimentos de alto desempenho com o uso do ROS, um *framework* para desenvolvimento de software para robótica, como peça central. O uso dessa nova arquitetura também possibilita a utilização de diversos pacotes para ROS distribuídos livremente na internet e torna mais fácil a integração de outros códigos feitos em diferentes linguagens de programação.

Para possibilitar a utilização da nova arquitetura pelos membros do MACRO, foi feita uma implementação de um cliente de ROS para a IMU 3DM-GX2. Entre os dispositivos considerados, este era o único o qual não havia nenhuma implementação já disponível para o ROS. Essa implementação foi feita tendo como prioridade a redução do tempo de execução do código, o que resultou em um programa de desempenho satisfatório para as taxas de transmissão da IMU e muito superior quando comparado a única implementação existente anteriormente, feita em MATLAB. Por fim foi verificada a qualidade dos dados processados pela nova implementação (ROS). Para isso foi feita uma comparação qualitativa dos dados obtidos por meio do ROS e do MATLAB. A comparação mostrou resultados similares entre os experimentos utilizando ROS e MATLAB, o que validou o processamento dos dados da nova implementação.

Também foi analisado os aspectos sociais, econômicos e culturais relacionados aos objetivos de diminuição da latência em sistemas para robótica. No geral não são esperados impactos negativos com a realização direta deste projeto ou da redução da latência no contexto deste trabalho.

Apêndice A

Aspectos Sociais, Econômicos e Culturais

Nas seções anteriores foi definido o que é a alta latência no contexto desse trabalho e como ela pode afetar o desempenho de demonstrações de técnicas de controle da área da robótica. Nesta seção será discutido como a diminuição da latência e a realização do objetivo deste trabalho, a viabilização de experimentos de alto desempenho, podem impactar na sociedade, economia e cultura.

A motivação para realização desta discussão é consequência do grande potencial de transformação social atrelado a projetos de engenharia (Filho et al., 2011). Esses projetos muitas vezes apresentam a capacidade para impactar positivamente ou negativamente na vida das pessoas e no meio ambiente. Como exemplo pode-se citar projetos de estradas, que podem tanto ligar duas regiões anteriormente inacessíveis, criando oportunidades de crescimento cultural e econômico, quanto podem desmatar, destruir e poluir por onde passam. Outro exemplo são projetos de rede de comunicação, esses podem tanto garantir a liberdade e expressão individual, quanto permitir a disseminação de ideologias extremistas. Assim, para garantir que o desenvolvimento e uso deste trabalho seja feito de forma ética e humana, julga-se necessário fazer uma análise dos impactos sociais, econômicos e culturais do mesmo.

A.1 Produção Científica

A realização desse projeto viabilizará a produção de experimentos de alto desempenho. O grupo MACRO será o primeiro beneficiado já que o trabalho está sendo desenvolvido dentro do grupo. Porém o objetivo é abrir os códigos e distribuí-los livremente para a comunidade de desenvolvedores da área de robótica. Com isso espera-se que esse projeto possa vir a ser útil a outros pesquisadores, salvando-os o tempo de implementação de ferramentas de uso comum como nós para ROS ou bibliotecas para MATLAB.

Espera-se também como resultado desse projeto o aumento da acessibilidade à produção de experimentos de alto desempenho. Essa acessibilidade viria como consequência do desacoplamento das áreas de conhecimento, de forma que um pesquisador não iria necessitar do conhecimento de como implementar uma arquitetura de software de baixa latência para que possa então conseguir desenvolver um experimento de alto desempenho. No caso, espera-se que o pesquisador possa escrever seu experimento utilizando a arquitetura e ferramentas desenvolvidas nesse trabalho e consequentemente ele obtenha um alto desempenho no seu experimento.

