UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

Ronan Fim Largura

Inserção de uma plataforma robótica móvel no espaço inteligente do laboratório Viros

Vitória-ES

Dezembro/2018

Ronan Fim Largura

Inserção de uma plataforma robótica móvel no espaço inteligente do laboratório Viros

Parte manuscrita do Projeto de Graduação do aluno Ronan Fim Largura, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Vitória-ES

Dezembro/2018

Ronan Fim Largura

Inserção de uma plataforma robótica móvel no espaço inteligente do laboratório Viros

Parte manuscrita do Projeto de Graduação do aluno Ronan Fim Largura, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovado em 14 de Dezembro de 2018
COMISSÃO EXAMINADORA:

Felippe Mendonça de Queiroz
Universidade Federal do Espírito Santo
Orientador

Raquel Frizera Vassallo

Profa. Dra. Raquel Frizera Vassallo
Universidade Federal do Espírito Santo
Coorientadora

Mariana Rampinelli Fernandes

Profa. Dra. Mariana Rampinelli Fernandes
Instituto Federal do Espírito Santo
Examinadora

Profa. Dra. Roberta Lima Gomes
Universidade Federal do Espírito Santo
Examinadora

Vitória-ES

Dezembro/2018

RESUMO

Neste trabalho será descrita a metodologia para a inserção da plataforma robótica desenvolvida pelo aluno Murilo Leonardelli Daltio do laboratório Viros no Espaço Inteligente deste, com o uso de uma placa *Raspberry Pi 2B* atuando como elemento intermediário entre o espaço e o controlador de baixo nível da plataforma. Serão descritas também uma tarefa de controle de posicionamento e outra de controle de trajetória, ambas realizadas para testar o funcionamento do robô dentro do espaço inteligente.

Palavras-chave: *Raspberry Pi*; Espaço Inteligente; Plataforma robótica; Robô.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

Ufes	Universidade Federal do Espírito Santo
Viros	*Vision and Robotics Systems* - Visão e Sistemas Robóticos
RAM	*Random Access Memory* - Memória de Acesso Aleatório
CPU	*Central Processing Unit* - Unidade Central de Processamento
RPC	*Remote Procedure Call* - Chamada de Procedimento Remoto
protobuf	*Protocol Buffers*

SUMÁRIO

1 INTRODUÇÃO

Frequentemente, observam-se muitos projetos de engenharia do mundo real que foram inspirados em histórias de ficção científica. Alguns sistemas originalmente retratados nessas obras, que na época eram ficção, tornaram-se reais com o avanço da tecnologia, tais como robôs e foguetes espaciais (LEE; HASHIMOTO, 2002). No romance Ubik, escrito em 1969 por Philip K. Dick, é descrita uma sociedade capaz de interagir com ambientes e dispositivos inteligentes de maneira natural, por meio de gestos e fala. Baseado nesse romance, Weiser especulou, em 1998, que no futuro as pessoas estariam tão imersas na tecnologia que interagiriam com ela naturalmente, tal como a sociedade descrita por Philip K. Dick (RAMPINELLI et al., 2014).

Visando criar um ambiente mais próximo do imaginado no romance Ubik, no qual a tecnologia está onipresente na vida das pessoas de modo imperceptível, criou-se o conceito de espaço inteligente, que pode ser definido como um ambiente equipado com vários sensores que permitem que o espaço perceba e compreenda o que está acontecendo nele (LEE; HASHIMOTO, 2002).

Todo esse esforço em tornar os ambientes mais tecnológicos tem, na grande maioria das vezes, o objetivo final de melhorar a qualidade de vida dos seres humanos que os habitam. Os robôs móveis vem assumindo ao longo dos anos o papel de auxiliadores dos seres humanos, ajudando-os em diversas tarefas, como por exemplo, transporte de cargas e condução de pessoas.

Para executar todas essas funções, um robô necessita obter informações do ambiente a sua volta. Um robô isolado no espaço conta apenas com as informações provenientes de seus próprios sensores, o que limita a compreensão do espaço ao seu redor à quantidade e à qualidade de seus sensores (MORIOKA; LEE; HASHIMOTO, 2008). Porém, se um robô estiver inserido dentro do espaço inteligente, ele pode ter acesso a todos os dados sensoriais daquele espaço, incluindo até mesmo dados coletados por outros robôs, possibilitando-o executar tarefas de um modo mais eficiente, podendo, inclusive, ampliar o escopo de atuação do mesmo. Além disso, a inserção de mais robôs no espaço permite a realização de tarefas mais complexas que envolvem múltiplos robôs trabalhando em sintonia, como por exemplo o transporte de uma carga de grandes dimensões por dois ou mais transportadores.

Para que os dispositivos inseridos no espaço inteligente consigam realizar essas tarefas, é necessário que eles se comuniquem de alguma maneira entre si e com o espaço. Somente por meio dessa comunicação é que o espaço poderá enviar comandos aos dispositivos inseridos nele e receber os dados coletados por eles. Existem diversas tecnologias que podem ser

utilizadas para realizar essa comunicação, como o *Bluetooth*, o *LoRa* e o *WiFi*.

O espaço inteligente que será discutido nesse trabalho é o espaço do laboratório Viros (Visão e Sistemas Robóticos), localizado no segundo andar do prédio CT-II da Ufes. Esse espaço possui 7,5 metros de comprimento, 4,8 metros de largura e é sensoriado por quatro câmeras BFLY-PGE-09S2C-CS da empresa *Point Grey*. Nesse espaço existe um *middleware* de comunicação e uma padronização na troca de mensagens que faz com que o espaço se comunique da mesma maneira com diferentes dispositivos. Nesse espaço, realizam-se diversas pesquisas em robótica móvel, em que os robôs inseridos dentro do mesmo são detectados por ele e recebem comandos de controle via tecnologia *WiFi*.

Entre os robôs existentes no laboratório há um *Pioneer P3-AT*, que já foi inserido com sucesso no espaço, e dois robôs menores que foram construídos pelo aluno Murilo Leonardelli Daltio durante um projeto de iniciação científica. Cada um desses dois robôs menores possui odometria, quatro sensores de distância ultrassônicos modelo HC-SR04 e um módulo WiFi modelo ESP8266 que foi adicionado com o objetivo de estabelecer uma comunicação com o espaço inteligente do laboratório. Apesar de o robô funcionar muito bem navegando autonomamente, foram encontradas dificuldades ao inserir o robô no espaço inteligente do laboratório, devido à alta latência apresentada pelo módulo WiFi utilizado. Além do problema com o ESP8266, a placa controladora adotada no projeto desse robô não permite o uso de sensores mais complexos, como câmeras ou sensores de varredura a laser, que podem ser inseridos no robô futuramente.

Com o objetivo de resolver os problemas supracitados será detalhado nesse trabalho a metodologia utilizada para a inserção desses robôs no espaço inteligente do laboratório Viros, realizando-se as modificações necessárias para adequar a plataforma robótica à arquitetura atual.

1.1 Objetivos

1.1.1 Objetivo geral

Inserir a plataforma robótica desenvolvida pelo aluno Murilo Leonardelli Daltio no espaço inteligente do laboratório Viros, na Ufes, realizando as modificações necessárias para adequá-la à arquitetura atual do espaço.

1.1.2 Objetivos específicos

- Estudar a documentação do robô e compreender a sua construção e seu código.
- Escolha das ferramentas, *hardware* adicional e protocolos de comunicação a serem utilizados.
- Implementar a comunicação entre o *hardware* inserido e o *Arduino Mega* presente na plataforma.
- Estabelecer de alguma maneira a comunicação do robô com o espaço inteligente do laboratório via *WiFi*.
- Estudar o espaço inteligente do laboratório, seu modelo de comunicação e os códigos já implementados nele.
- Estudar as padronizações já existentes para o robô *Pioneer P3-AT* para que sirvam de referência para a implementação a ser realizada para a plataforma robótica em questão.
- Com a finalidade de validação, executar um algoritmo de controle de posição final e de seguimento de trajetória, oferecido por um serviço já existente no espaço inteligente do laboratório Viros e que já é utilizado pelo robô *Pioneer P3-AT*.

1.2 Estrutura do Texto

A parte escrita desse Projeto de Graduação está dividida em cinco capítulos, descritos a seguir:

- **Introdução:** Capítulo inicial que tem como objetivo contextualizar o trabalho e apresentar o objetivo desse Projeto de Graduação.
- **Referencial Teórico:** Apresenta, de forma resumida, todo o embasamento teórico dos principais conceitos abordados no desenvolvimento nesse trabalho.
- **Implementação Desenvolvida:** Nesse capítulo é detalhado todo o trabalho realizado, com todas as implementações de *hardware* e *software* que foram feitas para cumprir com o objetivo do trabalho.
- **Experimentos e Resultados:** Apresenta os resultados dos experimentos de controle de posição final e controle de trajetória em oito, utilizados para validar a inserção do robô no espaço inteligente.

- **Conclusão e Trabalhos Futuros:** Esse capítulo conclui o trabalho com base nos resultados expostos e informa possíveis melhorias futuras que podem ser realizadas.

2 REFERENCIAL TEÓRICO

2.1 Espaços Inteligentes

Espaços inteligentes podem ser descritos como ambientes equipados com vários sensores (câmeras e microfones, por exemplo) e atuadores (robôs, telas, entre outros) que possibilitam interagir com as pessoas e realizar tarefas dentro desse ambiente. Além disso, os sensores e atuadores no espaço inteligente estão sujeitos a um sistema supervisor que analisa as informações dos sensores, compreendendo o que está acontecendo no ambiente e tomando decisões com base nessas observações (RAMPINELLI et al., 2014).

Diversos trabalhos sobre espaços inteligentes tem sido desenvolvidos em várias partes do mundo. Na Universidade de Alcalá (UAH), na Espanha, por exemplo, foi criado um espaço inteligente onde foi desenvolvido um trabalho visando encontrar a postura de um robô móvel com o uso de uma rede de câmeras instalada no espaço (PIZARRO et al., 2010). Em Jinan, na China, criou-se um protótipo de ambiente residencial inteligente para comando de um robô de tarefas domésticas, com o objetivo de melhorar a qualidade de vida de idosos e pessoas com alguma incapacidade física (LU et al., 2012).

Na Universidade Federal do Espírito Santo (Ufes), (RAMPINELLI et al., 2014) criou uma estrutura de espaço inteligente com 11 câmeras, um robô móvel e um microcomputador distribuídos em dois laboratórios com o objetivo de identificar, localizar e controlar robôs móveis dentro do espaço. Como pode ser observado nos exemplos citados, os dispositivos robóticos são bastante explorados dentro do conceito de espaço inteligente, estando presentes em diversas aplicações.

No trabalho de (RAMPINELLI et al., 2014) os dispositivos do espaço se comunicavam por uma LAN (*Local Area Network*), onde as câmeras e o microcomputador eram conectados em um *switch* e o robô móvel se conectava a essa rede por *WiFi*. Essa comunicação era essencial no espaço inteligente pois permitia que os dados das câmeras e do robô chegassem aos microcomputadores, além de permitir que o robô recebesse comandos de velocidade. Um problema desse trabalho é que não havia uma padronização bem definida caso fosse necessário adicionar mais dispositivos ao espaço, como mais câmeras ou robôs. A padronização na forma de comunicação é importante para que a inserção de novos elementos no ambiente inteligente possa ocorrer sem a necessidade de uma reprogramação completa.

Uma forma de padronizar a comunicação em ambiente inteligentes é utilizando protocolos de comunicação, que vão tornar a estrutura da comunicação mais organizada e melhorar a

escalabilidade do sistema, facilitando a inserção de novos dispositivos no espaço.

2.2 Comunicação em ambientes inteligentes

Existem diversos tipos diferentes de protocolos de comunicação que podem ser utilizados para prover a comunicação entre o espaço inteligente e seus dispositivos. O protocolo HTTP (do inglês *HyperText Transfer Protocol*), por exemplo, utiliza o padrão *Request/Response*, em que o cliente envia uma mensagem de requisição de um recurso e o servidor envia uma mensagem de resposta ao cliente com a solicitação (POZZEBOM, 2007). Já existem, atualmente, microcontroladores com interfaces de comunicação com a Internet que possibilitam, por exemplo, a criação de um servidor HTTP para coletar dados de um sensor (QUEIROZ, 2016).

No trabalho de (LU et al., 2012) foi utilizado o protocolo *ZigBee* para transferir os dados sensoriais do ambiente, como temperatura e umidade, e enviar comandos para os diversos dispositivos inseridos no espaço inteligente. O *ZigBee* é um protocolo de comunicação sem fio que ganhou bastante na popularidade nas aplicações em Internet das Coisas (IoT) devido ao seu baixo consumo energético. Porém, a baixa taxa de transmissão associada a esse protocolo é um fator que normalmente limita seu uso a aplicações envolvendo pouca quantidade de dados.

Entre os diversos tipos de protocolos existentes há aqueles baseados em mensagens. Esses protocolos geralmente possuem um elemento chamado *broker*, que possui a função de receber e encaminhar as mensagens ao destinatário correto. Nesse tipo de comunicação, para enviar uma mensagem é necessário apenas informar o endereço do *broker* e uma identificação do destinatário da mensagem (QUEIROZ, 2016). A forma como essa informação chegará ao destinatário final é gerenciada pelo *broker*, facilitando para os dispositivos que irão enviar e receber as mensagens e concentrando os problemas da comunicação em um único elemento.

Os protocolos de comunicação baseados em mensagens podem trabalhar com o padrão *publisher/subscriber* (*pub/sub*). Nesse padrão, os responsáveis por enviar as mensagens, chamados de *publishers* não as enviam diretamente a destinatários específicos. Ao invés disso, as mensagens são divididas em categorias, de acordo com os seus conteúdos. Aqueles que irão receber as mensagens, chamados de *subscribers*, assinam os conteúdos que têm

interesse e recebem mensagens dessas categorias. Assim, os produtores das mensagens não têm conhecimento de quem irá recebê-las, já que são os destinatários que escolherão quais mensagens desejam receber.

A seleção de quais mensagens os *subscribers* irão receber pode ser feita em dois modelos: baseado em tópicos ou baseado em conteúdo. No modelo baseado em tópicos, as mensagens são publicadas em tópicos específicos e os subscribers receberão as mensagens dos tópicos que eles assinarem. Já no modelo baseado em conteúdo, as mensagens apenas são entregues aos *subscribers* se o conteúdo das mesmas se encaixar nas restrições definidas pelos próprios *subscribers*. Alguns sistemas suportam ainda uma versão híbrida dos dois modelos, onde as mensagens são selecionadas pelo conteúdo para os *subscribers* que estão inscritos em determinados tópicos.

Existem diversos protocolos de comunicação que implementam o padrão de *publisher/subscriber*, como o AMQP (*Advanced Message Queuing Protocol*), o MQTT (*Message Queuing Telemetry Transport*) e o DDS (*Data Distribution System*). Cada um desses protocolos tem suas particularidades e suas vantagens e desvantagens. O AMQP, por exemplo, permite que haja interoperabilidade entre diferentes tecnologias e plataformas de modo confiável e seguro. Esse será o protocolo de comunicação utilizado nesse trabalho e, portanto, serão fornecidos mais detalhes sobre ele na próxima seção.

