DESENVOLVIMENTO DE UMA BANCADA EXPERIMENTAL PARA TESTES DE VEÍCULOS TERRESTRES AUTÔNOMOS

Gabriel Rodrigues de Lira
José Guilherme Tavares Monteiro

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação, da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro de Controle e Automação.

Orientador: Alessandro Jacoud Peixoto D.Sc.

Rio de Janeiro
Fevereiro de 2017

DESENVOLVIMENTO DE UMA BANCADA EXPERIMENTAL PARA TESTES DE VEÍCULOS TERRESTRES AUTÔNOMOS

Gabriel Rodrigues de Lira
José Guilherme Tavares Monteiro

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

Prof. Alessandro Jacoud Peixoto, D.Sc.

Prof. Ramon Romankevicius Costa, D.Sc.

Prof. Lilian Kawakami Carvalho, D.Sc.

Diego Pereira Dias, M.Sc.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2017

José Guilherme Tavares Monteiro, Gabriel Rodrigues de Lira

Desenvolvimento de uma bancada experimental para testes de veículos terrestres autônomos/Gabriel Rodrigues de Lira José Guilherme Tavares Monteiro. – Rio de Janeiro: UFRJ/Escola Politécnica, 2017.

XIV, 110 p.: il.; 29, 7cm.

Orientador: Alessandro Jacoud Peixoto D.Sc.

Projeto de graduação – UFRJ/Escola Politécnica/Curso de Engenharia de Controle e Automação, 2017.

Referências Bibliográficas: p. 106 – 110.

1. Robôs Autônomos. 2. Controle Linear. 3. Controle Angular. 4. *Cruise Control*. 5. Desenvolvimento de *Software*. 6. Visão Computacional. 7. Robótica. I. D.Sc., Alessandro Jacoud Peixoto. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

A Deus, nossa família e nossos amigos.

Agradecimentos

Gostaríamos de agradecer a Deus, por permitir que chegássemos até aqui.

À nossa família pelo incentivo, apoio, ajuda e compreensão durante esta importante etapa de nossas vidas.

Aos amigos e companheiros de curso que trilharam conosco este caminho, nos apoiando nos momentos difíceis e compartilhando muitos momentos alegres.

Ao nosso orientador, Alessandro Jacoud Peixoto, pela dedicação e orientação durante o desenvolvimento deste projeto.

Ao professor (vô) Odilon Antonio Paula Tavares pela carinhosa contribuição em revisar e assistir a construção do texto deste trabalho.

Aos professores Ramon Romankevicius Costa, Lilian Kawakami Carvalho e Diego Pereira Dias pela disposição em participar da banca examinadora e contribuir para o enobrecimento deste trabalho.

Resumo do Projeto de Graduação apresentado à POLI/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação.

DESENVOLVIMENTO DE UMA BANCADA EXPERIMENTAL PARA TESTES DE VEÍCULOS TERRESTRES AUTÔNOMOS

Gabriel Rodrigues de Lira
José Guilherme Tavares Monteiro

Fevereiro/2017

Orientador: Alessandro Jacoud Peixoto D.Sc.

Curso: Engenharia de Controle e Automação

Neste trabalho é realizado o desenvolvimento de uma bancada experimental para avaliar esquemas de controle para robôs terrestres autônomos visando potenciais aplicações no controle de um carro que se movimente de forma autônoma no trânsito de uma cidade. Na primeira etapa deste projeto, especifica-se todo o *hardware* da bancada, a aquisição dos robôs, a montagem de toda a bancada, assim como o desenvolvimento e codificação do *firmware* de comunicação dos robôs, a modelagem matemática e o desenvolvimento do simulador dos robôs. Como segunda etapa destaca-se o desenvolvimento de um *software* que engloba diversas funcionalidades, como: os algoritmos para obtenção da posição e orientação de cada robô, baseados em visão computacional, os algoritmos de controle, algoritmos de aquisição e visualização de dados e a interface com o usuário. A estratégia de controle desenvolvida visa permitir que o veículo evite colisões em um cruzamento, por exemplo, que permaneça a uma distância segura do veículo a sua frente em uma velocidade de cruzeiro pré estabelecida (*cruise control*) e que permita que o veículo realize ultrapassagens seguras. Os diversos sensores necessários em uma aplicação real são emulados pela informação de posição e orientação de cada veículo obtidas por meio de visão computacional. O esquema de controle de cada veículo utiliza apenas as informações do veículo mais próximo afim de emular uma situação real. Da modelagem completa de cada veículo, robôs terrestres tipo uniciclo, um modelo simples é obtido e utilizado para projeto do controle. O modelo completo constitui o simulador desenvolvido. Os resultados experimentais com dois robôs evidenciam a eficácia do esquema de controle utilizado.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Control and Automation Enginner.

DEVELOPMENT OF AN EXPERIMENTAL BENCH FOR AUTONOMOUS LAND VEHICLE TESTS

Gabriel Rodrigues de Lira
José Guilherme Tavares Monteiro

February/2017

Advisor: Alessandro Jacoud Peixoto D.Sc.

Course: Control and Automation Engineering

In this work the development of an experimental bench to evaluate control schemes for autonomous terrestrial robots aiming at potential applications in the control of a car that moves autonomously in the traffic of a city is carried out. In the first stage of this project, we specify the entire hardware, the acquisition of the robots, the assembly of the whole workbench, as well as the development and coding of the robots' communication firmware, modeling mathematics and the development of the robot simulator. The second step is the development of a software that encompasses several functionalities, such as algorithms for obtaining the position and orientation of each robot, based on computer vision, control algorithms, algorithms for acquisition and visualization of data and the user interface. The control strategy developed is intended to enable the vehicle to avoid collisions at a crossing, for example, that remains at a safe distance from the vehicle ahead of it at a pre-established cruise control speed and to allow the safe overtaking. The various sensors required in a real application are emulated by the position and orientation information of each vehicle obtained through computer vision. The control scheme of each vehicle uses only the information of the nearest vehicle in order to emulate a real situation. From the complete modeling of each vehicle, unicycle-type ground robots, a simple model is obtained and used for control design. The complete model is the developed simulator. The experimental results with two robots show the effectiveness of the control scheme used.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A linha de produção de automóveis em larga escala e a preços acessíveis foi lançada por Ransom Olds em sua fábrica Oldsmobile em 1902 e mais tarde (1914) expandida por Henry Ford trazendo diversas mudanças para a sociedade.[1]

Antes, a produção de um carro da Ford precisava de 12,5 horas-homem. Depois desse avanço somente 1 hora e 33 minutos eram o suficiente para produzir o mesmo carro. Além do aumento relevante na produtividade (8 vezes maior) houve também a redução da necessidade de recursos humanos, a redução de erros causados por falha humana, redução de acidentes de trabalho, além de alavancar o desenvolvimento em áreas relacionadas com a montagem de automóveis (ignição elétrica, suspensão independentes etc).

Contudo, junto com o avanço promovido por esse novo paradigma, vieram também diversos problemas. O aumento gigantesco no número de veículos circulando nas cidades somado com a falta de planejamento urbano para comportar um número cada vez maior de automóveis criou os trânsitos quiilométricos observados principlamente nas grandes cidades (em São Paulo por exemplo em 20/04/2016 foi registrado 317km de congestionamento [2]). O crescimento anual do trânsito nas cidades acarreta em diversos outros problemas como:

- Provoca estresse na população;
- Número de acidentes de trânsito crescente (a morte causada por acidentes de trânsito é uma das dez maiores causas de morte no mundo, segundo pesquisa feita pela OMS em Maio de 2016 [3]);
- Impacto Ambiental - promove a crescente poluição sonora e do ar;
- Prejuízo financeiro gigantesco (causado por tempo oscioso, problemas de saúde e aumento no gasto de combustível) principalmente para as grandes cidades. Um exemplo disso é que em 2012 o gasto do estado de São Paulo devido ao trânsito foi de 40 bilhões de reais [4];

O conceito de carros autônomos, também chamados de carros robóticos, veículos terrestres capazes de transportar pessoas ou objetos sem que exista um condutor humano envolvido no processo de deslocamento, surgiu como uma possibilidade de solução, ou minimização de diversos desses problemas causados pelo trânsito excessivo. Isso uma vez que os sistemas automáticos desses veículos possibilitam:

- Redução do número de acidentes causados por falhas humanas;
- Diminuição considerável dos engarrafamentos, pois a distância entre veículos poderia ser reduzida uma vez que o tempo de reação de um sistema computadorizado para frenagens é menor do que o dos humanos. Além disso, a utilização maciça de carros autônomos possibilitaria a redução da quantidade de semáforos, tendo em vista a possibilidade desses veículos conseguirem atravessar cruzamentos sem que haja a necessidade de semáforos para evitar colisões[5, 6];
- Aumento na produtividade devido a uma redução do tempo em engarrafamentos, já que esses seriam menores e os condutores passariam a usar o tempo dentro dos veículos realizando outras atividades no lugar da preocupação com a direção;
- Redução do estresse em geral;
- Diminuição da poluição do ar e sonora nas cidades, tendo em vista a redução nos engarrafamentos e a otimização dos recursos existentes nos veículos (por exemplo melhor aproveitamento do sistema de frenagem e aceleração devido a utilização adequada das partes que compõem esses sistemas)[7];

Isso sem contar com outros benefícios que surgem como a possibilidade de deficientes (motores ou visuais) de utilizarem um automóvel sem a necessidade do auxílio de terceiros.

Para uma implementação eficaz desse conceito, uma série de desafios se apresentam, tais como escolha dos sensores a serem utilizados com o intuito de "observar" obstáculos, sejam eles outros veículos, pedestres ou ciclistas, ou até mesmo objetos como cones, carros enguiçados etc a fim de obter uma noção dos objetos que influenciam o comportamento do veículo. Além dessa "visão do mundo" os sensores existem também para auxiliar na resolução de problemas como localização global, definição de trajetos a serem tomadas, decisões como ultrapassagem ou atravessar cruzamentos ou parada.

Somado a esses problemas descritos temos também o problema de *software*, ou seja, o sistema responsável por reunir todas as informações obtidas pelos sensores e escolhas do usuário (como destino da viagem por exemplo) e gerenciar a tomada de decisões de forma a obter o melhor resultado possível.

Por fim, entra em discussão também as implicações legais desse novo paradigma que está cada vez mais próximo de se tornar realidade e, eventualmente, comum na sociedade, visto que empresas como Google, Uber e Tesla entre outras já possuem veículos autônomos ou semi-autônomos sendo usados pela sociedade.

1.1 Objetivo

Como objetivo geral, trata-se do desenvolvimento de uma bancada experimental para avaliar esquemas de controle para robôs terrestres autônomos visando o controle de carros que se movimente de forma autônoma no trânsito de uma cidade. Como objetivos específicos destacam-se:

i Desenvolvimento de *software* para obtenção de posição e orientação dos robôs, para comunicação e controle;

ii Desenvolvimento de um esquema de controle que assegure que o veículo evite colisões, que permaneça a uma distância segura do veículo a sua frente em uma velocidade de cruzeiro pré estabelecida (*cruise control*) e que permita que o veículo realize ultrapassagens seguras.

1.2 Revisão Bibliográfica

Controle realimentado de sistemas dinâmicos tem como principal principio o fato de que a saída de um determinado sistema pode ser medida e essa medida pode ser interpretada por um controlador de forma a fazer com que esta variável medida atinja um valor desejado.

Um dos sistemas de controle que pode ser mais facilmente encontrado em carros modernos é o *CruiseControl* (Controle de Cruzeiro ou Controle de Velocidade de Cruzeiro). Sua função é regular o quanto o acelerador de um veículo é acionado de forma a fazer com que este veículo se locomova a uma velocidade constante [8]. Um sistema de Controle de Cruzeiro Adaptativo (*Adaptative Cruise Control - ACC*) é uma evolução do sistema de Cruise Control original. Este sistema, além de manter a velocidade do veículo constante, monitora a estrada a frente do mesmo de forma a evitar colisões e manter uma distância considerada segura do veículo a sua frente, além de atuar com um tempo de resposta menor do que uma pessoa no caso de uma frenagem de emergência[8, 9].

A utilização de computadores tem permitido a aplicação de métodos de projeto cada vez mais sofisticados além da implementação de leis de controle bastante complexas [10]. Várias técnicas avançadas e algorítmos de controle para diferentes tipos de robôs podem ser encontradas em [11]. Além disso, diferentes técnicas podem ser empregadas para

estimar a posição de um robô móvel. Um resumo destas técnicas pode ser encontrado em [12], em particular, por meio de Visão Computacional.

Entende-se por Visão Computacional a transformação de dados adquiridos através de imagens (foto ou vídeo) em decisões ou novas representações. Estas transformações podem ser desde a afirmação de que uma cor está presente na imagem até o fato de que existe uma pessoa com os braços em determinada posição se movimentando em uma determinada direção, por exemplo. A visão computacional se assemelha com o processo da visão humana, onde captamos informações visuais e transformamos em ações, decisões ou até mesmo informações [13].

Com o avanço da robótica, inteligencia artificial e muitos outros campos de pesquisas que tem tomado a atenção da comunidade científica, vem surgindo também um interesse cada vez maior no desenvolvimento de técnicas de visão computacional. Com isso diversos trabalhos tem sido publicados usando as mais diversas formas de identificação e segmentação de objetos por meio de imagens capturadas por câmeras 2D. Essas técnicas geralmente são separadas em dois conjuntos devido à semelhança em seus algorítimos, onde o primeiro conjunto se destaca pelas técnicas que utilizam a lógica de buscas por regiões ou agrupamentos de *pixels* vizinhos que contenham alguma(s) propriedade(s) em comum como por exemplo textura, cor e intensidade. Já a segunda coleção de técnicas são caracterizadas por evitarem a etapa de segmentação da imagem em grupamentos de pixel (uma vez que isso pode vir a ser uma fonte de erros) e buscarem regiões com grandes variações na imagem sem necessitarem definir as fronteiras de um determinado objeto. Geralmente o segundo grupo possui tempo de execução maior do que o primeiro. Entretanto, esse tempo é compensado por resultados melhores.

O primeiro exemplo para técnicas do primeiro grupo podemos encontrar em [14, 15] é uma técnica que usa regiões de crescimento (*region growing*). Nessa técnica algumas regiões da imgem são selecionadas (essas regiões são chamadas de *seeds*) e em seguida o algorítimo vai expandindo-as segundo um critério de semelhança entre os *pixels* próximos até que enfim as regiões desejadas estejam bem definidas. Uma segunda estratégia bastante implementada é a por divisão e fusão de regiões (*split and merge*) [16] onde a imagem é inicializada com uma única região e em seguida essa região é particionada e reagrupada em outras regiões até que o critério de homogeneidade definido seja alcançado. Para terminar esse grupo um terceiro método se apresenta. Esse, comumente chamado de *Active Contours Model* ou Modelos de Contornos Ativos [17] possui o seguinte procedimento. Um contorno inicial na imagem é definido. Esse contorno passará por uma série de deformações que buscam minimizar uma função de energia global. Essa função é composta pela energia interna (almejando a preservação do formato original) e pela energia externa que tem como finalidade encontrar as zonas de grande gradiente. Ao final da minimização é esperado que o contorno final corresponda as fronteiras do objeto buscado.

Já para o segundo grupo de técnicas, os exemplos mais comuns são:

- *Scale-Invariant Feature Transform*, geralmente chamado de SIFT [18] é uma tática onde inicialmente é feito uma detecção de pontos de interesse por intermédio de subtrações de um pirâmide de imagens filtradas e com escalas diferentes. Em seguida essas regiões são caracterizadas por histogramas de gradientes que são invariantes as mudanças de escala, distorções, orientação e até mesmo algumas variações de luminosidade.

- SURF (Speeded-Up Robust Feature)[19]. Por utilizar da técnica de imagens integrais para representação da imagem, essa estratégia é em geral mais rápida do que a descrita no item anterior. Inspirada no SIFT, e utilizando os *wavelets* de Harr para assegurar a invariância à rotação nas vizinhanças, essa estratégia se baseia na representação da distribuição de intensidade entorno dos pontos de interesse.

- Detectores:

  - Harris [20]: Em busca das bordas dos objetos a serem detectados, esse método calcula as variações de intensidade nas diferentes direções no entorno do ponto considerado. Uma característica desse detector é robustez em relação às mudanças de escala, iluminação e rotação.

  - Kadir-Brady [21] Baseado na entropia da imagem, esse detector procura pelas regiões mais raras na imagem. Atualmente algumas melhorias foram adicionadas a esse método a fim de torná-lo menos sensível às alterações de orientação e escala do objeto buscado.

Além desses existem alguns métodos de saliência visual [22], [23] e [24] que se baseiam no comportamento do cérebro humano buscando as regiões mais interessantes na imagem. Existe também uma estratégia que visa os pixels que possuem a maior probabilidade de conterem o máximo de uma determinada função objetivo definida a priori. Finalmente, é importante ressaltar que com a redução do custo de câmeras de visão 3D o desenvolvimento de técnicas de segmentação e identificação de objetos num mundo tridimensional tem crescido bastante [25] e [26].

Existem diversas bibliotecas e *frameworks* para se utilizar Visão Computacional. Dentre as principais, podemos destacar: *OpenCV (Open Computer Vision)*, MatLab, Accord.Net e EmguCV. A biblioteca OpenCV é uma biblioteca opensource, funciona com Windows, Linux e Mac OS X e existem interfaces para sua utilização com MatLab, Python, Ruby, C++ e outras linguagens. Um dos principais propósitos do OpenCV é ser uma biblioteca de visão computacional voltada para a eficiência computacional de forma a possibilitar a sua aplicação em sistemas em tempo real [13].

Uma alternativa para projetar e implementar algoritmos de visão computacional é utilizar a Programação Orientada à Objetos, uma prática de criação de uma arquitetura modular do sistema para alcançar flexibilidade do mesmo. Ao contrário da programação

procedural, a programação orientada à objetos faz com que os comportamentos específicos de cada parte da aplicação estejam encapsulados em classes diferentes, facilitando o entendimento, escrita, manutenção do código, além de possibilitar que o trabalho de desenvolvimento de software em equipe seja feito de forma clara e organizada [27].

A escalabilidade de um *software* sempre foi uma das principais preocupações quando se fala em desenvolvimento e projeto de algoritmos. Para alcançar este resultado, um dos recursos mais utilizados pela indústria é o OpenMP (*Open Multi-Processing*). A solução do OpenMP se destaca das demais técnicas disponíveis não somente pela performance, mas também pela facilidade de seu uso [28].

1.3 Organização do Trabalho

Este trabalho será organizado da seguinte maneira:

Equipamentos utilizados
Com a proposta de enfrentar e propor soluções para os problemas enfrentados no trânsito nas cidades em um ambiente controlado e em menor escala, apresentaremos neste capítulo o *hardware* que será utilizado nos experimentos que serão conduzidos com robôs móveis terrestres, bem como o *hardware* para a aquisição de dados (sensoriamento) e transmissão de dados.

Modelagem
Neste capitulo apresentaremos a modelagem matemática do problema proposto, ou seja, apresentaremos a modelagem de um robô móvel terrestre não-holonômico acionado por rodas (uniciclo).

Controle proposto para os robôs móveis
Serão apresentadas as técnicas que serão empregadas para o controle do sistema proposto, bem como o detalhamento da construção de um controle capaz de fazer com que o robô utilizado siga uma determinada trajetória a uma dada velocidade.

Simulador
Detalhamento da construção de um simulador para experimentar o controle proposto juntamente com a modelagem realizada de forma a comprovar a sua eficácia, estabilidade e robustez, bem como mostraremos e analisaremos os resultados das simulações realizadas.

Implementação do Programa
Decisões e os detalhes da implementação do programa construído com o objetivo de tornar possível a experimentação do controle de robôs móveis terrestres através do auxilio

da visão computacional para verificarmos as soluções propostas para os problemas discutidos neste trabalho.

Resultados
Resultados dos experimentos realizados, os dados coletados, bem como a análise dos mesmos.

Conclusão
Conclusões que puderam ser tiradas da elaboração deste trabalho, contribuições dadas e proposta para trabalhos futuros.

Capítulo 2

Equipamentos Utilizados

Com o intuito de desenvolver uma plataforma para testar os problemas apresentados no capítulo 1 e poder testar as soluções que serão apresentadas pelos autores, a proposta deste trabalho é controlar um robô móvel de pequena escala através de um programa de visão computacional.

2.1 Robô

No caso escolheu-se utilizar um robô holonomico uniciclo. Para evitar os problemas oriundos da elaboração e construção de um novo robô optou-se pela aquisição de um modelo comercial e mundialmente conhecido, o Pololu *3pi* [29]. O robô 3pi é mostrado na figura 2.1 e é uma plataforma móvel robusta para diversos tipos de aplicações, principalmente educacionais. O mesmo consiste em um robô uniciclo em formato circular com diâmetro aproximado de 94mm. O mesmo é alimentado por 4 pilhas AAA e possui dois motores com caixa de redução. Além disso, possui LCD, buzzer, 5 sensores de reflectância, potenciômetro e botões, todos estes programáveis. O robô é feito de forma que o seu chassi e sua placa de circuito sejam a mesma coisa, tornando-o leve, simples e robusto, sem muitas peças que se desmontam ou fios que possam quebrar. O microcontrolador é um Atmega328P programável através de um conector ISP[30].

2.2 Comunicação

Embora o robô *3pi* seja uma das mais completas soluções do mercado, o mesmo não dispõe de um sistema para comunicação sem-fio. Como no caso do projeto aqui apresentado, a proposta é controlar diversos robôs através de um programa de visão computacional. Para tanto, foi necessária a escolha e adaptação dos robôs de forma a prover um método de comunicação sem fio robusto, que possibilitasse o envio de informações de um

Figura 2.1: Robô Pololu 3pi

único transmissor para diversos robôs diferentes. Algumas possibilidades de comunicação sem fio foram analisadas e ponderadas.

A primeira delas foi a comunicação IR (infravermelho ou *infrared*). Nesse tipo de comunicação um emissor de infravermelho é posicionado na direção de um receptor sem que haja nenhum obstáculo entre eles, tendo em vista que o sinal transitado é direcional, ou seja, ele se propaga somente na direção em que o emissor está apontado. Assim, ao gerarmos uma mensagem podemos enviá-la usando o emissor de sinal e recebê-la através do receptor, mas nunca o caminho inverso. Geralmente essa técnica de comunicação possui uma implementação simples se comparada a outras, e o alcance do sinal não é muito longo.

A comunicação via RF (rádio frequência) por sua vez, não depende que o emissor e o receptor estejam direcionados um para o outro uma vez que o sinal é propagado omnidirecionalmente e também não exige a inexistência de obstáculos no caminho do sinal uma vez que esse é capaz de transitar através de obstáculos como paredes, móveis entre outros.

Este tipo de comunicação é baseada na indução de um campo eletromagnético, isto é, ao transitar uma corrente elétrica por um condutor, um campo eletromagnético é gerado ao redor desse na mesma frequência que a corrente. Esse campo gerado é capaz de induzir uma corrente elétrica de mesma frequência em um outro condutor que esteja no alcance desse campo mesmo que o condutor esteja atrás de um obstáculo, tendo em vista que esse tipo de onda é capaz de se propagar através da maioria dos materiais. Dessa forma, se produzirmos uma corrente elétrica de frequência variável num componente que chamaremos de emissor e colocarmos um outro componente, que pode ser denominado de emissor, na área afetada por esse campo, então somos capazes de transmitir uma mensagem sem a necessidade de contato físico entre os componentes. É interessante ressaltar que apesar de termos chamado um componente de emissor e o outro de receptor, na realidade eles podem ser transreceptores, ou seja, por serem condutores iguais, ambos possuem a capa-

cidade de enviar e receber mensagem diferentemente da transmissão IR, onde o emissor só pode enviar e o receptor só consegue receber.

Uma terceira possibilidade é do *wifi* que também possui a característica de ser omnidirecional, ter um longo alcance e possibilitar altas taxas de transmissão.

Por fim, outra comunicação a ser vislumbrada foi o *bluetooth*. Sendo também uma forma de comunicação via radiofrequência essa técnica utiliza protocolos específicos para transitar seus dados.

Para resumir as vantagens e desavantagens de cada meio de comunicação a tabela 2.1 foi construída:

Comunicação	Vantagens	Desvantagens
IR	Baixo Custo	Distância curta Emissor e Receptor precisam estar alinhados
RF	Possui grande alcance Fácil implementação Omnidirectional	Pode sofrer interferência de outros emissores Possui baixa taxa de transmissão
Bluetooth	Possui alta taxa de transmissão Omnidirectional	Necessita ser pareada O alcance sem obstruções é geralmente de 10m
Wifi	Alta taxa de transmissão	Difícil implementação

Tabela 2.1: Vantagens e Desvantagens dos diferentes tipos de comunicação

A opção escolhida para comunicação foi a utilização dos módulos XBee [31]. Estes módulos são baseados no padrão IEEE 802.15.4 e proporcionam uma comunicação RF robusta e segura com alcance considerável. O modelo escolhido foi o XBee Pro S1 (mostrado na figura 2.2).

Figura 2.2: XBee Pro S1

Este modelo funciona com uma voltagem de alimentação e de nível lógico de 3.3V. O nível lógico do robô *3pi* é de 5V. Sendo assim, para que fosse possível a utilização do Xbee com o mesmo foi utilizada uma placa intermediária responsável por converter o nível lógico de 5V para 3.3V e vice versa, além de prover 3.3V para a alimentação do módulo. A placa utilizada foi a *XBee Explorer* da fabricante *SparkFun* [32]. Além da conversão das tensões, esta placa também fornece fácil acesso aos 4 pinos necessários à comunicação: `DIN`, `DOUT`, `5V` e `GND` e também possui LEDs indicativos de fácil acesso para indicar a operação do módulo de comunicação. Os pinos `5V` e `GND` correspondem respectivamente á alimentação principal de 5 volts e ao sinal neutro. Já os pinos `DIN` e `DOUT` são os pinos de entrada e saída serial dos dados e que serão ligados ao microcontrolador.

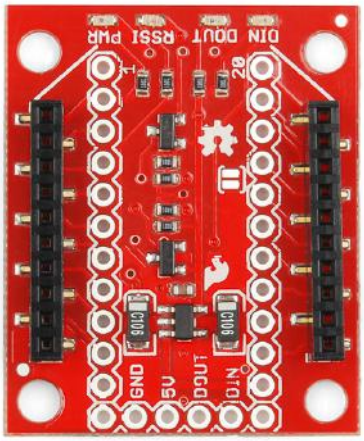
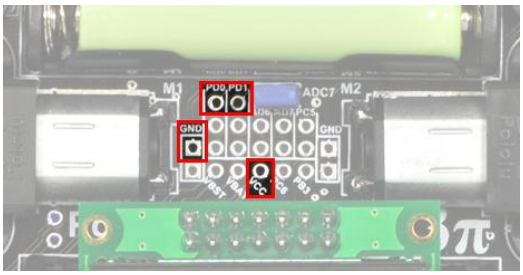
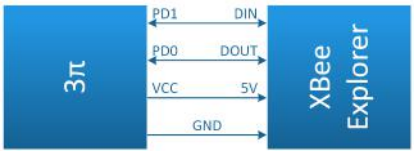


Figura 2.3: XBee Explorer Regulated

Para conectar o módulo Xbee e a placa de conversão de nível lógico ao robô *3pi* utilizamos o fato de que o robô deixa exposto na sua estrutura diversos pinos diretamente ligados ao microcontrolador justamente para facilitar modificações e *upgrades*. A figura 2.4a mostra os pinos que foram utilizados para esta adaptação. Já a figura 2.4b mostra o esquema de ligação entre estes pinos e a placa *Xbee Explorer Regulated*.



(a) Pinos do robô 3pi utilizados para conexão



(b) Esquemático de conexão dos pinos

Figura 2.4: Conexão entre robô 3pi e Xbee

Para a comunicação entre o programa desenvolvido neste trabalho (capítulo 6) e o robô, já adaptado para a utilização do módulo XBee, utilizamos um módulo, também da fabricante *SparkFun*, para conectar um XBee à uma porta USB convencional. O módulo trata-se do *SparkFun XBee Explorer Dongle* [33] e fornece uma interface serial através de uma porta COM (Windows). Este módulo é mostrado na figura 2.5

Figura 2.5: SparkFun XBee Explorer Dongle

O *hardware* do robô *3pi* permite que o microcontrolador (ATMEGA328) controle a velocidade de cada motor independentemente através *Pulse Width Modulation (PWM)*. PWM é uma técnica utilizada para se obter uma aproximação de um sinal analógico através de um sinal digital. A ideia principal consiste em chavear o sinal digital entre ligado e desligado (5V e 0V no nosso caso) de forma que a quantidade de tempo que o sinal fica ligado interfira na "média" deste sinal e, então, na aproximação analógica do mesmo. [34]

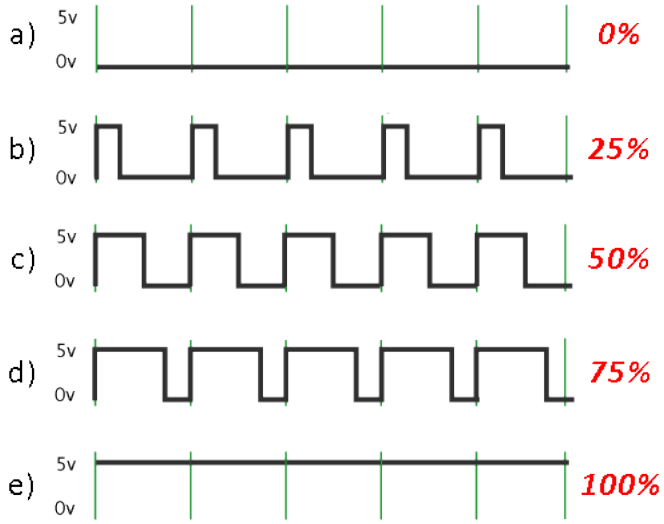


Figura 2.6: Pulse Width Modulation

A figura 2.6 mostra exemplos de sinais feitos com PWM. No exemplo 2.6.b temos o sinal digital como ligado 25% do tempo, o que daria um sinal analógico de $1,25$V. Seguindo o mesmo pensamento temos $2,5$V no exemplo 2.6.c e 5V no exemplo 2.6.e. O microcontrolador permite que o valor da largura da modulação do sinal (ou o percentual do tempo em que o sinal digital fica "ligado". seja controlado através de um byte, o que corresponde a um valor entre 0 e 255. Desta forma o exemplo 2.6.c corresponderia a um valor de 127, por exemplo.

O microcontrolador interage com os motores através de uma ponte-H. Simplificadamente falando, uma ponte-H é composta por quatro transistores. Controlando-se o sinal enviado para cada um destes transistores é possível controlar o sentido e a velocidade de rotação do motor acoplado à esta ponte.

A parte relacionada com o programa embarcado no robô e o protocolo utilizado para comunicação do mesmo serão explicitados neste trabalho na seção 6.2.7

2.3 Sensores

Para que seja possível a implementação do que é proposto neste trabalho é necessário que o sistema de sensoriamento utilizado fosse capaz de detectar o posicionamento de cada robô em relação à sua trajetória, bem como o posicionamento de outros robôs e outros objetos. Como a rede de sensores não é o foco deste trabalho escolheu-se a utilização de um único sensor, uma câmera posicionada sobre a pista de testes.

O modelo de câmera escolhido foi uma WebCam Microsoft LifeCam HD-3000 [35], mostrada na figura. Os fatores preponderantes para esta escolha foram a conexão USB, a resolução HD (1280x720) a 30FPS e principalmente o ângulo de visão de praticamente 70° que faz com que a câmera consiga capturar toda a extensão da pista quando posicionada à 2m de altura.

Figura 2.7: Microsoft LifeCam HD-3000

Para que o programa desenvolvido utilizando-se de uma imagem gerada por uma câmera com vista superior da pista seja capaz de identificar o posicionamento dos robôs que desejamos controlar e dos demais objetos que desejamos detectar a posição, utilizamos um padrão de cores (similar ao mostrado na figura 2.8) no topo de cada objeto. Desta forma, segmentando as cores e detectando o posicionamento dos objetos de cada cor é possível determinar a posição e o ângulo de cada objeto. Este procedimento será melhor explicitado na seção 6.2.1.

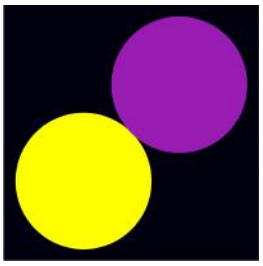


Figura 2.8: Exemplo de padrão de cores utilizado

Capítulo 3

Modelagem

A seguir serão apresentadas as considerações e cálculos a respeito da modelagem definida para aplicação no controle do sistema do robô deste projeto

A relação entre um vetor livre $\mathscr{V}$ (no plano) representado no sistema de coordenadas inerciais $(\mathscr{V})^A$ e o mesmo vetor representado no sistema de coordenadas do corpo $(\mathscr{V})^B$ é governada pela matriz de rotação em torno do eixo perpendicular ao plano de movimento, da seguinte forma:

$$(\mathscr{V})^A = R(\theta)(\mathscr{V})^B, \quad \text{onde} \quad R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix},$$

onde θ é a orientação do robô.