A demonstração de uma técnica de controle com um experimento de alto desempenho

pode melhor demonstrar a real capacidade dessa técnica para aplicações práticas. Acredita-se que essa melhor exibição do potencial da técnica pode diminuir o tempo entre a sua criação no laboratório e seu emprego em aplicações de robótica de uso doméstico ou industrial. A exemplo de como um experimento de baixo desempenho pode erroneamente demonstrar a capacidade da técnica, é ao exibir uma demonstração para leigos e estes expressarem que aquilo não pode ter utilidade por conta da velocidade de execução. Isso pode ser um problema em projetos de pesquisa financiados por empresas ou governos onde os pesquisadores devem regularmente demonstrar os resultados de seu trabalho para pessoas não-especialistas. Nesse caso é de grande importância que uma demonstração de melhor desempenho seja desenvolvida.

De uma forma geral este projeto tem a possibilidade de viabilizar ou facilitar o desenvolvimento de experimentos de alto desempenho por pesquisadores da área de robótica. Esses experimentos podem ser importantes para garantir o financiamento de pesquisas e também para demonstrar viabilidade de aplicações práticas em robótica. Algumas aplicações práticas de relevância social e econômica que são vislumbradas são aplicações de robótica assistiva, aplicações para melhorias das condições de trabalho e aplicações para aumento da produtividade de indústrias. Essas aplicações e seus impactos são discutidos nas próximas seções.

A.2 Interação Humano-Robô

Como já discutido na seção 2.2 o alto desempenho é essencial para aplicações que tenham interação entre homem e robô. Nesse tipo de aplicação o usuário espera que o robô reaja de forma instantânea e contínua a seus comandos, caso isso não ocorra o usuário terá uma experiência frustrante, podendo até levar a situações de risco a integridade física das pessoas próximas ao robô. Por exemplo, caso um robô receba um comando para exercer uma ação e durante essa ação uma pessoa se coloque em risco entrando na sua trajetória, o robô deve ter um tempo de reação rápido para corrigir sua trajetória e não colidir com a pessoa. Esse tempo de reação rápido só é alcançável com baixa latência, pois os sinais de controle são gerados somente após a amostragem e processamento das variáveis.

A.2.1 Robótica Assistiva

A robótica assistiva é a área da robótica que trabalha para suplementar as capacidades motoras e sensoriais de uma pessoa (Miller, 1998). Exemplos comuns de áreas de aplicações da robótica assistiva são no auxílio à mobilidade e no auxílio à manipulação de objetos. Devido a natureza de proximidade das interações dos robôs com as pessoas, é crítico para este tipo de aplicação a aceitabilidade dos usuários. Entre outros critérios, acredita-se que a aceitabilidade dos usuários pode ser alcançada com a apresentação de respostas rápidas em conjunto com movimentos fluidos e naturais por esse tipo de aplicação. E, para alcançar esse nível de desempenho, é de grande importância uma arquitetura de baixa latência.

A área de auxílio à mobilidade é a área que ajuda as pessoas com dificuldade de locomoção a se locomoverem para outros lugares ou na locomoção de algum objeto para o local onde a pessoa se encontra. De acordo com o Censo de 2010 do IBGE, no Brasil mais de 3,6 milhões de pessoas apresentam grande dificuldade de locomoção, enquanto que 734,4 mil se declararam totalmente incapazes de caminhar ou subir escadas. Já a área de auxílio à manipulação ajuda as pessoas com dificuldades motoras severas a realizarem tarefas básicas por

conta própria, por exemplo, abrir portas, beber líquidos de um recipiente (Lana et al., 2013) e se alimentar. No Brasil, 4,4 milhões de pessoas declararam ter uma deficiência motora severa no Censo de 2010.

As causas de problemas motores e de mobilidade nas pessoas são geralmente causados por deficiências congênitas ou adquiridas, doenças do sistema nervoso, acidentes, ou pela idade avançada (idosos). Como o número de idosos tende a crescer ao passo do aumento das expectativas de vida no Brasil e no mundo, espera-se também um aumento de pessoas com essas dificuldades. Visto os números atuais e a tendência de crescimento do número de pessoas com dificuldades de locomoção e motoras, pode-se afirmar que a robótica assistiva tem potencial para melhorar a vida de um grande número de pessoas tanto hoje quanto no futuro.