2.2.1 AMQP

O AMQP (do inglês, *Advanced Message Queuing Protocol*) é um protocolo de comunicação baseado em mensagens assíncrono. Isso significa que após enviar uma mensagem não se espera uma resposta imediata, o que traz como vantagem a possibilidade de enviar mensagens mesmo quando o destinatário não está disponível para recebê-las no momento.

Contudo, para que esse tipo de comunicação funcione é necessário haver um elemento central que receba as mensagens enviadas e as direcione aos destinatários corretos (SIMAS, 2015). No AMQP esse elemento central é o *broker*, que executa, portanto, um papel fundamental dentro do protocolo. Com o *broker* é possível monitorar o fluxo de mensagens e persisti-las, ou seja, manter a mensagem armazenada até o destinatário ficar disponível para recebê-la, além de possibilitar que diferentes aplicações consigam se comunicar de uma maneira mais fácil.

Porém, uma desvantagem apresentada por esse tipo de arquitetura é que o *broker*, devido ao seu papel no sistema, pode concentrar um alto fluxo de mensagens sobre ele. Essa

inconveniência, no entanto, pode ser amenizada com a utilização de dois ou mais *brokers* em pontos diferentes da rede (QUEIROZ, 2016).

Existem diversos *softwares* que podem ser utilizados como *broker* para o AMQP. A seguir serão fornecidos mais detalhes a respeito do *RabbitMQ*, que é o *software* de *broker* utilizado nesse trabalho.

2.2.2 RabbitMQ

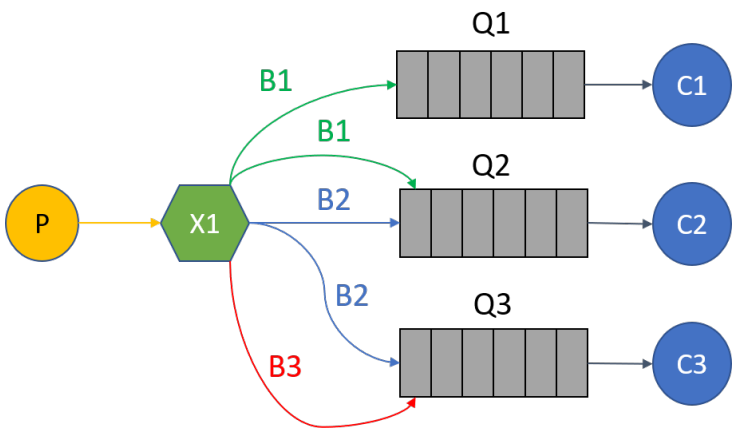
O RabbitMQ é um *software* em código aberto (*open source*) desenvolvido pela *Pivotal software* que funciona como *broker* para diversos protocolos de mensagens, tais como o AMQP, MQTT e STOMP, gerenciando o tráfego de mensagens dentro desses protocolos (BOSCHI; SANTOMAGGIO, 2013). Fazendo uma analogia com o mundo real, o RabbitMQ é como um serviço de correio que recebe uma carta e faz com que ela eventualmente chegue ao destinatário correto (Pivotal Software, 2018).

Para realizar esse serviço de entrega de mensagens, o RabbitMQ implementa filas internamente, onde as mensagens são armazenadas para serem encaminhadas aos destinatários. Além disso, como no padrão *publisher/subscriber* vários consumidores (*subscribers*) podem receber o mesmo dado de um produtor (*publisher*), é necessário haver elementos que encaminhem as mensagens dos produtores para as filas corretas. No RabbitMQ esses elementos são chamados de *exchanges* e ficam posicionados entre os produtores e as filas.

Entre os *exchanges* e as filas existem ainda regras de roteamento chamadas de *bindings*. Os *bindings* são necessários para que os *exchanges* saibam para qual fila eles devem encaminhar as mensagens recebidas. Uma fila pode ter mais de uma regra de roteamento e duas filas diferentes podem compartilhar a mesma regra de roteamento, o que faria a mensagem ser duplicada. Assim, uma mensagem enviada pelo produtor deve possuir, além de um *exchange*, um *routing_key*, que informará por qual *binding* a mensagem será enviada (QUEIROZ, 2016). Por exemplo, na Figura 1, a fila Q1 possui um *binding* B1, a fila Q2 possui dois *bindings*, B1 e B2, e a fila Q3 possui dois *bindings*, B2 e B3. Assim, uma mensagem enviada ao *broker* com o *exchange* X1 e *routing_key* B1 chegará aos consumidores C1 e C2, mas não chegará ao consumidor C3.

Um outro modelo que o RabbitMQ permite que seja implementado é o baseado em chamadas de procedimento remoto, ou RPC (do inglês *Remote Procedure Call*). Nesse modelo um cliente envia uma mensagem requisitando algum serviço a um servidor e o servidor retorna uma mensagem ao cliente após executar a chamada. A mensagem de

Figura 1 – Representação da comunicação entre um produtor e três consumidores, com um *exchange* e três *bindings*.

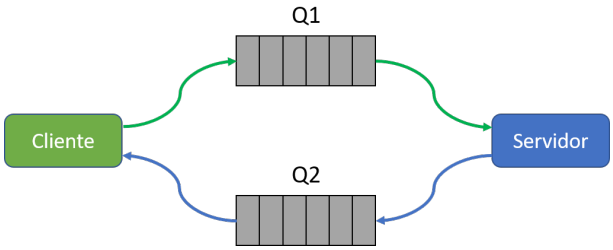


Fonte: Produção do próprio autor.

requisição enviada pelo cliente deve conter o nome da fila para a qual o servidor retornará a mensagem de resposta. Assim, o cliente saberá qual fila ele deve escutar para receber a resposta do servidor.

Como o protocolo é assíncrono, as mensagens de resposta do servidor podem não chegar na mesma ordem das mensagens de requisição do cliente. Assim, para que o cliente possa fazer a correlação entre as mensagens enviadas e recebidas, é enviado juntamente à mensagem um ID chamado de *correlation_ID*, que é inserido no cabeçalho do protocolo AMQP junto com a fila em que a resposta foi enviada *reply_to* (QUEIROZ, 2016). Na Figura 2 está representada uma comunicação entre cliente e servidor em uma implementação RPC.

Figura 2 – Representação de uma comunicação cliente/servidor em uma implementação RPC.



Fonte: Produção do próprio autor.

2.3 Padronização e Troca de Mensagens

Em sistemas de grande porte que utilizam protocolos de comunicação baseados em mensagens, a padronização das mesmas é fundamental, pois torna a estrutura de comunicação

mais organizada e mais fácil de ser compreendida. Consequentemente, a introdução de novos dispositivos na rede é facilitada, melhorando a escalabilidade do sistema. A padronização também permite que haja maior compatibilidade entre diferentes aplicações, pois para que a comunicação funcione basta que ambas as aplicações sigam a mesma padronização.

Uma ferramenta que auxilia na padronização de mensagens é a serialização. Por meio dela, diversos tipos de dados podem trafegar entre diferentes sistemas e aplicações, pois permite que estes conversem entre si em uma mesma linguagem.

2.3.1 Serialização

Serialização é um processo que permite transformar estruturas de dados ou objetos em uma sequência de bits para que possam ser armazenados ou transmitidos em um meio de comunicação com fio ou wireless (Standard C++ Foundation, 2015). O processo inverso também precisa ser garantido, ou seja, deve ser possível reconstruir o objeto original a partir de sua forma serializada, ainda que a reconstrução seja realizada em um dispositivo diferente do qual ele foi originalmente serializado.

A serialização se faz necessária para transmissão de dados porque é preciso estruturar os dados em uma sequência de bytes para que eles possam ser transmitidos (ARAÚJO, 2012). Além disso, as ferramentas de serialização de dados permitem que aplicações desenvolvidas em diferentes linguagem de programação possam se comunicar entre si.

Existem diversas maneiras diferentes de serializar dados, porém elas podem ser divididas em dois grandes grupos: serialização em formato de texto e serialização em formato binário. A serialização em formato de texto é legível para humanos e, portanto, mais fácil de depurar durante o desenvolvimento da aplicação. Já a serialização em formato binário não é legível para humanos e, portanto, mais difícil de depurar. Contudo, esse formato de serialização em geral consome menos ciclos de CPU (Standard C++ Foundation, 2015) sendo, assim, preferível em aplicações embarcadas com hardware limitado.

Dois exemplos famosos de métodos de serialização em formato de texto são o JSON (*JavaScript Object Notation*) e o XML (*eXtensible Markup Language*). Ambos os formatos são utilizados no compartilhamento de informações entre diferentes computadores e aplicações, porém o JSON se destaca em relação ao XML por ser mais leve e mais fácil de ser interpretado. Já entre os métodos de serialização em formato binário, tem-se como exemplos o *Protobuf* e o *Thrift*, sendo ambos métodos bem consolidados e amplamente

utilizados por empresas de grande porte, como *Google* e *Facebook*. O *protobuf* é o método de serialização que será aplicado nesse trabalho e, portanto, será detalhado na próxima seção.

2.3.2 Protocol Buffers

Protocol Buffers, também conhecido como *protobuf*, é um método de serialização de dados independente de linguagem e de plataforma criado pela *Google Inc* para uso em protocolos de comunicação, armazenagem de dados e diversas outras aplicações (Google Developers, 2018).

O *Protocol Buffers* foi criado inicialmente para uso interno da *Google*, porém a empresa decidiu posteriormente liberar o projeto para a comunidade oferecendo o gerador de código para diversas linguagens em licença de código aberto. A versão incial do *protobuf*, conhecida como *Proto1*, começou a ser desenvolvida pela *Google Inc* em 2001. Para que a ferramenta pudesse ser liberada para a comunidade foi necessário criar uma segunda versão (*Proto2*), que foi completamente reescrita visando a geração de um código mais limpo e independente das bibliotecas internas da *Google* (Google Developers, 2017). Atualmente, o *protobuf* está na sua terceira versão, conhecida como *Proto3*.

O método utilizado pelo *protobuf* consiste em especificar os dados a serem serializados em um arquivo *.proto*, que é então compilado para a linguagem desejada gerando arquivos de cabeçalho contendo as classes e os métodos utilizados no código para codificar e decodificar as mensagens (GLIGORIĆ; DEJANOVIĆ; KRČO, 2011). Entre as linguagens de programação que o *protobuf* oferece suporte estão as linguagens Java, C++, Python e C#, todas elas largamente utilizadas no desenvolvimento das mais diversas aplicações.

A Figura 3 mostra um exemplo de arquivo *.proto*. Esse arquivo estrutura duas mensagens. A primeira possui quatro campos, sendo um inteiro de 32 *bits* com sinal, um ponto flutuante, uma *string* e uma enumeração. Já a segunda mensagem possui dois campos, sendo um inteiro de 32 *bits* com sinal e uma mensagem do tipo definido anteriormente.

A primeira linha do código indica a sintaxe que está sendo utilizada (*proto2* ou *proto3*). No exemplo da Figura 3 está sendo utilizada a sintaxe da terceira versão do protobuf. A palavra *message* é utilizada para definir uma mensagem. A mensagem é semelhante a uma estrutura de dados, contendo um ou mais campos que podem ser de tipos escalares (tais como *int*, *float*, *double*, *string*, etc), enumerações, ou ainda de tipos pré-definidos anteriormente em outras mensagens. Cada um desses campos deve ser identificado por um

Figura 3 – Exemplo de arquivo *.proto*.

```
syntax = "proto3";

message Mensagem1 {
    int32 var1 = 1;
    float var2 = 2;
    string var3 = 3;
    enum Enum1 {
        op1 = 0;
        op2 = 1;
        op3 = 2;
    }
    Enum1 var4 = 4;
}

message Mensagem2 {
    int32 var5 = 1;
    Mensagem1 var6 = 2;
}
```

Fonte: Produção do próprio autor.

número que é único por mensagem (SOUZA, 2018).

Um problema do *protobuf* é que ele não é aplicável em sistemas embarcados e de memória restrita, por dificuldades como o tamanho do código gerado e a falta de suporte a linguagens de baixo nível. Para isso existem soluções que buscam implementar o *protobuf* nesse tipo de sistema, tais como o *NanoPB*.

2.3.3 NanoPB

Apesar do *protobuf* fornecer suporte para diversas linguagens e plataformas, é muito complicado utilizá-lo em microcontroladores ou sistemas de memória restrita. Isso ocorre pelo fato do *protobuf* não fornecer suporte à linguagem C, utilizada por grande parte dos compiladores de microcontroladores, e pelo tamanho do código gerado pelo *protobuf*, que ocupa uma porcentagem muito grande da memória limitada que esses dispositivos possuem.

O *nanopb* surgiu como uma possível solução para esses problemas. Ele é uma implementação do *protobuf* em ANSI-C que gera um código menor que o *protobuf* original, permitindo a utilização do *protobuf* em microcontroladores e sistemas de memória restrita (AIMONEN;

POOLE, 2018). Na página do *github* do *nanopb*[1] encontra-se a documentação do projeto, exemplos e instruções de como gerar as bibliotecas a partir dos arquivos *.proto*.

2.4 Hardware em sistemas embarcados

Os sistemas embarcados são caracterizados geralmente pela presença de um ou mais elementos microcontroladores ou microprocessadores responsáveis por gerenciar todo o funcionamento do sistema. Atualmente, existem diversas soluções de microprocessadores e microcontroladores que podem ser utilizadas em sistemas embarcados, como as diversas placas de prototipagem desenvolvidas pelas empresas *Arduino* e *Texas Instrument*. Entre essas placas, uma que tem ganhado destaque pelo seu alto poder de processamento e tamanho pequeno é a *Raspberry Pi*, que será detalhada na próxima seção.

2.4.1 A Raspberry Pi

A *Raspberry Pi* é um computador de baixo custo desenvolvido no Reino Unido pela fundação *Raspberry Pi* que reúne em uma placa do tamanho aproximado de um cartão de crédito um considerável poder computacional (UPTON; HALFACREE, 2012). Essa placa foi criada com o objetivo de possibilitar o ensino de computação e programação básica nas escolas e nos países subdesenvolvidos, porém ela pode ser utilizada em diversas aplicações eletrônicas que necessitam de um processador embarcado com alto processamento e de fácil utilização.

Atualmente existem vários modelos diferentes de *Raspberry Pi*. O Quadro 1 reúne as principais características de 4 modelos populares do produto.

Quadro 1 – Especificações de quatro modelos de *Raspberry Pi*

Especificações	**Raspberry Pi 3 Model B**	**Raspberry Pi Zero**	**Raspberry Pi 2 Model B**	**Raspberry Pi 1 Model B+**
Lançamento	2016	2015	2015	2014
System on Chip (SoC)	Broadcom BCM2837	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2835
CPU	Quad Cortex A53, 1.2GHz	ARM11, 1GHz	Quad Cortex A7, 900MHz	ARM11, 400MHz
GPU	400MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV
RAM	1GB SDRAM	512MB SDRAM	1GB SDRAM	512MB SDRAM
Portas USB	4	1 (Micro-USB)	4	4
Ethernet	10/100 Mbits/s	-	10/100 Mbits/s	10/100 Mbits/s
Bluetooth	4.1	-	-	-
WiFi	802.11n	-	-	-

Fonte: (TALIMAN, 2016)

[1] https://github.com/nanopb/nanopb

Existem diversos tipos de protocolos de comunicação que podem ser aplicados em sistemas embarcados com a *Raspberry Pi*. Muitos desses protocolos utilizam um formato de comunicação conhecido como serial, que será detalhado a seguir.