Verifica-se que a velocidade angular ω satisfaz

$$\dot{\theta} = \omega\,.$$

Definindo o vetor de coordenadas generalizadas $q := \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$, onde $\begin{bmatrix} x \\ y \end{bmatrix}$ é o vetor posição do centro de massa G representado nas coordenadas inerciais. Nota-se que $(a_G)^A = \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix}$ e que o vetor velocidade do centro de massa representado nas coordenadas inerciais é dado por $(v_G)^A := \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$. Pode-se escrever:

$$\dot{q} = \begin{bmatrix} (v_G)^A \\ \omega \end{bmatrix} = \mathscr{R}(\theta) \begin{bmatrix} (v_G)^B \\ \omega \end{bmatrix}.$$

Considerando a forças externas representadas no sistema de coordenadas do corpo (sistema móvel), a dinâmica do movimento de translação do centro de massa pode ser descrita

por:

$$m(a_G)^A = R(\theta)\sum(F_{ext})^B, \tag{3.1}$$

Assim, a dinâmica do robô pode ser reescrita nas coordenadas inerciais da seguinte maneira:

$$M\ddot{q} = \begin{bmatrix} R(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (F_{ext})^B \\ \sum \mathscr{M}_{extG} \end{bmatrix},$$

sendo

$$M := \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & \mathscr{I} \end{bmatrix},$$

e $\begin{bmatrix} R(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} (F_{ext})^B \\ \sum \mathscr{M}_{extG} \end{bmatrix}$ o vetor de forças generalizadas.

3.0.1 Modelo Cinemático

Consideram-se todas as forças externas representadas no sistema de coordenadas do corpo (sistema móvel). Assume-se que a componente $(x)^B$ da força de atrito na roda passiva (roda dianteira) é desprezível quando comparada com a respectiva componente da soma das forças de atrito das rodas tracionadas (rodas traseiras), denotada por $(f_{atx})^B = f_1 + f_2$, onde f_1 e f_2 são as componentes no eixo $(x)^B$ das forças de atrito das rodas esquerda e direita, respectivamente.

Para o uniciclo, há uma restrição cinemática, vito que o robô é incapaz de produzir velocidades laterais (no eixo y). A componente no eixo $(y)^B$, denotada por $(f_{aty})^B$, da soma das forças de atrito é responsável por restringir a velocidade do centro de massa do uniciclo ao eixo $(x)^B$, ou seja, $(v_G)^B = \begin{bmatrix} u \\ 0 \end{bmatrix}$, onde u. Esta restrição cinemática na velocidade do centro de massa é do tipo não-holonômica. O modelo cinemático do uniciclo pode ser então reescrito como:

$$\dot{q} = B(\theta)v, \qquad B(q) := \begin{bmatrix} R(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \tag{3.2}$$

sendo v o sinal de controle.

Capítulo 4

Controle proposto para os robôs móveis

Este capítulo destina-se à apresentação da técnica de controle empregada. Como parte do objetivo do trabalho era construir um controle capaz de fazer com que um carro seguisse uma determinada trajetória numa velocidade constante e, conforme vimos no capítulo 3, é possível se controlar o robô de forma desacoplada (translação e rotação), optou-se por utilizar o controle desta forma para que fosse possível a separação das "tarefas" de manter o veículo em uma velocidade de referência e fazer com que este veículo estivesse numa determinada angulação.

Primeiramente será apresentado o que é um controle proporcional, integral e derivativo mais conhecido como PID [36, 37]. Assim como o nome sugere essa técnica de controle se baseia em três parcelas. A primeira delas é uma resposta proporcional ao erro que quer-se zerar, ou seja, quanto maior o erro mais forte será a reação dessa parcela no controle, uma característica importante dessa parcela é que na maioria das situações reais essa parcela por se tornar muito pequena a medida que o erro diminui acaba por não conseguir zerar esse erro provocando assim um erro estacionário.

Já a parcela integral é baseada na soma dos erros anteriores, isto é quanto maior o tempo em que existir um valor de erro diferente de zero maior será a fração integral do controle sendo assim, essa parcela tem a capacidade de zerar completamente o erro pois caso o sistema viesse a ter um erro em estado estacionário essa parcela ficaria cada vez maior fazendo com que o sinal de controle aumentasse e assim conseguisse eliminar esse erro. Entretanto, a parte integral também pode provocar alguns problemas como por exemplo tornar o sistema oscilatório, isso uma vez que essa parcela vai crescendo indefinidamente assim ao atingir o valor da referência, ela não se torna zero como a parte proporcional e com isso faz com que o sistema ultrapasse o valor de referencia. Nesse instante e parcela integral começa a somar no sentido contrário ao anterior até que o erro reduza novamente, contudo, se for mal ajustado essa parcela pode fazer com que o sistema passe novamente do valor de referência e assim retorne à situação inicial do problema, promovendo assim um sistema oscilatório.

Por último a parcela derivativa é uma resposta ao sentido do movimento do sinal

de controle, ou seja, essa parte é responsável por suavizar o efeito causado pela parcela integral, o que pode tornar o sistema mais lento contudo este passa a ter uma resposta mais suave, com menos *overshoot* (situação em que a resposta do controle foi tão forte que o valor da variável controlada tenha um pico ultrapassando o valor de referência) e menos oscilatório. Aqui é necessário fazer uma ressalva em relação à parcela derivativa, dado que esta é baseada num efeito antecipativo ela é impossível de ser implementa da maneira ideal, uma vez que ela é fundamentada na previsão do futuro do sistema realimentado e dado que a previsão do futuro não é algo implementável então é necessário fazer uma aproximação do termo derivativo. Essa aproximação chamaremos de derivada suja. A imagem 4.1 ilustra um sistema com realimentação negativa usando um simples PID como controle:

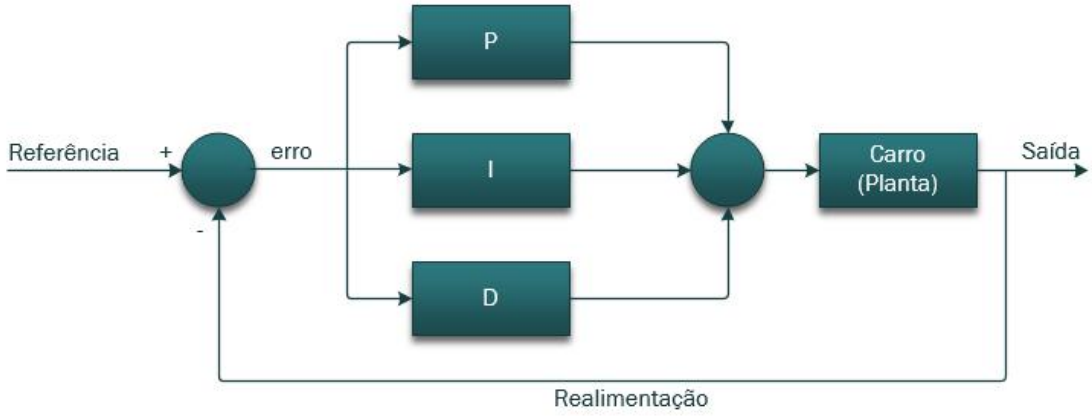


Figura 4.1: Diagrama de blocos PID numa realimentação negativa.

A figura 4.2 observada abaixo demonstra de maneira geral a estrutura implementada no controle de cada robô.Um primeiro controle, chamado de Controle de Cruzeiro (*Cruise Control*) foi empregado para controlar a velocidade linear do robô. Utilizou-se um segundo controle chamado Controle Angular para guiar o carrinho na a trajetória desejada por ajustes angulares.

4.1 Cruise Control

O Cruise Control é uma técnica de controle amplamente empregada nos veículos da atualidade. Esse tipo de controle destina-se a fazer com que a velocidade de um determinado carro se mantenha constante independentemente de alterações na declividade da

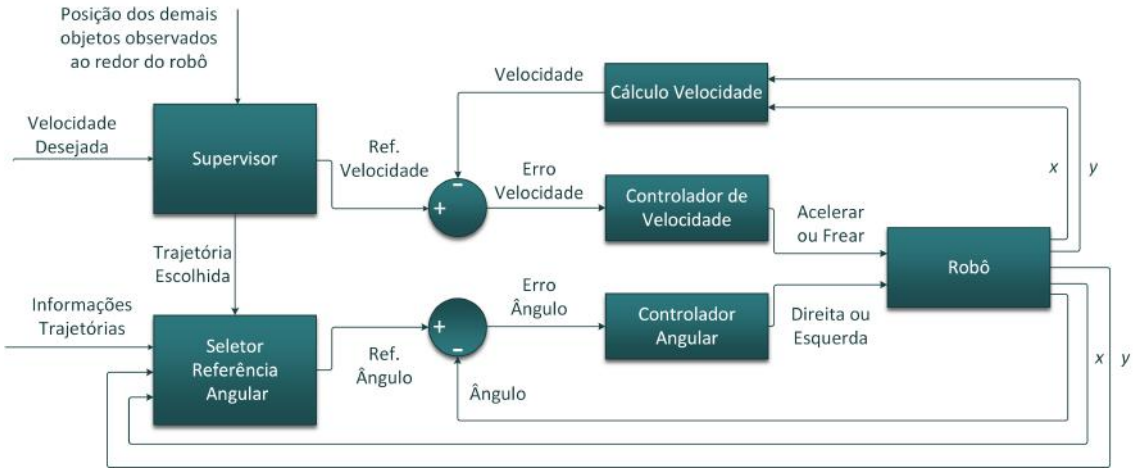


Figura 4.2: Visão Geral do Controle Implementado

pista, isto é, dado que o usuário escolhe um referencial de velocidade em que deseja andar, o cruise control é responsável em manter fixa a velocidade linear do veículo. Sendo assim, se a estrada possuir uma rampa por exemplo, esse controle deve aumentar a rotação do motor de forma a compensar a mudança na força contrária ao movimento do carro e assim sustentar o referencial escolhido.

Para isso, utiliza-se uma malha fechada como a da figura abaixo:

Figura 4.3: Diagrama de blocos para o Cruise Control.

Analisando a figura observa-se que um *setpoint* de velocidade é definido. Fazendo-se a diferença entre o valor estabelecido e a velocidade atual, que foi medida por intermédio de uma derivada suja, tem-se o "erro de velocidade" que será usado como a entrada do controle PID utilizado (demonstrado na seção acima). Assim, conseguimos manter a velocidade do carrinho mesmo que este esteja numa subida ou descida, já que nessas situações o que acontece é que o sinal de controle se torna maior para compensar a força extra.

Uma forma interessante de se analisar a aplicação de um controle PID para fazer o cruise control é analisar a interpretação de cada componente do PID individualmente. Começando pela parte proporcional, vemos que a ideia é que o controle responda de forma direta à diferença de velocidade entre o *setpoint* e a velocidade atual do carro, sendo o ganho proporcional (K_p) à intensidade dessa resposta, isto é quanto maior o ganho mais

forte será a saída do controle. Já a componente derivativa é a resposta à derivada da velocidade (aceleração), ou seja, essa componente é a resposta do PID à aceleração do carrinho multiplicada por um ganho derivativo (K_d). Por fim, a parcela integral sendo a integral da velocidade, logo a distância, seria uma reação do controle em relação ao que ainda falta a ser percorrido durante o intervalo de tempo percorrido vezes um ganho integral (K_i), ou seja, quanto maior a diferença entre a distância real percorrida pelo carro no intervalo de tempo $[t, t+\delta t]$ e a distância que ele deveria ter percorrido se estivesse na velocidade de *setpoint* durante esse mesmo intervalo mais forte será a resposta da parte integral do controle.

Por fim, é importante lembrar que o cruise control é um controle para a velocidade e não um sistema de piloto automático como algumas pessoas podem pensar, isto é, ele somente irá controlar a aceleração e freio do carro com a finalidade de manter a velocidade no valor especificado pelo usuário, entretanto as demais funções (como fazer curvas, frear caso haja um obstáculo à frente etc) ainda são de responsabilidade do motorista. A compreensão errada desse sistema pode acarretar em graves acidentes[38].

4.2 Controle Angular

Na seção 4.1 apresentou-se o controle por velocidade que é capaz de manter o robô numa determinada velocidade independentemente da direção para a qual o mesmo está apontado. Isto posto, será apresentado agora o controle responsável por manter o robô numa dada trajetória. No escopo deste trabalho lidou-se com trajetórias pré e bem definidas. Sendo assim, é de conhecimento prévio os pontos da mesma e os ângulos (tangentes a estes pontos) em que o robô deve estar em cada ponto para que siga a trajetória com sucesso.

A ideia principal é de fazer com que o robô sempre "tenda" para a trajetória estabelecida previamente. Para atingir este resultado é executado um algoritmo que será detalhado a seguir com a ajuda da figura 4.4.

Será tomado como exemplo o caso exibido na figura acima. Nesta tem-se um robô seguindo a trajetória no sentido horário. O robô está posicionado no sistema de coordenadas inercial na posição (x_r, y_r). Conhecendo a trajetória e seus pontos o primeiro passo é encontrar qual dos pontos é o ponto mais próximo das coordenadas do robô. Para isso mediu-se a distância entre a posição (x_r, y_r) e todos os pontos da trajetória até encontrar o melhor ponto e a menor distância. Este ponto é representado na figura por (x_t, y_t) e a menor distância por d.

Conhecendo-se o melhor ponto conhece-se também a tangente ao mesmo, representada na figura pela reta a com angulação α_t. Esta reta representa a direção na qual o robô deveria estar andando no caso de $(x_r, y_r) = (x_t, y_t)$. Entretanto, considerando $d > 0$ será

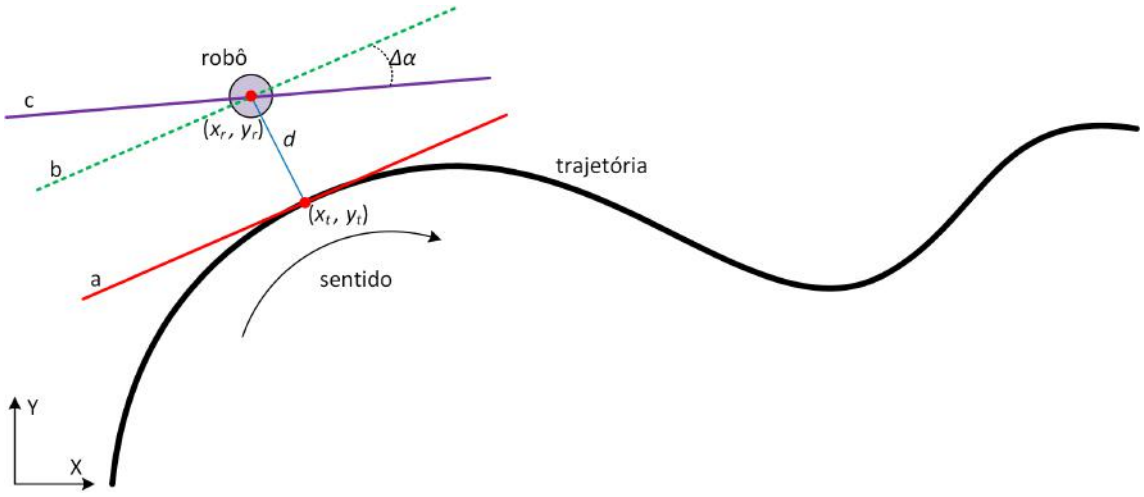


Figura 4.4: Esquemático alteração referência controle angular

necessário ajustar a referência do controle angular de forma a fazer com que $d \to 0$ e o robô ande exatamente sobre a trajetória desejada.

Para isso, o primeiro passo é transportar a reta a para a posição do robô (x_r, y_r). Essa reta transposta é representada na figura pela reta b. Isto feito, o próximo passo é ajustar a reta para trazer o robô para a trajetória. O ângulo $\Delta\alpha$ representa o quanto a tangente ao melhor ponto deverá ser ajustada e é calculado conforme apresentado na equação

$$\Delta\alpha = k \cdot d \cdot (-1)^{dentro} \cdot (-1)^{sentido}, \tag{4.1}$$

onde d é a distância previamente apresentada entre a posição do robô e o melhor ponto, k é uma constante utilizada para controlar a proporção do ajuste deste ângulo e tem valor empírico, $dentro$ tem valor 1 se o robô está dentro da trajetória e 0 se está fora da mesma, $sentido$ tem valor 1 se o robô estiver andando em sentido horário e 0 se andando em sentido anti-horário. $dentro$ e $sentido$ só assumirão os valores 1 e 0 de forma que estão na equação somente para ajustar o sinal de $\Delta\alpha$ conforme necessário.

Calculado o valor de $\Delta\alpha$ pode-se então proceder ao cálculo do ângulo que será utilizado como referência para o controle angular (α_r):

$$\alpha_r = \alpha_t - \Delta\alpha \tag{4.2}$$

4.3 Supervisor

Por mais que o *cruise control* traga benefícios como melhora do uso de combustível e controle de velocidade vale ressaltar que esse método é um controle "cego". Dessa forma a ideia de manter o referencial escolhido pelo usuário será seguida como uma verdade absoluta, ou seja, independentemente do que houver à frente do carro, o mesmo manterá a velocidade definida como referencial e com isso muitos acidentes podem ocorrer.

O conceito de *cruise control* adaptativo surge como uma tentativa de solucionar este problema. Essa nova técnica de controle utiliza radares e alguns outros sensores para identificar um obstáculo à frente e assim reduzir o *setpoint* de velocidade e dessa maneira evitar uma colisão. No entanto, esses sistemas ainda possuem problemas como por exemplo ir de parado a andando e vice-versa, ou situações de freadas bruscas exigindo que o motorista tome o controle a qualquer instante.

Na tentativa de solucionar esse problema no *cruise control* foi proposta no trabalho uma melhoria no mesmo em que, com o auxilio da câmera (simulando a utilização de alguns sensores) e conhecimento do veiculo (massa), o *setpoint* de velocidade por mais que tenha sido escolhido pelo motorista possa ser alterado levando inclusive o veiculo a parar se a situação demandar tal ato. Para isso, um "supervisor" foi acrescentado à malha de controle de velocidade. Esse supervisor tem, como o nome sugere, a função de analisar o mundo ao redor e assim decidir se é ou não possível manter a velocidade escolhida como referencial. A figura 4.5 retrata a técnica implementada para o *Cruise Control* Adaptativo:

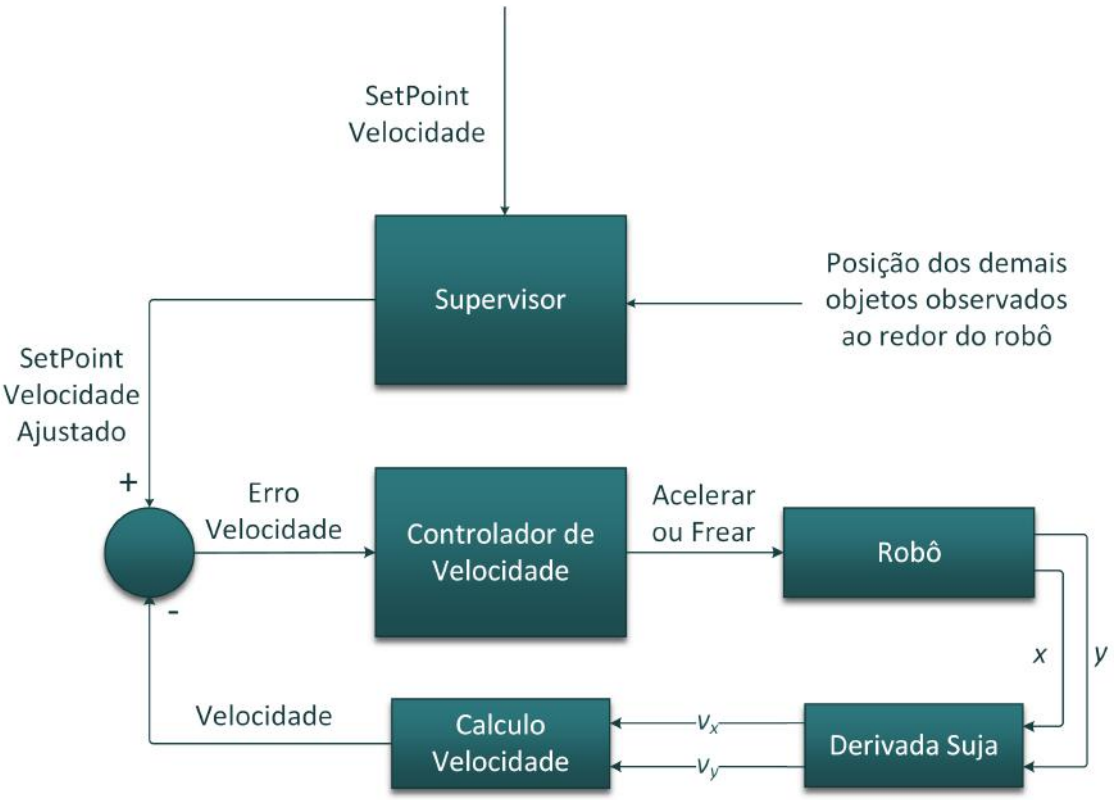


Figura 4.5: Diagrama de blocos do Cruise Control e Supervisor

Na figura acima pode-se notar que a saída da nossa "planta" (mundo observado por meio da câmera) são as posições X e Y dos veículos. Passando essas posições num derivador (derivada suja) obtem-se a velocidade linear de cada um.

Evitar Colisões

Com base nas informações passadas para o Supervisor, caso o veículo se depare com um outro carro à frente que esteja numa velocidade menor do que a usada como referencial, o sistema de supervisão tem como finalidade evitar uma colisão e manter uma possível distancia de frenagem segura. Para isto existem duas possibilidades ultrapassar o veículo mais lento (ou desviar do obstáculo) ou diminuir a sua velocidade (ou até mesmo frear).

A primeira dessas possibilidades a ser verificada para evitar uma colisão entre dois veículos na mesma trajetória e com velocidades diferentes é se o veículo mais rápido, estando atrás do mais lento, pode ou não ultrapassá-lo sem causar nenhuma situação de risco ou até mesmo uma colisão. Para isso, a estratégia usando o supervisor analisa se para o carro mais rápido existe primeiramente uma opção secundária de trajetória a ser seguida. Havendo essa trajetória, a estratégia irá analisar a seguir se na outra trajetória existe algum veículo dentro de um círculo de raio com tamanho igual ao da distância segura, explicada na seção 6.2.3.1. No caso de não existir, então a ultrapassagem é permitida e com isso a estratégia irá trocar a trajetória de referência do controle angular para a trajetória auxiliar a fim de realizar a ultrapassagem. Isso tudo é feito sem que haja necessidade de alterar o *setpoint* de velocidade definido pelo usuário.

No entanto, caso ao menos uma dessas condições descritas seja desobedecidas, ao invés de modificar a trajetória a ser seguida pelo carro a estratégia alterará o *setpoint* de velocidade do carrinho de forma a evitar uma colisão e manter também a distância segura entre os robôs.

Neste caso, o Supervisor analisa, para cada carro, se existe algum objeto em movimento à sua frente e que esteja se deslocando na mesma direção e trajetória. Caso não haja, o supervisor tem como saída o *setpoint* original. Entretanto, se houver, e o robô em analise estiver mais rápido que o obstaculo em sua frente, então o supervisor ajusta o *setpoint* da velocidade linear do automóvel de forma que o mesmo ainda seja capaz de frear numa situação de emergência (ou seja, mantendo uma distância segura) ou, se necessário, o supervisor colocará o valor do *setpoint* para 0 (zero) fazendo com que o veiculo freie. Caso o carro em analise esteja mais lento do que o obstaculo em sua frente o supervisor também não alterará a velocidade escolhida pelo usuário.

Para melhorar a resposta do veículo no caso das duas situações diferentes (Obstáculo em movimento e obstáculo parado), duas equações são propostas:

Obstáculo/robô parado

$$S_p = -\left(\frac{|Vel_{atual}|}{(d_s - d_{min})}\right) \cdot (d_s - d) \tag{4.3}$$

Robô em movimento

$$S_p = Vel_{atual} - \left(\frac{|Vel_{frente} - Vel_{atual}|}{(d_s - d_{min})} \right) \cdot (d_s - d) \tag{4.4}$$

Sabendo que, nas equações 4.3 e 4.4:

d Distância entre o robô atual e o robô dentro da sua região de segurança;

d_s Distância de segurança ;

d_{min} Distância mínima entre os robôs: 25px [1]

Vel_{frente} Velocidade do robô dentro da região de segurança do robô atual;

Vel_{atual} Velocidade do robô dentro da região de segurança do robô atual;

S_p Novo *SetPoint* calculado para o robô atual;

Para a equação 4.3 tem-se que primeiramente que a componente $\frac{V_{frente}}{d_s - d_{min}}$ é sempre positiva assim como $d_s - d$ será sempre positivo, uma vez que essa equação só é aplicada sobre o *setpoint* de velocidade do carro quando $d \leq d_s$. Sendo assim, pode-se concluir que o maior valor para a nova referência do carro será a velocidade atual do carro da frente, que no caso será sempre igual a 0 (zero) pois essa equação só é usada na situação em que existe um obstáculo parado na frente do robô analisado. Situação essa que só será verdadeira quando o carro em questão também estiver parado. Dessa maneira, garanti-se que o carrinho freie até que esteja com velocidade atual igual a zero.

Analisando agora a equação 4.4 vê-se que a estrutura usada nessa é similar àquela usada na equação anterior, entretanto o valor máximo a ser alcançado nessa equação é o valor da velocidade atual do carro no momento em que o carro na sua frente entrou na distância segura. Dessa maneira garanti-se que ao entrar na distância segura o carro freie suavemente a fim de alcançar a velocidade do carro da frente e assim consiga permanecer na velocidade do mesmo sempre respeitando a distância segura.

[1] Como a posição do robô é medida em relação ao centro do mesmo e este possui um raio de 10px, no caso dos dois robôs estarem encostados a distância medida entre eles seria de 20px. Sendo assim, a distância mínima entre os robôs foi escolhida como sendo 25px ($2r + r/2$) para evitar que os robôs se toquem.

Capítulo 5

Simulador

Com o intuito de comprovar a modelagem do problema realizada no capítulo 3 e a solução proposta no capítulo 4 foi construído um simulador para que fosse possível construir o sistema modelado e testar as técnicas de controle empregadas. O modelo utilizado na construção do simulador foi o Modelo Dinâmico, apresentado na seção **??**. Para tal foi utilizado o programa MatLab. Através da ferramenta Simulink foi desenvolvido o diagrama de blocos do sistema modelado do robô e então implementadas as abordagens de controle citadas neste trabalho.

Além dos objetivos já citados buscou-se também resolver eventuais problemas de implementação e, então, comprovar a estabilidade do sistema nas regiões em que serão atuadas para que estes resultados pudessem guiar a construção do programa e suas interações com o robô.

5.1 Estrutura do Simulador

O simulador foi estruturado de forma a possuir três etapas: Inicialização, Execução e Geração de Resultados. Estas etapas serão explicitadas a seguir.

5.1.1 Inicialização

Esta primeira etapa precede a execução da simulação com o `SimuLink` e é responsável por carregar os parâmetros para a simulação que será executada: o tempo total de simulação, o intervalo de simulação, a posição e ângulo inciais do robô, os parâmetros do modelo do robô (tamanho, peso, momento de inércia, torque). Nesta etapa também é gerada a lista de pontos para a trajetória desejada bem como o valor dos ângulos desejados de cada ponto (através da derivada da equação da trajetória).

O código 5.1 mostra um exemplo de script de inicialização para simular uma trajetória circular de raio 1m centrada na origem.

```
close all; clear; clc;

k = 0.0785; %constante de torque https://www.pololu.com/product/1117/faqs
r = 0.016; %raio da roda
R = 16.7; %Resistencia de armadura -> se a resistencia eletrica do motor for 9/0.540
m = 0.200; % massa do carro
l = 0.0475; %distancia ate o centro do carrinho
I = m*l^2/2;

Tfinal = 400;
Tstep = 0.003;

x0   = [0, 0.6, 0.6, 0, pi];
x1 = [235, 350, pi];
x0n   = [109300 31];    %kg

%% inicializando circulo de raio 1
 nPoints = 3000;
 syms t;
 r_circ = 1;
 x = r_circ*cos(t)+2;
 y = r_circ*sin(t)+2;
 xd = -r_circ*sin(t);
 yd = r_circ*cos(t);

 lstAtan2 = [];

 for i=0:nPoints
     t = (2*pi)*(i/nPoints); %normalizado
     lstAtan2 = [lstAtan2; eval(x) eval(y) atan2(eval(yd), eval(xd)) eval(yd) eval(xd)];
 end
```

Código 5.1: Exemplo script inicialização de simulação

5.1.2 Execução

A execução da simulação é feita através da execução de um sistema montado no SimuLink. O sistema construído pode ser observado na figura 5.1. O bloco principal, denominado de `robotsfunction` representa o modelo do robô apresentado no capítulo 3. A entrada é um vetor de dimensão dois com os sinais $v_1 + v_2$ e $v_1 - v_2$, onde v_1 e v_2 são as velocidades de cada roda. A saída deste bloco são as componentes x e y da posição bem como o ângulo θ medido do robô.

Os conteúdos dos blocos `calcularV1V2` e `calcUrUt` podem ser visualizados respectivamente nos trechos de código 5.2 e 5.3

```
function [v1,v2] = calcularV1V2(u_t,u_r)

v1 = ((0.5*u_t) + (u_r*0.5));
v2 = ((0.5*u_t)  - (u_r*0.5));
```

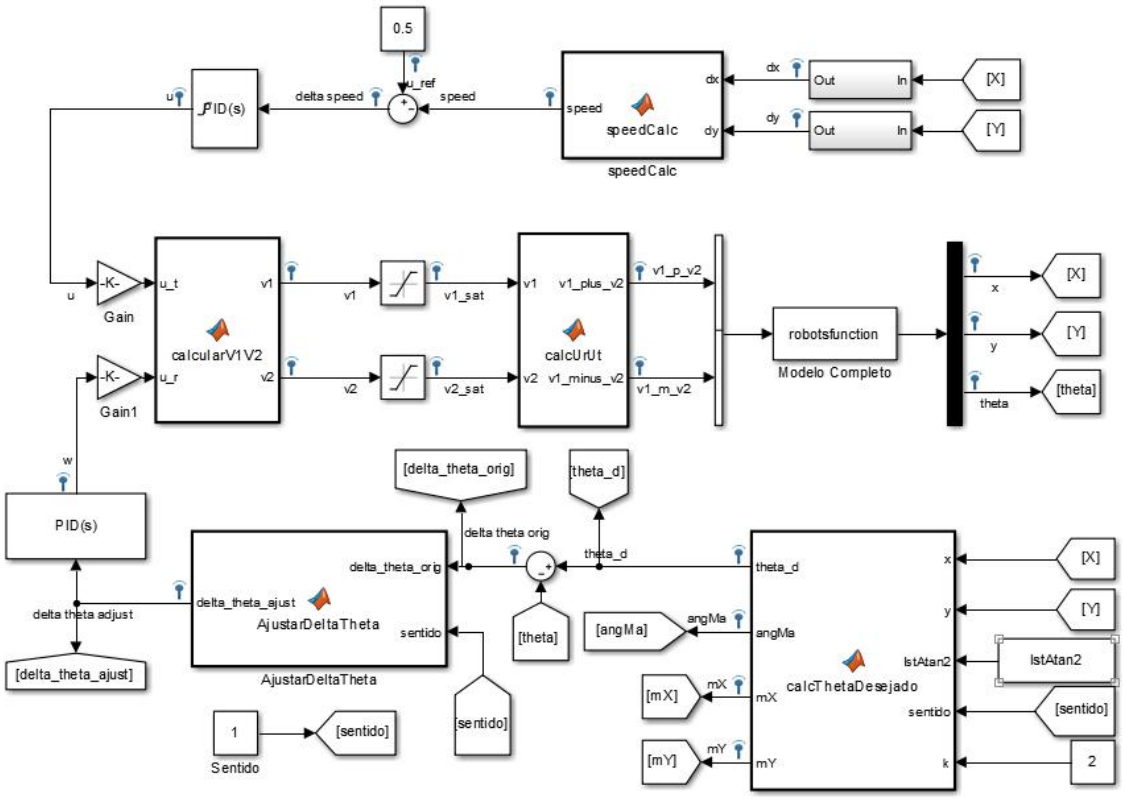


Figura 5.1: Estrutura do simulador na ferramenta Simulink

Código 5.2: Função CalcularV1V2

```
1 function [v1_plus_v2, v1_minus_v2] = calcUrUt(v1, v2)
2
3 v1_plus_v2 = v1+v2;
4 v1_minus_v2 = v1-v2;
```

Código 5.3: Função CalcularUrUt

Os blocos denominados `PID(s)` correspondem ao controlador PID. A equação dos mesmos pode ser visualizada na equação 5.1

$$P + I\frac{1}{s} + D\frac{N}{1+N\frac{1}{s}} \tag{5.1}$$

Na parte superior do diagrama pode-se ver que a velocidade atual do robô é conseguida através da utilização da "derivada suja" das componentes x e y e então sua composição através da função `speedCalc` (Código 5.4).

```
1 function speed = speedCalc(dx, dy)
2 speed = sqrt(dx^2 + dy^2);
```

Código 5.4: Função speedCalc

A velocidade é então comparada à uma referência e, então, o erro ($\Delta speed$) é passado para o controlador PID. A saída do PID é então passada ao modelo passando pelos devidos cálculos.