A.2.2 Melhores Condições de Trabalho

Robôs capazes de trabalhar próximos a pessoas de forma segura podem possibilitar melhores condições de trabalho. Hoje em dia robôs industriais são capazes de executar tarefas repetitivas ou de grande exigência física. Porém, a maioria desses robôs não podem operar em colaboração com uma pessoa, a razão disso é o risco em que a pessoa coloca sua integridade física ao entrar na área de trabalho dos robôs. Para um robô trabalhar com segurança junto a pessoas este deve ser capaz de detectar quando estiver prestes a colidir com alguém e reagir para impedir essa colisão em um pequeno instante de tempo. Uma das limitações técnicas para implementação de soluções desse problema é o tempo de reação do robô que está diretamente relacionado à latência desses sistemas.

Com a realização de robôs capazes de trabalhar em ambientes compartilhados com pessoas, espera-se abrir possibilidades de automação de menor escala. Indústrias menores, comércios e até escritórios poderiam, sem alocações de espaços exclusivos para robôs, utilizar de robôs para automatizar tarefas de grande desgaste físico, que envolvem riscos ou repetitividade para os seus trabalhadores (Autor, 2015). Essa automação tem o potencial para reduzir a fadiga física, lesões por esforço repetitivo e acidentes de trabalho, consequentemente criando melhores condições de trabalho para as pessoas.

A.3 Produtividade

A latência de um sistema também está relacionada com o quão rápido este sistema pode realizar uma tarefa. Um sistema de controle com menor latência pode ter seu controlador configurado com um maior ganho e consequentemente convergir com maior velocidade para a referência. Logo, um sistema com menor latência pode executar uma tarefa em menor tempo quando comparado a um mesmo sistema com maior latência. Então, pode-se concluir que reduzir o período de amostragem viabiliza a redução do tempo de execução de tarefas.

Na indústria, com robôs mais rápidos na execução de tarefas tem-se maior produtividade. Maior produtividade nesse contexto significa produzir um maior volume em um menor tempo. A maior produtividade das indústrias podem levar à redução de preços dos produtos finais para os consumidores, consequentemente expandindo o mercado para esse produto e gerando crescimento econômico (Brynjolfsson & McAfee, 2014).

No caso em que o mercado não é expansível podem haver ganhos ambientais. Por exemplo, considere uma indústria com duas linhas de produção, se essa indústria dobra sua produ-

tividade, mas o mercado não absorve um volume maior que o anterior, essa indústria poderá desativar um das linhas de produção e provavelmente ocupar a metade do espaço que antes ocupava. Com uma menor área ocupada por indústrias, se torna possível ter na cidade mais áreas para habitação, comércio, lazer ou áreas verdes, cada uma trazendo consigo seus respectivos benefícios para a cidade.

Referências Bibliográficas

Autor, D. H. (2015). Why are there still so many jobs? the history and future of workplace automation. *Journal of Economic Perspectives*, 29(3), 3–30. A.2.2

Brynjolfsson, E. & McAfee, A. (2014). *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*. W. W. Norton. A.3

Filho, N. G. d. S., de Santana, J. G. L., & da Silva, L. R. B. (2011). A responsabilidade social na vida de um engenheiro. In *XXXIX Congresso Brasileiro de Educação em Engenharia*. A

Lana, E. P., Adorno, B. V., & Tierra-Criollo, C. J. (2013). Assistance task using a manipulator robot and user kinematics feedback. In *XI Simpósio Brasileiro de Automação Inteligente*. A.2.1

Markley, F. L. (2008). Unit quaternion from rotation matrix. *Journal of guidance, control, and dynamics*, 31(2), 440–442. 2.3.3

MicroStrain (2010). *3DM-GX2 Data Communications Protocol*. MicroStrain Inc., Williston, VT, 119 edition. 2.3.1

Miller, D. P. (1998). *Assistive robotics: An overview*, (pp. 126–136). Springer Berlin Heidelberg: Berlin, Heidelberg. A.2.1

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3 (pp. 1–5).: Kobe, Japan. 1.1

Shepperd, S. W. (1978). Quaternion from rotation matrix. *Journal of Guidance and Control*, 1(3), 223–224. 2.3.3