2.5 Comunicação Serial

A comunicação serial consiste em enviar dados bit a bit, de modo sequencial, por um canal de comunicação ou barramento. Devido a essa característica, esse tipo de comunicação utiliza geralmente um menor número de fios quando comparado com a comunicação paralela, onde vários bits são enviados simultaneamente.

Existem diversos protocolos de comunicação serial, os quais podem ser síncronos ou assíncronos. Nos protocolos síncronos existe um sinal de *clock* que possui a função de sincronizar a transmissão e a recepção informando quando o dado transmitido pode ser lido pelo receptor. Já nos protocolos assíncronos não existe esse tipo de sinal, de modo que é preciso pensar em outras estratégias para que a comunicação funcione corretamente.

Pode-se, também, dividir os protocolos em *full-duplex* (quando o protocolo permite que o dispositivo transmita e receba dados ao mesmo tempo), *half-duplex* (quando o dispositivo pode transmitir e receber dados, porém não ao mesmo tempo) e *simplex* (quando a comunicação é unidirecional apenas) (Robocore, 2018). No Quadro 2 são comparados diversos protocolos de comunicação serial.

Quadro 2 – Comparação entre protocolos de comunicação serial

Protocolo	**Sincronismo**	**Sentido de Transmissão**	**Taxa de comunicação**
UART	Assíncrono	Full-duplex	1200 a 115200 bps
SPI	Síncrono	Full-duplex	Até 10 Mbps
I2C	Síncrono	Half-duplex	100 a 400 kbps
RS 232	Assíncrono ou Síncrono	Full-duplex	1200 a 115200 bps
USB 2.0	Assíncrono	Half-duplex	Até 480 Mbps
USB 3.0	Assíncrono	Full-duplex	Até 4,8 Gbps

Fonte: (Robocore, 2018)

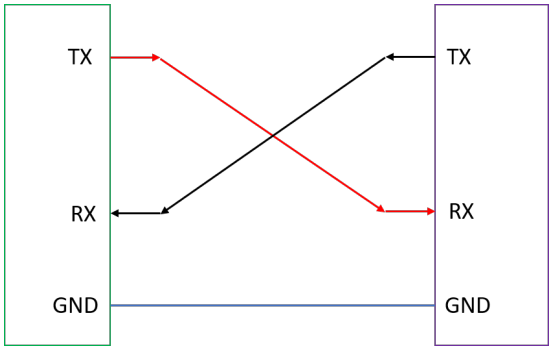
Entre os diversos protocolos de comunicação serial descritos no Quadro 2, o UART ganhou destaque pela sua simplicidade e possibilidade de transmitir e receber dados ao mesmo tempo (*Full-duplex*).

2.5.1 UART

O UART, do inglês *Universal Asynchronous Receiver Transmitter*, é um protocolo de comunicação serial assíncrono amplamente utilizado em aplicações caracterizadas por baixo custo, baixa velocidade e curtas distâncias (WAKHLE; AGGARWAL; GABA, 2012). A simplicidade desse protocolo e possibilidade de comunicar simultaneamente nos dois sentidos (*full-duplex*) também contribuíram para a sua popularidade.

Os dispositivos que se comunicam via protocolo UART possuem dois pinos: RX (terminal de recepção) e TX (terminal de transmissão). A sequência de bits a ser transmitida é enviada pelo pino TX do transmissor e recebida pelo terminal RX do receptor. Portanto, para a correta comunicação, é necessário que o terminal TX de um dispositivo seja conectado ao terminal RX de outro e vice versa, como mostra a Figura 4.

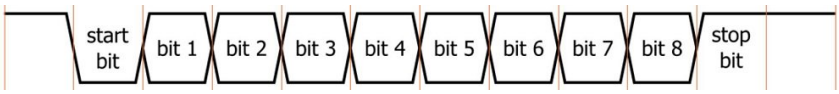
Figura 4 – Conexão entre dispositivos para comunicação serial.



Fonte: Produção do próprio autor.

A transferência de dados nesse protocolo funciona da seguinte forma: na ausência de dados, o canal de comunicação apresenta um nível lógico alto constantemente. Para iniciar a comunicação, é enviando um bit no nível lógico baixo, chamado *start bit*, e em seguida os bits de dados em série, do menos significativo ao mais significativo. A transmissão se encerra com um bit em nível lógico alto chamado de *stop bit* (FANG; CHEN, 2011). Pode, ainda, ser enviado um bit de paridade par ou ímpar, destinado à detecção de eventuais erros na comunicação. A Figura 5 mostra um exemplo de transmissão de 8 bits em série pelo protocolo UART.

Figura 5 – Transmissão de oito bits via UART.



Fonte: (KEIM, 2016)

Para saber o tempo de cada bit e manter o sincronismo entre a transmissão e a recepção na ausência de um sinal de *clock*, é necessário que os dois dispositivos que vão se comunicar estabeleçam uma taxa de transmissão em comum, chamada de *baud rate*, que pode chegar até 115200 bits por segundo.

3 IMPLEMENTAÇÃO DESENVOLVIDA

A motivação para a realização desse trabalho surgiu da necessidade de inserir dois robôs no espaço inteligente do laboratório Viros, localizado no segundo andar do prédio CT-II da Ufes. Os robôs em questão são do tipo tração diferencial e foram criados durante o projeto de iniciação científica do aluno Murilo Leonardelli Daltio. Os dois robôs são idênticos e cada um deles possui uma placa *Arduino Mega* atuando como controlador, encoders de rotativos e quatro sensores de distância ultrassônicos modelo HC-SR04, três na frente do robô e um atrás comandado por um servo. A inserção desses robôs no espaço inteligente pode trazer várias possibilidades interessantes de projetos futuros, como o mapeamento de áreas no espaço que as câmeras não conseguem alcançar, por exemplo.

Além disso, o robô também possui um módulo *WiFi* modelo ESP8266. Na época de seu desenvolvimento, tentou-se inserir esse robô no espaço inteligente do laboratório utilizando o ESP8266, porém, devido a alta latência de comunicação apresentada, verificou-se que não era possível realizar tarefas básicas de controle com o robô utilizando a infraestrutura do espaço inteligente.

Dessa maneira, a fim de se realizar o problema de comunicação *wireless* do robô com o espaço inteligente, bem como aumentar a sua capacidade computacional tendo em vista a futura inserção de sensores mais complexos, como câmeras e sensores *laser*, optou-se por adicionar uma placa *Raspberry Pi* 2B. Eliminou-se, então, o ESP8266 e manteve-se o *Arduino* para realizar a interface com os sensores e atuadores. A comunicação entre a *Raspberry Pi* e o robô é feita via porta serial, enquanto que a comunicação entre a *Raspberry Pi* e a rede do espaço inteligente é feita utilizando um módulo *WiFi* conectado a uma porta USB.

Apesar do grande poder de processamento da *Raspberry Pi* despertar a ideia de utilizar apenas uma placa para comandar todo o robô, foi decidido manter o *Arduino Mega* para tratar as operações de baixo nível da plataforma e deixar a *Raspberry Pi* responsável apenas pela comunicação com o espaço inteligente e processos de alto nível. A justificativa para essa decisão de projeto se baseia em três pontos importantes: a diferença entre a *Raspberry Pi* e o *Arduino*, a confiabilidade necessária nas operações de baixo nível como o cálculo da odometria e o controle dos motores e o desejo de causar a menor interferência possível na plataforma já existente.

A diferença básica entre o *Arduino* e a *Raspberry Pi* é que enquanto o primeiro é basicamente uma placa com um microcontrolador e interfaces de entrada e saída que roda um código em loop, o outro é um computador completo de uso geral que roda um sistema operacional

Quadro 3 – Quadro comparativo entre o *Arduino Mega* e o *Raspberry Pi 2B*

Especificações	**Arduino Mega**	**Raspberry Pi 2 Model B**
CPU	Atmega 2560	Quad Cortex A7
Clock	16MHz	900MHz
RAM	8KB SRAM	1GB SDRAM
Memória Flash	128KB	Cartão SD Externo
Pinos digitais	54	40
Pinos analógicos	16	-
Comprimento	102mm	85mm
Largura	53,3mm	56mm

Fonte: (ARDUINO COMPANY, 2006) e (RASPBERRY PI FOUNDATION, 2015)

(geralmente Linux) e capaz de executar vários processos simultaneamente. No Quadro 3 observa-se que enquanto a *Raspberry Pi* 2B possui maior poder de processamento, com CPU mais veloz e maior quantidade de RAM, o *Arduino Mega* possui mais pinos de entrada e saída digitais, inclusive pinos de entrada analógica.

Comparando as características e especificações das duas placas pode-se concluir que a *Raspberry Pi* é mais adequada para realizar tarefas que exigem maior processamento, como a execução de algoritmos de controle e tomada de decisões. Já o *Arduino Mega* consegue desempenhar muito bem a função de elemento de baixo nível, realizando a interface entre o sistema de inteligência da plataforma e o hardware.

Além disso, deixar uma placa dedicada ao baixo nível da plataforma gera uma maior confiabilidade na odometria do robô, pois, com a utilização dos pinos de interrupção e das saídas PWM do *Arduino*, pode-se obter uma leitura confiável e precisa da posição e da orientação do robô e o correto comando de velocidade das rodas do mesmo ainda que o sistema de alto nível trave durante a execução de algum algoritmo mais complexo. Assim, aumenta-se a robustez da plataforma e diminui o erro acumulativo que pode ser gerado pela odometria.

Ademais, para que a *Raspberry Pi* consiga realizar um cálculo preciso da odometria e um perfeito controle PWM dos motores da plataforma, seria necessário utilizar um sistema operacional de tempo real, pois pequenos erros temporais na leitura dos *encoders* rotativos da plataforma podem se acumular e gerar diferenças relevantes no cálculo da posição do robô. Outro ponto a ser observado é que nesse trabalho procurou-se causar a menor interferência possível na infraestrutura já existente na plataforma, de modo a aproveitar ao máximo o que já foi realizado e testado anteriormente.

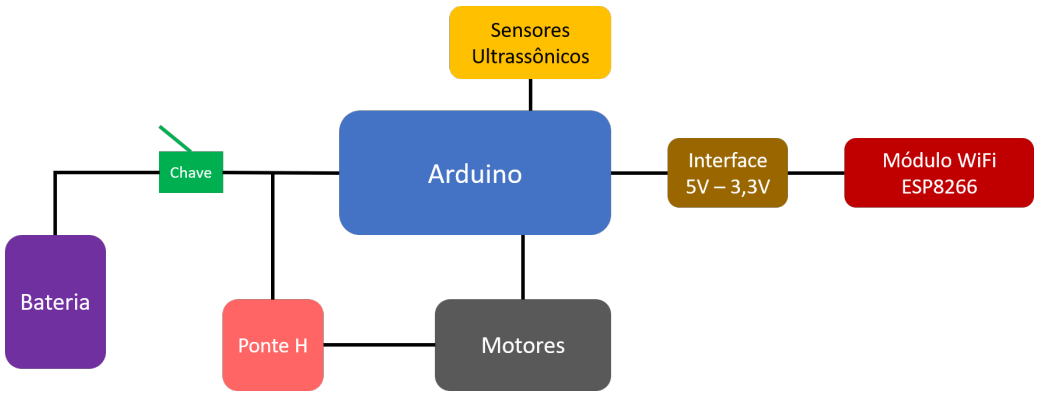
Com a adição da *Raspberry Pi* na plataforma, boa parte da inteligência do robô que

antes era executada pelo *Arduino Mega* foi transferida para a camada de alto nível, como o controlador de posição final, o controlador de trajetória e todo o código referente à comunicação com o módulo *WiFi*. Com isso, foram liberados recursos computacionais do *Arduino Mega*, o que abre espaço para a substituição deste por uma placa mais simples, tornando mais barata a construção de uma nova plataforma com as modificações inseridas nesse trabalho.

3.1 Estrutura Física

Antes das alterações realizadas nesse trabalho, a plataforma era composta por um *Arduino Mega*, quatro sensores ultrassônicos modelo HC-SR04, dois motores com *encoders* rotativos, um módulo de ponte H modelo L298 e um módulo *WiFi* modelo ESP8266, ligado ao *Arduino* por uma placa que realizava a interface entre os níveis de tensão do controlador e do módulo. Toda a plataforma era alimentada por uma bateria de 1000mAh e energizada por uma chave geral, como pode ser observado no esquema da Figura 6.

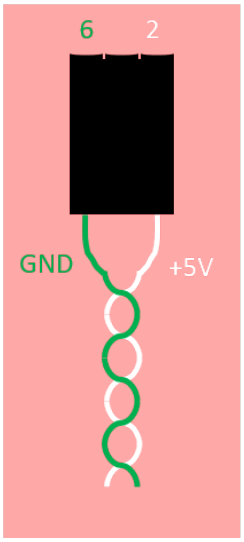
Figura 6 – Esquemático da estrutura antiga do robô.



Fonte: Produção do próprio autor.

Nas modificações realizadas, a maior parte do *hardware* original do robô foi aproveitado, realizando-se o mínimo de alterações possíveis no mesmo para adicionar os novos recursos. Para alimentar a *Raspberry Pi*, utilizou-se um módulo regulador de tensão LM2596 ajustado para 5V conectado na saída do interruptor geral do robô. Assim, a energia para a alimentação da *Raspberry Pi* é suprida direto pela bateria e ela é energizada no instante em que o robô é ligado. Da saída do regulador de tensão sai um chicote com dois fios, um branco e um verde, trançados terminando nas extremidades de um conector com espaço para três pinos, como mostra a Figura 7. Nesse conectador, o fio verde deve ser conectado ao *Ground* da placa (pino 6) e o fio branco ao *DC Power* (pino 2), como pode ser observado na Figura 8.

Figura 7 – Conector dos fios de alimentação da *Raspberry Pi*.



Fonte: Produção do próprio autor.

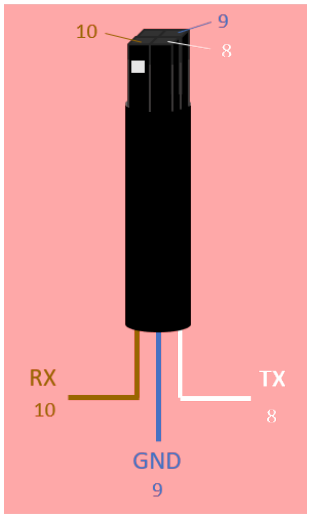
Figura 8 – Conexão dos fios nos pinos da *Raspberry Pi*.

Fonte: Produção do próprio autor.