Já para o controle angular, as informações da simulação (sentido de rotação, pontos e ângulos calculados para a trajetória e posição atual do robô) são passadas para o bloco `calcThetaDesejado`. Este bloco é responsável por, dada uma trajetória desejada e a posição atual do robô, determinar qual o ângulo em que o robô deveria estar para tender à trajetória. Lembrando que, conforme explicado na seção 4.2, este ângulo depende do ângulo do melhor ponto escolhido entre os pontos da trajetória, a distância entre o ponto escolhido e a posição atual do robô, o sentido de rotação e um ganho k. Nas simulações utilizou-se o valor $k = 2$. Este valor foi obtido empiricamente. O código do bloco `calcThetaDesejado`pode ser visualizado no trecho de código 5.5.

```
1  function [theta_d, angMa, mX, mY ] = calcThetaDesejado(x,y, lstAtan2, sentido,k)
2      theta_d = 0; mX = 0; mY = 0; mA = 0; angMa = 0; mDist = 0;
3
4      coder.extrinsic('melhorPonto');
5      coder.extrinsic('melhorPontoAtan2');
6      [mDist, mX, mY, angMa] = melhorPontoAtan2(x, y, lstAtan2);
7
8      coder.extrinsic('inpolygon');
9
10     dentro = int16(1);
11     diferenca = double(1);
12     satAngDeg = 85;
13     satAngRad = deg2rad(satAngDeg);
14
15     if inpolygon(x,y,lstAtan2(:,1),lstAtan2(:,2))
16         dentro = 1;
17     else
18         dentro = 0;
19     end
20
21     diferenca = k*mDist*(-1)^(dentro)*(-1)^(sentido);
22
23     if (abs(diferenca) > satAngRad)
24         diferenca = sign(diferenca)*satAngRad;
25     end
26
27     if (sentido == 0)
28         angMa = 0 - (pi - angMa);
29     end
30
31     theta_d =  (angMa - diferenca);
```

Código 5.5: Função CalcularThetaDesejado

Uma das saídas do bloco `calcThetaDesejado` é θ_d. θ_d corresponde à referência para o controle angular. A diferença entre $theta_d$ e $theta$ ($\Delta\theta$) passa então pelo bloco `AjustarDeltaTheta`. Este bloco é responsável por resolver a descontinuidade do `atan2`. O código do bloco `AjustarDeltaTheta` pode ser contemplado no trecho de código 5.6.

```
1  function delta_theta_ajust = AjustarDeltaTheta(delta_theta_orig, sentido)
2
3  delta_theta_ajust = delta_theta_orig;
4
5  while (delta_theta_ajust <= -pi)
6      delta_theta_ajust = delta_theta_ajust + 2 * pi;
7  end
8
9  while (delta_theta_ajust >= pi)
10     delta_theta_ajust = delta_theta_ajust - 2 * pi;
11 end
```

Código 5.6: Função AjustarDeltaTheta

O valor de $\Delta\theta$ já ajustado é então a entrada do bloco PID responsável pelo controle angular.

5.1.3 Geração de Resultados

Ao término da execução do `SimuLink` é executado também um script que colherá os dados salvos durante a simulação e transformará os mesmos em gráficos para facilitar a analise dos dados. Além disso, para que seja possível avaliar quantitativamente os resultados obtidos, este mesmo script calcula os valores do *Mean Squared Error (MSE)* [39] para cada uma das quatro variáveis principais (x, y, θ e Vel). O MSE nada mais é do que a média dos quadrados dos erros de cada medida. A equação 7.1 ilustra o erro médio quadrático de um sinal X, onde X_d representa o valor desejado e X_a representa o valor atual, ou valor medido. O MSE para cada um dos sinais pode ser obtido de forma simples através da utilização da função `immse` do MatLab.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(X_d - X_a)^2 \tag{5.2}$$

5.2 Resultados Obtidos

5.2.1 Círculo de raio de 1m e sentido anti-horário

Para esta primeira simulação escolheu-se uma trajetória simples, um círculo de raio 1m centrado em $(2;2)$. O *setpoint* de velocidade dado para o robô foi o de 0,5 m/s. A posição inicial do robô está em $(2;0,5)$. O sentido de rotação é anti-horário.

O código utilizado para gerar os pontos e os ângulos da trajetória pode ser observado no trecho de código 5.7.

```
%% inicializando circulo de raio 1
nPoints = 3000;
syms t;
r_circ = 1;
x = r_circ*cos(t)+2; %x(t)
y = r_circ*sin(t)+2; %y(t)
xd = -r_circ*sin(t); %derivada de x(t)
yd = r_circ*cos(t); %derivada de y(t)

lstAtan2 = [];

for i=0:nPoints
    t = (2*pi)*(i/nPoints); %normalizado
    lstAtan2 = [lstAtan2; eval(x) eval(y) atan2(eval(yd), eval(xd)) eval(yd) eval(xd)];
end
```

Código 5.7: Script para geração da trajetória circular com raio 1 centrado em $(2;2)$

Variável	MSE
x	0,0002
y	0,0146
θ	0,0225
Vel	1,4878

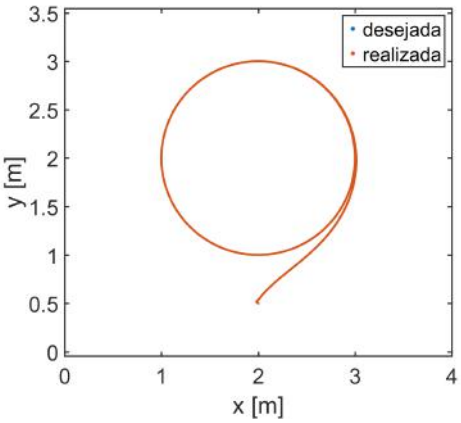
(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	60	92
I	395	90
D	1,5	1,5

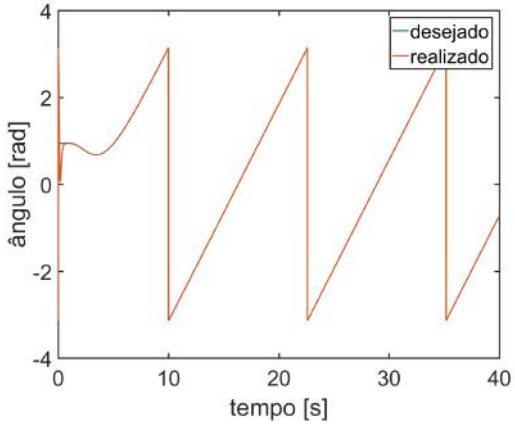
(b) Parâmetros utilizados nos controladores

Figura 5.2: MSE das variáveis medidas na simulação e parâmetros utilizados pelos controladores

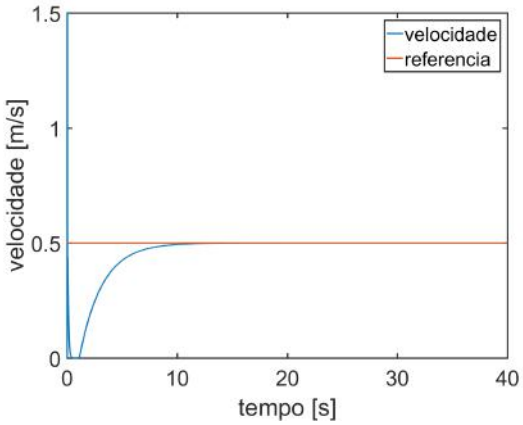
Analisando os resultados exibidos na figura 5.3 e na tabela 5.2 pode-se perceber a estrutura do controle proposto possui um resultado excelente. Conforme pode-se observar nos gráficos 5.3a e 5.3e, o robô foi capaz de seguir a trajetória proposta sem maiores dificuldades. O erro médio quadrático para tanto a posição e o ângulo alcançaram valores praticamente desprezíveis. A figura 5.3c mostra que o controlador de velocidade proposto também foi satisfatório. O sistema tendeu para a velocidade de referência sem overshoot e sem oscilações após atingir o valor de referência. O valor do erro médio quadrático para a velocidade é de uma ordem de grandeza um pouco maior devido ao fato de que o transitório da resposta (aproximadamente os 10 segundos iniciais) é um pouco maior do que no caso do controle angular. Isto se deve ao fato de que o controlador foi projetado para que não houvessem alterações bruscas de velocidade (para não diminuir o conforto dos passageiros de um veículo, por exemplo). Por fim, os sinais de controle se mostraram



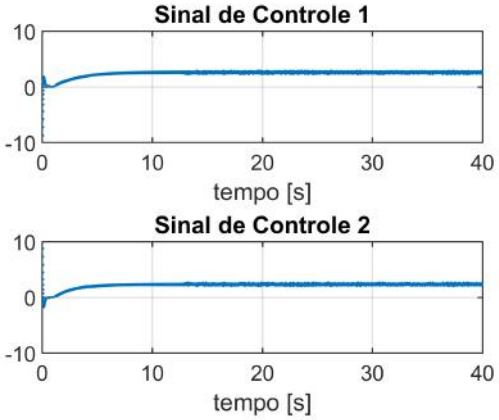
(a) Trajetória



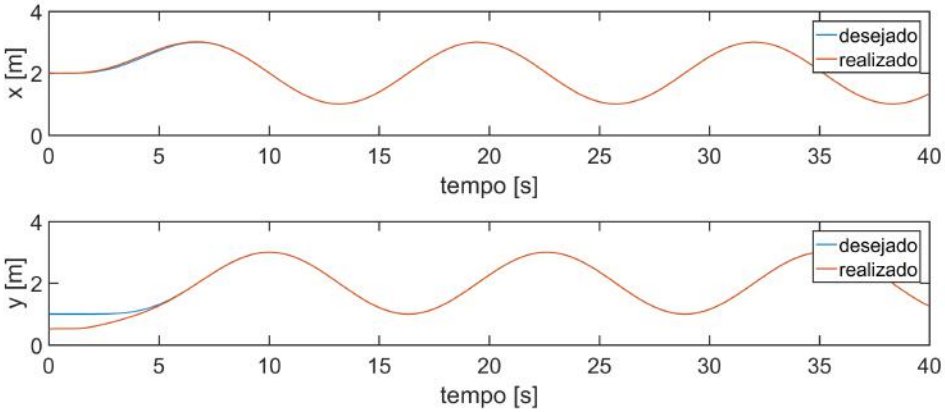
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 5.3: Resultados círculo de raio 1m centrado em $(2;2)$ com sentido anti-horário

em valores aceitáveis durante toda a simulação.

5.2.2 Círculo de raio de 1m e sentido horário

Para esta simulação utilizou-se a mesma trajetória gerada na seção 5.2.1 , a mesma posição inicial e o mesmo *setpoint* de velocidade. Entretanto, agora alterou-se o sentido para que seja horário.

Variável	MSE
x	0,0002
y	0,0135
θ	9,8116
Vel	1,4877

(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	60	92
I	395	90
D	1,5	1,5

(b) Parâmetros utilizados nos controladores

Figura 5.4: MSE das variáveis medidas na simulação e parâmetros utilizados pelos controladores

O objetivo principal deste segundo teste foi o de comprovar o fato de que o sistema também se comportaria de forma satisfatória quando o robô estivesse percorrendo a trajetória agora no sentido horário. Os resultados exibidos na figura 5.5 são muito parecidos com os encontrados na figura 5.3, ou seja, conforme quis-se comprovar, o sistema se comportou de forma semelhante quando o robô percorreu a trajetória desejada em ambos os sentidos.

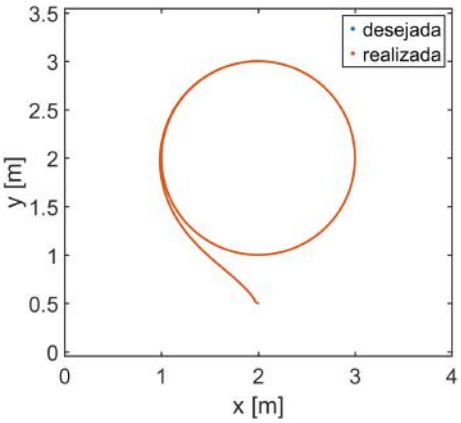
5.2.3 Círculo de raio 1m iniciando em (1,5;1,5) com alteração de velocidade

Como segundo experimento de simulação escolheu-se o mesmo círculo de raio 1m centrado na origem. Porém, agora o robô iniciará a simulação dentro do círculo ($(1,5;1,5)$ e o setpoint de velocidade será alterado durante a simulação. O *setpoint* iniciará como 0,5 m/s, será alterado para 1m/s aos 10 segundos de simulação e retornará para 0,5 m/s após os 20 segundos de simulação. O sentido de rotação é anti-horário.

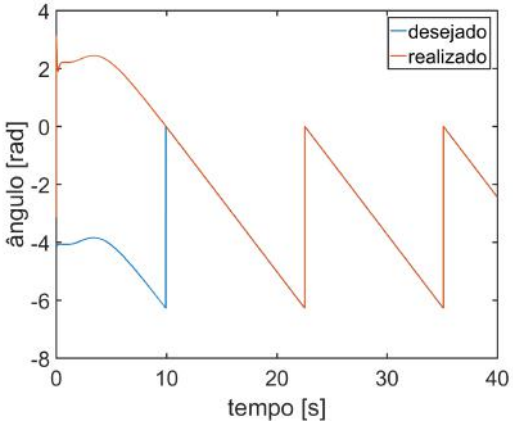
Este teste tinha como objetivos:

1. Comprovar que o sistema funcionava bem com o robô na parte de dentro da trajetória

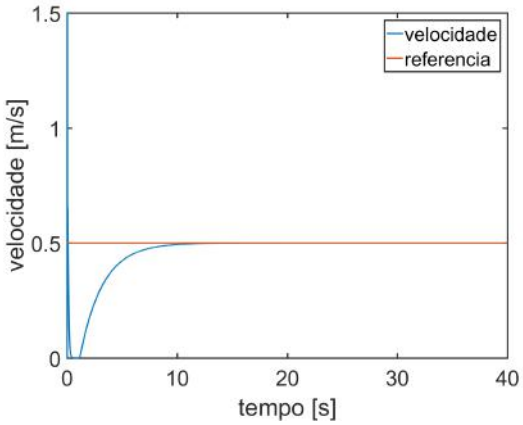
2. Comprovar que o sistema lidaria com mudanças no valor de referência de velocidade.



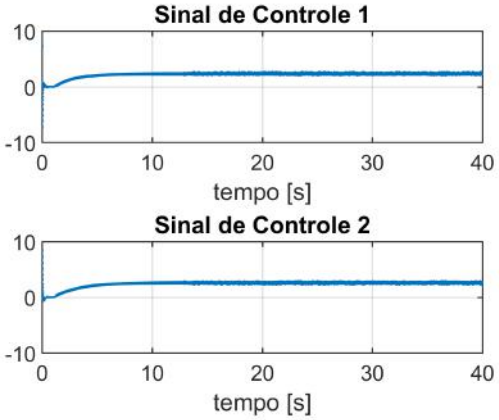
(a) Trajetória



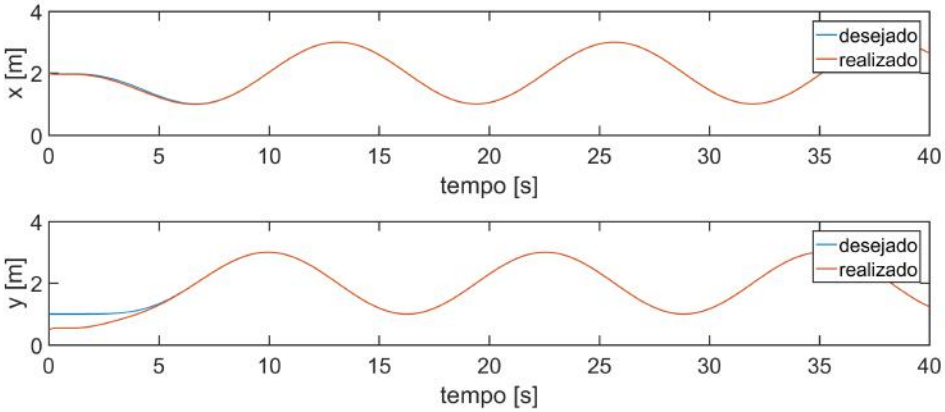
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 5.5: Resultados círculo de raio 1m centrado em $(2;2)$ com sentido horário

Variável	MSE
x	0,0018
y	0,0027
θ	0,0173
Vel	1,5794

(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	60	92
I	395	90
D	1,5	1,5

(b) Parâmetros utilizados nos controladores

Figura 5.6: MSE das variáveis medidas na simulação e parâmetros utilizados pelos controladores

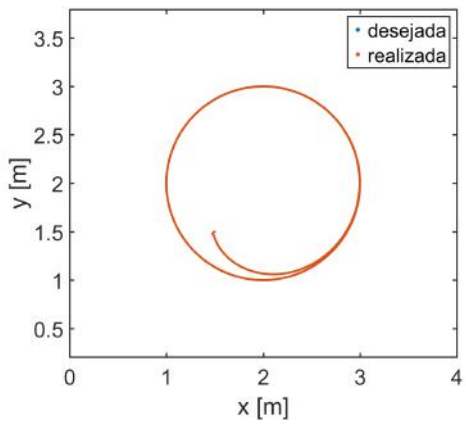
Os resultados exibidos na figura 5.7 e na tabela 5.6 podem nos ajudar na comprovação destes objetivos apresentados. Os gráficos 5.7a e 5.7e mostram que mesmo com o robô posicionado dentro da trajetória o sistema fez com que o mesmo seguisse a trajetória desejada satisfatoriamente. O gráfico 5.7b mostra que o robô conseguiu seguir com sucesso a referência angular dada ao mesmo. O gráfico 5.7c mostra que o sistema acompanhou as mudanças no valor de referência da velocidade tanto quando a alteração foi para um valor maior quanto quando o valor de referência foi diminuído. Isto aconteceu sem overshoot, oscilações ou erro em estado estacionário. A figura 5.7d mostra que os sinais de controle responderam à alteração de referência, entretanto se mantiveram em valores aceitáveis.

Os dados da tabela 5.6 mostram um valor maior do erro médio quadrático para a velocidade. Isto se deve ao fato de que, como houveram duas mudanças de velocidade durante a simulação o sistema estava durante grande parte do tempo da simulação em um estado transitório.

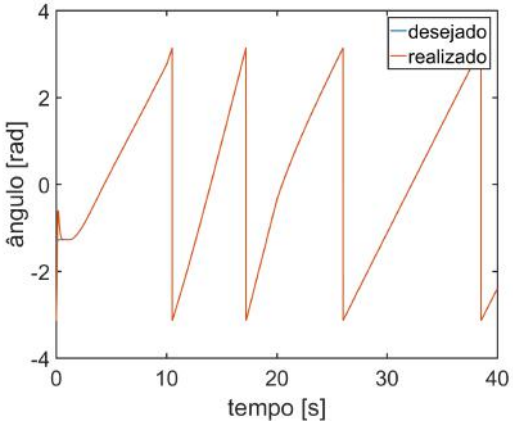
5.2.4 Oval de Cassini

Neste experimento buscou-se testar o comportamento do sistema modelado em uma trajetória um pouco mais elaborada do que um círculo. Para isso escolheu-se uma Oval de Cassini. As ovais de Cassini são uma família de curvas denominadas com o nome do matemático e astrônomo italiano Giovanni Domenico Cassini. Uma Oval de Cassini é o lugar geométrico dos pontos tais que o produto das distâncias destes pontos à dois pontos fixos é constante [40].

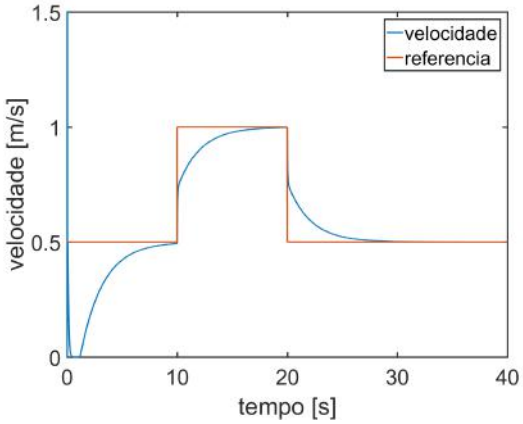
Para gerar os pontos da trajetória utilizou-se o script que pode ser visto no trecho de código 5.8. Neste utilizou-se a equação da trajetória para gerar uma lista de pontos. A partir desta lista de pontos encontrou-se um polinômio que se enquadre aos dados de $x(t)$ e outro que se enquadre aos dados de $y(t)$ através de um `polyfit`. Com estes dois polinômios em mãos encontrou-se as derivadas dos mesmos e então montou-se a lista com os pontos e os respectivos ângulos em que o robô deve estar em cada ponto da trajetória.



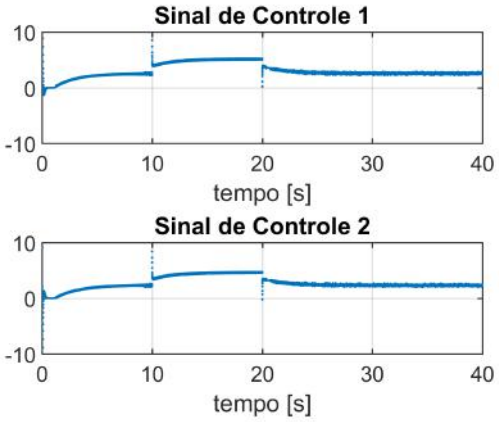
(a) Trajetória



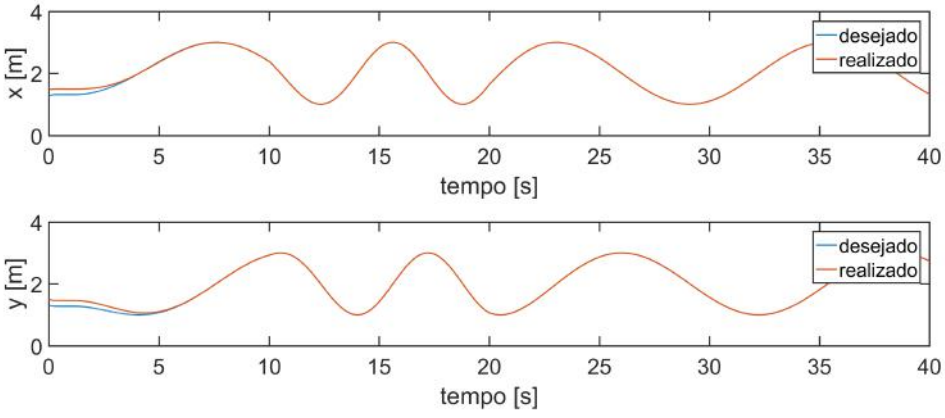
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 5.7: Resultados círculo de raio 1m iniciando em $(1,5;1,5)$ com alteração de velocidade e sentido anti-horário

Da mesma forma que na seção 5.2.3, o *setpoint* de velocidade será alterado durante a simulação. O *setpoint* iniciará como 0,5 m/s, será alterado para 1m/s aos 10 segundos de simulação e retornará para 0,5 m/s após os 20 segundos de simulação. O sentido de rotação é anti-horário.

```
syms t; a = 1.7; b = 2;
c = a^2*cos(2*t) + sqrt(b^4 - a^2*sin(2*t)^2);
xc = 1.5; yc = 1.5; s = 0.5;
x = xc + s*cos(t)*sqrt(c);  y = yc + s*sin(t)*sqrt(c);
t = linspace(0,2*pi,1000);
lista = [eval(x)' eval(y)'];
nPoints = max(size(lista));

t_x = [];  t_y = [];
for i = 1:nPoints
        t_x = [t_x; i lista(i,1)];
        t_y = [t_y; i lista(i,2)];
end

 t_x = [t_x; i+1 lista(1,1)];  t_y = [t_y; i+1 lista(1,2)];

 n = nPoints + 1;  degree = 20;

 px = polyfit(t_x(1:n,1),t_x(1:n,2),degree);
 py = polyfit(t_y(1:n,1),t_y(1:n,2),degree);

 xEq = poly2sym(px);  yEq = poly2sym(py);

 xDiff = diff(xEq);  yDiff = diff(yEq);  pVar = 1000;

 xPoints = polyval(px,linspace(1,n,pVar))';
 yPoints = polyval(py,linspace(1,n,pVar))';

 xDiffPoints = polyval(sym2poly(xDiff),linspace(1,n,pVar))';
 yDiffPoints = polyval(sym2poly(yDiff),linspace(1,n,pVar))';

 lstAtan2 = [xPoints yPoints atan2(yDiffPoints,xDiffPoints) yDiffPoints xDiffPoints];
```

Código 5.8: Script para geração da trajetória Oval de Cassini

O objetivo principal desta simulação era comprovar o funcionamento do sistema em um cenário mais complexo, com uma trajetória não circular (e portanto com curvas para os dois lados) e também com a alteração no valor de referência da velocidade do robô.

Os dados das figuras 5.9a 5.9e mostram que o robô seguiu com sucesso a trajetória gerada através de uma Oval de Cassini. A figura 5.9b explicita o fato de que o controle angular foi capaz de lidar com as mudanças de angulação de referência para ambos os lados sem maiores problemas. Conforme já havia sido comprovado anteriormente, a figura 5.9c mostra que a velocidade de referência foi seguida pelo robô. Os sinais de controle, exibidos na figura 5.9d, oscilaram um pouco mais do que nas simulações anteriores. Entretanto isto pode ser explicado pelo fato de que neste teste, diferentemente dos demais

Variável	MSE
x	0,0001
y	0,0059
θ	0,0564
Vel	1,4878

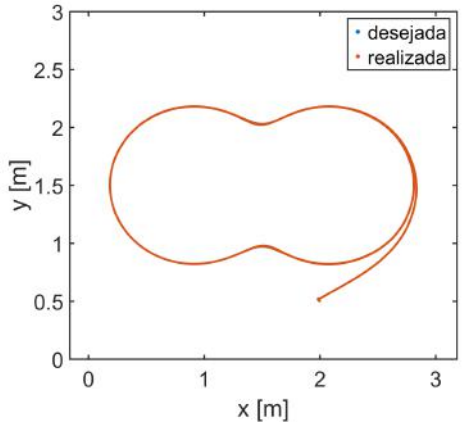
(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	60	92
I	395	90
D	1,5	1,5

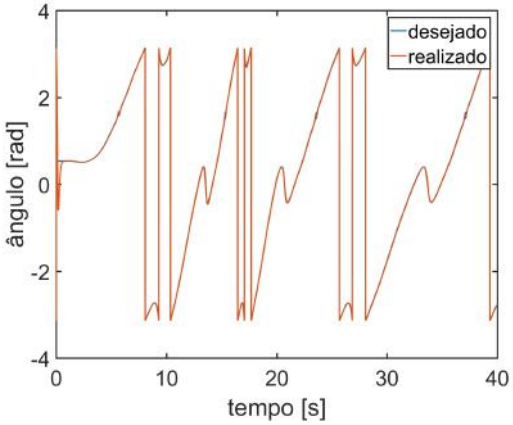
(b) Parâmetros utilizados nos controladores

Figura 5.8: MSE das variáveis medidas na simulação e parâmetros utilizados pelos controladores

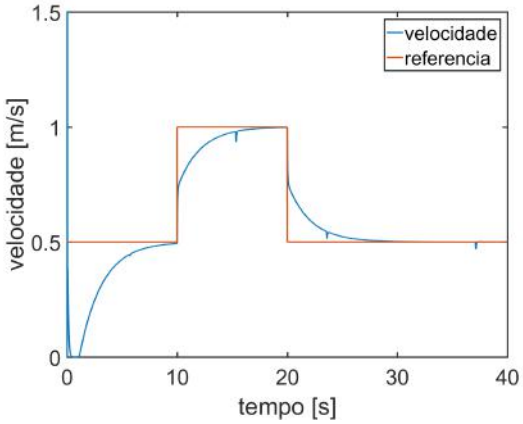
testes com trajetória circular, as mudanças na trajetória fazem com que os sinais enviados para cada motor fiquem sendo alterados constantemente. Os dados da tabela 5.8 mostram que os valores do erro médio quadrático para todas as quatro variáveis analisadas ficaram próximos dos valores encontrados anteriormente, mostrando que o aumento na complexidade da tarefa não ocasionou um resultado pior para esta simulação.



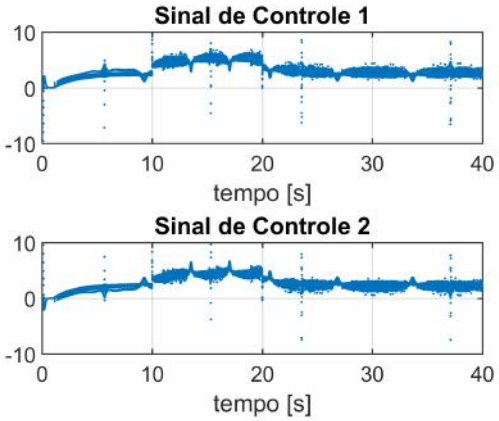
(a) Trajetória



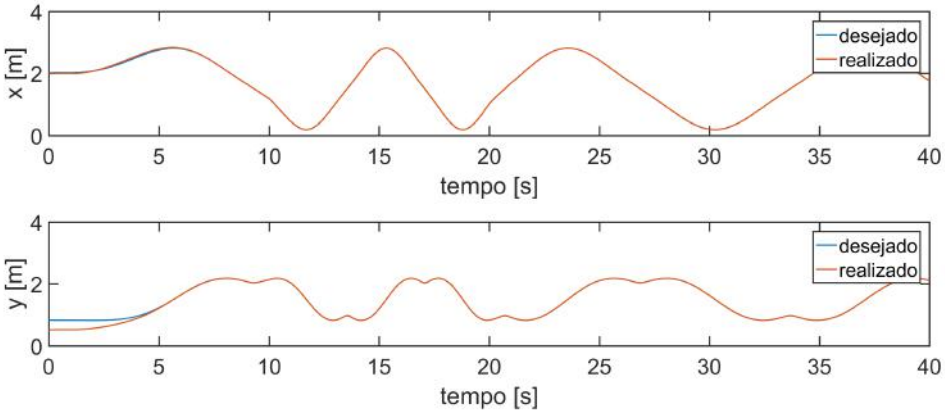
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 5.9: Resultados Oval de Cassini com alteração de setpoint de velocidade durante a simulação

Capítulo 6

Implementação do programa

Com o intuito de auxiliar na implementação prática das propostas deste trabalho foi desenvolvido também um programa para ser responsável por todo o ciclo de processamento que vai desde o sensoreamento (adquirir os dados do mundo real) até o envio dos comandos para os robôs passando por processamento de imagem, modelagem computacional, decisões sobre trajetórias, controle, etc.

Conforme pode-se perceber pela observação da Figura 6.1, o programa desenvolvido é responsável por receber a imagem captada pela câmera e, baseando-se no que foi recebido, tomar a decisão de enviar um comando específico para um robô específico. Desta forma a câmera percebe o "mundo real" e traduz isto de forma que o programa possa compreender (uma imagem no nosso caso) a decisão que o programa toma altera o comportamento de cada robô que, por consequência, interfere novamente no "mundo real". A forma com que essas interfaces estão estabelecidas possibilita que o sensor seja alterado da câmera para outro de interesse com impacto pequeno no resto do programa. Da mesma forma a comunicação do programa com o "mundo real" pode também ser alterada com pequenos efeitos sobre o mesmo.