Já para realizar a conexão serial entre a *Raspberry Pi* e o *Arduino Mega* foi aproveitada uma placa que era utilizada para adequar os níveis de tensão entre o *Arduino*, que trabalha com 5V, e o antigo módulo WiFi da placa, que utilizava 3,3V como tensão de nível lógico alto. Como os pinos de GPIO da *Raspberry Pi* também funcionam com tensão de 3,3V, manteve-se essa placa no projeto para realizar a interface entre os pinos TX e RX da *Raspberry Pi* e os pinos RX e TX do *Arduino Mega*. Assim, dessa placa saem três fios, um azul, um branco e um marrom, que foram unidos em um único chicote usando termo-retráteis. No final do chicote foi adicionado um conector a mais além dos três fios para formar um quadrado e deixar o conector mais firme e estável, como pode ser visualizado na Figura 9. Nesse chicote, o fio azul deve ser conectado ao *Ground* da *Raspberry Pi* (pino 9), o fio branco deve ser conectado ao TX da *Raspberry Pi* (pino 8) e o fio marrom deve ser conectado ao RX da *Raspberry Pi* (pino 10). Para saber a orientação com que se deve encaixar os quatro conectores na placa, o terminal que possui uma pequena placa de metal aparente na lateral do conector (Figura 9) corresponde ao fio marrom e, portanto, deve

ser conectado no pino 10 da *Raspberry Pi*, como pode ser visto na Figura 8.

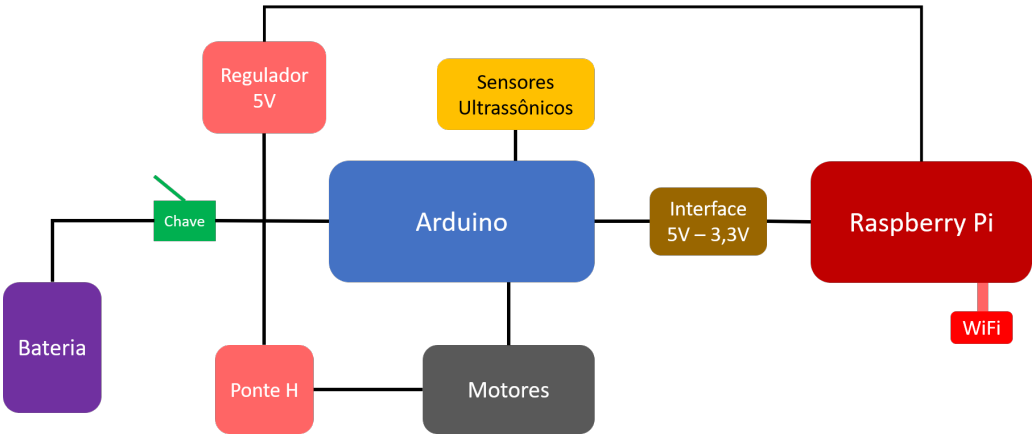
Figura 9 – Conector dos fios de comunicação serial da *Raspberry Pi*.



Fonte: Produção do próprio autor.

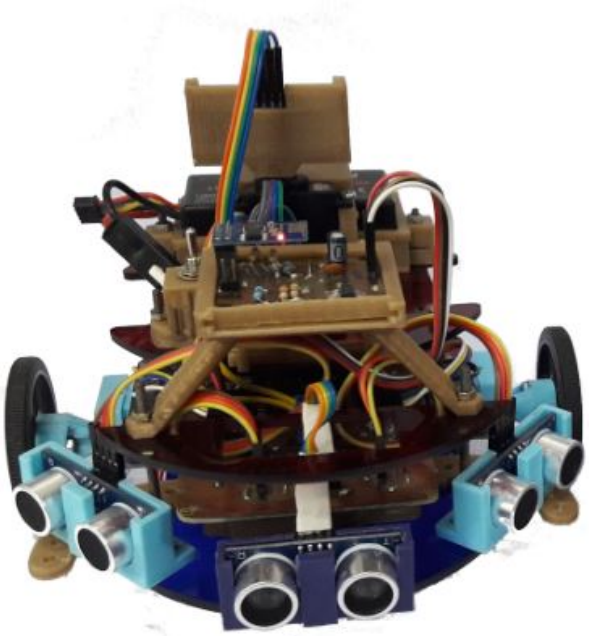
O adaptador WiFi utilizado na *Raspberry Pi* é TL-WN821N da marca TP-Link. Esse adaptador foi simplesmente conectado em uma porta USB da placa, não sendo necessário nenhum ajuste de *hardware* para funcionar. Na Figura 10 pode-se observar resumidamente a estrutura nova da plataforma, com as modificações realizadas. As Figuras 11 e 12 mostram fotos do robô antes e depois das modificações realizadas.

Figura 10 – Esquemático da nova estrutura do robô.



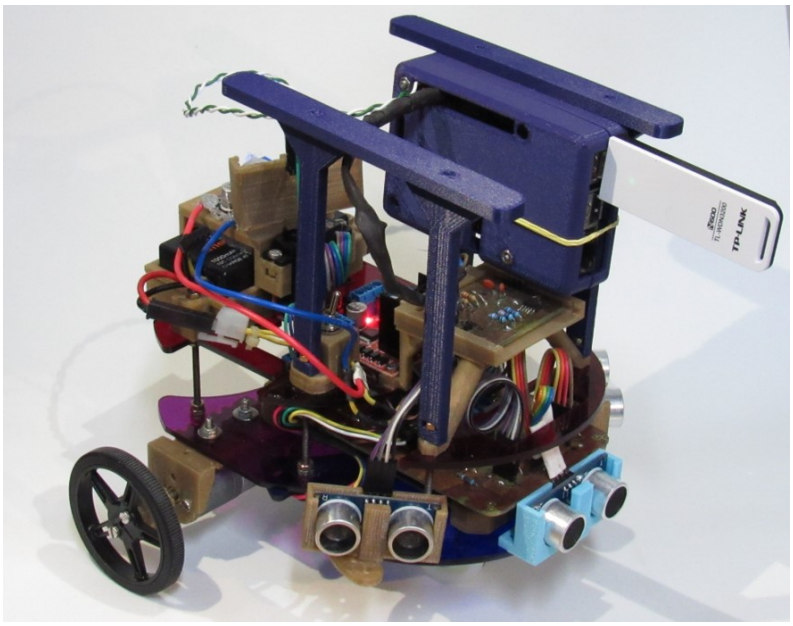
Fonte: Produção do próprio autor.

Figura 11 – Robô com a estrutura antiga.



Fonte: (DALTIO, 2017)

Figura 12 – Robô com a nova estrutura já montada.



Fonte: Produção do próprio autor.

3.2 Comunicação Arduino - Raspberry

Neste trabalho decidiu-se manter a placa *Arduino Mega*, que já estava presente na plataforma, atuando como um elemento de baixo nível do robô, recebendo os dados dos sensores,

computando a odometria e comandando os motores da plataforma. Já a *Raspberry Pi* atua na camada de alto nível, realizando a comunicação com o espaço inteligente.

Para que esse tipo de arquitetura funcione é necessário que haja um canal de comunicação entre o *Arduino Mega* e a *Raspberry Pi* por onde transitam os dados dos sensores e os comandos de atuação dos motores da plataforma. Neste trabalho optou-se por realizar essa comunicação de forma serial via protocolo UART.

Por meio desse canal serial são transmitidos do *Arduino Mega* para a *Raspberry Pi* a posição e orientação do robô tendo como referência o instante no qual ele é ligado, as distâncias medidas pelos três sensores ultrassom do robô e suas velocidades linear e angular. Essas informações são serializadas em uma única mensagem *protobuf* para serem enviadas como um pacote de dados que será posteriormente desserializado na *Raspberry Pi*.

Da *Raspberry Pi* para o *Arduino Mega* serão enviados os comandos de velocidades linear e angular que o robô deverá executar. Esses valores de velocidades são também serializados em uma outra mensagem *protobuf* para serem transmitidos. O arquivo *.proto* contendo as duas mensagens descritas anteriormente pode ser visto na Figura 13.

Figura 13 – Mensagens *protobuf* utilizadas para serializar os dados a serem transmitidos na comunicação serial.

```
syntax = "proto3";

message Odometry {
    int32 pos_x = 1;
    int32 pos_y = 2;
    int32 heading = 3;
    int32 vel_linear = 4;
    int32 vel_angular = 5;
    uint32 dist_US_Frente = 6;
    uint32 dist_US_Direito = 7;
    uint32 dist_US_Esquerdo = 8;
}

message PIDs {
    int32 velLin = 1;
    int32 velAng = 2;
}
```

Fonte: Produção do próprio autor.

Um dos motivos pelos quais optou-se por utilizar o *protobuf* na transferência de dados entre a *Raspberry Pi* e o *Arduino* é que ele permite a criação de mensagens que suportam adição

de campos sem perda de retrocompatibilidade, desde que se mantenha as identificações dos campos antigos. Ou seja, caso futuramente seja necessário adicionar mais dados de sensores ou comandos para atuadores, expandindo a mensagem *protobuf*, a parte de comunicação se manteria a mesma e a leitura dos dados e execução dos comandos continuaria funcional.

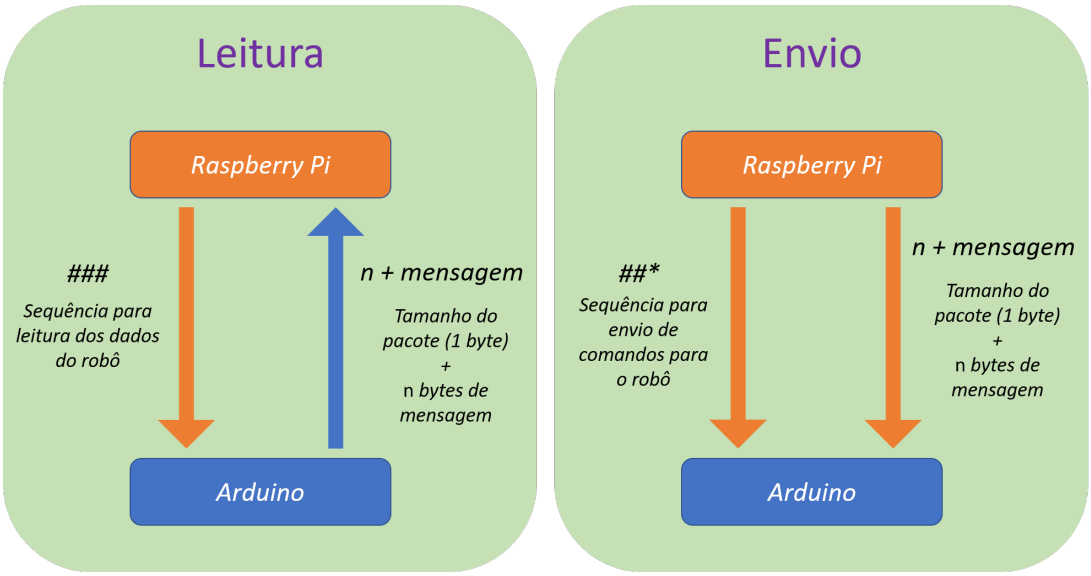
Para tornar a comunicação mais organizada e robusta, optou-se por criar um protocolo do tipo mestre e escravo, onde a *Raspberry Pi* atua como mestre, informando quando deseja enviar comandos de velocidade ou receber dados dos sensores, e o *Arduino Mega* atua como escravo, esperando um pedido do mestre para só então receber ou enviar dados pelo canal de comunicação.

Para sinalizar que deseja enviar ou receber um pacote, o mestre envia pelo canal serial uma sequência pré-definida de três caracteres, que é aguardada pelo escravo para tomar uma ação. Caso a *Raspberry Pi* envie uma sequência de três '#' (###), o *Arduino Mega* entende que o mestre deseja receber dados dos sensores. Assim, o escravo enviará um byte indicando o tamanho do pacote e em seguida enviará os dados serializados com o *nanopb*. A *Raspberry Pi* efetuará, então, a leitura do pacote para logo em seguida desserializá-lo via *protobuf*.

Para enviar um comando de velocidade para o robô o mestre inicia a comunicação transmitindo uma sequência de dois '#' e um '*' (##*) seguida de um byte indicando o tamanho do pacote e a mensagem com os comandos de velocidades linear e angular serializados via *protobuf*. O escravo então efetuará a leitura das mensagens e a desserializará com o *nanopb*. Desse modo, o *Arduino Mega* fica constantemente esperando receber algumas das sequências pré-definidas para só então executar uma ação de leitura ou escrita. A Figura 14 mostra esse padrão de comunicação em detalhe.

Como um *byte* pode assumir 256 valores diferentes, um único caractere seria capaz de representar todos os comandos existentes. Seria possível, por exemplo, usar apenas um '#' para representar um comando de leitura e um '*' para representar um comando de escrita. Porém, esses dois caracteres podem estar presentes nas mensagens binárias serializadas pelo *protobuf*, o que poderia gerar problemas caso o escravo interpretasse esse caractere no meio da mensagem como o início de um comando. Definindo uma sequência específica de três caracteres para representar os comandos torna-se muito mais improvável que esse tipo de problema ocorra. Além disso, ainda que essas sequências apareçam na mensagem serializada, o *Arduino Mega* não verifica a existência de um comando nos *bytes* recebidos enquanto o pacote está sendo lido. Essa verificação só volta a acontecer após o término do envio ou recebimento do pacote, quando o canal serial normalmente entre em estado ocioso (*idle*).

Figura 14 – Padrão de comunicação serial mestre/escravo entre a *Raspberry Pi* e o *Arduino Mega*.



Fonte: Produção do próprio autor.

3.3 Comunicação Raspberry - Espaço Inteligente

No espaço inteligente presente no laboratório Viros é utilizado o protocolo AMQP com um *broker* RabbitMQ para realizar a comunicação entre os diversos dispositivos e serviços do espaço. As mensagens que transitam nessa rede são definidas com arquivos *protobuf* e encontram-se padronizadas no repositório *is-msgs*, disponível no *github*[1] do laboratório.

O tráfego dessas mensagens é feita no padrão *publisher/subscriber*, onde a *Raspberry Pi* publica para o espaço os dados da odometria e dos sensores de distância ultrassônicos do robô nos tópicos específicos de cada um e recebe os comandos de velocidades linear e angular do espaço via uma chamada de procedimento remoto (RPC). Por RPC o espaço também consegue obter os valores de velocidade do robô no instante da solicitação.

Existe também um padrão para os tópicos das mensagens publicadas dentro do espaço, onde um dos campos do tópico corresponde ao ID do robô. Dessa forma, é possível ter vários robôs sendo controlados ao mesmo tempo no espaço, já que cada um deles pode ser identificado por um ID único.

Como já havia uma grande quantidade de serviços prontos e bem consolidados para o robô *Pioneer P3-AT*, buscou-se seguir essa padronização do laboratório tendo em vista a utilização desses serviços já existentes também no novo robô, como o controlador de

[1] https://github.com/labviros/is-msgs

posição final e de trajetória.

Seguindo a estrutura organizacional desenvolvida anteriormente para o robô *Pioneer P3-AT*, para realizar a comunicação do novo robô com o espaço inteligente do laboratório foram criados um *driver* e um *gateway*, ambos em *python3*.

O *driver* é responsável por realizar a interface da *Raspberry Pi* com o baixo nível do robô. É por meio dele que o espaço inteligente se comunicava com o *Arduino* via porta serial, apresentando funcionalidades que possibilitam ao espaço ler informações do robô e enviar comandos a ele.

O *gateway* representa a interface do robô com o espaço inteligente. Ele é responsável por acessar os recursos do robô através das interfaces padronizadas no *driver* e expor, também de forma padronizada, os serviços e os dados dos sensores no ambiente inteligente.

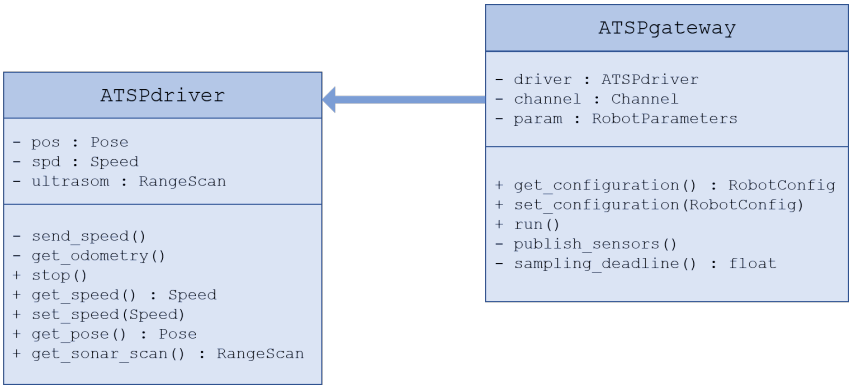
As abstrações de *driver* e *gateway* que acabaram de ser descritas já haviam sido implementadas no laboratório em C++ para utilização no robô *Pioneer P3-AT*. Porém, como o desenvolvimento desse trabalho foi feito em *python*, foi necessário reimplementar todos os códigos relativos a essas abstrações. Como o desenvolvimento desses códigos seguiu a padronização já existente no laboratório, existe a possibilidade de futuramente utilizar esses mesmos códigos na inserção de outros robôs semelhantes, bastando apenas implementar um *driver* para o novo robô em questão. Assim, caso seja necessário adicionar mais robôs no espaço, o *gateway* já estará pronto nas duas linguagens (C++ e *python*), sendo necessário apenas criar um novo *driver*, que é individual de cada robô.

A camada de comunicação entre a *Raspberry Pi* e o espaço inteligente pôde ser abstraída pelo uso da biblioteca *is-wire*, já anteriormente desenvolvida e disponível no *github*[2] do laboratório. Essa biblioteca possui versões em C++ e em *python*, sendo esta última a utilizada nesse trabalho.

No diagrama da Figura 15 pode-se observar os parâmetros e métodos das classes de *driver* e de *gateway* que foram implementadas nesse trabalho. Os objetos dos tipos *Pose*, *Speed*, *RangeScan* e *RobotConfig* fazem parte do repositório *is-msgs*, criado no laboratório. A classe *Channel* encontra-se na biblioteca *is-wire* e a classe *RobotParameters* é criada por um arquivo *protobuf* que encapsula informações de configuração da comunicação, como endereço do *broker*, código de identificação do robô no espaço e a porta serial utilizada.

[2] https://github.com/labviros/is-wire

Figura 15 – Diagrama mostrando os atributos e métodos das classes de *driver* e de *gateway* que foram implementadas.



Fonte: Produção do próprio autor.

4 EXPERIMENTOS E RESULTADOS

Para testar se a inserção do robô no espaço inteligente do laboratório foi realizada com sucesso, foram executados algoritmos de controle de posição final e de trajetória em formato de oito dentro do espaço. Essas tarefas foram feitas utilizando um serviço já existente no laboratório, o *is-robot-controller*, disponível no *github*[1] do laboratório Viros. Esse serviço, que é executado pelo espaço inteligente, obtêm a posição do robô por meio das câmeras disponíveis e a utiliza para computar os comandos de velocidades linear e angular que serão enviados ao robô para executar uma determinada tarefa.

O *is-robot-controller* já foi extensivamente utilizado no *Pioneer P3-AT*, apresentando excelentes resultados com esse robô. Além de controlar o robô, esse serviço publica periodicamente uma mensagem contendo o progresso da tarefa de controle que está sendo executada. Essa mensagem contém informações como as velocidades linear e angular atuais do robô, sua posição no espaço, sua orientação e o erro em relação à posição desejada.

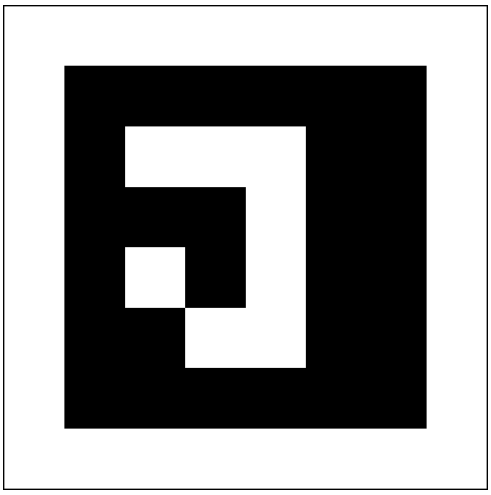
Como tanto o *Pioneer* quanto o novo robô inserido são do mesmo tipo (tração diferencial), o mesmo controlador implementado para o *Pioneer* deve funcionar no novo robô. Ainda que a estrutura do *Pioneer* seja um pouco diferente, já que ele possui quatro rodas enquanto o novo robô possui duas, ele se movimenta como um robô de tração diferencial pois as rodas deste são mecanicamente conectadas.

Em ambos os experimentos o espaço detectava a posição do robô no laboratório por meio de um padrão fixado sobre ele. O padrão utilizado é um *ArUco* (ROMERO-RAMIREZ; MUÑOZ-SALINAS; MEDINA-CARNICER, 2018) com o formato da Figura 16. Esse padrão tem o formato de um quadrado de 18 centímetros de lado e foi construído utilizando-se cartolina preta e branca fosca para refletir pouca luz incidente e colado sobre um quadrado de papelão para deixá-lo mais reto e firme. A detecção desse padrão pelo espaço inteligente é feita pelo serviço *is-aruco-detector*, já desenvolvido e exaustivamente testado no laboratório. Uma vez detectado o padrão, é possível determinar a localização do robô no espaço por meio do serviço *is-frame-transformation*, também desenvolvido e testado previamente no laboratório. Ambos os serviços encontram-se disponíveis no *github* do laboratório Viros. A Figura 17 mostra uma foto do robô com o *ArUco* fixado sobre ele.

O *is-robot-controller* recebe como parâmetros a identificação do robô (*robot_id*), a distância do centro do robô ao ponto que será controlado (*center_offset*), os valores máximos de velocidades linear e angular do robô (*speed_limits*), os ganhos do controlador (*gains*) e o tempo máximo em que será permitido continuar controlando o robô sem receber um novo

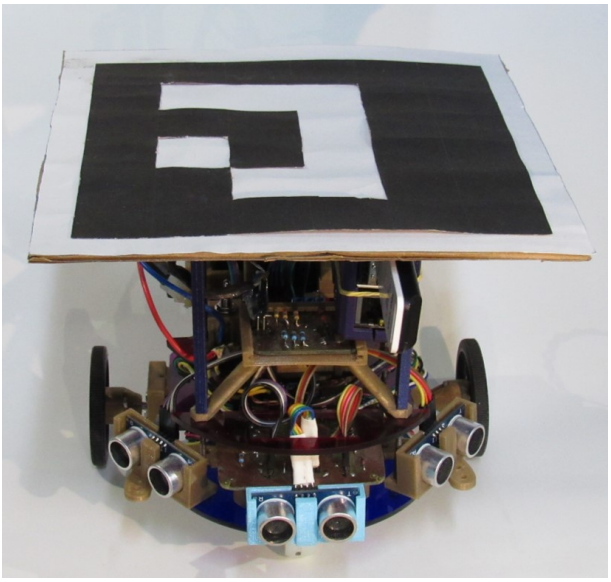
[1] https://github.com/labviros/is-robot-controller

Figura 16 – Padrão para detecção do robô no espaço.



Fonte: Produção do próprio autor.

Figura 17 – Robô com o padrão fixado sobre ele.



Fonte: Produção do próprio autor.

valor de posição do mesmo (*allowed_measurement_disruption*). Para os experimentos que serão detalhados nesse capítulo, esses parâmetros foram definidos com os valores da Tabela 1.

O serviço *is-robot-controller* permite que sejam executadas tarefas tanto de posição final, quanto de seguimento de trajetória. Para isso existe uma mensagem bem definida que é enviada ao controlador para configurá-lo de acordo com a tarefa a ser executada. A seguir serão detalhados os dois experimentos realizados no robô e os resultados obtidos por eles.

Tabela 1 – Parâmetros do controlador utilizados nos experimentos.

robot_id	0
center_offset	0,1 m
speed_limits	linear: 0,45 m/s, angular: 0,25 m/s
gains	kx: 0,4, ky: 0,4
allowed_measurement_disruption	0,2 s

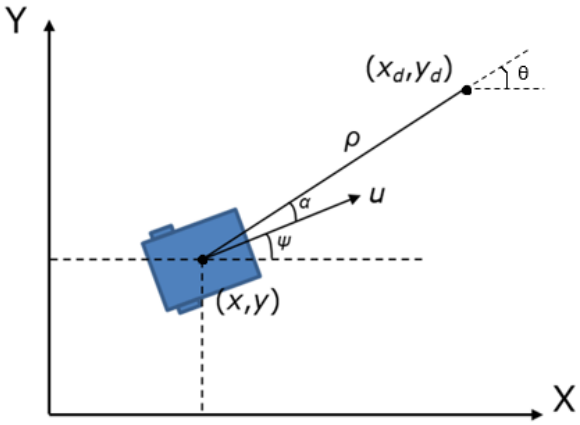
4.1 Controle de Posição Final

O controle de posição final consiste em levar o robô até um ponto desejado no espaço. As leis de controle desse controlador estão descritas nas equações 4.1 e 4.2, onde u é a velocidade linear do robô e ω corresponde a velocidade angular do mesmo. Os valores de ρ e α correspondem, respectivamente, à distância entre o destino e a posição atual do robô (erro de posição) e ao ângulo entre o vetor velocidade do robô e a reta que une os pontos de origem e de destino (erro de orientação), como pode ser visto na Figura 18.

$$u = u_{max}.tanh(\rho).cos(\alpha) \tag{4.1}$$

$$\omega = k_{\omega}.\alpha + u_{max}.\frac{tanh(\rho)}{\rho}.sin(\alpha).cos(\alpha) \tag{4.2}$$

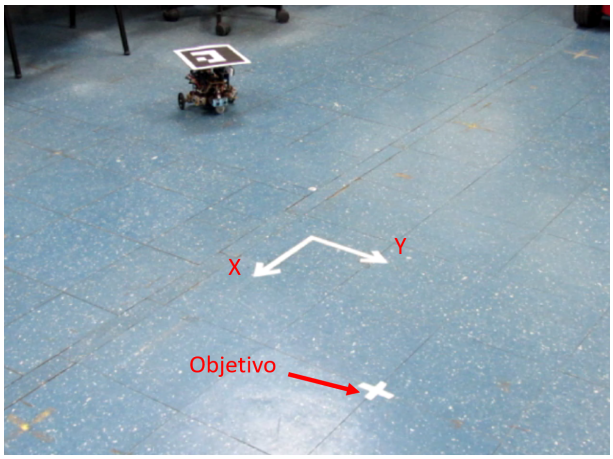
Figura 18 – Variáveis analisadas no controle de posicionamento.



Fonte: Produção do próprio autor.

Nesse experimento o robô partiu do ponto (-80;-80) centímetros e foi executado um algoritmo de controle de posição final com o objetivo de levá-lo ao ponto (+50;+50) centímetros, como pode ser observado na Figura 19.

Figura 19 – Foto do início do experimento de controle de posição final com o ponto a ser atingido pelo robô em destaque.



Fonte: Produção do próprio autor.

O algoritmo utilizado nesse experimento já havia sido desenvolvido anteriormente no laboratório e utilizado para o controle do robô *Pioneer P3-AT*. Quando foi feita a inserção da nova plataforma robótica no espaço inteligente, um dos objetivos é que o espaço consiga tratar todo robô desse tipo da mesma maneira. Portanto, o mesmo algoritmo deve funcionar para os dois robôs, necessitando-se modificar apenas alguns parâmetros de configuração do serviço, que estão relacionados às dimensões físicas do robô e a suas limitações de velocidade.

Durante o experimento os dados de odometria do robô foram registrados para posterior análise. Esses dados foram salvos consumindo-se a mensagem publicada periodicamente pelo controlador com o *status* da tarefa de controle e encontram-se disponíveis para consulta no apêndice A desse trabalho. Nas Figuras 20, 21 e 22 podem ser observados os gráficos da coordenada X, da coordenada Y e do erro, respectivamente, gerados por meio dos dados coletados.

Na Figura 23 pode-se ver a trajetória executada pelo robô no plano cartesiano. Esse gráfico foi criado utilizando-se a posição estimada do robô pelas câmeras por meio do *ArUco* posicionado sobre ele e utilizando-se os serviços *is-aruco-detector* e *is-frame-transformation* do laboratório. Devido a diversas fontes de imprecisão inerentes a esse processo, como o tamanho e a qualidade do padrão utilizado, a trajetória gerada pelos dados coletados parece sinuosa, porém no experimento o robô executou a tarefa de modo mais suave. O que ocorre é que a localização visual do robô é utilizada no controlador para determinar o erro de posicionamento da plataforma e, a partir desse valor, calcular os sinais de controle que serão enviados ao robô. Isso ocorre a cada período de amostragem do *loop* de controle do sistema, porém a taxa de envio de comandos ao robô é muito alta comparada à velocidade

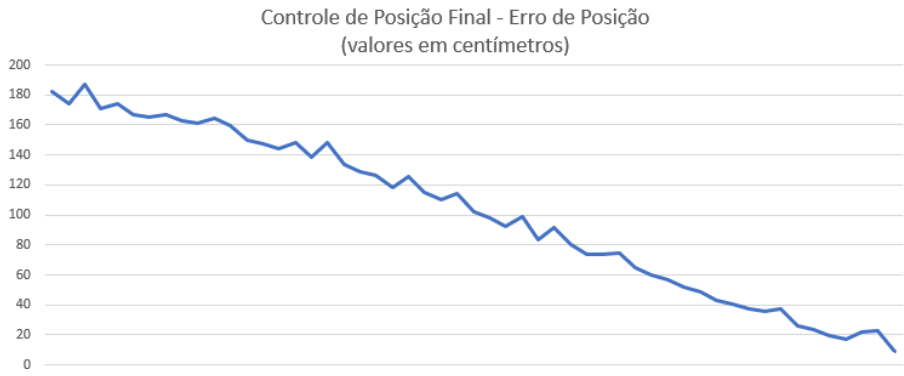
Figura 20 – Gráfico do valor da coordenada X ao longo do tempo no experimento de controle de posição final.

Fonte: Produção do próprio autor.

Figura 21 – Gráfico do valor da coordenada Y ao longo do tempo no experimento de controle de posição final.

Fonte: Produção do próprio autor.

Figura 22 – Gráfico do valor do erro ao longo do tempo no experimento de controle de posição final.



Fonte: Produção do próprio autor.

com que o mesmo se move. Dessa forma, ocorre naturalmente uma filtragem devido à dinâmica do robô, pois até que um sinal de controle surta efeito, outro comando já foi enviado ao mesmo. Assim, a plataforma não apresenta o comportamento ruidoso mostrado

Figura 23 – Trajetória do robô no plano cartesiano no experimento de controle de posição final.