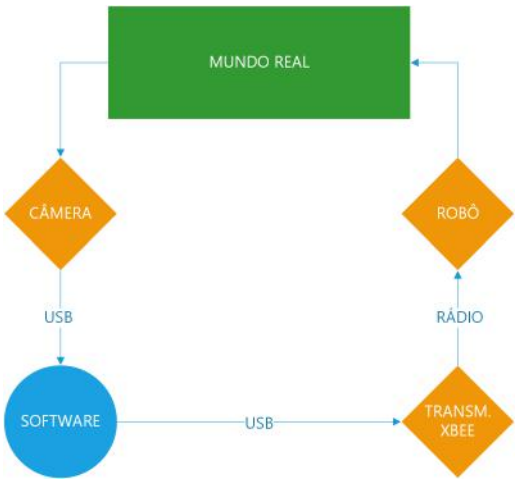


Figura 6.1: Fluxo de informações do programa.

Para a implementação do programa em questão diversas decisões tiveram que ser tomadas. Estas serão apresentadas e explicadas a seguir.

6.1 Decisões de Implementação

O programa desenvolvido no âmbito deste trabalho foi feito com a seguinte implementação:

- **Linguagem de Programação:** C++
- **Visão Computacional:** OpenCV3
- **Interface Gráfica:** Qt
- **Plataforma:** Windows
- **IDE:** Visual Studio 2013
- **Geração das trajetórias:** MatLab
- **Banco de dados:** SQLite3

6.1.1 Linguagem de Programação

A linguagem de programação escolhida para a implementação do trabalho foi a linguagem C++, isso tendo em vistas alguns benefícios trazidos por essa tais como:

- Possui uma comunidade imensa ao longo do mundo. No StackOverflow (site de perguntas e respostas voltadas para programação) é o sexto marcador mais procurado. Além disso também possui a sexta posição no GitHub (repositório usado para armazenar e versionar códigos).
- Possui amplo controle sobre os recursos disponíveis nos computadores (como controle sobre a memória do computador por exemplo), ou seja, com essa linguagem é possível utilizar os recursos de um computador de forma bastante otimizada, o que nem sempre é possível em outras linguagens.
- Devido ao benefício anterior, é uma linguagem geralmente de execução mais rápida do que códigos escritos nas demais linguagens.

Além disso, há também o fato dos autores possuírem conhecimento prévio nessa linguagem e em algumas bibliotecas que foram usadas na construção do programa implementado.

6.1.2 Visão Computacional

A biblioteca para visão computacional usada no desenvolvimento do trabalho foi o OpenCV (*Open Source Computer Vision Library*), tendo em vista que além de ser projetada com o intuito de fornecer a infraestrutura necessária para a implementação da visão computacional de forma otimizada, também possui parte de suas funcionalidades capazes de atuar com CUDA[1], ou seja, ela está sendo evoluída para fornecer uma estrutura capaz de fazer processamento em paralelo. Por fim, a biblioteca OpenCV possui suporte para Mac, Linux e Windows e pode ser usada com as linguagens: C++, Matlab, Python, C e Java.

Soma-se a isso o fato dos autores terem experiência anterior com o uso da biblioteca durante uma bolsa de iniciação científica em que puderam desenvolver um programa de reconhecimento de pessoas mesmo que estas estejam em movimento para produzir uma medição da radiação que estas recebem ao passarem no interior de uma sala com um reator nuclear. [41]

6.1.3 Interface Gráfica

Para o desenvolvimento da interface gráfica da nossa aplicação adotou-se o Qt [42] (pronuncia-se como a palavra inglesa *cute*). Qt foi escolhido por ser uma ferramenta de desenvolvimento *cross-platform*, possuir extensa documentação, ser de uso gratuito para fins não comerciais e por atender aos requisitos que desejá-se para a interface que seria desenvolvida. Além disso, pesou na escolha a experiência prévia dos autores com a ferramenta.

6.1.4 IDE

A escolha do Visual Studio como IDE (*Integrated Development Environment*) foi bastante simples, não só pelos autores já estarem familiarizados com o uso da ferramenta mas também por alguns outros benefícios como:

- Possuir suporte a um vasto grupo de linguagens diferentes tais como: Visual Basic, C#, C++, PHP, Javascript etc.
- Possuir uma grande comunidade facilitando a configuração do ambiente para desenvolver um projeto.
- Possuir suporte e uma extensão para a biblioteca *Qt* usada na construção da interface gráfica do programa.

[1] CUDA®(*Compute Unified Device Architecture*) é uma plataforma de computação paralela e um modelo de programação. Ela permite grandes aumentos no desempenho de computação por aproveitar a potência da unidade de processamento gráfico (GPU).

- Ser compatível com desenvolvimento utilizando *CUDA* (Processamento em paralelo) o que pode ser bastante útil para uma continuidade no desenvolvimento do programa pois aumentaria a taxa de amostragem (no caso fps) o que em aplicações de visão computacional possui um papel muito importante.

6.1.5 Geração de Trajetórias

No escopo deste trabalho está-se propondo manter determinados robôs em trajetórias pré-definidas. Desta forma, faz-se necessário conhecer os pontos que formam esta trajetória e a angulação que cada robô deve assumir caso esteja em cada uma dessas posições. A explicação da metodologia utilizada para fazer com que o ângulo de referência para cada robô seja ajustado de forma a manter o mesmo sempre andando sobre a trajetória desejada é melhor explicado na seção 4.2.

Levando-se em consideração o fato de que as trajetórias só precisam ser geradas uma única vez, decidiu-se que não faria sentido nossa aplicação gerar estes pontos e as informações sobre os mesmos. Ao invés disso, decidiu-se utilizar o *MatLab* para gerar estas trajetórias visto que muitas delas foram geradas através de equações ou de *curve fitting*, o que é tarefa fácil para um programa como o *MatLab*. Também levou-se em consideração o fato de que o *MatLab* possui integração nativa com o banco de dados *SQLite*, conforme explicado na seção 6.1.6.

6.1.6 Banco de Dados

Com intuito de possibilitar a persistência dos dados da aplicação fez-se necessário a adoção de um banco de dados. Neste banco estão armazenadas informações sobre as cores, configuração de cada robô, as informações sobre as trajetórias e os pontos que as compõem e parâmetros para os controles PID. Para armazenar estes dados a escolha dos autores foi a de utilizar um banco de dados relacional devido ao fato das relações entre as entidades mapeadas permanecem as mesmas sempre, obedecendo à um esquema único definido previamente.

Como no caso da aplicação desenvolvida no escopo deste projeto não se fez necessária uma grande complexidade de dados e nem uma capacidade de armazenamento de grande volume de dados, escolheu-se um banco de dados que não fosse preciso instanciar um servidor para utilizar o mesmo. A alternativa OpenSource que foi encontrada e que atenderia melhor foi o SQLite [43]. Isto se deve ao fato do mesmo possuir integração nativa com o MatLab [44], possuir fácil integração com o C++ através de uma biblioteca com documentação disponível e de já ser de conhecimento dos autores.

6.1.7 Manutenção do programa

Para manter um histórico do desenvolvimento do código e possibilitar que os autores pudessem desenvolver ao mesmo tempo o programa sem que houvesse conflito entre as alterações feitas nos códigos, decidiu-se utilizar o sistema de versionamento (SVN) utilizando para isso o programa tortoise SVN, e para armazenar esse histórico utilizou-se o servidor hospedeiro do Assembla.

6.2 Implementação

Com base nas decisões tomadas e explicitadas na seção 6.1, buscou-se implementar o programa para resolver os problemas propostos. Esta implementação buscou seguir os paradigmas da orientação a objetos. Este paradigma da programação tem como conceito principal os *Objetos*. *Objetos* são estrutura da dados que podem conter informações (na forma de atributos) e ações (procedimentos, geralmente chamados de métodos). Além de seguir os paradigmas da orientação a objetos, buscou-se também seguir sempre as boas práticas de programação e metodologias de trabalho em equipe para *software*.

Utilizou-se a estrutura de versionamento de dados Subversion [45] através da interface TortoiseSVN [46] para Windows para possibilitar não somente o historiamento dos dados bem como facilitar o trabalho em equipe no transcorrer do desenvolvimento da aplicação em questão.

Além disso, metodologias de Scrum [47, 48] foram utilizadas para organizar o trabalho a ser feito. Em que o *Developer Team* eram os autores do projeto e o *Scrum Master* era o orientador. O *Product Backlog* foi definido na definição do programa a ser desenvolvido. Os clientes era uma alternância entre os autores, por exemplo se o autor 1 precisava de uma funcionalidade que o outro estava desenvolvendo então o autor 1 era o cliente, mas se o autor 2 dependesse de algo do 1 então a situação invertia. Os *Sprint Planning Metting* eram encontros aos sábados dos autores e orientador.

A ferramenta *Trello* [49] foi utilizada para realizar uma espécie de KanBan [50].

6.2.1 Visão Computacional

O processo de filtragem de uma cor é bastante simples. O programa transforma a imagem capturada através da câmera numa matriz tridimensional contendo os valores de tom, saturação e brilho (HSV) [51] de cada pixel e usando as informações cadastradas pelo usuário "passa" essa matriz por um filtro passa-faixa de valores HSV retornando na saída uma imagem binária, ou seja, com o auxílio do OpenCV o programa verifica se o valor de cada pixel da matriz adquirida se encontra entre os valores mínimo e máximo de H, de S e de V cadastrados para a cor em questão. Se o pixel estiver dentro desses intervalos o valor desse índice na matriz será transformado em 1(um) e caso contrário

será 0(zero). Dessa forma, na saída do filtro possuí-se uma matriz binária. Ao mostrar essa nova matriz numa tela de vídeo, onde o índice estiver com o valor 1(um) a imagem mostrada será branca e será preta caso o desse valor seja 0(zero).

Após esse primeiro filtro, o sistema procura na imagem retornada os agrupamentos de pixel com características iguais. Esses agrupamentos são geralmente chamados de *blobs* e que possuem uma área medida em [$pixel^2$] passam agora por um filtro de área. Esse novo que também possui seu intervalo definido pelo usuário (valores mínimo e máximo de área) retira os *blobs* que estiverem fora dessa faixa, ou seja, os agrupamentos que possuam uma área menor do que o valor mínimo definido ou maior do que o valor máximo.

Feito o processo de filtragem, o sistema organiza os *blobs* encontrados em diversas listas cada uma contendo os agrupamentos encontrados para uma cor específica assim como os centroides de cada um. Todo este processo que se inicia com a entrada da imagem capturada e termina com as listas de agrupamentos de *blobs* por cor é feita de forma paralela de forma a aumentar a performance da execução deste processo e possibilitar uma maior escalabilidade da utilização do programa. Esse processo é paralelizado através da biblioteca OpenMP (*Open Multi-Processing*) [28]. O trecho de código 6.1 mostra este processo sendo realizado.

```
1  vector<vector<cv::Point>> Vision::getPosicaoCores(vector<Cor> pVCor, bool pDesenha){
2      Mat processedMat = cv::Mat(originalMatHSV.cols, originalMatHSV.rows, CV_8UC3);
3      vector<vector<cv::Point>> rtnVct(pVCor.size());
4
5  #pragma omp parallel for firstprivate(originalMatHSV,processedMat)
6      for (int idxCor = 0; idxCor < pVCor.size(); idxCor++)
7      {
8          Cor cor = pVCor.at(idxCor);
9          cv::inRange(originalMatHSV, cv::Scalar(cor.minH, cor.minS, cor.minV), cv::↩
               Scalar(cor.maxH, cor.maxS, cor.maxV), processedMat);
10         std::vector< std::vector<Point> > contours;
11         std::vector<Vec4i> hierarchy;
12         std::vector<int> IndicesValidosContorno;
13
14         findContours(processedMat, contours, hierarchy, CV_RETR_EXTERNAL, ↩
               CV_CHAIN_APPROX_TC89_KCOS);
15
16         if (!contours.empty()) {        // Encontra os  contornos
17             for (size_t i = 0; i < contours.size(); ++i) {
18                 double contour_area = contourArea(contours[i]);
19                 if (contour_area >= cor.minArea && contour_area <= cor.maxArea)
20                     IndicesValidosContorno.push_back(i);
21             }
22         }
23         // Desenha os contornos e seus centros
24         for (size_t i = 0; i < IndicesValidosContorno.size(); ++i) {
25             float centroX, centroY;
26             cv::Moments momentos = cv::moments(contours[IndicesValidosContorno[i]]);
27             centroX = momentos.m10 / momentos.m00;  centroY = momentos.m01 / momentos.↩
                   m00;
28             cv::Point centroObjeto = cv::Point(centroX, centroY);
```

```
29              rtnVct.at(idxCor).push_back(centroObjeto);
30          }
31      }
32      return rtnVct;
33  }
```

Código 6.1: Identificação da posição dos objetos de cada cor

Neste trecho de código 6.1 vale destacar a linha 5. Esta linha é onde está a marcação do OpenMP para dizer que este loop será realizado em paralelo. Nesta linha também está definido que ao paralelizar este loop as duas variáveis (`originalMatHSV` e `processedMat`) são `firstprivate`. Estas duas variáveis estão definidas fora deste loop que será paralelizado e serão acessadas por todas as suas iterações. Entretanto, como as iterações vão somente consultar as informações destas variáveis estas são ditas `firstprivate`, ou seja, será criada uma cópia da variável para cada iteração do loop de forma que não seja necessário nenhum controle de fluxo para organizar o acesso à este recurso.

Após essa etapa o código vai para o próximo passo que é a identificação de objetos. Para tal, o sistema seleciona o primeiro objeto da lista de objetos definidas no banco de dados e percorre as listas das cores que identificam aquele objeto com a finalidade de poder definir se esse objeto está ou não na imagem capturada pela câmera e em seguida passa para o próximo objeto da lista de objetos.

A identificação mencionada acima é feita da seguinte maneira: após selecionar as listas de cores correspondentes àquele objeto, o sistema busca nessas listas as duplas ou trios de *blobs* - dependendo de quantas cores o objeto possuir - que definem a menor distância entre os centroides dos *blobs*. A matriz 6.1 mostra uma possível situação em que o código está varrendo duas listas de cores na busca de identificação dos *blobs* que compõem um determinado objeto. Nela, cada elemento a_{ij} contém a distância em pixels do $blob_i$ para o $blob_j$. Para este exemplo em particular pode-se observar que o valor do índice a_{32} é o menor, logo os *blobs* que correspondem ao objeto que está sendo analisado são $blob_3$ para a cor primária e o $blob_2$ para a cor secundária.

		Cor Secundária		
		$blob_{p1}$	$blob_{p2}$	$blob_{p3}$
	$blob_{s1}$	800	256	84
Cor	$blob_{s2}$	52	413	68
Primária	$blob_{s3}$	777	21	354
	$blob_{s4}$	127	512	229

Tabela 6.1: Exemplo de buscas de *blob* com menor distância

Encontrada essa relação, esses *blobs* são retirados de suas respectivas listas, e fazendo as operações matemáticas vistas nas equações 6.2 e 6.3 obteu-se o centro e ângulo do objeto em análise:

$$X_{centro} = \frac{X_{centroide1} + X_{centroide2}}{2} \tag{6.1}$$

$$Y_{centro} = \frac{Y_{centroide1} + Y_{centroide2}}{2} \tag{6.2}$$

Aqui, X_{centro}, $X_{centroide1}$ e $X_{centroide2}$ são as coordenadas X do centro do objeto, do centroide da cor primária e do centroide da cor secundária respectivamente e Y_{centro}, $Y_{centroide1}$ e $Y_{centroide2}$ são as coordenadas Y do centro do objeto, do centroide da cor primária e do centroide da cor secundária respectivamente.

$$\Theta = ATan\left(\frac{X_{centroide1} - X_{centroide2}}{Y_{centroide1} - Y_{centroide2}}\right) + \frac{\pi}{4} \tag{6.3}$$

onde ATan significa o arco tangente. A imagem 6.2 foi usada para esclarecer a questão da soma de $\frac{\pi}{4}$:

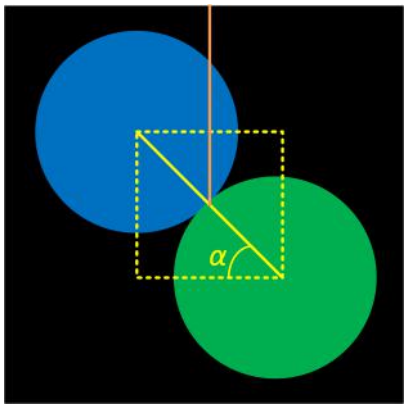


Figura 6.2: Diferença entre o ATan(x) e a direção do carro.

Como observado na figura acima, o ângulo α resultante do arco tangente é um dos ângulos agudos do triângulo retângulo isósceles formado por duas laterais de um quadrado e a diagonal do mesmo, sendo assim para obter-se a direção real (linha laranja indo do centro do carro até uma de suas laterais) do carrinho faz-se necessário somar os 45° vistos na equação 6.3 a fim de enfim adquirir a direção real do carro em relação as coordenadas do mundo.

Passada essa explicação, fica muito mais simples entender o que realmente é o processo de calibração das cores mencionado na seção 9.2.0.4, já que o ato de modificar o valor do máximo ou mínimo dos parâmetros H, S, ou V significa que está-se indicando ao programa qual é o intervalo desses parâmetros que correspondem a uma determinada cor. E o máximo e mínimo de área simbolizam o tamanho que deseja-se para identificar os círculos.

Por fim, é importante ressaltar que para um leitor mais observador o método utilizado para identificar um objeto poderia eventualmente vir a causar uma inconsistência já que

existiria a possibilidade de um segundo objeto muito próximo ao objeto que está sendo identificado ter uma cor igual à usada no objeto em análise e assim o programa talvez viesse a confundir os dois objetos como se fossem um só. Contudo, para esse leitor vale lembrar que o método para determinar que uma dupla ou trio formam o objeto que está-se procurando utiliza a menor distância entre os centroides dos *blobs* e como essa distância é sempre menor do que o raio dos carros utilizados então não haveria a possibilidade dessa confusão a não ser que um entrasse no outro o que é inviável.

6.2.2 Controle proposto para os robôs móveis

Tendo as técnicas de controle descritas nas seções 4.1 e 4.2 em mente pode-se expor aqui mais detalhadamente a implementação das mesmas.

Começando pelo *Cruise Control* vê-se de cara a necessidade de obter a velocidade linear de cada objeto. Para isso, pode-se utilizar os resultados obtidos na etapa de visão retratada na seção 6.2.1 sobre as posições de cada carrinho. Isolando um carrinho e servindo-se de dois quadros consecutivos pode-se obter a velocidade de duas maneiras distintas: a primeira seria calculando a distância $\Delta D = \sqrt{\Delta X + \Delta Y}$ percorrida pelo carro e dividindo esse valor por ΔT, isto é o tempo que se passou entre o primeiro e segundo quadro. Já a segunda via para se calcular a velocidade, seria com o auxílio de uma derivada suja no deslocamento em X e uma outra no deslocamento em Y, calcular as componentes de velocidades nos eixos X e Y e combiná-los de forma a obter a velocidade resultante do carro. Devido a existência de um filtro nessa segunda forma, gerando assim um resultado menos ruidoso, os autores decidiram utilizá-la na implementação do programa e assim teria-se:

Equação da Derivada Suja:

$$Valor_{Derivada} = \frac{\delta P_{[h-1]} + H P_{[h]}}{\delta + H} \tag{6.4}$$

onde, $Valor_{Derivada}$ é o valor resultante da derivada suja, H é uma constante relacionada com a taxa de amostragem do sinal (no caso o fps), δ é o inverso da constante H, $P_{[h-1]}$ é a posição anterior do carro e $P_{[h]}$ é a posição atual da carro.

$$\overrightarrow{V_X} = \frac{-1}{\delta} Valor_{DerivadaX} + \frac{1}{\delta} X_{[h]} \tag{6.5}$$

$$\overrightarrow{V_Y} = \frac{-1}{\delta} Valor_{DerivadaX} + \frac{1}{\delta} Y_{[h]} \tag{6.6}$$

$$\overrightarrow{V_R} = \sqrt{\overrightarrow{V_X}^2 + \overrightarrow{V_Y}^2} \tag{6.7}$$

em que, $\overrightarrow{V_X}$ é a componente de velocidade no eixo X, $\overrightarrow{V_Y}$ é a componente de velocidade no eixo Y e $\overrightarrow{V_R}$ é a velocidade resultante, representando a velocidade linear do carrinho.

Como está-se falando de um controle de velocidade de cruzeiro de um veículo, a medida da velocidade não é alterada bruscamente (inclusive para economizar combustível e dar conforto aos ocupantes). Sendo assim, utilizou-se a média das últimas quatro medidas da velocidade do robô como o valor de comparação da velocidade.

$$\overrightarrow{V_R^{med}} = \frac{1}{4} \sum_{i=t-3}^{t} \overrightarrow{V_R}(t) \tag{6.8}$$

Em nossos experimentos utilizou-se a média das últimas quatro medidas da velocidade. Entretanto, vale notar que, se tratando de um sistema funcionando perto de 30FPS, a média das últimas quatro medidas de velocidade equivale à velocidade média dos últimos 0,13s (130ms), ou seja, a diferença do tempo de reação da medida da velocidade média e uma alteração entre a velocidade real do robô é praticamente imperceptível. Considerando a suavidade do controle que já foi anteriormente mencionada, poderia-se adotar uma média de mais medidas sem prejuízo à percepção de uma eventual mudança de velocidade do robô.

Adquirida a velocidade atual do carrinho, o sistema calcula o erro do Cruise Control por meio da seguinte equação:

$$e_{vel} = \overrightarrow{V_{Ref}} - \overrightarrow{V_R^{med}}, \text{ em que } \overrightarrow{V_{Ref}} \text{ é o } \textit{setpoint} \text{ de velocidade definido pelo usuário.} \tag{6.9}$$

E assim, usa esse resultado como a entrada do controlador PID Linear onde por meio da equação 6.17 obtém enfim a saída de controle linear.

$$P_{L_L} = e_{vel} K_{P_L} \tag{6.10}$$

$$P_{I_L} = \sum_{n=0}^{h} e_{vel_{[n]}} K_{I_L} \tag{6.11}$$

$$P_{D_L} = \dot{e_{vel}} K_{D_L} \tag{6.12}$$

$$S_L = P_{L_L} + P_{I_L} + P_{D_L} \tag{6.13}$$

onde:

K_{P_L} É o ganho proporcional linear.

K_{I_L} É o ganho integral linear.

K_{D_L} É o ganho derivativo linear.

P_{L_L} É a parcela proporcional do PID linear.

P_{I_L} É a parcela integral do PID linear.

P_{D_L} É a parcela derivativa do PID linear.

$\sum$ É a soma de todos os erros desde o início até o presente momento.

$\dot{e_{vel}}$ É a derivada suja do erro de velocidade naquele instante.

S_L É a saída do controle linear.

Seguindo agora para o controle angular e lembrando do que foi visto na seção 4.2 tem-se que o erro para esse controle é a diferença entre o ângulo calculado ($\Delta\alpha$) e o ângulo da tangente do melhor ponto da trajetória em relação ao carrinho. Para encontrar o que chamou-se de "melhor ponto", ou seja, o ponto qual a menor distância entre o robô e a trajetória faz-se uma busca simples, calculando a distância para os pontos da trajetória e escolhendo o ponto com a menor distância. Este procedimento pode ser visto no trecho de código 6.2. Nesta função inicializa-se a variável `minDist` com o valor máximo possível para uma variável do tipo `double`. Então, para cada ponto da trajetória calculou-se a distância entre a posição atual e este ponto e, caso a distância seja menos do que o valor armazenado em `minDist` o valor desta variável é atualizado. Desta forma, ao final do loop pelos pontos tem-se o valor da menor distância e o ponto correspondente à esta distância.

```
1  tuple<Position, double> Trajetoria::getMelhorPonto(Position pPos){
2      double minDist = DBL_MAX;
3      double dist;
4      Position minPos;
5      for each (Position pos in pontos)
6      {
7          dist = pos.distance(pPos);
8
9          if (dist < minDist) {
10             minDist = dist;
11             minPos = pos;
12         }
13
14     }
15     return make_tuple(minPos, minDist);
```

Código 6.2: Busca melhor ponto na trajetória

O ângulo desejado (ângulo da tangente à trajetória no melhor ponto ajustado por um fator $\Delta\alpha(dist)$, onde *dist* é a distância entre a posição do robô e o ponto mais próximo da trajetória é então encontrado através da função que mostrada no trecho de código 6.3.

```
1  float Objeto::CalculaAngDesejado(bool sentido, Position mPos, double mDist, double ↩
       pKAngDes){
2      double k  = pKAngDes;
3      bool dentro = traj.contem(posAtual.x, posAtual.y);
4      double angDesejado, diferenca;
5
```

```
6       float satAngRad = mathHelper::Deg2Rad(85);  // Saturacao da diferenca angular em ↩
            radianos
7
8       if (k*mDist < 2) { int a = 1;}
9
10      double angMa = mPos.ang;
11
12      diferenca = k * mDist * pow(-1, dentro) * pow(-1, sentido);
13
14      if (abs(diferenca) > satAngRad){    diferenca = mathHelper::sgn(diferenca) * ↩
            satAngRad; }
15
16      if (sentido == 0){ angMa = 0 - (M_PI - angMa);  }
17      angDesejado = (angMa - diferenca);
18
19      return angDesejado;
20  }
```

Código 6.3: Cálculo referência de ângulo

Obtido esse erro, faz-se a seguinte filtragem:

Enquanto o valor do erro for menor do que $-\pi$ somou-se 2π e enquanto o valor desse erro for maior do que π retirou-se -2π. O trecho de código responsável por este tratamento pode ser visto em 6.4.

```
1   float Objeto::AjustaAngulo(float erroAng){
2
3       while (erroAng <= -M_PI){
4           erroAng = erroAng + 2 * M_PI;
5       }
6
7       while (erroAng >= M_PI) {
8           erroAng = erroAng - 2 * M_PI;
9       }
10
11      return erroAng;
12  }
```

Código 6.4: Ajuste erro do controlador angular

Dessa maneira consegue-se lidar com a descontinuidade causada pelo uso da função *atan2* no ponto π ou $-\pi$. Feito esse ajuste, assim como no *cruise control* passa-se o erro como entrada para o controle PID Angular e assim tem-se:

$$P_{L_A} = e_{vel} K_{P_A} \tag{6.14}$$

$$P_{I_A} = \sum_{n=0}^{h} e_{vel_{[n]}} K_{I_A} \tag{6.15}$$

$$P_{D_A} = \dot{e_{ang}} K_{D_A} \tag{6.16}$$

$$S_A = P_{L_A} + P_{I_A} + P_{D_A} \tag{6.17}$$

onde:

K_{P_A} É o ganho proporcional angular.

K_{I_A} É o ganho integral angular.

K_{D_A} É o ganho derivativo angular.

P_{L_A} É a parcela proporcional do PID angular.

P_{I_A} É a parcela integral do PID angular.

P_{D_A} É a parcela derivativa do PID angular.

$\sum_{n=0}^{h}$ É a soma de todos os erros desde o início até o presente momento.

$\dot{e_{ang}}$ É a derivada suja do erro angular naquele instante.

S_A É a saída do controle angular.

Com as saídas linear e angular encontradas, pode-se por fim calcular as saídas de tensão de cada roda do veículo v_1 e v_2 utilizando as seguintes expressões:

$$v_1 = \frac{0.05S_L + 0.05S_A}{K_{dim}} \tag{6.18}$$

$$v_2 = \frac{0.05S_L - 0.05S_A}{K_{dim}} \tag{6.19}$$

em que, K_{dim} é uma constante relacionado com a dimensão em metros do carrinho usado. Limitando os valores de v_1 e v_2 entre -255 e 255 devido ao protocolo de comunicação que será explicado na seção 6.2.7 pode-se enfim enviar uma mensagem ao carrinho com esses valores.

Para possibilitar que o controle trabalhe sempre com uma taxa de amostragem constante foi implementado um trecho de código que faz com que a taxa de FPS (*Frames* por segundo) fique constante. A ideia principal é a de que é melhor reduzir um pouco o valor do FPS e manter o mesmo constante do que possuir um FPS mais alto que oscile bastante. Sendo assim, no começo do loop principal da aplicação é verificado o tempo que se passou desde o início do loop anterior. Caso necessário o sistema ficará parado neste ponto até que o tempo total desde o início do loop anterior seja condizente com a taxa de quadros por segundo desejada. O trecho de código 6.5 mostra a implementação que foi realizada dentro do loop principal do programa. Vale ressaltar que caso o tempo passado desde o início do loop anterior seja maior do que o desejado o sistema continuará sua execução, porém sofrerá uma queda na quantidade de quadros por segundo. O código que foi implementado atua somente para abaixar o FPS para um valor que seja sempre possível de se manter de forma a oferecer uma taxa de amostragem mais constante.

```
double taxaH = ((clock() - tempo) / ((double)CLOCKS_PER_SEC));
while (taxaH <= (1.0 / 30.0))
{
    taxaH = ((clock() - tempo) / ((double)CLOCKS_PER_SEC));
}

tempo = clock();
```

Código 6.5: Implementação para fixar a taxa de quadro por segundos

6.2.3 Estratégia

6.2.3.1 Região Segura

Definiu-se uma região na qual será buscada por outros carros ou obstáculos para evitar uma possível colisão. Esta região, exibida na figura 6.3, é definida por um polígono de 5 pontos, a saber:

1. Ponto exatamente à frente do robô à uma distância D_s do centro do mesmo
2. Ponto 1 rotacionado $+45°$ em relação ao centro do robô, também com uma distância D_s do centro do mesmo
3. Ponto 1 rotacionado $-45°$ em relação ao centro do robô, também com uma distância D_s do centro do mesmo
4. Ponto 1 rotacionado $+90°$ em relação ao centro do robô, com uma distância $D_s/3$ do centro do mesmo
5. Ponto 1 rotacionado $-90°$ em relação ao centro do robô, com uma distância $D_s/3$ do centro do mesmo

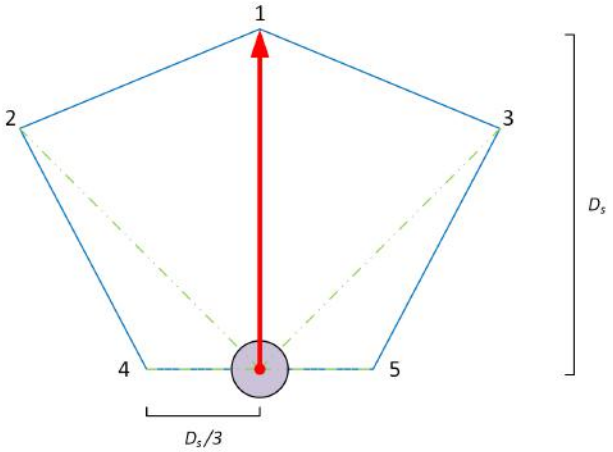


Figura 6.3: Região Segura

6.2.3.2 Evitar colisões

O trecho de código 6.6 mostra o algoritmo utilizado para decidir se um robô pode ou não ultrapassar outro robô que esteja mais lento à sua frente. A teoria utilizada para modelar esta solução foi explicitada na seção 4.3

```
if (objetos[idxObj].traj.nome == objetos[idxObj2].traj.nome) //mesma ótrajetria
{
    bool existeTrajAux = objetos[idxObj].trajAux.nome != ``''.
    double minDistU = 500; // ultrapassagem
    if (existeTrajAux) {
        for (int idxObj3 = 0; idxObj3 < objetos.size(); idxObj3++) {
            if (objetos[idxObj3].nome != objetos[idxObj].nome && objetos[idxObj3].nome ↩
                != objetos[idxObj2].nome){
                double dist = objetos[idxObj3].posAtual.distance(objetos[idxObj].↩
                    posAtual);
                if (dist < minDistU){
                    minDistU = dist;
                }
            }
        }
    }

    if (existeTrajAux && minDistU > distSegura) {
        objetos[idxObj].usaTrajAux = true;
    }
```

Código 6.6: Ultrapassagem

Já o trecho de código 6.7 mostra o algoritmo utilizado para decidir o *setpoint* de velocidade de um dado robô baseado no fato de existir um outro robô mais lento à sua frente no caso da impossibilidade de se realizar uma ultrapassagem. A teoria utilizada para modelar esta solução foi explicitada na seção 4.3

```
if (objetos[idxObj2].velAtual < limiarVelChange && objetos[idxObj2].velAtual > -↩
    limiarVelChange){
    velMax = objetos[idxObj].velAtual;
    kCoefFren = 0.5;
    objetos[idxObj].saidaLinearI = 0.25*objetos[idxObj].saidaLinearI;
}
else{
    velMax = objetos[idxObj2].velAtual;
    kCoefFren = 1.5;

    if (objetos[idxObj].saidaLinearI > 550) {
        objetos[idxObj].saidaLinearI = 550;
    }
}

if (objetos[idxObj].traj.nome != objetos[idxObj2].traj.nome) {
    coefFrenagem = objetos[idxObj].velAtual / (distSegura - minDist);
}
else{
```

```
19      coefFrenagem = mathHelper::abs(objetos[idxObj2].velAtual - objetos[idxObj].velAtual↩
            ) / (distSegura - minDist);
20  }
21
22  distance = objetos[idxObj].posAtual.distance(objetos[idxObj2].posAtual);
23  newSetPoint = velMax - (kCoefFren*coefFrenagem*(distSegura - distance));
24  objetos[idxObj].setPointVel = mathHelper::upperSat(newSetPoint, objetos[idxObj].↩
        setPointVelOrig);
```

Código 6.7: Frenagem

6.2.4 Plotter

Com o propósito de facilitar e agilizar a análise dos resultados experimentais obtidos durante todo o desenvolvimento do trabalho foi feito um sistema semiautomático de registrar e gerar gráficos. Esse sistema é divido em duas partes:

A primeira parte, feita na linguagem C++ é o registro dos dados experimentais. Para isso, uma classe denominada "*Logger*" foi desenvolvida. Nela um arquivo de texto (.txt) é construído contendo em sua 1ª linha o cabeçalho do que será registrado, por exemplo se desejar-se registrar os quadros, o nome do robô, a trajetória objetivo, as coordenadas (X, Y) desejadas e atuais, as velocidade lineares desejada e atual e os ângulos desejado e atual teria-se um arquivo com a seguinte aparência:

FrameNum	Robo	trajetoria	Xd	Yd	Xa	Ya	velAtualDerivSuja	velDesejada	angAtual	angDesejado
28	CarroTeste	circulo_r68_xc160_yc120	176	53	177	47	3.910935	80.0	0.380506	0.360416

Tabela 6.2: Exemplo de Log (arquivo de texto)

Após ter rodado um teste, passa-se à segunda fase em que pega-se o arquivo de texto com o nome "AnoMêsDia_HoraMinutoSegundo.txt" (por exemplo "20170129_-214544.txt" seria um log gerado no dia 29/01/2017 às 21:45:44) e usando um *script* 9.1 desenvolvido pelos autores no Matlab, gera-se os gráficos em função do tempo e/ou em função de uma outra grandeza por exemplo (X x Y) a fim de analisar os resultados. Para utilizar esse *script* basta abri-lo, alterar o nome do arquivo de log que será utilizado na geração de gráficos e apertar o botão de "*Run*". Os exemplos de gráficos gerados serão exemplificados na seção de Resultados.