Fonte: Produção do próprio autor.

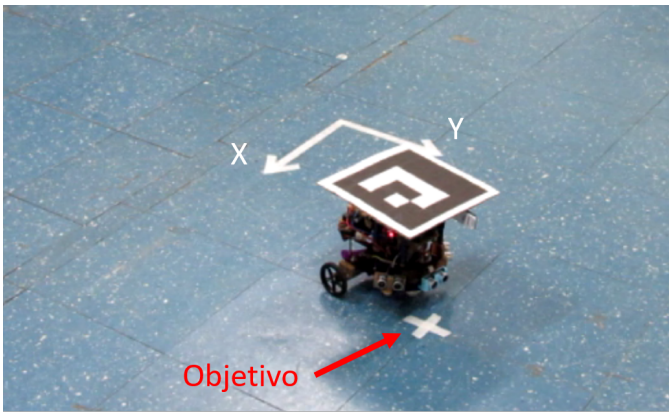
no gráfico da Figura 23, movendo-se de modo mais suave. No canal do *YouTube*[2] do autor desse trabalho pode ser visto um vídeo mostrando a trajetória real do robô no experimento.

Na Figura 24 pode-se observar a posição atingida pelo robô no final do experimento. Observa-se pelos gráficos e pela Figura 24 que o robô cumpriu a tarefa desejada, chegando próximo ao ponto destino, e que apresentou um erro final de aproximadamente 10 centímetros. Uma possível causa para esse erro é a imprecisão no cálculo da odometria pelas câmeras do laboratório. Entre os fatores que prejudicam esse cálculo estão o tamanho do padrão e a distância do padrão até as câmeras de vídeo.

O *ArUco* utilizado no robô *Pioneer* possui o formato de um quadrado de 30 centímetros de lado e fica posicionado a uma altura de 64 centímetros do chão, enquanto que o *ArUco* criado para o novo robô inserido possui 18 centímetros de lado e fica posicionado a uma altura de 20 centímetros do chão. Assim, como o padrão do novo robô é menor do que o utilizado pelo *Pioneer* e fica mais próximo do chão, consequentemente mais distante das câmeras, o cálculo preciso da posição do robô pelas câmeras é dificultado. A resolução da câmera também tem um papel importante na detecção do padrão. Uma câmera de melhor resolução poderia detectar com mais precisão a posição e a orientação do padrão, diminuindo o erro final do experimento.

[2] https://www.youtube.com/watch?v=RUs0vaEjyIc

Figura 24 – Foto do final do experimento de controle de posição final, com o ponto objetivo em destaque.



Fonte: Produção do próprio autor.

4.2 Controle de Trajetória em oito

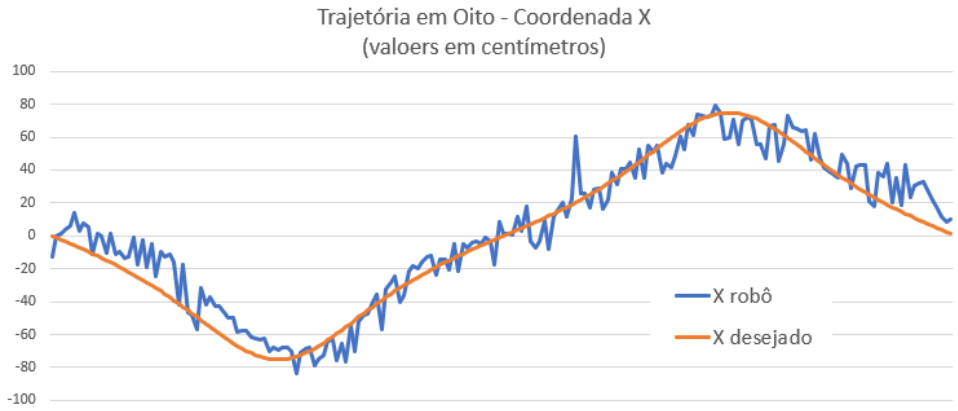
O controle de trajetória consiste em fazer o robô desenvolver um caminho desejado no espaço a uma certa velocidade. Nesse experimento foi executado um algoritmo para que o robô realizasse uma trajetória em formato de oito, partindo e finalizando na origem do plano cartesiano do espaço. A trajetória em oito a ser realizada é centrada na origem, possui 1,5 metro de comprimento no eixo X e 1 metro de largura no eixo Y e foi configurada para ser completada pelo robô em 40 segundos.

Assim como no outro experimento, o algoritmo utilizado nesse teste já havia sido desenvolvido anteriormente para o robô *Pioneer P3-AT* e, segundo a filosofia de que o espaço inteligente deve tratar qualquer robô desse tipo da mesma maneira, esse mesmo código deveria funcionar também no novo robô inserido, modificando-se apenas alguns parâmetros dimensionais.

Durante o experimento foram coletados os valores das coordenadas do robô a cada instante, bem como os valores das coordenadas da trajetória que deveria ser seguida. Esses dados encontram-se disponíveis para consulta no apêndice A desse trabalho. Nos gráficos das Figuras 25 e 26 pode-se comparar os valores das coordenadas X e Y, respectivamente, do robô em relação ao valor desejado.

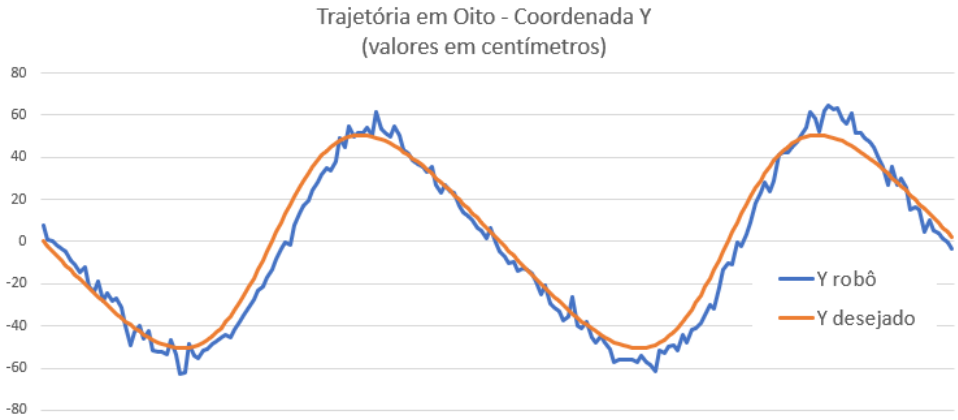
Na Figura 27 pode ser observada a trajetória executada pelo robô juntamente com a trajetória desejada. O gráfico da trajetória executada pelo robô foi criado utilizando-se a posição estimada do mesmo pelas câmeras por meio do *ArUco* posicionado sobre ele e utilizando-se os serviços *is-aruco-detector* e *is-frame-transformation* disponíveis no laboratório. Devido a vários fatores que podem gerar imprecisão nessa estimativa, como

Figura 25 – Gráfico do valor da coordenada X do robô ao longo do tempo no experimento de trajetória em oito.



Fonte: Produção do próprio autor.

Figura 26 – Gráfico do valor da coordenada Y do robô ao longo do tempo no experimento de trajetória em oito.



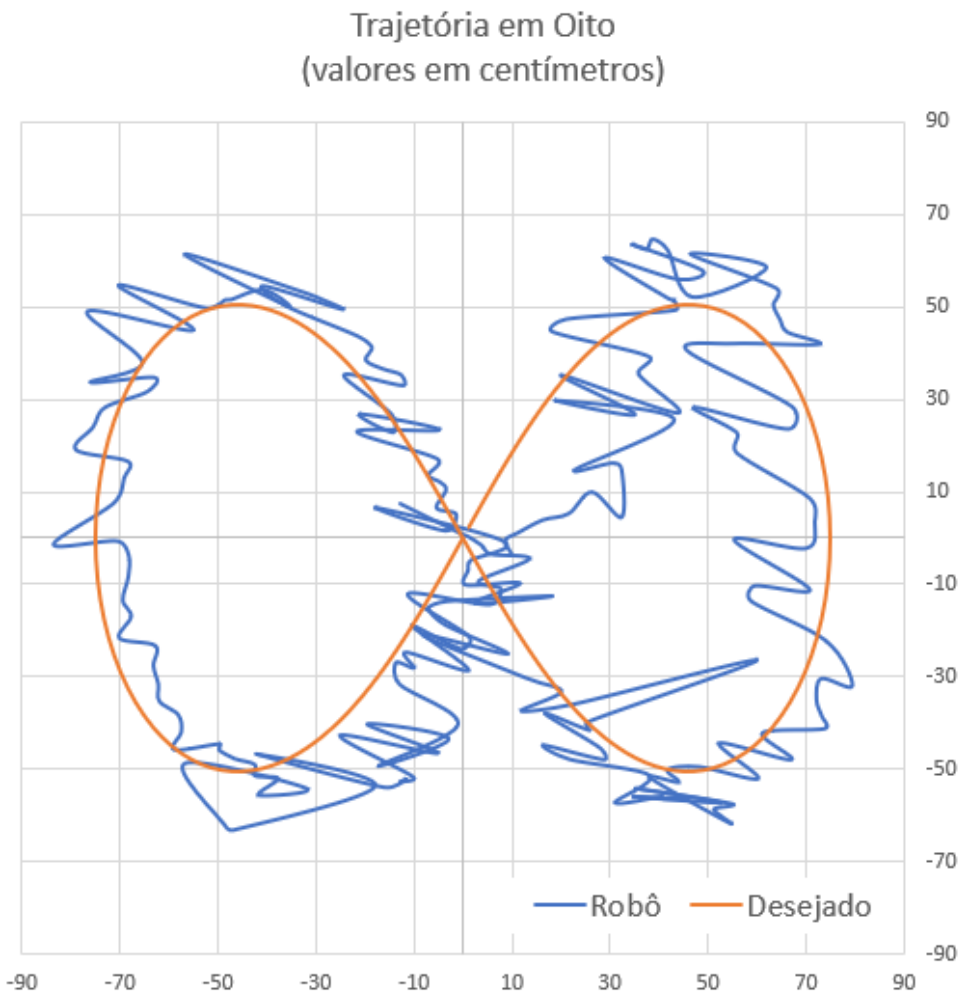
Fonte: Produção do próprio autor.

o tamanho e a qualidade do padrão utilizado, a trajetória gerada pelos dados coletados parece bastante oscilatória. Porém, no experimento real o robô executou a tarefa de maneira mais suave de modo que essas oscilações não aparecem. Conforme já foi explicado no experimento anterior, a dinâmica do robô atua como um filtro para o comportamento ruidoso dos sinais de controle do sistema, pois a plataforma se move lentamente comparada à taxa de atualização do comando de controle. No canal do *YouTube*[3] do autor desse trabalho pode ser visto um vídeo mostrando a trajetória real do robô no experimento.

Observa-se pelos gráficos gerados que, apesar de se distanciar do caminho proposto em alguns pontos, o robô cumpriu o objetivo de realizar a trajetória em oito em um tempo total de 40 segundos, mostrando que a inserção do robô no espaço inteligente do laboratório foi realizada com sucesso. Assim como na tarefa anterior, o erro entre a trajetória realizada

[3] https://www.youtube.com/watch?v=UsjkXmvCNM8

Figura 27 – Trajetória executada pelo robô no experimento de trajetória em oito.



Fonte: Produção do próprio autor.

e a desejada pode ter sido ocasionado pela imprecisão na detecção do robô pelas câmeras do espaço inteligente. Fatores como o tamanho do padrão, sua distância até as câmeras e o material com que ele foi construído prejudicam o cálculo da posição e orientação do robô, levando a erros na execução da tarefa.

5 CONCLUSÕES E TRABALHO FUTUROS

A proposta desse trabalho foi realizar a inserção dos robôs desenvolvidos pelo aluno Murilo Leonardelli Daltio no espaço inteligente do laboratório Viros, localizado no segundo andar do prédio CT-II da Universidade Federal do Espírito Santo. Apesar da inserção ter sido feita efetivamente em apenas um dos dois robôs existentes, a mesma metodologia pode ser utilizada para inserir o segundo robô no espaço, já que os mesmos são idênticos. Para verificar se a inserção foi realizada com sucesso, foram utilizados serviços já existentes no espaço inteligente do laboratório. Primeiro foi realizada uma tarefa de controle de posição final e depois uma trajetória em oito, com o espaço inteligente detectando a posição e orientação do robô e enviando a ele comandos de velocidades linear e angular.

De acordo com os resultados apresentados e discutidos no Capítulo 4 desse trabalho, pode-se observar que o robô executou as tarefas propostas a ele de modo satisfatório. Conclui-se, portanto, que a inserção do robô no espaço inteligente do laboratório foi realizada com sucesso e o objetivo do trabalho foi cumprido.

Ainda que esse trabalho tenha atingido seu propósito, há algumas melhorias que ainda podem ser realizadas. São destacadas a seguir algumas propostas de trabalhos futuros:

- Refazer o padrão de detecção do robô com um material mais fosco, como o EVA, por exemplo, e fixá-lo sobre um material bem reto, como uma prancheta de madeira. Isso poderia melhorar a detecção do padrão pelas câmeras, aperfeiçoando a precisão no cálculo da odometria do robô pelo espaço e consequentemente melhorando a execução das tarefas de controle.
- Substituir o *Arduino Mega* por outro elemento controlador de baixo nível com dimensões menores. Apesar do *Arduino Mega* executar sua função no robô de modo muito satisfatório, ele ocupa uma área relativamente grande em um robô de pequeno porte. Uma melhoria que pode ser realizada é utilizar alguma outra placa microcontroladora menor que atenda aos requisitos do robô para executar as tarefas de baixo nível (computar a odometria, comandar os motores e efetuar a comunicação serial com a *Raspberry Pi*).
- Fazer fusão sensorial entre os dados da odometria do robô e a posição estimada do mesmo pelas câmeras do laboratório, de modo a obter um resultado mais exato do posicionamento da plataforma no espaço.

REFERÊNCIAS BIBLIOGRÁFICAS

AIMONEN, P.; POOLE, M. *Nanopb — Protocol Buffers for Embedded Systems*. 2018. Disponível em: <https://github.com/nanopb/nanopb>. 27

ARAÚJO, E. C. de. *Serialização – Solução Para Persistência de Objetos*. 2012. Disponível em: <http://www.linhadecodigo.com.br/artigo/3401/serializacao-solucao-para-persistencia-de-objetos.aspx>. 24

ARDUINO COMPANY. *Arduino Mega*. 2006. Disponível em: <http://www.arduino.cc/en/Main/arduinoBoardMega>. 32

BOSCHI, S.; SANTOMAGGIO, G. *RabbitMQ cookbook*. [S.l.]: Packt Publishing Ltd, 2013. 22

DALTIO, M. L. *Projeto AT & SP: Construção de Plataformas Robóticas com Conectividade Wi-Fi*. 2017. 36

FANG, Y.-y.; CHEN, X.-j. Design and simulation of uart serial communication module based on vhdl. In: IEEE. *Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on*. [S.l.], 2011. p. 1–4. 29

GLIGORIĆ, N.; DEJANOVIĆ, I.; KRČO, S. Performance evaluation of compact binary xml representation for constrained devices. In: IEEE. *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. [S.l.], 2011. p. 1–5. 25

Google Developers. *Protocol Buffers — Frequently Asked Questions*. 2017. Disponível em: <https://developers.google.com/protocol-buffers/docs/faq>. 25

Google Developers. *Protocol Buffers – Developer Guide*. 2018. Disponível em: <https://developers.google.com/protocol-buffers/docs/overview>. 25

KEIM, R. *Back to Basics: The Universal Asynchronous Receiver/Transmitter (UART)*. 2016. Disponível em: <https://www.allaboutcircuits.com/technical-articles/back-to-basics-the-universal-asynchronous-receiver-transmitter-uart/>. 29

LEE, J.-H.; HASHIMOTO, H. Intelligent space—concept and contents. *Advanced Robotics*, Taylor & Francis, v. 16, n. 3, p. 265–280, 2002. 15

LU, F. et al. Building an intelligent home space for service robot based on multi-pattern information model and wireless sensor networks. *Intelligent Control and Automation*, Scientific Research Publishing, v. 3, n. 01, p. 90, 2012. 19, 20

MORIOKA, K.; LEE, J.-H.; HASHIMOTO, H. Intelligent space for human centered robotics. In: *Advances in service robotics*. [S.l.]: InTech, 2008. 15

Pivotal Software. *RabbitMQ Tutorial - Introduction*. 2018. Disponível em: <https://www.rabbitmq.com/tutorials/tutorial-one-python.html>. 22

PIZARRO, D. et al. Localization of mobile robots using odometry and an external vision sensor. *Sensors*, Molecular Diversity Preservation International, v. 10, n. 4, p. 3655–3680, 2010. 19

POZZEBOM, R. *O protocolo HTTP*. 2007. Disponível em: <https://www.oficinadanet.com.br/artigo/459/o_protocolo_http>. 20

QUEIROZ, F. M. de. *Desenvolvimento da Infraestrutura de um Espaço Inteligente baseado em Visão Computacional e IoT*. 2016. 20, 22, 23

RAMPINELLI, M. et al. An intelligent space for mobile robot localization using a multi-camera system. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 14, n. 8, p. 15039–15064, 2014. 15, 19

RASPBERRY PI FOUNDATION. *Raspberry Pi 2 Model B*. 2015. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. 32

Robocore. *Comparação Entre Protocolos de Comunicação Serial*. 2018. Disponível em: <https://www.robocore.net/tutoriais/comparacao-entre-protocolos-de-comunicacao-serial.html>. 28

ROMERO-RAMIREZ, F. J.; MUÑOZ-SALINAS, R.; MEDINA-CARNICER, R. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, v. 76, p. 38 – 47, 2018. ISSN 0262-8856. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0262885618300799>. 43

SIMAS, G. *Introdução ao AMQP com RabbitMQ*. 2015. Disponível em: <https://www.devmedia.com.br/introducao-ao-amqp-com-rabbitmq/33036>. 21

SOUZA, E. F. *Protobuf — Uma alternativa ao JSON e XML*. 2018. Disponível em: <https://medium.com/trainingcenter/protobuf-uma-alternativa-ao-json-e-xml-a35c66edab4d>. 26

Standard C++ Foundation. *Serialization and Unserialization*. 2015. Disponível em: <https://isocpp.org/wiki/faq/serialization>. 24

TALIMAN, S. *Introduction And Design Simulation Of Raspberry PI*. 2016. Disponível em: <http://www.c-sharpcorner.com/article/introduction-and-design-simulation-of-raspberry-pi/>. 27

UPTON, E.; HALFACREE, G. Meet the raspberry pi. *Raspberry Pi® User Guide*, Wiley Online Library, p. 11–22, 2012. 27

WAKHLE, G. B.; AGGARWAL, I.; GABA, S. Synthesis and implementation of uart using vhdl codes. In: IEEE. *Computer, Consumer and Control (IS3C), 2012 International Symposium on*. [S.l.], 2012. p. 1–3. 29

Apêndices

APÊNDICE A – DADOS DOS EXPERIMENTOS

Nas Tabelas 2 e 3 mostradas a seguir, estão expostos os dados obtidos nos dois experimentos realizados no robô para validar a sua inserção no espaço inteligente do laboratório. Nos dois experimentos foram coletados dados das coordenadas X e Y do robô, sua orientação e suas velocidades linear e angular. Também foram coletados os valores das coordenadas X e Y desejadas que o robô deveria atingir e o erro entre o valor esperado e o valor real.

A Tabela 2 mostra os dados do primeiro experimento, em que foi dado o comando para que o robô chegasse ao ponto (50;50) centímetros partindo do ponto (-80;-80) centímetros. Já a Tabela 3 mostra os dados do segundo experimento, onde foi pedido ao robô para executar uma trajetória em oito com 1,5 metros de comprimento no eixo X e 1 metro de largura no eixo Y, partindo e finalizando na origem do plano cartesiano.

Tabela 2 – Dados do experimento de controle de posição final.

Y robô (m)	X robô (m)	Orientação (rad)	Vel linear (m/s)	Vel angular (rad/s)	Erro (m)
-0.7912	-0.7900	0.9961	0.4030	-1.7688	1.8252
-0.8643	-0.5795	0.9936	0.3870	-1.4762	1.7398
-0.9207	-0.7194	0.9525	0.4067	-1.4949	1.8722
-0.8572	-0.5324	0.7893	0.4026	-0.5995	1.7052
-0.8794	-0.5615	0.5610	0.4106	0.3019	1.7406
-0.8506	-0.4745	0.3875	0.3834	1.0648	1.6655
-0.9053	-0.3740	0.1388	0.3239	2.0162	1.6549
-0.8832	-0.4297	-0.2741	0.2279	3.1763	1.6666
-0.9196	-0.2925	0.1590	0.3085	1.9839	1.6258
-0.8662	-0.3548	0.2089	0.3327	1.7866	1.6116
-0.8825	-0.3840	-1.0445	-0.0627	3.7787	1.6410
-0.7420	-0.4964	0.3774	0.3854	1.0622	1.5923
-0.7927	-0.2638	0.4009	0.3393	1.1917	1.5015
-0.8008	-0.1925	0.6138	0.3412	0.5607	1.4737
-0.7608	-0.1988	0.6244	0.3426	0.5058	1.4415
-0.6646	-0.4112	0.6830	0.3841	-0.0534	1.4787
-0.7285	-0.1455	0.6286	0.3299	0.5735	1.3877
-0.7519	-0.2994	0.6589	0.3650	0.2226	1.4853

-0.7287	-0.0223	0.6609	0.3016	0.7002	1.3351
-0.6946	0.0155	0.6832	0.2927	0.7027	1.2891
-0.5500	-0.2061	0.7057	0.3418	0.1513	1.2653
-0.6024	0.0719	0.6780	0.2751	0.8111	1.1826
-0.6030	-0.0992	0.7067	0.3199	0.3688	1.2552
-0.5473	0.0240	0.7026	0.2878	0.6168	1.1504
-0.4486	-0.0543	0.7556	0.3052	0.2446	1.0986
-0.5378	0.0212	0.7249	0.2895	0.5425	1.1429
-0.4489	0.1254	0.7330	0.2594	0.7205	1.0202
-0.4194	0.1613	0.7383	0.2487	0.7785	0.9798
-0.3453	0.1217	0.7766	0.2573	0.5374	0.9261
-0.4107	0.1214	0.7632	0.2605	0.6116	0.9863
-0.3097	0.3104	0.7663	0.2033	1.0289	0.8316
-0.3524	0.1711	-1.5169	-0.2121	1.3955	0.9137
-0.3018	0.4182	0.8527	0.1829	1.1640	0.8059
-0.1914	0.2374	1.0015	0.2246	0.2129	0.7396
-0.2137	0.3270	1.0353	0.2103	0.4492	0.7343
-0.2434	0.4378	1.1294	0.1983	0.6619	0.7460
-0.1399	0.3870	1.1705	0.1952	0.3368	0.6498
-0.0623	0.2887	1.2375	0.1965	-0.2037	0.6007
-0.0573	0.4027	1.2712	0.1816	0.1546	0.5657
0.0194	0.3127	1.3112	0.1752	-0.3026	0.5158
0.0659	0.2795	1.3329	0.1665	-0.4924	0.4869
0.1240	0.2923	1.3352	0.1500	-0.4848	0.4296
0.0967	0.4313	1.3142	0.1444	0.0953	0.4091
0.1314	0.4498	1.2922	0.1328	0.1712	0.3720
0.1396	0.4983	1.3021	0.1256	0.3385	0.3604
0.1557	0.3512	1.3144	0.1362	-0.2545	0.3751
0.2419	0.4874	1.3230	0.0960	0.1909	0.2584
0.3273	0.3387	0.9612	0.0919	-0.1397	0.2364
0.3051	0.5088	1.3566	0.0730	0.1949	0.1951
0.3307	0.5009	1.3869	0.0649	0.1244	0.1693
0.3160	0.3791	1.3027	0.0817	-0.2748	0.2201
0.3068	0.3772	1.4064	0.0819	-0.3601	0.2289
0.4094	0.5251	1.4036	0.0338	0.1590	0.0940

Tabela 3 – Dados do experimento de controle de trajetória em oito.

Erro (m)	Vel linear (m/s)	Vel angular (rad/s)	Orientação (rad)	Y robô (m)	X robô (m)	Y esp (m)	X esp (m)
0.1476	0.0794	1.1845	-2.6104	0.0757	-0.1267	0.0000	0.0000
0.0351	0.1173	0.7688	-2.6081	0.0119	-0.0044	-0.0224	-0.0118
0.0610	0.1402	0.5552	-2.4556	0.0048	0.0119	-0.0449	-0.0236
0.0918	0.1596	0.1134	-2.2365	-0.0151	0.0402	-0.0672	-0.0354
0.1193	0.1653	-0.3034	-2.0401	-0.0334	0.0579	-0.0895	-0.0474
0.2079	0.1713	-0.9359	-1.8567	-0.0435	0.1370	-0.1117	-0.0594
0.1126	0.1413	-0.8177	-1.7265	-0.0919	0.0330	-0.1338	-0.0715
0.1698	0.1401	-1.1325	-1.6694	-0.1083	0.0792	-0.1558	-0.0838
0.1559	0.1298	-1.1514	-1.6332	-0.1425	0.0557	-0.1775	-0.0962
0.0797	0.1417	-0.5627	-1.6163	-0.1195	-0.1113	-0.1991	-0.1089
0.1356	0.1176	-1.1074	-1.6397	-0.2113	0.0136	-0.2204	-0.1217
0.1324	0.1157	-1.0903	-1.6575	-0.2383	-0.0024	-0.2415	-0.1348
0.0868	0.1332	-0.8405	-1.5817	-0.1884	-0.1025	-0.2623	-0.1481
0.1748	0.1076	-1.3249	-1.6249	-0.2843	0.0132	-0.2826	-0.1617
0.0835	0.1262	-0.8835	-1.6221	-0.2456	-0.1145	-0.3026	-0.1756
0.1008	0.1228	-0.9764	-1.6613	-0.2792	-0.0986	-0.3221	-0.1898
0.1008	0.1379	-0.7957	-1.7376	-0.2678	-0.1352	-0.3411	-0.2044
0.1020	0.1311	-0.8444	-1.8012	-0.3148	-0.1277	-0.3596	-0.2193
0.2273	0.1206	-1.3869	-1.8525	-0.4019	-0.0086	-0.3773	-0.2346
0.1249	0.0805	-0.8865	-1.9248	-0.4913	-0.1715	-0.3944	-0.2503
0.2400	0.1401	-1.2738	-2.0236	-0.4324	-0.0275	-0.4106	-0.2664
0.0928	0.1341	-0.5818	-2.1181	-0.4009	-0.1937	-0.4259	-0.2830
0.2516	0.1596	-1.1027	-2.2246	-0.4638	-0.0494	-0.4403	-0.3000
0.0737	0.1333	-0.3313	-2.3252	-0.4254	-0.2492	-0.4535	-0.3174
0.2410	0.1630	-0.9172	-2.4300	-0.5153	-0.0995	-0.4655	-0.3353
0.2325	0.1634	-0.8605	-2.4928	-0.5255	-0.1264	-0.4762	-0.3536
0.2597	0.1785	-0.8534	-2.5521	-0.5203	-0.1149	-0.4854	-0.3723
0.2367	0.1716	-0.7918	-2.6242	-0.5376	-0.1589	-0.4930	-0.3914
0.0349	0.1019	-0.0440	-2.7144	-0.4655	-0.4207	-0.4989	-0.4109
0.2557	0.1900	-0.6244	-2.8111	-0.5341	-0.1770	-0.5029	-0.4308
0.1261	0.0834	-0.6426	-2.9255	-0.6300	-0.4667	-0.5049	-0.4510
0.1152	0.0898	-0.6427	-3.0111	-0.6191	-0.4845	-0.5047	-0.4714
0.0754	0.0726	-0.1859	-3.0546	-0.4857	-0.5656	-0.5021	-0.4920
0.1993	0.0941	-1.6184	-2.3590	-0.5395	-0.3180	-0.4971	-0.5127
0.1344	0.1662	0.0173	2.6909	-0.5545	-0.4158	-0.4895	-0.5334
0.1832	0.1857	-0.3549	2.9248	-0.5185	-0.3751	-0.4791	-0.5541