6.2.5 Banco de Dados

Como o banco de dados utilizado é armazenado em um arquivo, ao iniciar o programa é feita a verificação se o arquivo existe. Caso o arquivo não exista o próprio programa cria um arquivo vazio. Após o programa encontrar o arquivo (seja o que já existia ou o que ele mesmo criou) passa-se à etapa de verificação das tabelas. O programa verifica se todas as tabelas que ele necessita para operar em sua normalidade existem. Caso alguma tabela

não seja encontrada ele a cria com o mesmo esquema que já foi previamente definido e apresentado neste trabalho. O trecho de código 6.8 mostra um exemplo da funcionalidade que verifica se uma determinada tabela existe e, caso contrário, cria a mesma.

```
1   const char *sqlSelect = ``SELECT * FROM tb_cor;''.
2   char **results = NULL;
3   int rows, columns;
4
5   rc = sqlite3_get_table(db, sqlSelect, &results, &rows, &columns, &error);
6
7   if (rc) {
8       std::string str(error);
9       Logger::Log(''.rror executing SQLite3 statement: `` + str);
10      sqlite3_free(error);
11
12      Logger::Log(''.riando tabela tb_cor ``);
13
14      const char *sqlCreateTable = ``CREATE TABLE tb_cor (nome STRING PRIMARY KEY, ↩
            minH REAL, maxH REAL, minS REAL, maxS REAL, minV REAL, maxV REAL, minArea ↩
            REAL, maxArea REAL);''.
15
16      rc = sqlite3_exec(db, sqlCreateTable, NULL, NULL, &error);
17      if (rc) {
18          string str(sqlite3_errmsg(db));
19          Logger::Log(''.rror executing SQLite3 statement: `` + str);
20          sqlite3_free(error);
21      }
22      else {
23          Logger::Log(''.b_cor criada com sucesso''.;
24      }
25  }
26  else    {
27      Logger::Log(''.abela tb_cor ja existe.''.;
28  }
```

Código 6.8: Exemplo funcionalidade que verifica existência tabela

Além das transações realizadas pelo programa desenvolvido pelos autores e com a integração ao MatLab, foi também utilizado uma terceira ferramenta para auxiliar no desenvolvimento do modelo de dados. Essa ferramenta é um gerenciador de banco de dados específico para o SQLite e que se integra ao browser Mozilla Firefox [52]. Com essa ferramente pôde-se de forma fácil realizar alterações e consultas ao modelo de dados sem precisar alterar a aplicação.

6.2.5.1 Estrutura dos dados

Na Figura 6.4 pode-se ver o modelo de dados criado para a aplicação. O modelo consiste em cinco tabelas: `tb_cor`, `tb_objeto`, `tb_paramcontrole`, `tb_ponto`, e `tb_trajetoria`. O objetivo de cada tabela, bem como considerações sobre os relacionamentos entre as mesmas serão explicitados em seguida.

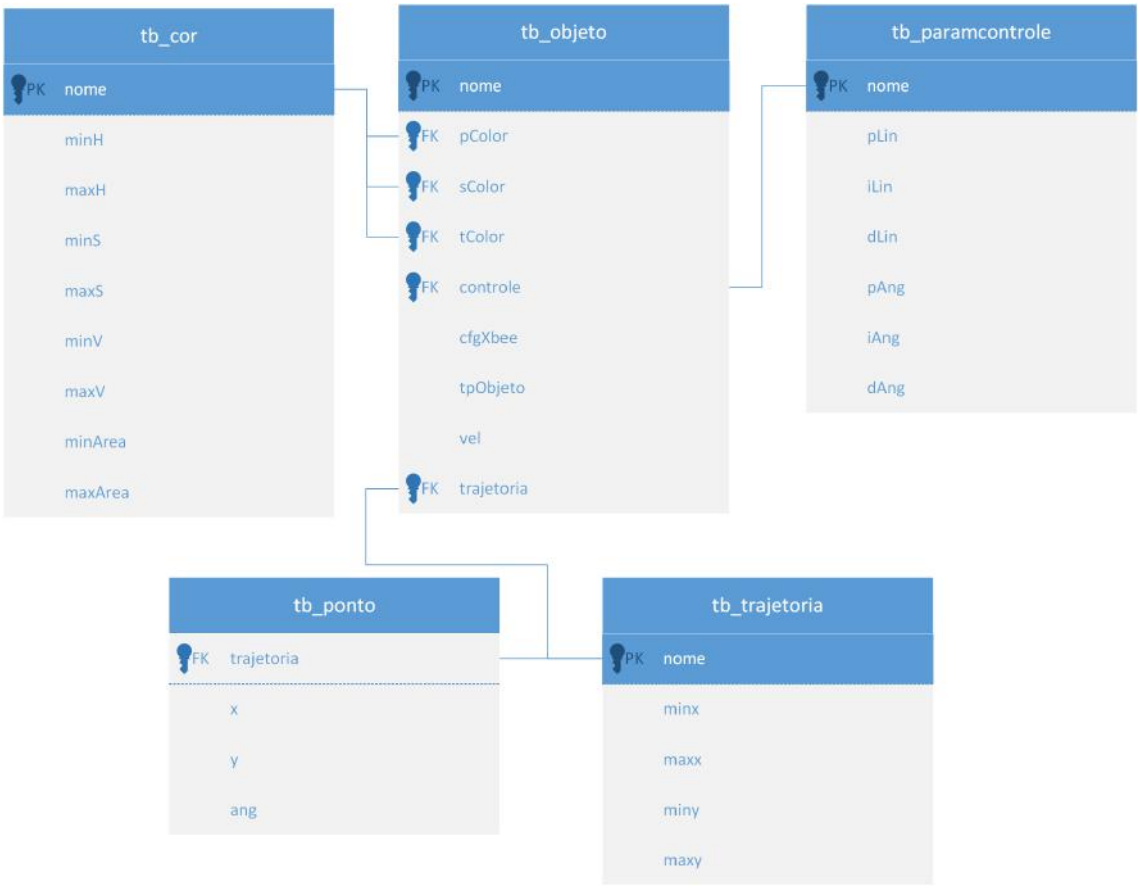


Figura 6.4: Modelo de Dados

tb_trajetoria A `tb_trajetoria` foi criada com o objetivo de armazenar as informações sobre todas as trajetórias disponíveis na aplicação. A coluna `nome` é a chave primária e é o nome que foi dado para cada trajetória.

tb_ponto A tabela `tb_ponto` foi criada para armazenar todos os pontos que foram gerados para cada trajetória bem como a informação do ângulo da tangente da curva em um determinado ponto. Para isto a coluna `trajetoria` é chave estrangeira com a `tb_trajetoria`, referenciando o nome da trajetória e as colunas `x`, `y` e `ang` armazenam as informação das coordenadas (x,y) e o ângulo em cada ponto respectivamente.

tb_cor A tabela `tb_cor` é responsável por armazenar o cadastro de todas as cores utilizadas pelo programa bem como as informações de filtro já configuradas para as mesmas. `nome` é a chave primária e representa um nome para a cor (e.g. "Azul".. `minH`, `maxH`, `minS`, `maxS`, `minV`, `maxV` representam os valores mínimos e máximos para as três componentes de cada cor (*Hue, Saturation e Value*. `minArea` e `maxArea` representam os valores mínimo e máximo da área de um blob em pixels.

tb_paramcontrole A tabela `tb_paramcontrole` tem como objetivo gravar conjuntos de parâmetros para os controladores PID de cada um dos robôs. A coluna `nome` é a chave primária da tabela e armazena um nome para a configuração em questão. As colunas `pLin`, `iLin` e `dLin` correspondem aos parâmetros PID do controlador de velocidade (CruiseControl) de cada robô e as colunas `pAng`, `iAng` e `dAng` correspondem aos parâmetros PID do controlador angular.

tb_objeto A `tb_objeto` tem como o objetivo guardar as informações relativas a todos os objetos com os quais a nossa aplicação lida, sejam eles robôs, obstáculos ou outros objetos. A chave primária á a coluna `nome`. `pColor`, `sColor` e `tColor` são chaves estrangeiras com a `tb_cor` e se referem as cores primária, secundária e terciária (quando necessário), respectivamente. A coluna `cfgXbee` é responsável por armazenar a string com o endereço configurado para o rádio transmissor Xbee instalado em cada um dos robôs (e.g. "ACAC".. A coluna `vel` armazena o *setpoint* de velocidade para o controle de velocidade de cruseiro também dos robôs. Já `trajetoria` é chave estrangeira com a `tb_trajetoria` e representa a trajetória que o robô está configurado para seguir.

6.2.5.2 Mapeamento para objetos

Embora a aplicação utilize o banco de dados descrito para salvar e persistir seus dados e configurações, usar os dados somente no banco de dados e consulta-los quando necessário se mostraria custoso para a performance do programa . No caso de um robô seguindo

uma dada trajetória, por exemplo, para escolher o melhor ponto que deve-se considerar a tangente precisa-se varrer boa parte dos pontos da trajetória. Em se tratando de 1000 pontos em média , ler estes dados através de consultas SQL ao banco de dados se mostraria impeditivo para a taxa de amostragem que se desejou obter com este trabalho.

Sendo assim, conforme os dados se mostram necessários, o programa busca os dados no banco de dados e carrega entidades em memória com atributos que guardam as informações antes salvas no banco. Um exemplo seria a entidade `objeto`. Esta entidade possui os atributos de cores que por sua vez também são entidades. `Objeto` também possui atributo `Trajetória` que detém as informações básicas da trajetória bem como um `vector` de `Positions`. `Position` por sua vez é uma entidade que armazena coordenadas x, y e um ângulo.

6.2.6 Geração de Trajetórias

Conforme já foi explicado na seção 6.1.5, utilizou-se para a geração prévia das trajetória scripts de *MatLab* salvando os dados no banco de dados em *SQLite* criado para esta aplicação. Para isto, primeiramente precisou-se definir a forma da trajetória. Uma forma de se gerar os pontos para a trajetória e, conhecendo-se os pontos por onde a mesma passa, utilizar um *polyfit* para encontrar dois polinômios que representem $x(t)$ e $y(t)$ da trajetória desejada. Entretanto, caso se conheça a forma da trajetória pode-se utilizar diretamente a equação da mesma. No exemplo 6.9 será utilizada a equação conhecida de um círculo. Dado que está-se trabalhando com uma resolução de 320x240 pixels, irá-se criar uma trajetória que seja um circulo de raio 60px, centrado no centro da imagem $(160, 120)$.

```
1  close all; clear; clc; format long;
2
3  nPoints = 1000;
4
5  syms t;
6  r_circ = 60; %raio do circulo
7  yc =120; %coordenada y do centro
8  xc = 160; %coordenada x do centro
9  x = r_circ*cos(t)+xc; % equacao de x(t)
10 y = r_circ*sin(t)+yc; % equacao de y(t)
11 xd = -r_circ*sin(t);    % derivada da equacao de x(t)
12 yd = r_circ*cos(t);   % derivada da equacao de y(t)
13
14 lstAtan2 = [];
15
16 for i=0:nPoints
17     t = (2*pi)*(i/nPoints); %normalizado
18     lstAtan2 = [lstAtan2; eval(x) eval(y) atan2(eval(yd), eval(xd))];
19 end
20
21 conn = sqlite('MyDb.db'); %abre a conexao com a base SQLite
22
23 %nome da trajetoria que sera salva
```

```
24 nomeTrajetoria = ['circulo_r' num2str(r_circ) '_xc' num2str(xc) '_yc' num2str(yc)];
25
26 %apaga todos os pontos e informacoes da trajetoria caso existam
27 exec(conn, ['delete from tb_ponto where trajetoria = ''' nomeTrajetoria ''''])
28 exec(conn, ['delete from tb_trajetoria where nome = ''' nomeTrajetoria ''''])
29
30 minx = min(lstAtan2(:,1));    %
31 maxx = max(lstAtan2(:,1));   % calcula os valore de maximo de minimo
32 miny = min(lstAtan2(:,2));    % tanto para x quanto  para y
33 maxy = max(lstAtan2(:,2));   %
34
35 %criacao do registo da trajetoria
36 exec(conn, ['insert into tb_trajetoria values (''' nomeTrajetoria ''', ' num2str(minx) '
       , ' num2str(maxx) ', ' num2str(miny) ', ' num2str(maxy) ', ''S'', ''320x240'' )'])
37
38 data = {};
39 colnames = {'trajetoria', 'x', 'y', 'ang'};
40
41 for idxLst=1:nPoints + 1 % monta a cell com os dados
42     data{idxLst,1} = nomeTrajetoria;
43     data{idxLst,2} = num2str(lstAtan2(idxLst,1),16);
44     data{idxLst,3} = num2str(lstAtan2(idxLst,2),16);
45     data{idxLst,4} = num2str(lstAtan2(idxLst,3),16);
46 end
47
48 data_table = cell2table(data,'VariableNames',colnames); % converte para um table
49 tablename = 'tb_ponto';
50 insert(conn,tablename,colnames,data_table); %insere os dados
51 close(conn) %fecha a conexao
```

Código 6.9: Exemplo de geração de trajetória

Dado que está-se medindo a posição dos objetos em questão também em pixels pode-se afirmar que todas as medidas de posição são números inteiros. Desta forma é correto afirmar que qualquer intervalo entre dois pontos que seja menor do que 1px fará com que o mesmo pixel possua dois pontos da trajetória em questão. Logo, com o objetivo de economizar processamento ao gerar a trajetória, espaço no banco de dados para armazenar os pontos da mesma e também processamento ao comparar a distância da posição do objeto com os pontos da trajetória, utilizou-se um valor tal de pontos que seja próximo a se ter somente um ponto por pixel, mas com alguma margem de segurança.

Conforme viu-se no exemplo 6.9, geraram-se informações para apenas 1000 pontos de cada trajetória. Para ilustrar o motivo que levou a escolher este número, imagina-se uma elipse de tamanho tal que tangenciasse todos os quatro lados da pista utilizada para nossos testes (figura 6.5.

Como a câmera é posicionada a uma distância tal da pista que faça com que a pista caiba perfeitamente em um *frame* e, dado que está-se utilizando a resolução de 320x240 px, tem-se que o menor eixo desta elipse terá tamanho $120px$ e o maior $160px$. Vai-se então calcular o perímetro (C) desta elipse. Tomando a como o maior eixo e b como o menor eixo tem-se:

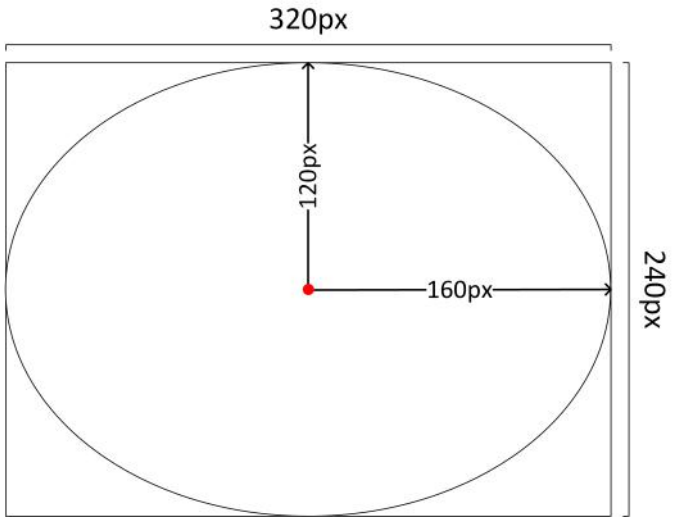


Figura 6.5: Maior elipse dentro da pista

$$C \approx \pi(a+b)\left(3\frac{(a-b)^2}{(a+b)^2\left(\sqrt{-3\frac{(a-b)^2}{(a+b)^2}+4}+10\right)}+1\right)$$

$$= \pi\cdot(160+120)\cdot\left(3\cdot\frac{(160-120)^2}{(160+120)^2\cdot\left(\sqrt{-3\cdot\frac{(160-120)^2}{(160+120)^2}+4}+10\right)}+1\right)$$

$$\approx 884.13969$$

Como pode-se perceber, o perímetro da elipse apresentada anteriormente é de $884.13969px$. Para que se tenha alguma margem de segurança ao gerar várias trajetórias considerou-se então o valor de 1000 pontos para cada trajetória.

6.2.7 Comunicação

O objetivo do presente trabalho não foi o desenvolvimento de programa embarcado para um robô móvel. Sendo assim, todo o processamento dos dados obtidos e toda a tomada de decisão é realizada no computador onde o programa principal é executado. Conforme já foi explicitado anteriormente neste mesmo capítulo, o programa principal é executado em *loop*. Este *loop* se inicia com a aquisição de uma imagem da câmera, o que poderia ser substituído pela aquisição de dados de outro(s) sensores, passa pelos algorítimos de estratégia e de controle e então são calculados, baseando-se no modelo do robô, os valores das tensões dos dois motores do robô de forma que este execute o que o programa calculou.

O robô *3pi* utiliza um sinal PWM explicado na seção 2.2 para controlar a velocidade e um sinal digital simples para controlar o sentido de rotação de cada um dos motores. Sendo assim, para que fosse possível se aproveitar de todo o intervalo de velocidades que o hardware do robô nos permite utilizar seria necessário que o programa principal enviasse um bit (0 ou 1) de sentido e um byte (0-255) com a velocidade de rotação para cada motor. Desta forma, como a comunicação serial é feita em bytes, precisaria-se de pelo menos 3 bytes de informação pura para garantir que não estaria-se discretizando demais as velocidades.

Além disso, como os experimentos seriam realizados com vários robôs, o protocolo de comunicação deveria prever alguma forma de endereçamento das mensagens de forma que cada robô soubesse quando a mensagem deveria ser interpretada por ele ou não. Por fim, medidas de segurança também seriam bem vindas ao protocolo pois se tratando de vários bytes poderia acontecer de algum deles não ser transmitido com sucesso. Por isso também fez-se necessária a utilização de um *checksum* para verificar se a mensagem recebida era realmente a mensagem enviada pelo transmissor.

Conforme já explicado na seção 2.2, escolheu-se o XBee como forma de comunicação serial. O XBee, se configurado corretamente, possui um próprio protocolo de comunicação que implementa parte das verificações que desejou-se para nossa comunicação. Cada dispositivo pode ser configurado para fazer parte de uma rede. Nesta rede cada um deles possui um endereço único que será utilizado para direcionar as mensagens aos mesmos. Além disso dispositivos podem ter funções distintas dentro de uma rede:

Coordenador Pode enviar mensagens a todos os dispositivos da sua rede e receber mensagens dos mesmos.

Dispositivo final Somente recebe as mensagens que foram direcionadas para o seu endereço e envia mensagens somente ao seu coordenador

Roteador Recebe todas as mensagens enviadas no seu grupo, as que são direcionadas para o seu endereço são processadas e as que não são são retransmitidas. Dispositivos roteadores são uma forma de aumentar o raio de alcance do sinal de comunicação.

Em nosso caso, como está-se lidando somente com uma fonte transmissora e a uma distância pequena da mesma em relação aos dispositivos receptores instalados em cada robô, escolheu-se configurar o dispositivo ligado ao computador (e portanto ao programa principal) como Coordenador e os demais dispositivos como Dispositivos Finais (*End-Point*). O protocolo próprio do XBee não é totalmente embarcado nos dispositivos de forma a se ter que se preocupar somente com as informações que devem ser transmitidas, entretanto caso a mensagem seja mensagem corretamente é garantido que esta só será

"entregue" ao dispositivo ao qual foi endereçada e que esteja na mesma rede do dispositivo que transmitiu esta mensagem[53] . Abaixo tem-se a estrutura da mensagem montada no programa principal:

0x7E Cabeçalho da mensagem, representa o início da mesma

0x00 Byte mais significativo do tamanho da mensagem

0x08 Byte menos significativo do tamanho da mensagem

0x01 Frame Type

0x01 Frame ID

End_1 Byte mais significativo do endereço do dispositivo que irá receber a mensagem

End_2 Byte menos significativo do endereço do dispositivo que irá receber a mensagem

0x01 Disable ACK

Sentido Ambos para frente: 0xFF, ambos para trás: 0x00, M_1 para frente e M_2 para trás: 0xF0, M_2 para frente e M_1 para trás: 0x0F

Vel_1 Velocidade (0-255) para o Motor 1

Vel_2 Velocidade (0-255) para o Motor 2

Sum *Checksum* da mensagem

O trecho de código 6.10 mostra a implementação da função que realiza o envio da mensagem através do protocolo acima mostrado.

```
1  bool Serial::EnviarMensagem(float velocidade1, float velocidade2, std::string address)
2  {
3      unsigned char sum = 0;
4      char msg[12] = ``'';
5
6      msg[0] = 0x7E;
7      msg[1] = 0x00;
8      msg[2] = 0x08;
9      msg[3] = 0x01;
10     msg[4] = 0x01;
11     msg[5] = std::stoi(address.substr(0, 2), 0, 16);
12     msg[6] = std::stoi(address.substr(2, 2), 0, 16);
13     msg[7] = 0x01;
14     if ((velocidade1 > 0)  && (velocidade2 > 0) ) {
15         msg[8] = 0xFF;
16     } else if ((velocidade1 > 0) && !(velocidade2 > 0) )     {
17         msg[8] = 0xF0;
```

```
    } else if (!(velocidade1 > 0)  && (velocidade2 > 0) ) {
        msg[8] = 0x0F;
    } else if (!(velocidade1 > 0)  && !(velocidade2 > 0) ) {
        msg[8] = 0x00;
    }
    msg[9] = abs(velocidade1);
    msg[10] = abs(velocidade2);

    for (int i = 0; i < 12; i++){
        if (i>2 && i < 11) sum = sum + msg[i];
    }
    sum = 0xFF - sum;
    sum = 0xFF & sum;
    msg[11] = sum;

    return (WriteData(msg, 12));
}
```

Código 6.10: Envio de mensagem via serial

Código embarcado no robô:

```
#include <pololu/3pi.h>
unsigned char rec_buffer_pos = 0; char rec_buffer[12]; int p = 0; int direcao;
int vel1; int vel2;

int main() {
    serial_set_baud_rate(9600);
    serial_receive_ring(receive_buffer, sizeof(receive_buffer));
    while (1) { check_for_new_bytes_received(); }
}

void check_for_new_bytes_received()
{
    unsigned char c = serial_get_received_bytes();

    if (c == 0) {
        receive_buffer_position = 0;
    }
    else if (c > 10) {
        char byte = rec_buffer[rec_buffer_pos];
        p = (int)rec_buffer_pos;

        if ((int)(byte) == 126) {
            if ((int)(rec_buffer[p+1]) == 0 && (int)(rec_buffer[p+2]) == 8 &&
                (int)(rec_buffer[p+4]) == 204 && (int)(rec_buffer[p+5]) == 204) {

                direcao = (int) rec_buffer[p+8];
                vel1 = (int) rec_buffer[p+9];
                vel2 = (int) rec_buffer[p+10];

                switch (direcao) {
                case 255: //0xFF ->Os dois para frente
                    set_motors(vel1, vel2);
                    break;
                case 0: //0x00 ->Os dois para átrs
                    set_motors(-vel1, -vel2);
                    break;
```

```
37                  case 240: //0xF0 -> Motor1 para frente e Motor2 para átrs
38                      set_motors(vel1, -vel2);
39                      break;
40                  case 15: //0x0F -> Motor1 para átrs e Motor2 para frente
41                      set_motors(-vel1, vel2);
42                      break;
43                  }
44                  rec_buffer_pos ++;
45              }
46          }
47          else {rec_buffer_pos++;}
48      }
49  }
```

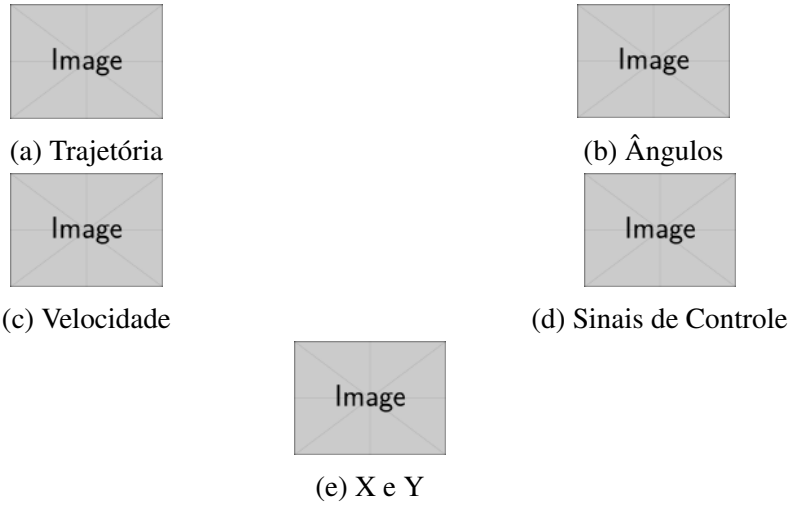
Código 6.11: Código Embarcado no robô

Capítulo 7

Resultados

Neste capítulo serão explicados os procedimentos experimentais realizados, os resultados destes experimentos, bem como a metodologia utilizada para analisar e avaliar estes resultados. Os experimentos foram conduzidos utilizando o hardware descrito no capítulo 2, o programa descrito no capítulo 6, bem como a modelagem e o controle explicitados nos capítulos 3 e 4 do presente trabalho.

Conforme explicado na seção 6.2.4, para que fosse possível coletar os dados e posteriormente analisar os dados dos experimentos realizados, salvou-se os dados num arquivo de texto e posteriormente processaram-se os mesmos com um script MatLab. O resultado deste script são cinco gráficos distintos para cada robô envolvido no experimento. Estes gráficos serão utilizados na análise dos resultados e serão dispostas conforme exemplo na figura 7.1 .



(a) Trajetória (b) Ângulos

(c) Velocidade (d) Sinais de Controle

(e) X e Y

Figura 7.1: Exemplo disposição resultados experimentais

A primeira das cinco figuras (7.1a), exibida no canto superior esquerdo, é o gráfico de Trajetória. Conforme explicado na seção 4.2, dado que tem-se a posição atual do robô o algoritmo escolhe qual o melhor ponto na trajetória para servir como referência para

o controle angular. Neste gráfico a posição do robô em cada frame é mostrada como um ponto vermelho e o melhor ponto escolhido como referência é mostrado em azul. O eixo horizontal representa a componente x da posição em pixels (px). O eixo vertical representa a componente y da posição também em pixels (px).

A segunda figura (7.1b), localizada no canto superior direito, é o gráfico de Ângulos. Este gráfico mostra a comparação entre o ângulo do robô em cada *frame* (representado na cor vermelha) com o ângulo do melhor ponto escolhido (representado na cor azul). Os ângulos são medidos em radianos (rad). O eixo horizontal corresponde à medida de tempo que, em nossos experimentos, estamos utilizando o número sequencial do *frame*.

A figura 7.1c representa o gráfico de velocidade. O eixo horizontal, tal como na figura 7.1b, representa a medida de tempo (*frames*). Já o eixo vertical representa a medida de velocidade (pixels/s). Em azul é mostrada a velocidade desejada, ou seja, o *setpoint* do controlador de velocidade explicado na seção 4.1. Em vermelho é mostrada a medida da velocidade atual do robô. Conforme explicitado na seção 6.2.2 a medida de velocidade é obtida através da média dos últimos quatro valores da velocidade calculada através das composições das velocidades nos eixos X e Y obtidas através da derivada suja.

Já a figura 7.1d representa os Sinais de Controle. O fim do loop de controle implementado monta dois sinais com valores entre -255 e 255. Estes dois valores são os valores que são transmitidos para cada robô através do XBee utilizando o protocolo descrito na seção 6.2.7. Estes dois valores são os dois mostrados nos gráficos da figura 7.1d. O eixo horizontal também representa o tempo (*frame*) e o eixo vertical representa o valor enviado ao robô e que será convertido em tensão nos terminais de cada motor pela Ponte-H.

Por fim, a figura 7.1e representa as componentes X e Y da posição do robô ao longo do tempo. O gráfico superior mostra a componente X da posição atual do robô para cada *frame* em vermelho e a componente X do ponto escolhido da trajetória, onde idealmente o robô deveria estar, em azul. De forma similar o gráfico inferior mostra a componente Y da posição atual do robô ao longo do tempo (*frame*) em vermelho enquanto em azul é mostrado a componente Y do melhor ponto escolhido.

Para avaliar quantitativamente os resultados obtidos escolheu-se a metodologia do *Mean Squared Error (MSE)* [39]. O MSE nada mais é do que a média dos quadrados dos erros de cada medida. A equação 7.1 ilustra o erro médio quadrático de um sinal X, onde X_d representa o valor desejado e X_a representa o valor atual, ou valor medido. O MSE para cada um dos sinais pode ser obtido de forma simples através da utilização da função `immse` do MatLab.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(X_d - X_a)^2 \tag{7.1}$$

Juntamente com os gráficos acima explicitados será mostrada também uma pequena tabela com os valores de MSE para cada um dos sinais principais (x, y, θ e Vel), bem como os parâmetros utilizados nos controladores de velocidade e de ângulo.

7.1 Círculo de raio 60 px centrado em (160,120)

Neste experimento um robô deverá percorrer a trajetória mostrada na figura 7.2 buscando alcançar a velocidade de 50px/s. O robô está inicialmente parado (velocidade 0) na posição (132,146).