0.1588	0.1808	-0.4370	2.8621	-0.5123	-0.4226	-0.4659	-0.5745
0.1755	0.1927	-0.3912	2.7874	-0.4881	-0.4233	-0.4498	-0.5945
0.1561	0.1929	-0.4184	2.6940	-0.4755	-0.4646	-0.4306	-0.6141
0.1449	0.1955	-0.4824	2.6319	-0.4562	-0.4962	-0.4084	-0.6330
0.1672	0.2102	-0.5636	2.5936	-0.4439	-0.4953	-0.3832	-0.6511
0.1293	0.2116	-0.6194	2.4037	-0.4562	-0.5876	-0.3551	-0.6682
0.1452	0.2201	-0.7608	2.4487	-0.4173	-0.5728	-0.3241	-0.6841
0.1499	0.2277	-0.7995	2.4134	-0.3808	-0.5792	-0.2903	-0.6987
0.1353	0.2320	-0.8352	2.3279	-0.3513	-0.6177	-0.2540	-0.7117
0.1432	0.2439	-0.7508	2.2445	-0.3145	-0.6197	-0.2155	-0.7231
0.1435	0.2520	-0.6709	2.1581	-0.2749	-0.6297	-0.1749	-0.7326
0.1509	0.2601	-0.5307	2.0707	-0.2313	-0.6259	-0.1327	-0.7401
0.1342	0.2620	-0.7580	2.0017	-0.2151	-0.6991	-0.0892	-0.7456
0.1438	0.2684	-0.5981	1.9379	-0.1682	-0.6752	-0.0448	-0.7489
0.1450	0.2705	-0.6308	1.8832	-0.1330	-0.6921	0.0000	-0.7500
0.1443	0.2686	-0.5380	1.8273	-0.0824	-0.6808	0.0448	-0.7489
0.1411	0.2647	-0.5017	1.7736	-0.0358	-0.6801	0.0892	-0.7456
0.1424	0.2625	-0.6107	1.7315	-0.0052	-0.7044	0.1327	-0.7401
0.2129	0.2672	-1.1798	1.6988	-0.0127	-0.8334	0.1749	-0.7326
0.1389	0.2495	-0.6149	1.6276	0.0771	-0.7105	0.2155	-0.7231
0.1241	0.2350	-0.5478	1.5822	0.1319	-0.6892	0.2540	-0.7117
0.1243	0.2270	-0.4985	1.5152	0.1672	-0.6818	0.2903	-0.6987
0.1678	0.2252	-0.9475	1.4661	0.1922	-0.7880	0.3241	-0.6841
0.1410	0.2096	-0.8225	1.4022	0.2419	-0.7523	0.3551	-0.6682
0.1257	0.1961	-0.7445	1.3313	0.2839	-0.7281	0.3832	-0.6511
0.0912	0.1772	-0.4184	1.2447	0.3177	-0.6424	0.4084	-0.6330
0.0833	0.1653	-0.3936	1.1611	0.3485	-0.6281	0.4306	-0.6141
0.1979	0.1935	-0.8390	1.0876	0.3383	-0.7581	0.4498	-0.5945
0.1150	0.1616	-0.5352	1.0002	0.3810	-0.6521	0.4659	-0.5745
0.2102	0.1555	-1.1491	0.9185	0.4918	-0.7639	0.4791	-0.5541
0.0414	0.1190	-0.3777	0.8119	0.4508	-0.5480	0.4895	-0.5334
0.1957	0.1390	-1.1264	0.7437	0.5479	-0.7017	0.4971	-0.5127
0.0312	0.1063	-0.5004	0.6120	0.4988	-0.5230	0.5021	-0.4920
0.0162	0.0948	-0.5010	0.5062	0.5162	-0.4828	0.5047	-0.4714
0.0238	0.0977	-0.5088	0.3955	0.5169	-0.4715	0.5049	-0.4510
0.0439	0.0770	-0.5483	0.3036	0.5386	-0.4052	0.5029	-0.4308
0.0526	0.0705	-0.4215	0.2053	0.5041	-0.3586	0.4989	-0.4109
0.2129	0.1555	-1.0038	0.1135	0.6151	-0.5659	0.4930	-0.3914
0.0675	0.0761	-0.6063	-0.0081	0.5330	-0.3245	0.4854	-0.3723

0.0777	0.0735	-0.5875	-0.1129	0.5182	-0.2882	0.4762	-0.3536
0.0940	0.0689	-0.5538	-0.2225	0.4964	-0.2465	0.4655	-0.3353
0.1287	0.1495	-0.5455	-0.3306	0.5465	-0.4064	0.4535	-0.3174
0.0872	0.1448	0.0358	-0.7333	0.5036	-0.3600	0.4403	-0.3000
0.0707	0.0953	-0.1717	-0.7763	0.4379	-0.2134	0.4259	-0.2830
0.0821	0.0870	-0.3589	-0.6332	0.4142	-0.1844	0.4106	-0.2664
0.0544	0.0958	-0.2395	-0.6963	0.3857	-0.1966	0.3944	-0.2503
0.0729	0.0929	-0.2526	-0.7752	0.3673	-0.1624	0.3773	-0.2346
0.0950	0.0922	-0.2694	-0.8829	0.3541	-0.1245	0.3596	-0.2193
0.0841	0.0963	-0.1954	-0.9472	0.3311	-0.1208	0.3411	-0.2044
0.0604	0.1418	0.1807	-0.9934	0.3537	-0.2413	0.3221	-0.1898
0.0459	0.1030	0.0486	-1.0232	0.2691	-0.1442	0.3026	-0.1756
0.0566	0.0994	0.1330	-1.0538	0.2297	-0.1416	0.2826	-0.1617
0.0600	0.1364	0.2909	-1.0689	0.2690	-0.2077	0.2623	-0.1481
0.0886	0.1058	-0.1249	-1.1612	0.2352	-0.0464	0.2415	-0.1348
0.0938	0.1291	0.7688	-1.3627	0.2316	-0.2148	0.2204	-0.1217
0.0644	0.1038	0.1290	-1.2993	0.1722	-0.0504	0.1991	-0.1089
0.0476	0.1057	0.1027	-1.1562	0.1381	-0.0696	0.1775	-0.0962
0.0520	0.1062	0.0437	-1.1642	0.1218	-0.0444	0.1558	-0.0838
0.0482	0.1085	0.0373	-1.1714	0.1048	-0.0331	0.1338	-0.0715
0.0459	0.1068	0.2225	-1.2208	0.0664	-0.0523	0.1117	-0.0594
0.0506	0.1067	0.0754	-1.1983	0.0529	-0.0124	0.0895	-0.0474
0.0501	0.1028	0.3638	-1.3393	0.0172	-0.0344	0.0672	-0.0354
0.1538	0.1563	0.6713	-1.1964	0.0672	-0.1758	0.0449	-0.0236
0.1004	0.1018	-0.1596	-1.2148	-0.0056	0.0846	0.0224	-0.0118
0.0500	0.1064	0.1125	-1.1667	-0.0471	0.0168	0.0000	0.0000
0.0458	0.1093	0.1807	-1.1748	-0.0682	0.0121	-0.0224	0.0118
0.0574	0.1087	0.2443	-1.1616	-0.0998	0.0072	-0.0449	0.0236
0.0876	0.1025	-0.1417	-1.1751	-0.0964	0.1181	-0.0672	0.0354
0.0536	0.1113	0.2544	-1.1554	-0.1389	0.0265	-0.0895	0.0474
0.1246	0.1015	-0.3199	-1.1623	-0.1241	0.1834	-0.1117	0.0594
0.1029	0.1397	0.5379	-1.1882	-0.1354	-0.0314	-0.1338	0.0715
0.1528	0.1508	0.6889	-1.1684	-0.1497	-0.0688	-0.1558	0.0838
0.1325	0.1428	0.6287	-1.1540	-0.1857	-0.0360	-0.1775	0.0962
0.0524	0.1016	0.4751	-1.3319	-0.2488	0.0924	-0.1991	0.1089
0.2028	0.1700	0.6655	-1.0275	-0.2078	-0.0807	-0.2204	0.1217
0.0569	0.1086	0.2148	-1.0429	-0.2954	0.1164	-0.2415	0.1348
0.0475	0.1062	0.0797	-0.9840	-0.3095	0.1528	-0.2623	0.1481
0.0636	0.0969	0.0128	-1.0107	-0.3317	0.2022	-0.2826	0.1617

0.0891	0.1117	0.3299	-0.9420	-0.3726	0.1204	-0.3026	0.1756
0.0514	0.0999	-0.0556	-0.9060	-0.3579	0.2266	-0.3221	0.1898
0.4038	0.0494	-1.3886	-0.9036	-0.2629	0.6006	-0.3411	0.2044
0.0513	0.0975	0.0046	-0.8935	-0.3967	0.2547	-0.3596	0.2193
0.0445	0.0975	0.1685	-0.9909	-0.4142	0.2596	-0.3773	0.2346
0.0834	0.1365	0.4490	-0.9924	-0.3781	0.1686	-0.3944	0.2503
0.0452	0.0911	0.3539	-1.0584	-0.4531	0.2819	-0.4106	0.2664
0.0521	0.0898	0.3739	-0.9779	-0.4776	0.2895	-0.4259	0.2830
0.1347	0.1224	0.9440	-1.1015	-0.4465	0.1654	-0.4403	0.3000
0.1018	0.1231	0.6389	-0.8781	-0.4768	0.2182	-0.4535	0.3174
0.0665	0.0767	0.2623	-0.8194	-0.5124	0.3825	-0.4655	0.3353
0.1030	0.0671	0.8665	-1.0245	-0.5707	0.3126	-0.4762	0.3536
0.0800	0.0393	0.7194	-1.2363	-0.5576	0.4068	-0.4854	0.3723
0.0714	0.0718	0.5471	-0.7203	-0.5625	0.4079	-0.4930	0.3914
0.0725	0.0669	0.5157	-0.6517	-0.5624	0.4461	-0.4989	0.4109
0.0975	0.1053	0.8043	-0.6011	-0.5576	0.3501	-0.5029	0.4308
0.1013	0.0488	0.5554	-0.5532	-0.5705	0.5282	-0.5049	0.4510
0.1242	0.1264	0.8363	-0.4469	-0.5398	0.3523	-0.5047	0.4714
0.0918	-0.0210	0.9056	-1.2671	-0.5732	0.5501	-0.5021	0.4920
0.0877	0.0830	0.9082	-0.2606	-0.5848	0.5128	-0.4971	0.5127
0.1259	0.0837	1.0784	-0.1476	-0.6148	0.5455	-0.4895	0.5334
0.1743	0.1590	0.9595	-0.1319	-0.5175	0.3841	-0.4791	0.5541
0.1462	0.1651	0.7773	0.1237	-0.5265	0.4415	-0.4659	0.5745
0.1826	0.1907	0.6181	0.2552	-0.4992	0.4188	-0.4498	0.5945
0.1450	0.1849	0.6558	0.3718	-0.4932	0.4834	-0.4306	0.6141
0.1137	0.1622	1.0386	0.4291	-0.5184	0.6039	-0.4084	0.6330
0.1399	0.2025	0.5357	0.5889	-0.4424	0.5243	-0.3832	0.6511
0.1227	0.1863	1.0274	0.6740	-0.4778	0.6714	-0.3551	0.6682
0.1200	0.2194	0.5469	0.8431	-0.4201	0.6121	-0.3241	0.6841
0.1270	0.2098	0.9161	0.9267	-0.4106	0.7397	-0.2903	0.6987
0.1317	0.2321	0.7629	1.0368	-0.3840	0.7333	-0.2540	0.7117
0.1332	0.2504	0.5333	1.1545	-0.3486	0.7225	-0.2155	0.7231
0.1276	0.2590	0.3309	1.2789	-0.3025	0.7322	-0.1749	0.7326
0.1939	0.2841	0.4057	1.3927	-0.3184	0.7956	-0.1327	0.7401
0.1329	0.2733	0.0413	1.4719	-0.2220	0.7433	-0.0892	0.7456
0.1835	0.2610	-0.6745	1.5432	-0.1348	0.5889	-0.0448	0.7489
0.1845	0.2632	-0.6787	1.5964	-0.1020	0.5963	0.0000	0.7500
0.1595	0.2837	0.0948	1.5201	-0.1097	0.7089	0.0448	0.7489
0.2120	0.2541	-0.6309	1.6071	-0.0021	0.5542	0.0892	0.7456

0.1584	0.2752	0.1010	1.5958	-0.0209	0.7011	0.1327	0.7401
0.1393	0.2626	0.3122	1.5938	0.0363	0.7189	0.1749	0.7326
0.1286	0.2495	0.4046	1.5942	0.0877	0.7082	0.2155	0.7231
0.1666	0.2146	-0.0792	1.6112	0.1853	0.5600	0.2540	0.7117
0.1515	0.2002	-0.0205	1.6510	0.2290	0.5601	0.2903	0.6987
0.2145	0.1764	-0.2629	1.6798	0.2838	0.4735	0.3241	0.6841
0.1180	0.2027	0.5876	1.6846	0.2371	0.6650	0.3551	0.6682
0.1005	0.1838	0.7272	1.7056	0.2851	0.6727	0.3832	0.6511
0.1759	0.1207	0.0523	1.7179	0.4080	0.4571	0.4084	0.6330
0.0584	0.1161	0.5738	1.7131	0.4218	0.5563	0.4306	0.6141
0.1365	0.1177	1.3899	1.6928	0.4205	0.7278	0.4498	0.5945
0.0908	0.0964	1.2641	1.6868	0.4464	0.6632	0.4659	0.5745
0.0944	0.0742	1.3391	1.6652	0.4733	0.6482	0.4791	0.5541
0.1024	0.0631	1.3475	1.7487	0.5062	0.6344	0.4895	0.5334
0.1358	0.0622	1.4253	1.8921	0.5409	0.6413	0.4971	0.5127
0.1164	0.0202	0.9419	2.0561	0.6154	0.4654	0.5021	0.4920
0.1702	0.0733	1.4667	2.1930	0.5868	0.6205	0.5047	0.4714
0.0254	0.0713	0.8303	2.3864	0.5225	0.4693	0.5049	0.4510
0.1211	0.0470	1.0412	2.5755	0.6234	0.4193	0.5029	0.4308
0.1502	0.0533	1.1001	2.7702	0.6475	0.3887	0.4989	0.4109
0.1334	0.0686	1.0424	2.9142	0.6255	0.3757	0.4930	0.3914
0.1519	0.0611	1.1738	2.9207	0.6355	0.3489	0.4854	0.3723
0.1716	0.1267	1.1761	2.9448	0.5788	0.4910	0.4762	0.3536
0.1425	-0.1577	0.4121	0.8737	0.5603	0.4417	0.4655	0.3353
0.1559	0.1006	1.0547	-2.9404	0.6071	0.2905	0.4535	0.3174
0.1442	0.1604	0.4820	-2.7914	0.5189	0.4208	0.4403	0.3000
0.1737	0.1793	0.1836	-2.5899	0.5150	0.4321	0.4259	0.2830
0.1883	0.1854	-0.1751	-2.4008	0.4942	0.4352	0.4106	0.2664
0.0888	0.1304	0.1961	-2.2291	0.4745	0.2120	0.3944	0.2503
0.0872	0.1279	0.1231	-2.1265	0.4477	0.1833	0.3773	0.2346
0.1648	0.1566	-0.7640	-2.0468	0.3937	0.3806	0.3596	0.2193
0.1556	0.1436	-0.8723	-1.9610	0.3562	0.3593	0.3411	0.2044
0.2546	0.1201	-1.4269	-1.8492	0.2719	0.4394	0.3221	0.1898
0.0572	0.1328	-0.5529	-1.7717	0.3536	0.2016	0.3026	0.1756
0.1916	0.1204	-1.2429	-1.7573	0.2683	0.3528	0.2826	0.1617
0.0554	0.1329	-0.5649	-1.7888	0.2990	0.1896	0.2623	0.1481
0.2981	0.1587	-1.4284	-1.8611	0.2582	0.4324	0.2415	0.1348
0.1306	0.1080	-0.7868	-1.8896	0.1499	0.2317	0.2204	0.1217
0.2006	0.1429	-0.9137	-1.9919	0.1634	0.3062	0.1991	0.1089

0.2304	0.1579	-0.8882	-2.0627	0.1497	0.3250	0.1775	0.0962
0.2651	0.1429	-0.9254	-2.1685	0.0464	0.3253	0.1558	0.0838
0.1961	0.1635	-0.3991	-2.2885	0.1007	0.2649	0.1338	0.0715
0.1681	0.1496	-0.2253	-2.3724	0.0543	0.2174	0.1117	0.0594
0.1268	0.1405	-0.0182	-2.4176	0.0404	0.1643	0.0895	0.0474
0.0959	0.1278	0.1149	-2.4509	0.0134	0.1149	0.0672	0.0354
0.0801	0.1236	0.2041	-2.4748	-0.0040	0.0870	0.0449	0.0236
0.1038	0.1297	0.1521	-2.4936	-0.0320	0.1002	0.0224	0.0118