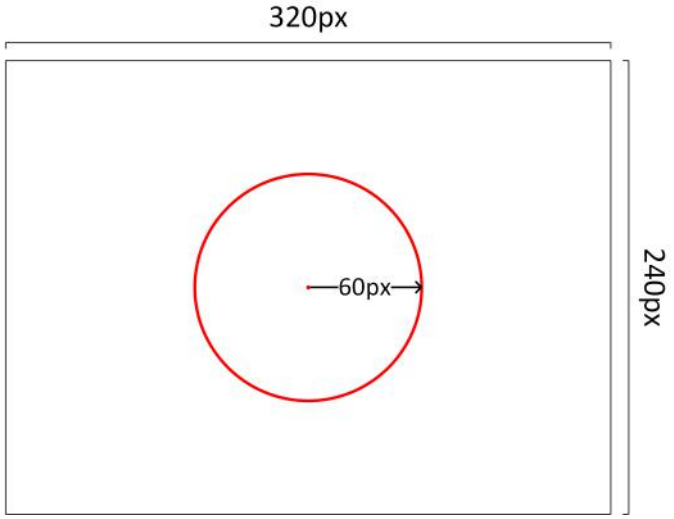


Figura 7.2: Circulo de raio 60 px centrado em (160,120)

Resultados

Variável	MSE
x	20,0562
y	14,0870
θ	5,0247
Vel	361,4447

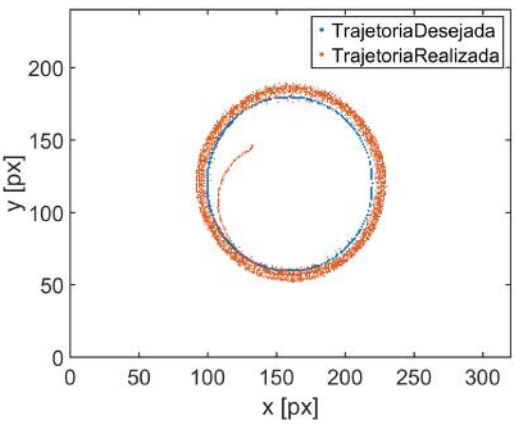
(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

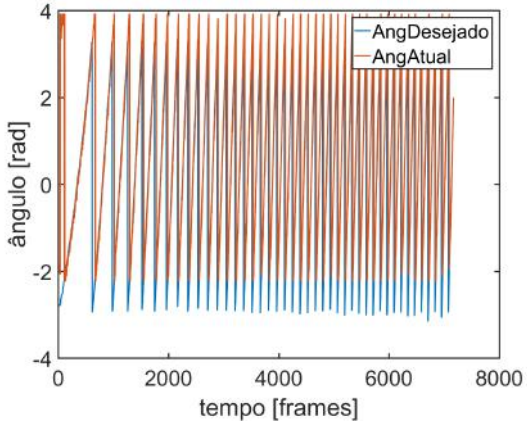
(b) Parâmetros utilizados nos controladores

Figura 7.3: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores

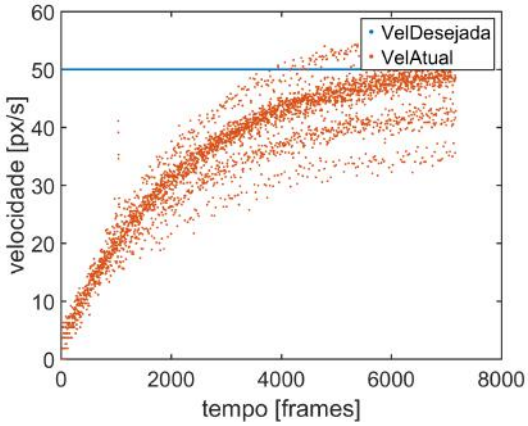
Analisando primeiramente a trajetória desejada pode-se observar pela figura 7.4a que o resultado do experimento foi bastante bom uma vez que o carro não só segue a trajetória desejada como chega na mesma bastante rápido. Além disso, ao observar as figuras 7.4b e 7.4 verifica-se que tanto as coordenadas X e Y são seguidas com um erro bem pequeno (em torno de 4 pixels para cada coordenada) reforçando a análise sobre o seguimento da



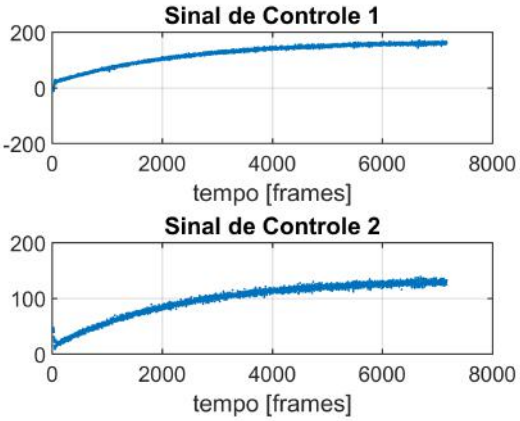
(a) Trajetória



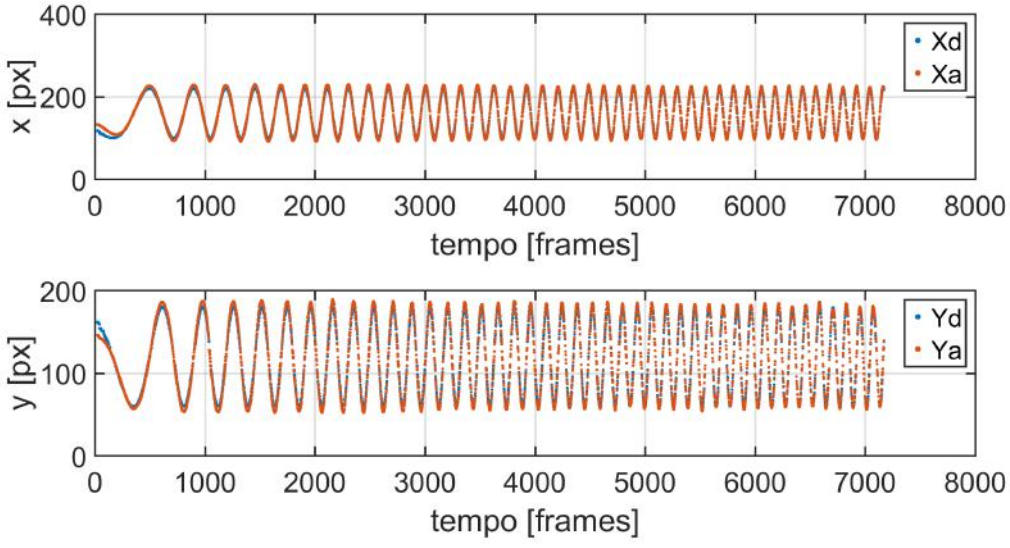
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 7.4: Resultados Resultado experimento círculo de raio 60 px centrado em (160,120)

trajetória. Somado ao que foi observado até agora, vê-se que os sinais de controle estão bem comportados, isto é sempre dentro dos limites estabelecidos (-255 até 255).

Entretanto aqui vale uma observação quanto ao ângulo (figura 7.4b) pode-se observar uma defasem entre o valor desejado e o valor atual do carrinho, isso tendo em vista que como é um sistema real, o controle não é capaz de prever o futuro da referência, sendo assim tem-se sempre um atraso na resposta do carrinho em relação a uma mudança de referência.

Por fim, analisando o gráfico da velocidade linear do carro (figura 7.4c) pode-se inicialmente ver que existe uma dispersão muito grande dos pontos medidos, isso se deve à imprecisão da medida de velocidade feita por intermédio de uma câmera, seguida de uma derivada suja como mencionado na seção 4. Contudo, apesar dessa incerteza pode-se notar que a média dos pontos tende para o valor de *setpoint* definido no experimento. Além disso, o valor de erro observado para a velocidade se deve não só ao fato desse ruído da medida mas também por ter-se a medida somente do período transitório da velocidade.

7.2 Oval de Cassini

Neste experimento buscou-se testar o comportamento do sistema modelado em uma trajetória um pouco mais elaborada do que um círculo. Para isso escolheu-se uma Oval de Cassini. A trajetória gerada e utilizada para este teste pode ser observada na figura 7.5

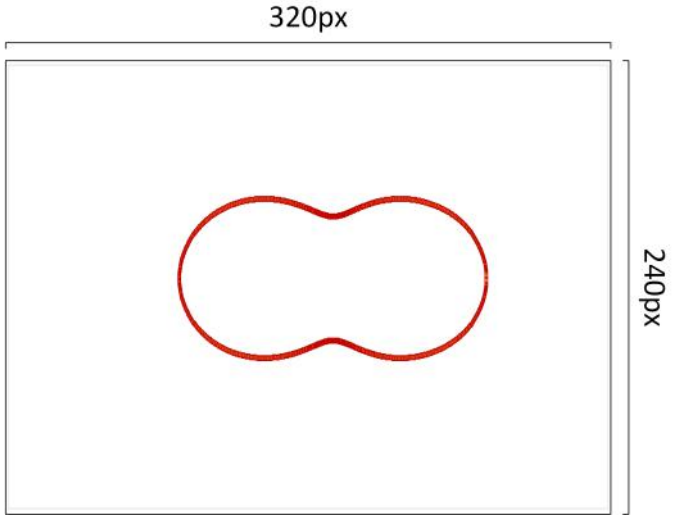


Figura 7.5: Oval de Cassini

Resultados

Nesse experimento pode-se observar um resultado parecido ao visto no experimento 7.1, isto é, em geral o carro segue a trajetória desejada bastante bem (erros nas coordena-

Variável	MSE
x	45,1818
y	20,2571
θ	5,7853
Vel	89,2947

(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

(b) Parâmetros utilizados nos controladores

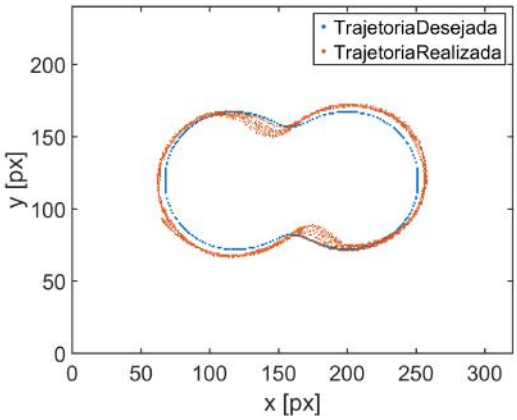
Figura 7.6: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores

das não muito grandes) sem que os valores de controle precisem ser saturados. Contudo, observou-se que nos sinais de controle por exemplo, passaram a ocorre picos nos instantes em que a mudança de referência angular se torna muito rápida e acentuada. Nesses mesmos instante notou-se que o seguimento do ângulo, devido ao atraso e a uma taxa de quadros não muito alta (em torno de 17 fps), não acompanha com perfeição as transições na referência, contudo, tem um resultado bastante satisfatório dado que o carro continua seguindo a trajetória desejada sem se perder. Reparou-se por fim, que apesar de ainda bastante disperso os resultados vistos na medida de velocidade estão mais coesos e assim como visto no experimento anterior tendem ao valor de *setpoint*.

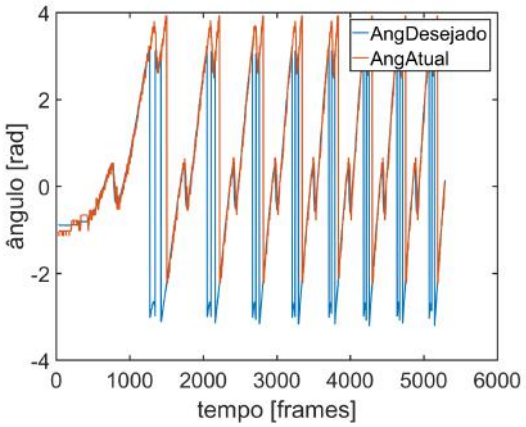
7.3 Círculo de raio 60px centrado em (160,120) com inserção de obstáculo

Resultados

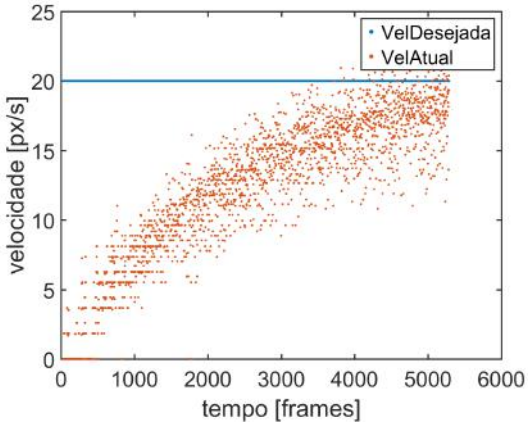
Nesse experimento usou-se a mesma trajetória que a usada no experimento 7.1, fazendo com que o início do experimento tivesse um resultado bem parecido com aquele visto no experimento mencionado. Entretanto no instante 2,9s um obstáculo foi posto em cima da trajetória desejada nas coordenadas (194,70). Dito isso, pode-se observar que assim como o resultado visto no primeiro experimento realizado, o carrinho segue com uma performance bastante aceitável a trajetória a ele imposta mantendo os sinais de controle sempre dentro de valores toleráveis. Essa afirmação é reforçada ao observar os gráficos (a), (b), (d) e (e) até o instante 2,9s. Nesse momento, pode-se observar que o sistema do carro ao detectar o obstáculo no caminho força o *setpoint* de velocidade do carrinho para o valor de 0pixels/s com o propósito de evitar a colisão. Nota-se também, que ao se alterar o *setpoint* de velocidade, o carro segue rapidamente para esse valor sem que



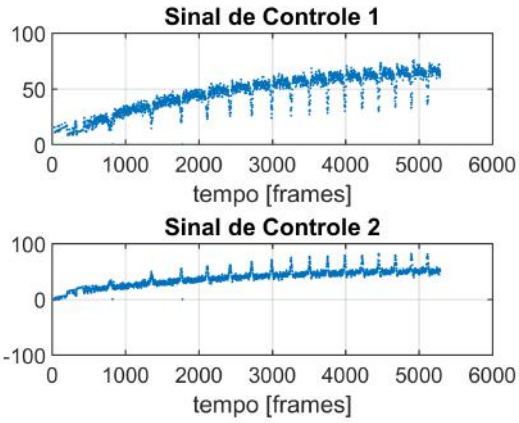
(a) Trajetória



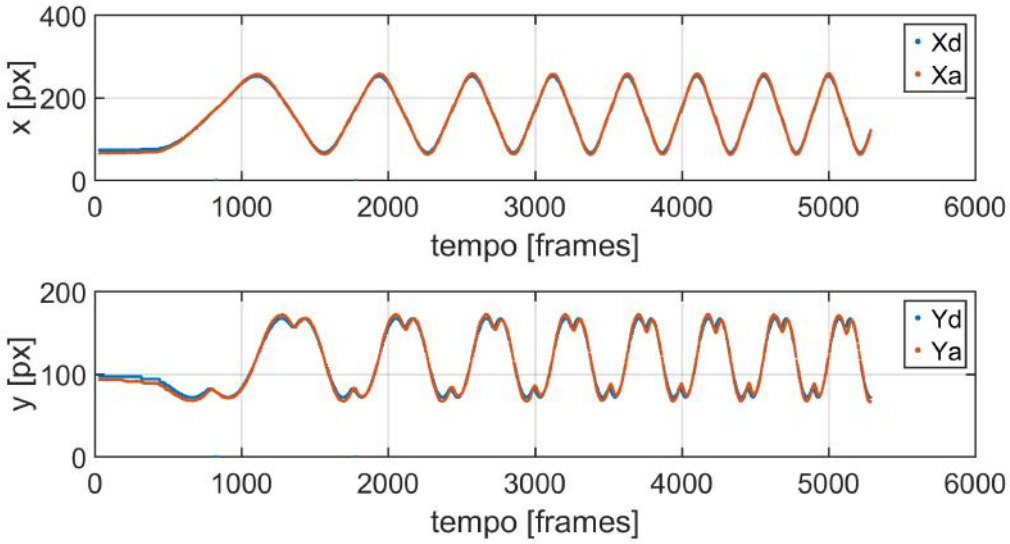
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 7.7: Resultados Resultado do experimento oval de Cassini

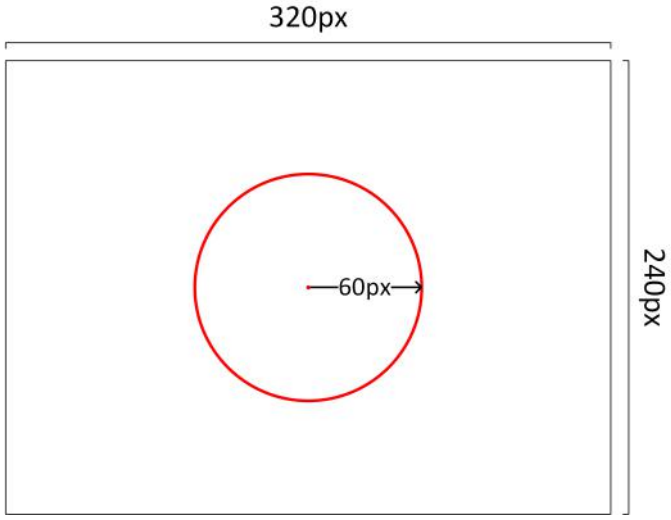


Figura 7.8: Circulo de raio 60 px centrado em (160,120)

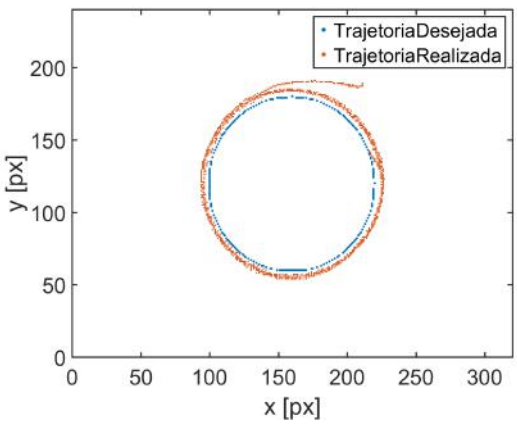
Variável	MSE
x	15,0521
y	24,7393
θ	5,3290
Vel	673,3185

(a) MSE principais variáveis

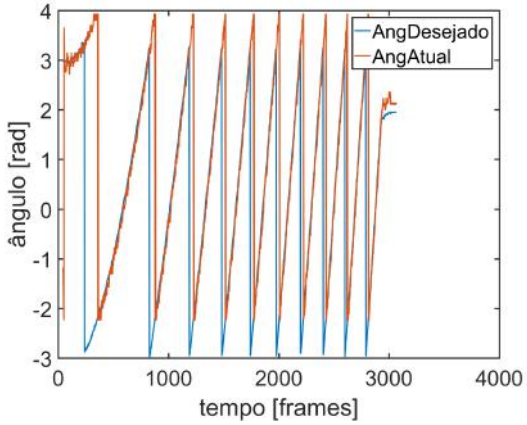
Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

(b) Parâmetros utilizados nos controladores

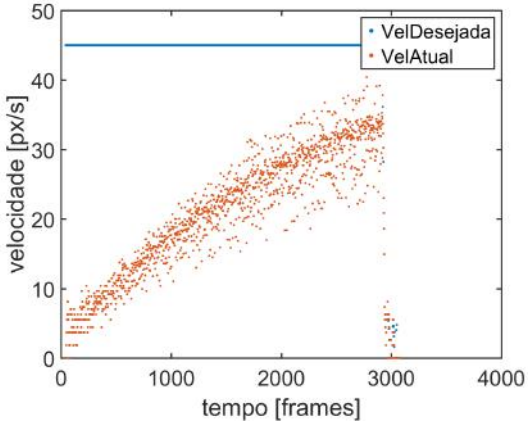
Figura 7.9: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores



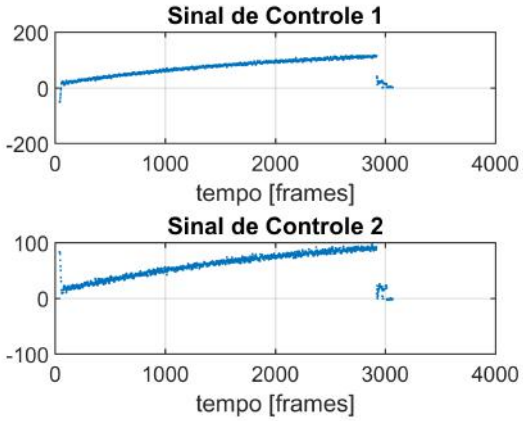
(a) Trajetória



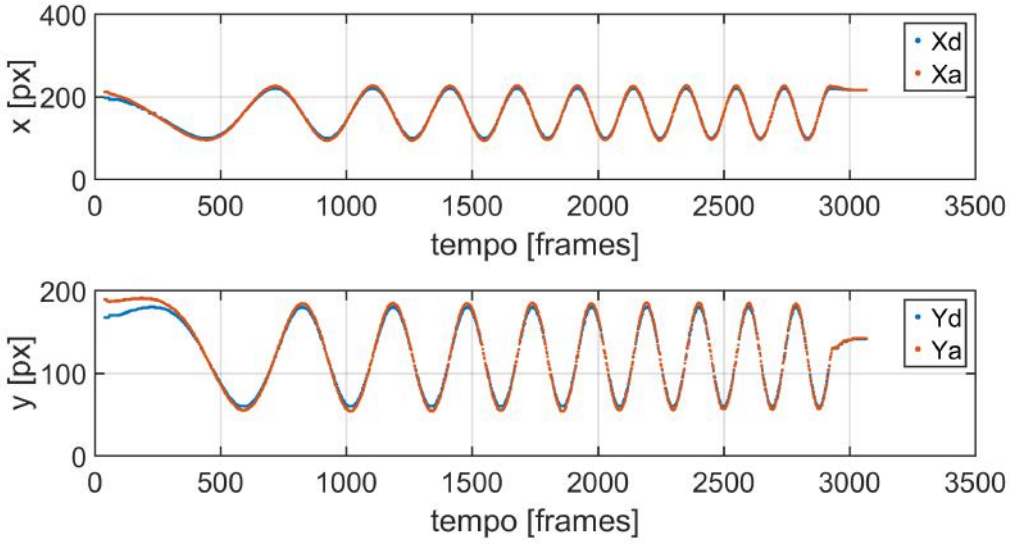
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 7.10: Resultados Resultados do experimento círculo de raio 60px centrado em (160,120) com inserção de obstáculo

houvesse um resultado indesejado nos sinais de controle indicando uma resposta bastante boa do carrinho a essa mudança repentina da referência.

7.4 Robôs com velocidades diferentes na mesma trajetória

A partir desse experimento, 2 carros serão utilizados para simular as situações que deseja-se testar.

Resultados

Variável	MSE
x	4,6988
y	4,2637
θ	5,0294
Vel	1151,4255

(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

(b) Parâmetros utilizados nos controladores

Figura 7.11: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores do Robô 1

Variável	MSE
x	182,7305
y	23,4778
θ	4,7964
Vel	269,4979

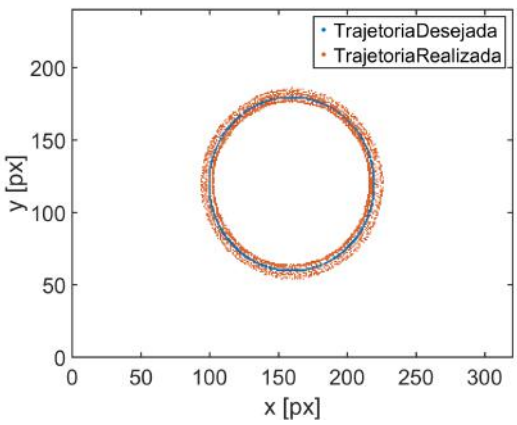
(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

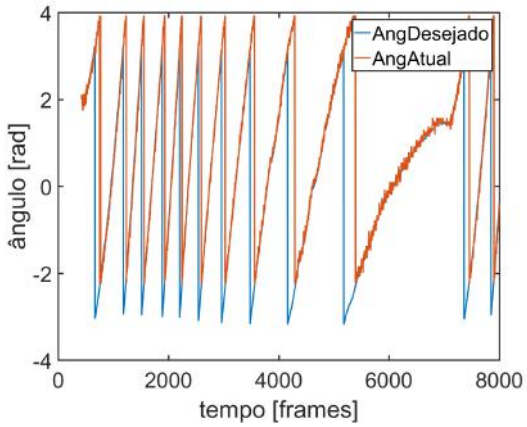
(b) Parâmetros utilizados nos controladores

Figura 7.12: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores do Robô 2

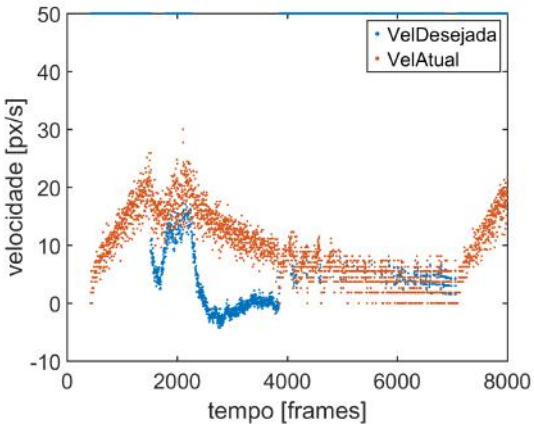
Para esse experimento fez-se com que dois carros andassem na mesma trajetória no mesmo sentido, porém com *setpoints* de velocidade diferentes. Isso foi feito com a finalidade de observar o comportamento do carro com velocidade maior passar a seguir o carro mais lento sem que haja colisão. Para isso, além de deixar os carros com referências de



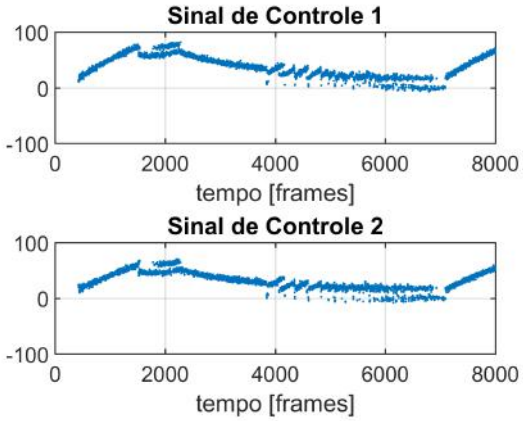
(a) Trajetória



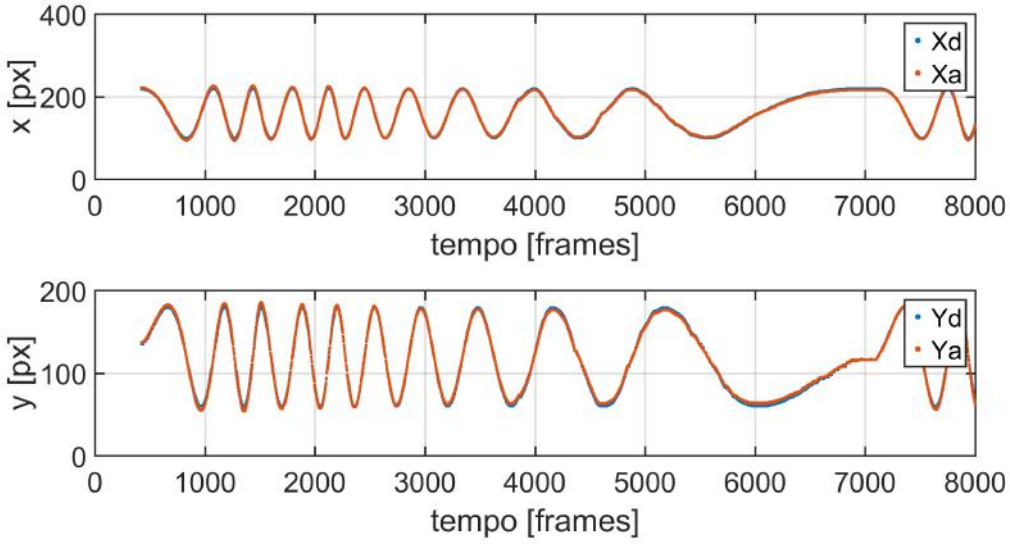
(b) Ângulos



(c) Velocidade

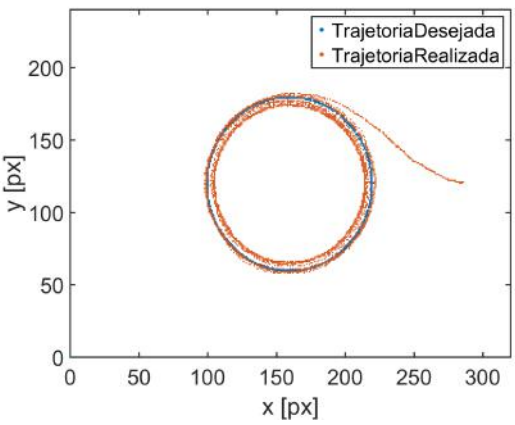


(d) Sinais de Controle

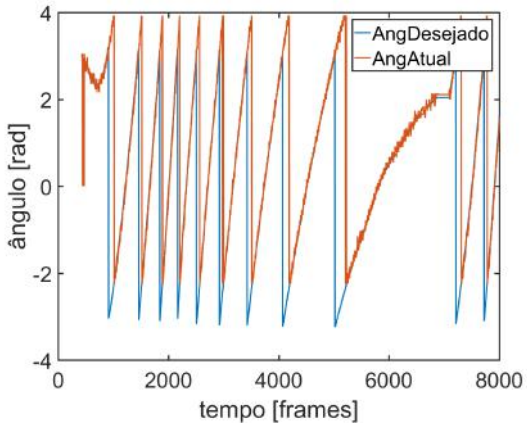


(e) X e Y

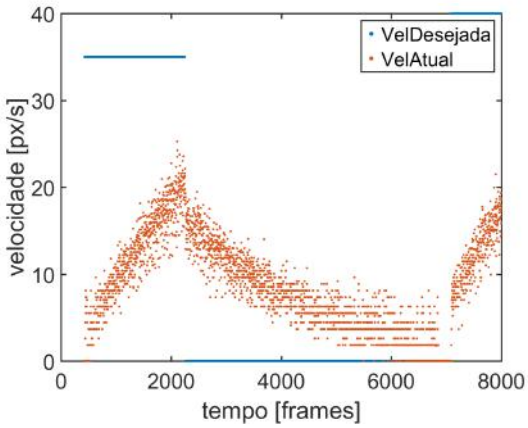
Figura 7.13: Resultados Resultado do experimento em que dois robôs com velocidades diferentes se locomovem na mesma trajetória - robô 1 (mais rápido)



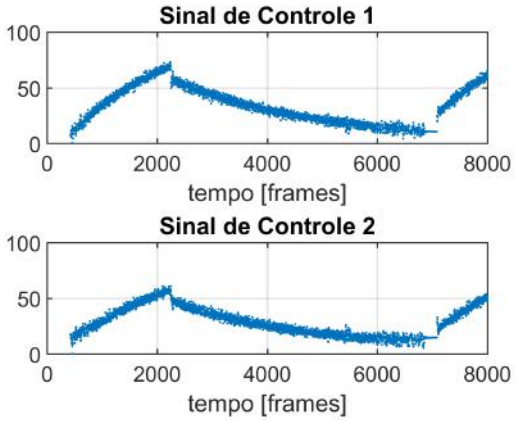
(a) Trajetória



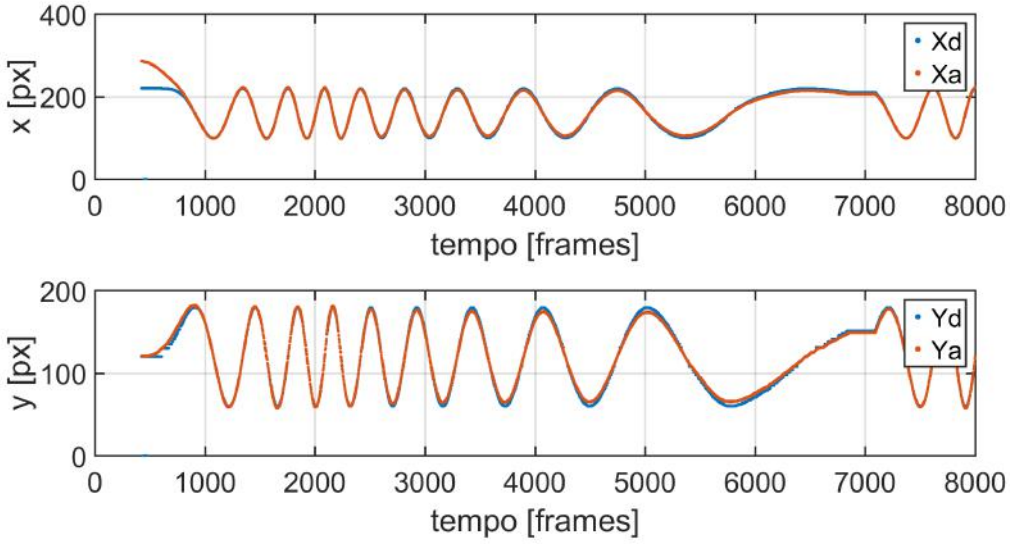
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 7.14: Resultados Resultado do experimento em que dois robôs com velocidades diferentes se locomovem na mesma trajetória - robô 2

velocidades diferentes, também fez-se com que durante o teste um dos carros passa a ter seu *setpoint* de velocidade igual a zero.

Feitas as observações a respeito do experimento, pode-se notar com o mesmo, que até o instante 2,3s ambos os carros têm o comportamento esperado, isto é, o comportamento observado nos testes anteriores, em que a trajetória desejada é seguida mantendo os valores de controle sempre bem comportados.

Já a partir do instante 2,3s o resultado experimental se torna mais interessante, uma vez que nesse instante como pode ser observado no gráfico de velocidade do robô 2 trocou-se a referência de velocidade desse carro de 35pixels/s para 0pixels/s. Dessa forma, notou-se que o carro começa a desacelerar, reduzindo os sinais de controle. No entanto, ele não deixa de seguir a trajetória desejada. O robô 1 por sua vez, ao notar um carro em movimento mais lento em sua frente, tem automaticamente seu *setpoint* reduzido para se ajustar a velocidade do veículo a frente sem que haja colisão entre eles (observar gráfico de velocidade do robô 1) e sempre mantendo a distância segura de frenagem. Da mesma maneira, os sinais de controle desse carro se reduzem porém a trajetória seguida pelo carrinho se mantém inalterada como era esperado.

Mais para o final do experimento (instante 7,1s), retornou-se o valor de referência de velocidade do segundo veículo para 35pixels/s e a partir desse instante notou-se que ambos os carros começam a acelerar juntos seguindo a trajetória a eles imposta. Vale observar também que durante esse processo o carro de trás (robô 1) sempre mantém a distancia segura em relação ao robô 2.

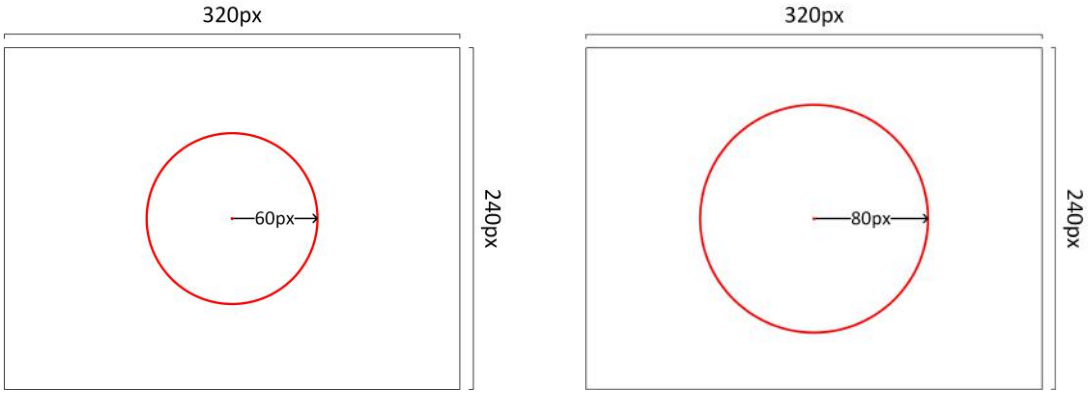
Assim como mencionado nos experimentos anteriores, a medida de velocidade é bastante dispersa devido à baixa taxa de amostragem obtida durante os testes, contudo ao analisar os valores médio de velocidade nota-se que ela realmente tende ao valor desejado.

7.5 Robôs com velocidades diferentes na mesma trajetória - Ultrapassagem

Como o título nos induz a pensar, nesse experimento tem-se dois carros numa mesma trajetória, mas com velocidade diferentes a fim de observar um comportamento de ultrapassagem.

Resultados

Nota-se aqui que o robô 1 possui os gráficos de X e Y razoavelmente diferentes dos gráficos até agora observados, onde em grande parte os valores dessas coordenadas eram senoides. Além disso, observa-se também que a trajetória desejada assim como a trajetória seguida ficam oscilando entre dois círculos concêntricos. Isso é devido ao fato de



(a) Trajetória principal para ambos os robôs (b) Trajetória auxiliar para o Robô 1

Figura 7.15: Trajetórias utilizadas no experimento

Variável	MSE
x	21,3057
y	16,6540
θ	5,5514
Vel	1115,5910

(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

(b) Parâmetros utilizados nos controladores

Figura 7.16: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores do Robô 1

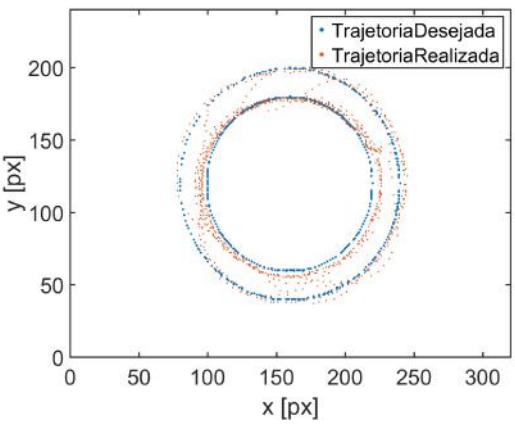
Variável	MSE
x	283,4667
y	63,1632
θ	5,1411
Vel	414,8240

(a) MSE principais variáveis

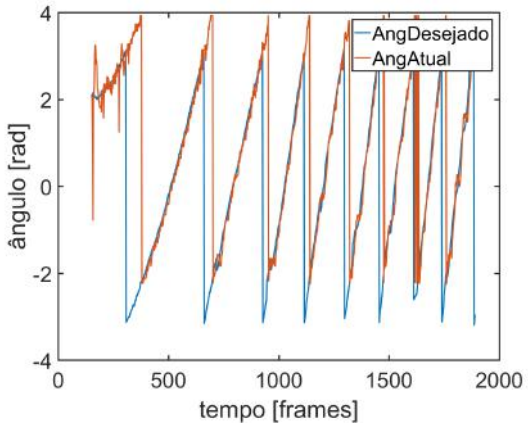
Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

(b) Parâmetros utilizados nos controladores

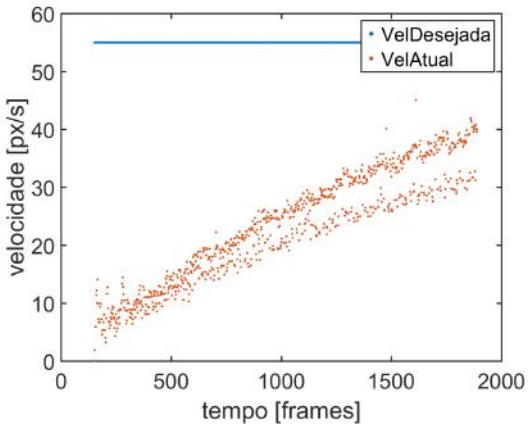
Figura 7.17: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores do Robô 2



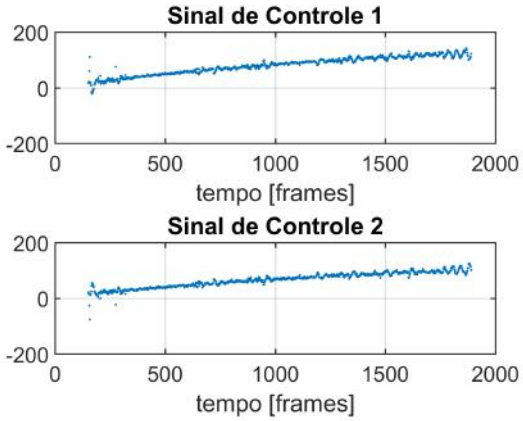
(a) Trajetória



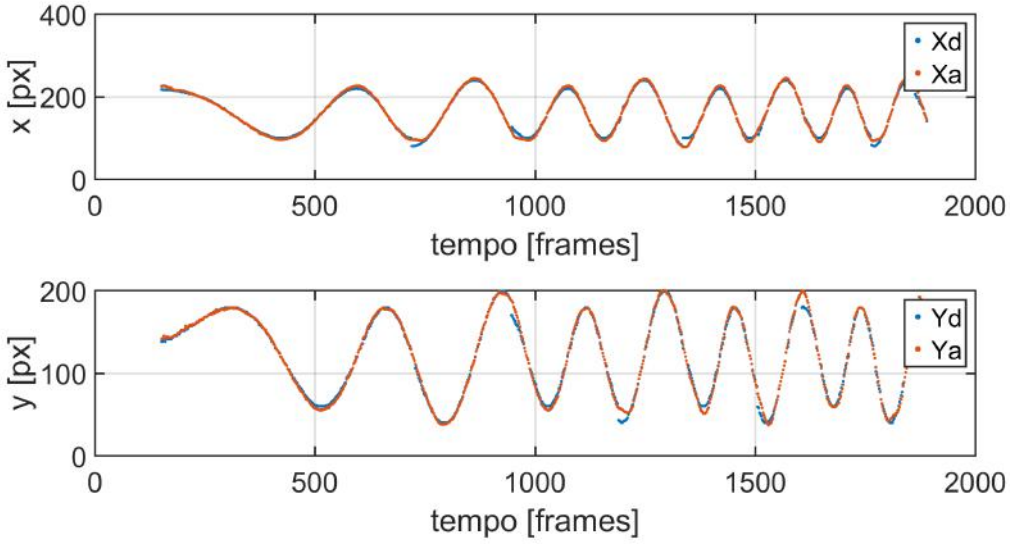
(b) Ângulos



(c) Velocidade

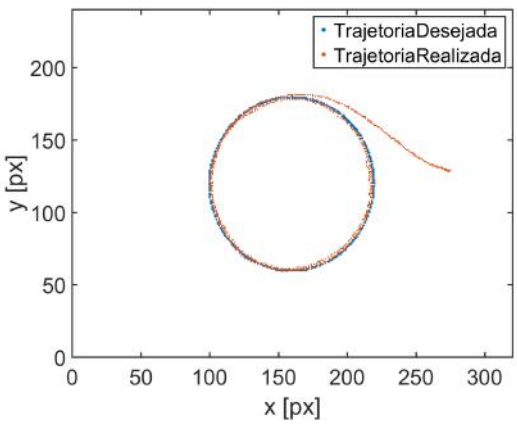


(d) Sinais de Controle

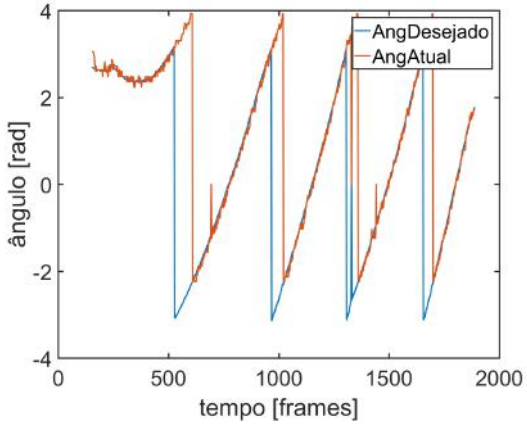


(e) X e Y

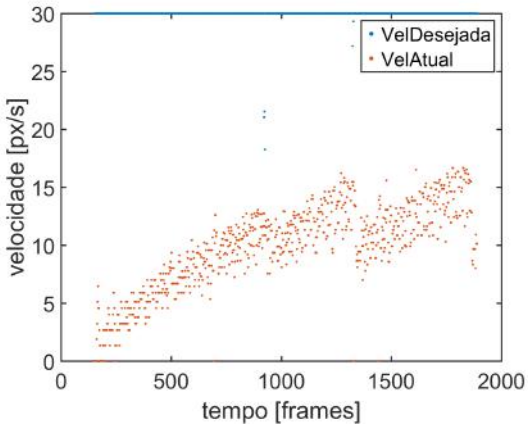
Figura 7.18: Resultados Resultados robô 1 (mais rápido) para o experimento em que dois robôs com velocidades diferentes se locomovem na mesma trajetória - Ultrapassagem



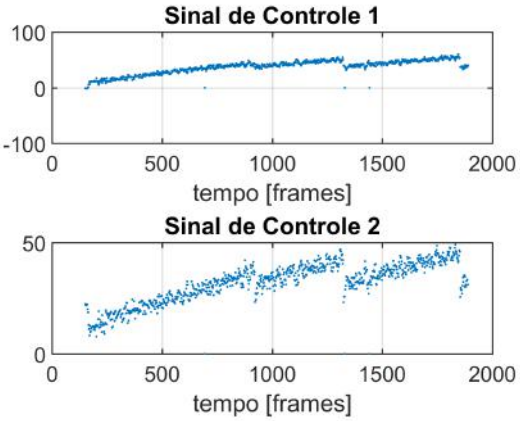
(a) Trajetória



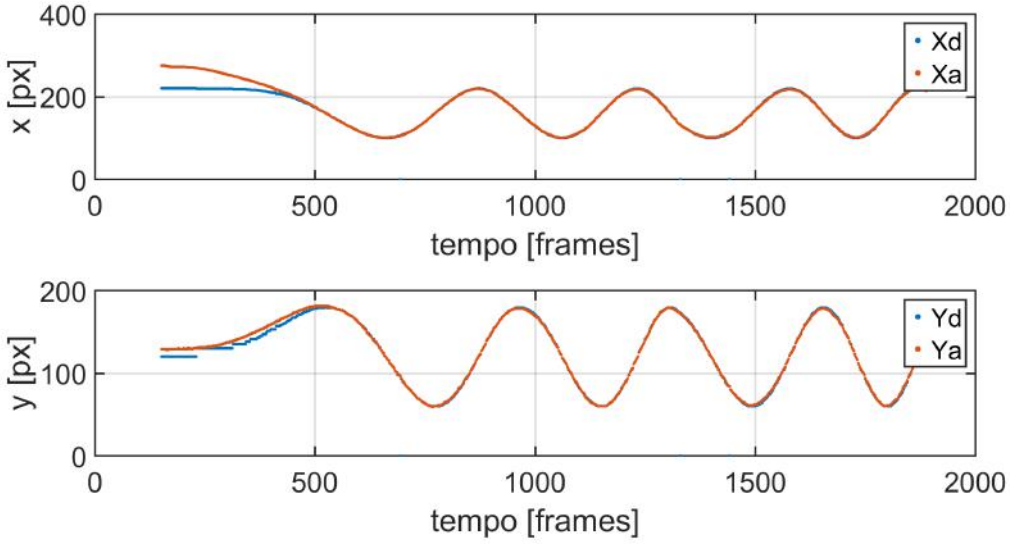
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 7.19: Resultados Resultados robô 2 para o experimento em que dois robôs com velocidades diferentes se locomovem na mesma trajetória - Ultrapassagem

nesse experimento simular uma situação de ultrapassagem. Dessa forma, nos momentos em que a inteligencia do carro decide que ele pode fazer uma ultrapassagem por estar mais rápido que o carro à frente e não ter nenhum outro carro numa segunda opção de trajetória, a trajetória desejada do mesmo é trocada e como observado nos gráficos do ângulo, X e Y o carro tem uma resposta bastante aceitável, fazendo um ajuste suave em seu ângulo para atingir os novos valores de X e Y que correspondem à nova trajetória. Nota-se também, que durante esse período de mudança os valores dos sinais de controle se mantêm bem comportados apesar de passarem a oscilar e que o cruise control não foi afetado.

Para terminar a análise do carro 1, nota-se que um tempo após ultrapassar o veículo mais lento o carro 1 tem sua trajetória desejada retornada para a trajetória inicial e com isso todo o comportamento visto durante o intervalo de tempo da primeira mudança de referência se repete a fim de que o carro retorne à trajetória inicial.

Observando agora o segundo robô utilizado durante o experimento vê-se que esse possui um comportamento bem parecido com aquele visto no 1 experimento. Todavia, nota-se que as velocidades desejada e realizada desse veículo possuem algumas oscilações, no caso quedas. Isso ocorre uma vez que no momento de reentrada do carro 1 na trajetória inicialmente seguida por ambos, esse carro acaba entrando numa distância menor do que a distância segura considerada pelo carro 2, fazendo com que esse tenha leves freadas nesses momentos de retorno do carro 1 a trajetória definida pelo círculo de menor raio.

A fim de ilustarar o comportamento do sistema durante uma ultrapassagem, as figuras abaixo mostram o comportamento do *software* durante este procedimento.

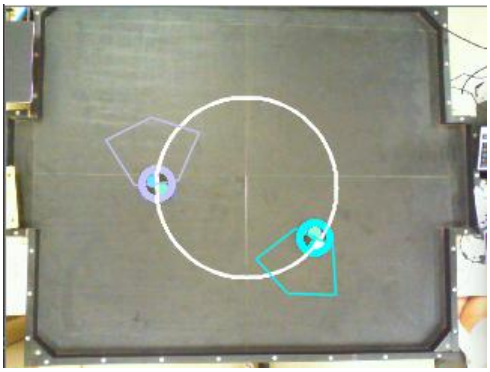


Figura 7.20: Interface mostrando informações da trajetória e do robô antes do inicio da ultrapassagem

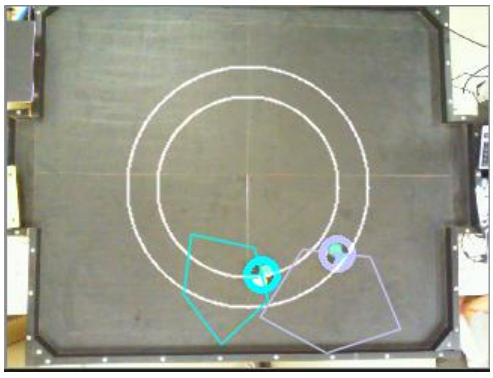


Figura 7.21: Interface mostrando informações da trajetória e do robô durante o inicio da ultrapassagem

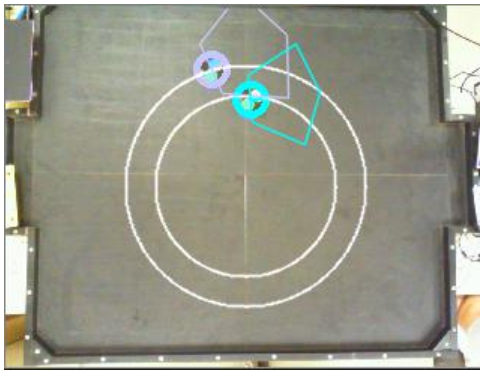


Figura 7.22: Interface mostrando informações da trajetória e do robô durante a ultrapassagem

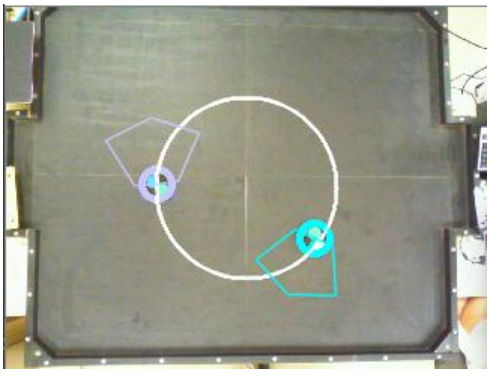


Figura 7.23: Interface mostrando informações da trajetória e do robô após a ultrapassagem

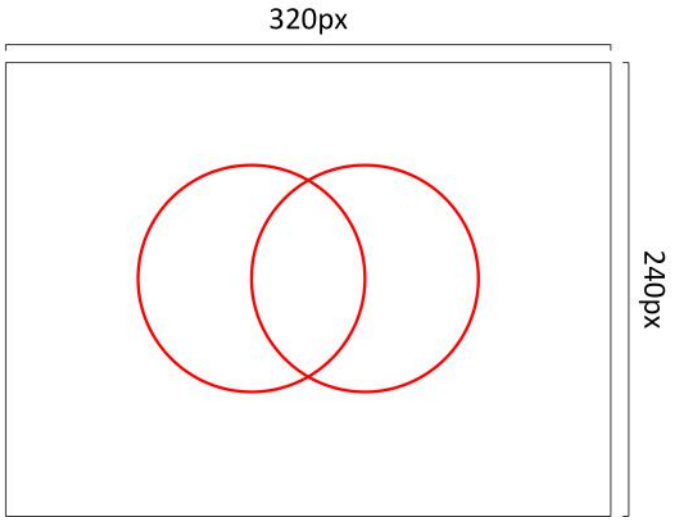


Figura 7.24: Trajetórias utilizadas no experimento

7.6 Robôs em trajetórias diferentes que se interceptam

Para essa experiência usaram-se 2 carros, cada um com um *setpoint* de velocidade e cada um seguindo uma trajetória.

Resultado

Variável	MSE
x	2,0053
y	2,4840
θ	5,5319
Vel	1208,5879

(a) MSE principais variáveis

Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

(b) Parâmetros utilizados nos controladores

Figura 7.25: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores do Robô 1

Observou-se que o carro 2 possui um comportamento bastante parecido com aquele observado no primeiro experimento.

Já o carro 1 tem um comportamento mais interessante, em que durante alguns momentos ocorre uma queda nas velocidade desejada desse carro e por consequencia uma redução não só da velocidade atual do carro como também uns picos de queda nos sinais de controle para que assim a velocidade atual do carrinho pudesse se manter se não junto, ao menos o mais próximo possível desses novos valores de referência de velocidades.

Variável	MSE
x	2,2574
y	0,5883
θ	4,5245
Vel	453,3740

(a) MSE principais variáveis

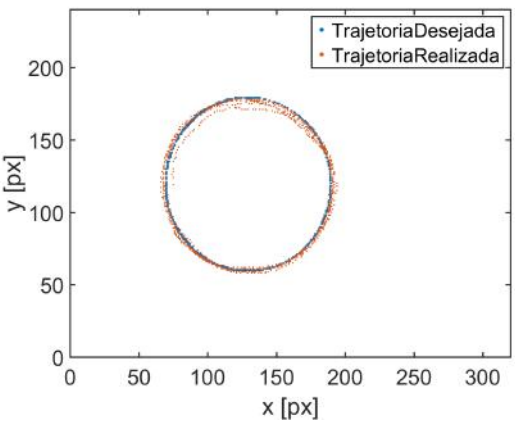
Parâmetro	Angular	Velocidade
P	30	6
I	0,01	0,05
D	1	1

(b) Parâmetros utilizados nos controladores

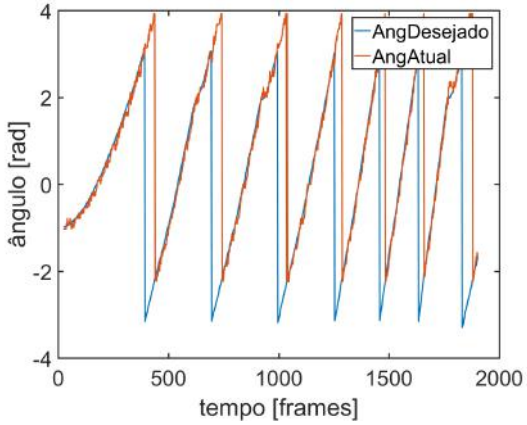
Figura 7.26: MSE das variáveis medidas no experimento e parâmetros utilizados pelos controladores do Robô 2

Esse processo todo que acabou de ser mencionado é devido ao fato de que as trajetórias desses carrinho se cruzam em 2 pontos como pode-se observar na figura 7.24, assim nos momentos em que os carros irão passar nesses cruzamentos ao mesmo tempo, um deles (no caso o carro mais a esquerda, dado que a direita possui a preferencia) precisa reduzir sua velocidade a fim de evitar a colisão.

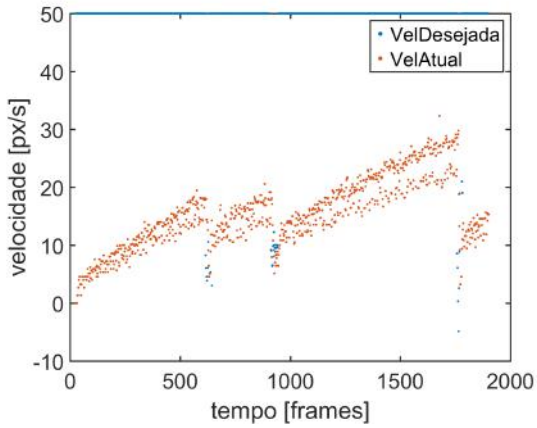
Tem-se por fim , que para ambos os carrinhos a ação das malhas de controles neles implementadas possui um resultado bastante satisfatório, mantendo-os sempre nas trajetórias desejadas (com erros pequenos que podem ser causados pela incerteza na medição da posição do carrinho por intermédio da imagem capturada), com velocidades próximas das desejadas, ou tendendo as mesmas, sem que os sinais de controle possuam comportamentos explosivos ou extremamente oscilatórios.



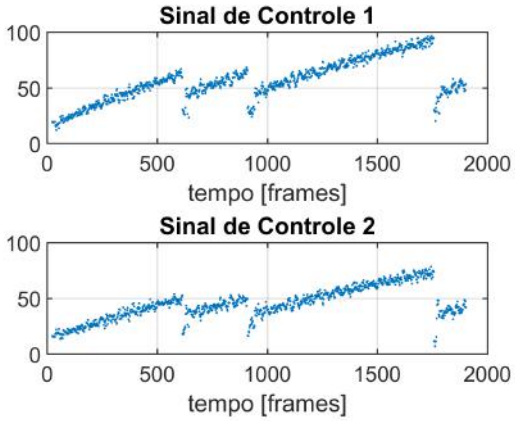
(a) Trajetória



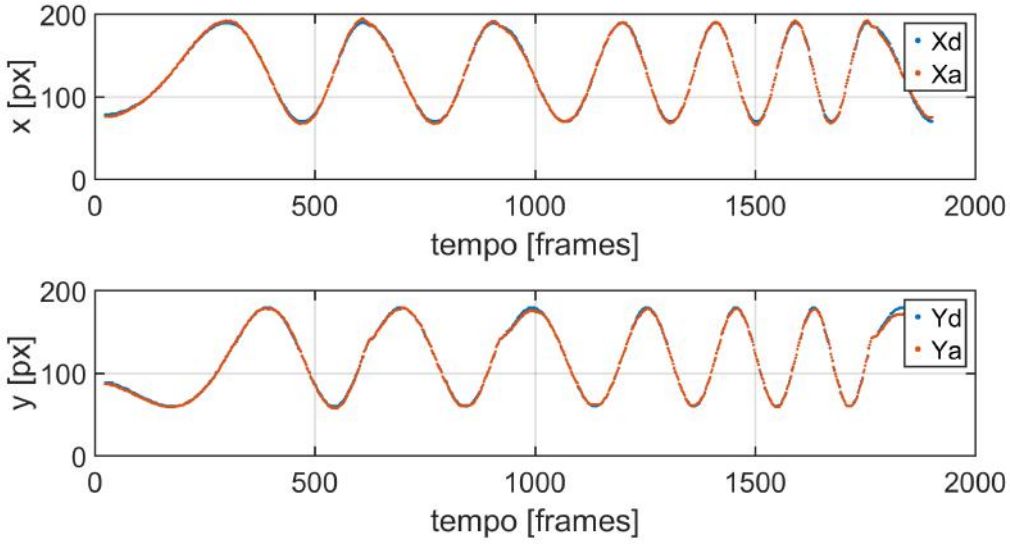
(b) Ângulos



(c) Velocidade

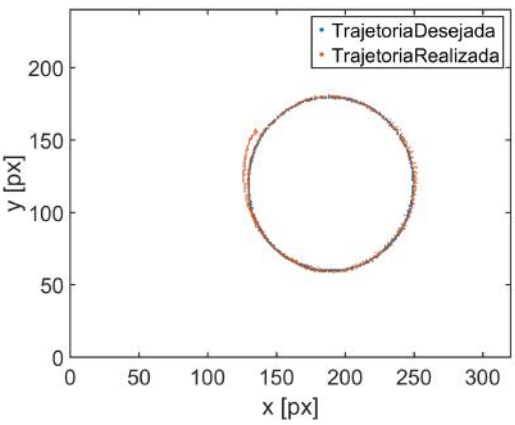


(d) Sinais de Controle

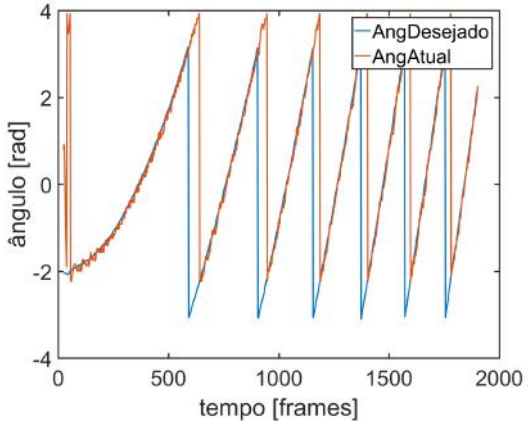


(e) X e Y

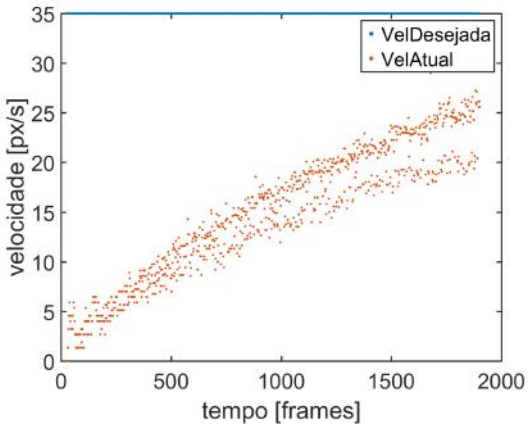
Figura 7.27: Resultados Resultados do experimento em que dois robôs se locomovem em trajetórias diferentes que se interceptam - robô 1



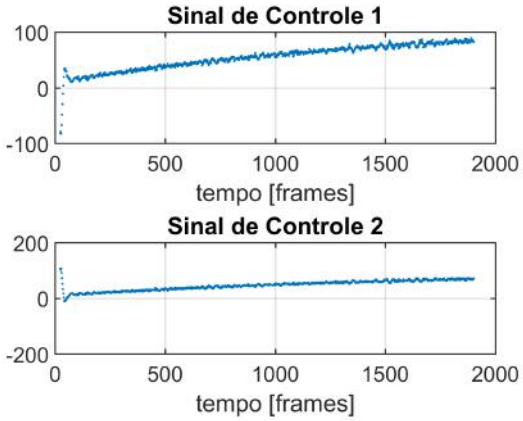
(a) Trajetória



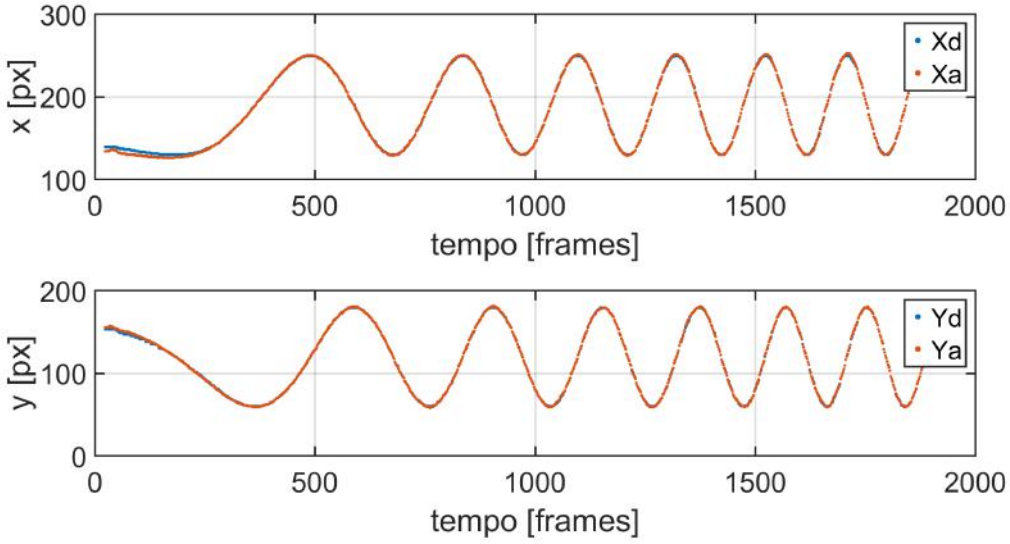
(b) Ângulos



(c) Velocidade



(d) Sinais de Controle



(e) X e Y

Figura 7.28: Resultados Resultados do experimento em que dois robôs se locomovem em trajetórias diferentes que se interceptam - robô 1

Capítulo 8

Conclusões, Contribuições e Trabalhos Futuros

8.1 Conclusões

Este trabalho apresentou o desenvolvimento de uma bancada experimental para avaliar esquemas de controle para robôs terrestres autônomos. O objetivo da bancada é permitir avaliar esquemas de controle que possam ser aplicados no controle de um carro que se movimente de forma autônoma no trânsito de uma cidade. Na primeira etapa deste projeto, foram especificados todo o *hardware* da bancada, a modelagem matemática e o desenvolvimento do simulador dos robôs, além do desenvolvimento e codificação do *firmware* de comunicação dos robôs. Em seguida, o *software* que engloba diversas funcionalidades, em particular, os algoritmos para obtenção da posição e orientação de cada robô, baseados em visão computacional, foi desenvolvido.

A estratégia de controle foi desenvolvida visando permitir que o veículo evite colisões, permaneça a uma distância segura do veículo a sua frente (*Cruise Control*) e permita que o veículo realize ultrapassagens. O esquema de controle de cada veículo considerou apenas as informações do veículo mais próximo afim de emular uma situação real. Um simulador foi desenvolvido e experimentas foram conduzidos com dois robôs ilustrando a eficácia do esquema de controle utilizado.

Ao longo do trabalho pode-se concluir primeiramente que o simulador desenvolvido, por mais que idealizado, demonstra suficientemente bem que a técnica de controle considerada assegura resultados satisfatórios em malha fechada. Além disso, resultados experimentais com um robô real movendo-se ao longo de uma trajetória foram obtidos e mostraram-se aceitáveis.

Durante os estudos realizados para o controle angular foi observada uma singularidade no método utilizado, entretanto a mesma pôde ser contornada por um ajuste bastante simples na medição dos ângulos.

Observou-se também que o *software* desenvolvido foi capaz de tornar mais de um carro inteligente o suficiente para evitar colisões com obstáculos parados além de, dependendo das situações, ser capaz de ultrapassar algum obstáculo ou até mesmo um outro carro em movimento mantendo sempre uma distância de segurança para eventuais necessidades de freadas bruscas.

Ainda sobre o controle do carro, observou-se que a técnica de *cruise control* proposta no trabalho é possível de ser implementada com o auxílio de alguns sensores de proximidades, fazendo com que os carros autônomos sejam capazes de não só manterem a velocidade definida como *setpoint* mas também caso haja um outro carro se movendo mais lentamente na frente do veículo este não colida e passe a seguir a velocidade do carro à sua frente mantendo sempre a distância segura

8.2 Contribuições

- Desenvolvimento de um simulador em MatLab/Simulink que possibilita o estudo do comportamento de robôs móveis terrestres e que pode ser utilizado para o estudo e desenvolvimento de diversas técnicas de controle;
- Desenvolvimento de um *software* amigável para o estudo e experimentação com robôs moveis terrestres, além de possibilitar a fácil atualização, manutenção e expansão do mesmo para que seja possível sua utilização em outros projetos e outros tipos de experimentos;
- Desenvolvimento de uma ferramenta de análise dos dados gerados pelo *software*, gerando gráficos e indicadores quantitativos dos experimentos realizados;
- Proposta de um esquema de controle que evita colisão e permite ultrapassagens.

8.3 Trabalhos Futuros

Para continuar o que foi proposto e realizado no âmbito deste trabalho alguns trabalhos futuros poderiam ser realizados. Conforme viu-se na seção 7, um dos principais problemas encontrados foi a taxa de amostragem. Utilizando um pouco menos de 30 quadros por segundo fica difícil o controle preciso do posicionamento e da velocidade dos robôs. Um dos trabalhos futuros seria então tentar utilizar uma câmera que fosse capaz de oferecer uma taxa de quadros por segundo maior (alguns modelos comerciais chegam a 60FPS e outros a 120FPS) e otimizar ainda mais os algoritmos de forma a obter uma taxa de amostragem que possibilite o controle dos robôs com maior precisão e em uma maior velocidade. Uma das otimizações que pode ser feita é a utilização de computação paralela através da placa de vídeo (GPU) do computador.

As simulações foram realizadas utilizando um *step* bem pequeno (0,003s). Seria interessante refazer as simulações utilizando um *step* condizente com o que pode ser alcançado na realidade em ter de *hardware* e *software* de forma que seja possível resolver em ambiente simulado alguns problemas da discretização do sistema e do controle do mesmo.

O simulador foi construído de forma que a entrada no modelo sejam as tensões que são enviadas para cada motor. Entretanto, na hora de implementação, o sistema teve que ser alterado para que a saída dos controladores montasse o sinal entre -255 e 255 que é enviado via XBee para cada motor. O simulador poderia ser alterado para trabalhar também com esta faixa de valores de forma que os ganhos ótimos encontrados no simulador pudessem ser mais facilmente utilizados na implementação do sistema.

O simulador construído no escopo deste projeto se destinou a simulação do controle angular e o controle de velocidade. Um possível trabalho futuro seria a construção da camada de estratégia de forma que fosse possível simular os casos implementados (frenagem de emergência, ultrapassagem, carro mais lento à frente) com mais robôs e com mais situações diversas acontecendo ao mesmo tempo.

O programa desenvolvido poderia possibilitar que o usuário clicasse em diversos pontos numa imagem e o algoritmo geraria a trajetória com os pontos para que o robô seguisse esta trajetória.

A verificação do melhor ponto para uma determinada trajetória poderia ser feita de forma otimizada, sem ter que efetivamente medir a distância entre a posição do robô e todos os pontos da trajetória.

Outra possibilidade de trabalho futuro seria a construção de uma pista mais complexa do que a utilizada neste trabalho de forma a possibilitar a experimentação de uma subida (para verificar se o controle de velocidade mantém a velocidade inclusive quando o carro está em aclive ou declive) e um ou mais cruzamentos de trajetórias para possibilitar também a experimentação de cenários mais parecidos com a realidade do cotidiano do trânsito em nossas cidades.

Capítulo 9

Apêndice

9.1 Script de Log (*Matlab*)

```
1
2
3  %%
4  % Condicoes encontradas na geracao do LOG.txt:
5  % 1 - Ao menos a 1 linha da ultima colunas precisa estar preenchida
6  % 2 - O Xd nao pode ser vazio, colocar pra quando nao encontrar o -1 mesmo
7
8  clear
9  close all
10 clc
11
12 % TimeStamp | Ticks | FrameNum | FPS | LogaDados | Robo | achou | trajetoria | Xd | Yd |
       Xa | Ya | DistMelhorPonto | velAtual | velDesejada | erroVel | SaidaControleLinear
       | angAtual | angDesejado | erroAng | SaidaControleAngular | SinalDeControle1 |
       SinalDeControle2 |
13 filename = '20170212_112221.txt';
14 delimiterIn = '|';
15 headerlinesIn = 1;
16 dados = importdata(filename,delimiterIn,headerlinesIn);
17
18 headers = dados.textdata(1, :);
19
20
21 %% TRATAMENTO PARA O TAMANHO DAS COLUNAS (as q vieram como string)
22
23 stringHeaders = {'TimeStamp', 'Robo', 'trajetoria'}; % Headers do tipo String
24 % trajetoria aqui e o ultimo elemento string do log
25 for i=1:find(strcmp(dados.textdata(1, :), 'trajetoria'))
26     maxSize = size(cell2mat(dados.textdata(2, i)), 2);      % pega o tamanho do primeiro
           elemento da coluna
27     % Loop para coletar o tamanho do maior elemento da coluna
28     for elemento=3:length(dados.textdata(:, 4))
29         if maxSize < size(cell2mat(dados.textdata(elemento, i)), 2)
30             maxSize = size(cell2mat(dados.textdata(elemento, i)), 2);
31         end
32     end
33
```

```
34      if ismember(strtrim(headers(1, i)), stringHeaders)
35          for j=1:length(dados.textdata(1:end, i)) % varre todas as linhas
36              while(maxSize > size(cell2mat(dados.textdata(j, i)), 2))
37                  % Pra cada linha adiciona n elementos, em que n eh a
38                  % diferenca entre o tamanho max da coluna e o tamanho do
39                  % individo em analise
40                  dados.textdata(j, i) = strcat(dados.textdata(j, i), '-');
41              end
42          end
43      else
44          for j=1:length(dados.textdata(1:end, i)) % varre todas as linhas
45              while(maxSize > size(cell2mat(dados.textdata(j, i)), 2))
46                  % Pra cada linha adiciona n elementos, em que n eh a
47                  % diferenca entre o tamanho max da coluna e o tamanho do
48                  % individo em analise
49                  %dados.textdata(j, i) = strcat(dados.textdata(j, i), '0');
50                  dados.textdata(j, i) = strcat('0', dados.textdata(j, i));
51              end
52          end
53      end
54  end
55
56  strCounter = 1;
57  numCounter = 1;
58
59  % Varre os headers para atribuir um array pra cada um
60  for idx = 1:size(headers, 2)-1
61          headers{2, idx} = []; % Inicializa com vazio
62          isString = false;
63
64          % Verifica se eh do tipo String e caso seja atribui o array de
65          % string nele
66          for strIdx = 1:length(stringHeaders)
67              if ismember(strtrim(headers(1, idx)), stringHeaders)
68                  isString = true;
69                  headers{2, idx} = dados.textdata(2:end,strCounter);
70                  strCounter = strCounter + 1;
71                  break;
72              end
73
74          end
75          if ~isString
76              if (idx <= 8)
77                  temp = cell2mat(dados.textdata(2:end, idx));
78                  headers(2, idx) = {str2num(temp)};
79                  strCounter = strCounter + 1;
80              else
81                  if numCounter <= size(dados.data, 2)
82                      headers{2, idx} = dados.data(:, numCounter);
83                  end
84                  numCounter = numCounter + 1;
85              end
86          end
87  end
88
89
90  %% PLOTS
91
92
```

```matlab
% filtrando os nomes dos carros
carros = {};
for i=1:size(headers{2, 6},1)
    if ~ismember(headers{2, 6}(i), carros)
        carros = [carros; headers{2, 6}(i)];
    end
end

containerCarros = {};
% ESTRUTURA DO CONTAINER
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%1 % Nome(1)                   | Nome(2)                   | ...
%2 % Indices(1)                | Indices(2)                | ...
%3 % Ticks(1)                  | Ticks(2)                  | ...
%4 % Frame(1)                  | Frame(2)                  | ...
%5 % FPS(1)                    | FPS(2)                    | ...
%6 % Achou(1)                  | Achou(2)                  | ...
%7 % Trajetoria(1)             | Trajetoria(2)             | ...
%8 % Xd(1)                     | Xd(2)                     | ...
%9 % Yd(1)                     | Yd(2)                     | ...
%10% Xa(1)                     | Xa(2)                     | ...
%11% Ya(1)                     | Ya(2)                     | ...
%12% DistMelhorPonto(1)        | DistMelhorPonto(2)        | ...
%13% velAtual(1)               | velAtual(2)               | ...
%14% velDerivadaSuja(1)        | velDerivadaSuja(2)        | ...
%15% velDesejada(1)            | velDesejada(2)            | ...
%16% erroVel(1)                | erroVel(2)                | ...
%17% SaidaControleLinear(1)    | SaidaControleLinear(2)    | ...
%18% angAtual(1)               | angAtual(2)               | ...
%19% angDesejado(1)            | angDesejado(2)            | ...
%20% erroAng(1)                | erroAng(2)                | ...
%21% SaidaControleAngular(1)   | SaidaControleAngular(2)   | ...
%22% SinaldeControle1(1)       | SinaldeControle1(2)       | ...
%23% SinaldeControle2(1)       | SinaldeControle2(2)       | ...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

indicesDesnecessarios = [5, 6, 24];
for i=1:size(carros, 1)
    containerCarros{1, i}  = carros(i);
    containerCarros{2, i}  = find(strcmp(headers{2, 6}, carros(i)));
    containerCarros{3, i}  = headers{2, 2}(containerCarros{2, i});   % ticks
    containerCarros{4, i}  = headers{2, 3}(containerCarros{2, i});   % frame
    containerCarros{5, i}  = headers{2, 4}(containerCarros{2, i});   % fps
    for j=7:size(headers,2)
        if (~isempty(headers{2, j}) && ~ismember(j, indicesDesnecessarios))
            containerCarros{j-1, i} = headers{2, j}(containerCarros{2, i});
        else
            containerCarros{j-1, i} = [];
        end
    end
end

% SEPARACAO POR CARROS

set(0,'defaultfigurecolor',[1 1 1]) % muda o fundo das figuras
indicesParaNaoPlotar = [1, 2, 3, 7];
for i=1:size(carros, 1) % para cada carro plotar n figuras
    tempo = containerCarros{4, i};

```

```
152     %Plot de (XdxYd) vs (XaxYa)
153     figure
154     plot(containerCarros{8, i}, containerCarros{9, i},'-r', containerCarros{10, i},
            containerCarros{11, i},'-b')
155     axis([0, 640, 0, 480])
156     legend('\fontsize{8}TrajetoriaDesejada','\fontsize{8}TrajetoriaRealizada');
157     title([ '\fontsize{12}TrajetoriaDesejada x TrajetoriaRealizada (' cell2mat(
            containerCarros{1, i}) ')'])
158
159     % Plot de (Xd, Xa), (Yd, Ya) e (Angd, Anga)
160     figure
161     subplot(3,1,1)
162         plot(tempo,containerCarros{8, i},'-r', tempo,containerCarros{10, i},'-b')
163         legend('\fontsize{8}Xd','\fontsize{8}Xa');
164         title(['\fontsize{12}Xd vs Xa(' cell2mat(containerCarros{1, i}) ')'])
165         grid on
166
167     subplot(3,1,2)
168         plot(tempo,containerCarros{9, i},'-r', tempo,containerCarros{11, i},'-b')
169         legend('\fontsize{8}Yd','\fontsize{8}Ya');
170         title(['\fontsize{12}Yd vs Ya(' cell2mat(containerCarros{1, i}) ')'])
171         grid on
172
173     subplot(3,1,3)
174         plot(tempo,containerCarros{19, i},'-r', tempo,containerCarros{18, i},'-b')
175         legend('\fontsize{8}AngDesejado','\fontsize{8}AngAtual');
176         title(['\fontsize{12}AngDesejado vs AngAtual (' cell2mat(containerCarros{1, i})
                ')'])
177         grid on
178
179     % VelDesejada vs VelAtual
180     figure
181     %plot(tempo,containerCarros{15, i},'-r', tempo,containerCarros{14, i},'-b')
182     hold on
183     scatter(tempo,containerCarros{15, i});
184     scatter(tempo,containerCarros{14, i});
185         legend('\fontsize{8}VelDesejada','\fontsize{8}VelAtual(SUJA)');
186     title([ '\fontsize{12}VelDesejada x VelAtual (' cell2mat(containerCarros{1, i}) ')'
            ])
187
188     % AngDesejado vs AngAtual
189     figure
190     plot(tempo,containerCarros{19, i},'-r', tempo,containerCarros{18, i},'-b')
191     legend('\fontsize{8}AngDesejado','\fontsize{8}AngAtual');
192     title([ '\fontsize{12}AngDesejado x AngAtual (' cell2mat(containerCarros{1, i}) ')'
            ])
193
194     % Plot de (V1xTempo), (V2xTempo)
195     if (~isempty(containerCarros{22, i}) && ~isempty(containerCarros{23, i}))
196         figure
197         subplot(2,1,1)
198             plot(tempo,containerCarros{22, i})
199             title(['\fontsize{12}Sinal de Tensao 1 (' cell2mat(containerCarros{1, i}) ')
                    '])
200             grid on
201
202         subplot(2,1,2)
203             plot(tempo,containerCarros{23, i})
```

```
204             title(['\fontsize{12}Sinal de Tensao 2 (' cell2mat(containerCarros{1, i}) ')
                    '])
205             grid on
206     end
```

Código 9.1: Implementação do script de geração de gráficos para análise dos resultados dos experimentos

9.2 Manual do Usuário

Explicada a implementação será apresentada agora a interface gráfica do programa desenvolvido. Para a construção dessa interface utilizou-se a biblioteca do QT mencionada anteriormente neste artigo.

9.2.0.1 Tela Principal

Figura 9.1: Tela Principal

Na figura 9.1 visualizada acima tem-se a tela principal do programa, responsável por rodar o *loop* principal do programa e dar acesso as demais telas existentes. Ao iniciar o programa ela é aberta automaticamente e encontram-se na mesma 4(quatro) *checkboxes* com seus respectivos identificadores, 2(dois) botões, 1(uma) caixa de texto e 1(uma) barra de ferramentas. O botão "Iniciar" faz com que o *loop* principal do programa seja executado, ou seja, identificar cada objeto, traçar a estratégia de cada um e enviar suas respectivas mensagens. Já o botão "Pausar", como o nome sugere, pausa a execução do *loop* principal podendo ser reiniciado de onde parou clicando novamente no botão. A barra de ferramentas com o título "Configurações" ao ser selecionada mostra 3 opções:

- Calibrar Filtros de Cor → Abre a tela de Calibração do Filtro de Cores;
- Calibrar Controle dos robôs → Abre a tela de Ajuste de PID;
- Cadastro Robôs → Abre a tela de Registro de novos objetos;

A utilização da aba de ferramentas mencionada pode ser visualizada na figura 9.2:

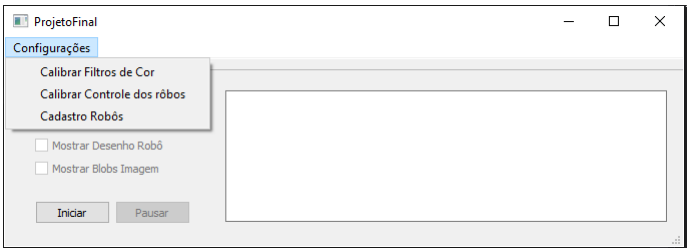


Figura 9.2: Opções para acessar outras telas na barra de ferramentas.

Por fim, os 4(quatro) *checkboxes* existentes nessa tela têm as seguintes características e funções:

1. Identificador → Mostrar a imagem da Câmera:
   - Característica → Inicializa desmarcado;
   - Característica → Inicialmente habilitado;
   - Função → Mostrar a imagem capturada pelo programa numa nova tela de tamanho 320x240 (seção 9.2.0.2);
2. Identificador → Mostrar Trajetória Inicial;
   - Característica → Inicializa desmarcado;
   - Característica → Inicialmente desabilitado. Este será habilitado caso o usuário marque o *checkbox* com o identificador "Mostrar a imagem da Câmera".
   - Função → Mostrar na imagem aberta a(s) trajetória(s) do(s) carrinho(s);
3. Identificador → Mostrar Desenho Robô;
   - Característica → Inicializa desmarcado;
   - Característica → Inicialmente desabilitado. Este será habilitado caso o usuário marque o *checkbox* com o identificador "Mostrar a imagem da Câmera".
   - Função → Mostrar na imagem aberta a(s) marcação(ões) de identificação do(s) carrinho(s), essas marcações são feitas com um círculo ao redor do objeto identificado e com uma linha que vai do centro do objeto até o círculo feito, indicando assim a direção de movimentação do veículo como visto na figura 9.3;
4. Identificador → Mostrar *Blobs* Imagem;
   - Característica → Inicializa desmarcado;
   - Característica → Inicialmente desabilitado. Este será habilitado caso o usuário marque o *checkbox* com o identificador "Mostrar a imagem da Câmera".

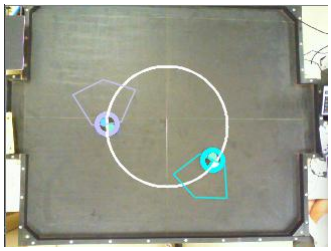


Figura 9.3: Marcações de Identificação do(s) robô(s)

– Função → Mostrar na imagem aberta o(s) *blob(s)* encontrados pelo filtro de cores;

9.2.0.2 Tela de exibição da imagem da câmera

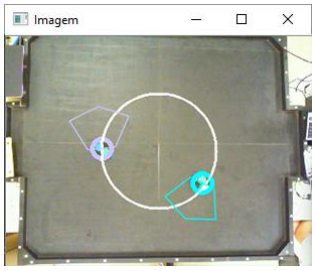


Figura 9.4: Tela de exibição de imagem

A figura 9.4 mostra a tela que utiliza-se para mostrar a imagem proveniente da câmera. A janela possui exatamente o tamanho da imagem capturada (no nosso caso 320x240px) e pode ser movida independentemente das demais.

9.2.0.3 Tela de registro de novos objetos

A tela observada abaixo, foi construída com o intuito de Registrar/Configurar os objetos. Estes objetos podem ser robôs, obstáculos que queira-se observar, ou até algum objeto sem definição precisa como seria o caso dos pedestres.

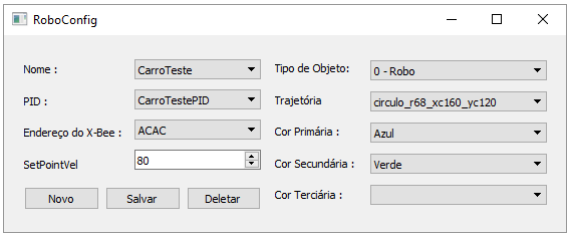


Figura 9.5: Interface para registro/configuração de novos objetos.

Nesta tela encontram-se alguns *dropdowns*(objeto que possui uma lista de opções, em que o usuário seleciona uma delas) que possuem as seguintes características:

- *Dropdown* com identificador “Nome”.
  - ▶ Habilitado;
  - ▶ Preenchimento obrigatório;
  - ▶ Possui os “nomes” dados aos objetos, esses nomes são uma forma de identificação de qual objeto estamos configurando;
- *Dropdown* com identificador “PID”.
  - ▶ Habilitado caso seja um robô e desabilitado caso contrário;
  - ▶ Preenchimento obrigatório;
  - ▶ Possui os “nomes” dados aos controladores, esses nomes são as identificações das estratégias de controle configuradas e a diferença entre as configurações são os ganhos Proporcional, Derivativo e Integral para os controladores Linear e Angular separadamente. A seleção de uma dessas configurações indica qual estratégia de controle será utilizada para o carro em questão;
- *Dropdown* com identificador “Endereço do X-Bee”.
  - ▶ Habilitado caso seja um robô e desabilitado caso contrário;
  - ▶ Preenchimento obrigatório;
  - ▶ Possui os canais de comunicação possíveis;
- *DoubleSpinBox* com identificador “SetPointVel”.
  - ▶ Habilitado;
  - ▶ Preenchimento obrigatório;
  - ▶ É um campo preenchido com o valor de *setpoint* de velocidade para o objeto em questão. A dimensão de velocidade é *pixels/s*, ou seja ao preencher com o valor de 80 significa que o usuário quer que aquele objeto se desloque com 80 *pixels/s* linearmente;
- *Dropdown* com identificador “Tipo de Objeto”.
  - ▶ Habilitado;
  - ▶ Preenchimento obrigatório;
  - ▶ Possui os tipos possíveis de objetos - Robô(0) ou Obstáculo(1) ou Indefinido(2);
- *Dropdown* com identificador “Trajetória”.
  - ▶ Habilitado;

- ► Preenchimento obrigatório;
- ► Possui as possíveis trajetórias. Ao escolher uma e salvar, o usuário está indicando qual trajetória aquele objeto deve executar;

- *Dropdown* com identificador “Cor Primária”.
  - ► Habilitado;
  - ► Preenchimento obrigatório;
  - ► Possui as possíveis cores cadastradas no banco. Ao escolher uma e salvar, o usuário está indicando a primeira cor que será usada na identificação daquele objeto (é importante ressaltar que as combinações de cores de cada objeto devem ser diferentes, uma vez que 2(dois) objetos com “nomes” diferentes mas com a mesma combinação de cores fazem com que o sistema não seja capaz de identificar esses objetos);

- *Dropdown* com identificador “Cor Secundária”.
  - ► Habilitado;
  - ► Preenchimento obrigatório;
  - ► Possui as possíveis cores cadastradas no banco. Ao escolher uma e salvar, o usuário está indicando a segunda cor que será usada na identificação daquele objeto (é importante ressaltar que as combinações de cores de cada objeto devem ser diferentes, uma vez que 2(dois) objetos com “nomes” diferentes mas com a mesma combinação de cores fazem com que o sistema não seja capaz de identificar esses objetos);

- *Dropdown* com identificador “Cor Terciária”.
  - ► Habilitado;
  - ► Preenchimento opcional;
  - ► Possui as possíveis cores cadastradas no banco. Ao escolher uma e salvar, o usuário está indicando a terceira cor que será usada na identificação daquele objeto (é importante ressaltar que as combinações de cores de cada objeto devem ser diferentes, uma vez que 2(dois) objetos com “nomes” diferentes mas com a mesma combinação de cores fazem com que o sistema não seja capaz de identificar esses objetos);

Além dos elementos listados acima existem também 3(três) botões nessa tela. O mais a esquerda é o “Novo”. Ao ser clicado, esse botão mostra uma nova tela (figura 9.6) pedindo que o usuário insira o nome que deseja dar ao novo objeto criado. O botão seguinte com o nome “Salvar”. salva no banco as configurações dos objetos, se o objeto for

novo então ele será inserido no banco de dados, caso contrário ele será somente atualizado. Por fim, o botão “Deletar” deleta o objeto selecionado do banco assim como suas configurações.

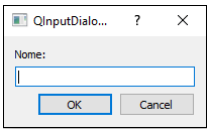


Figura 9.6: Tela de inserção do nome do objeto.

Neste ponto mostrou-se interessante demonstrar a utilização da tela descrita. Para isso, 2(dois) exemplos foram utilizados, sendo o primeiro sobre a criação de um novo objeto e o segundo sobre a configuração/reconfiguração de um objeto já existente.

Exemplo 1 (Criação de um novo objeto do tipo Robô): Para criar um novo objeto a partir da tela principal, deve-se primeiramente ir na aba “Configurações” e clicar na opção “Cadastro Robôs”. Após a abertura da nova tela, clica-se no botão “Novo” e com isso uma nova tela com uma caixa de texto surgirá (vide figura 9.6). Dado um nome para o novo objeto, o passo a seguir é apertar o botão “Ok” se desejar continuar com a criação do novo objeto ou o botão “Cancelar” se desistir de fazê-lo. No caso, optou-se por seguir, e com isso o sistema retorna para a tela da figura9.7. Como o objetivo é criar um objeto do tipo robô, deve-se agora selecionar a opção “0 - Robô” no *dropdown* com a identificação “Tipo de Objeto”, depois deve-se escolher a configuração de PID um endereço para o X-Bee, a trajetória desejada para o robô, as 2(duas) ou 3(cores) que serão usadas na identificação do objeto e por fim preencher o campo para o *setPoint* de velocidade (vale ressaltar que a ordem de preenchimento descrita não é obrigatória). Assim, chega-se à situação vista na figura 9.8, em que pode-se clicar no botão “Salvar” e assim o nosso novo objeto será inserido no banco de dados.

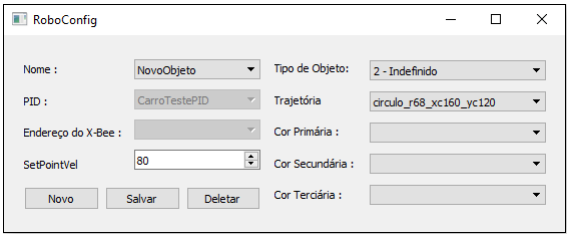
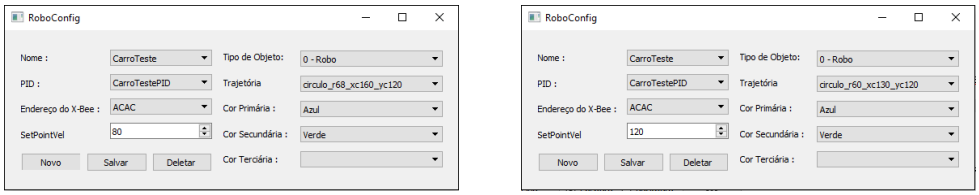


Figura 9.7: Tela de configuração de objetos.

Figura 9.8: Tela de configuração de objetos.

Exemplo 2 (Reconfiguração/Deleção de um objeto do tipo Robô): Esse exemplo irá partir diretamente da tela "RoboConfig" uma vez que o caminho para chegar nela foi descrito no exemplo anterior. Nesse estado, seleciona-se um objeto no *dropdown* "Nome". Feito isso, se deseja retirar esse objeto do banco de dados basta clicar no botão "Deletar", mas pode-se também alterar as configurações do mesmo. Para fins ilustrativos será alterado o *setPoint* de velocidade de um robô assim como sua trajetória. Para isso, no campo "Setpoint" trocou-se o valor de 80 para 120, em seguida clicou-se no *dropdown* de trajetória e trocou-se de `circulo\_r68\_xc160\_yc120` (Círculo de raio 68 *pixels*, centrado em [160, 120]) para `circulo\_r60\_xc130\_yc120` (Círculo de raio 60 *pixels*, centrado em [130, 120]) e finalizou-se o processo apertando no botão "Salvar" para atualizar as informações do banco de dados. Esse procedimento pode ser observado na sequencia de imagens abaixo:



(a) Situação Inicial (b) Situação Final

Figura 9.9: Passos para criação de um novo objeto

9.2.0.4 Tela de calibração do filtro de cores

Como pode-se observar na figura 9.10 a tela para calibração de cores possui:

- 1 *dropdown*, sempre habilitado. Esse *dropdown* possui os nomes das cores já existentes no banco de dados.
- 8 *sliders*, em que 3(três) deles são os valores mínimos de HSV (*hue* (matiz ou tom), *saturation* (saturação) e *value* (valor ou brilho)), outros 3(três) possuem os valores máximos de HSV e os 2(dois) restantes possuem os valores mínimo e máximo da área do *blob*.
- 1 tela, onde é possível visualizar o resultado do filtro aplicado.

- 3 botões, “Salvar”. “Adicionar” e “Deletar”.

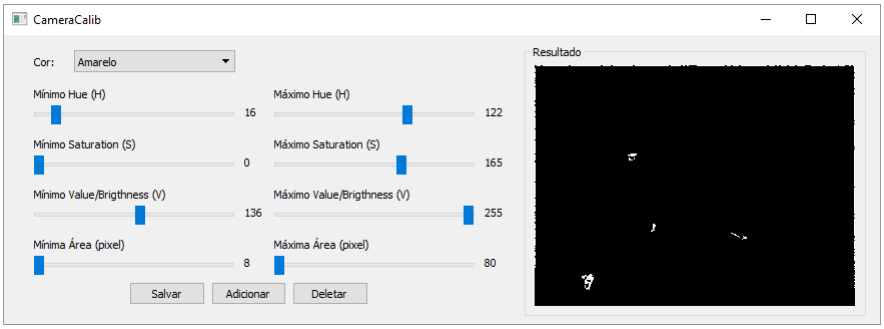


Figura 9.10: Tela para inserir/deletar/calibrar os filtros de cores.

O processo de calibração de uma cor é fácil. Inicialmente, verifica-se se a cor desejada já existe no banco de dados, para isso clica-se no *dropdown* com o identificador “Cor” e procura-se o nome da cor almejada.

Caso a mesma seja encontrada, seleciona-se o índice correspondente no *dropdown* e assim começa-se a alterar os valores dos *sliders* de forma e encontrar os mínimos e máximos de tom (H), saturação (S), brilho (V) e área que retornem a cor e tamanho desejados e retire o restante. Para facilitar esse procedimento a tela de título “Resultado” mostra uma imagem preto e branca correspondente a matriz binária resultante da saída do filtro passa-faixa utilizado para filtrar a imagem capturada pela câmera. Nessa tela, as partes brancas são os *pixels* que não foram filtrados, isto é, estavam dentro dos intervalos de HSV definidos nos valores dos *sliders* e os *pixels* pretos são aqueles que foram eliminados pela aplicação do filtro. Dessa maneira, possui-se um *feedback* instantâneo sobre o filtro que está sendo empregado na cor escolhida. Por fim, ajustados todos os valores da calibração precisa-se clicar no botão “Salvar” a fim dos novos valores de HSV e área associados àquela cor sejam registrados no banco.

Entretanto, na situação em que a cor cobiçada não seja achada, antes de começar a modificar os valores dos *sliders* é necessário que primeiramente a cor seja “criada”. Para tal, deve-se clicar no botão “Adicionar” e na nova tela (9.11) que surgirá preencher o nome da cor que deseja-se calibrar. Depois de fazer isso, segue-se as etapas descritas no parágrafo anterior (situação em que a cor já existia no banco de dados).

Para terminar as funcionalidades dessa tela para o caso em que se deseja deletar uma cor do banco de dados, basta selecionar a mesma no *dropdown* e após isso clicar no botão “Deletar”.

9.2.0.5 Tela de Ajuste de PID

A última tela do programa desenvolvido a ser apresentada é a tela de ajustes/configuração dos controles PID vista abaixo.

Para melhor explicar essa tela, essa será dividida em áreas:

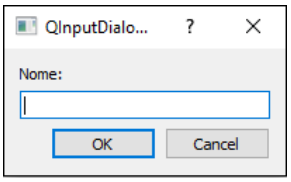


Figura 9.11: Tela de inserção do nome da cor.

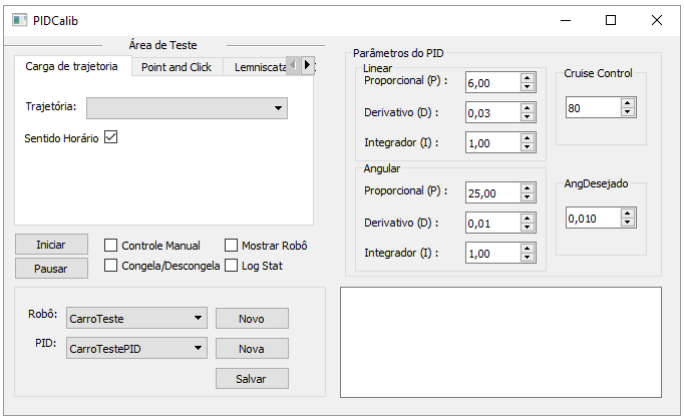
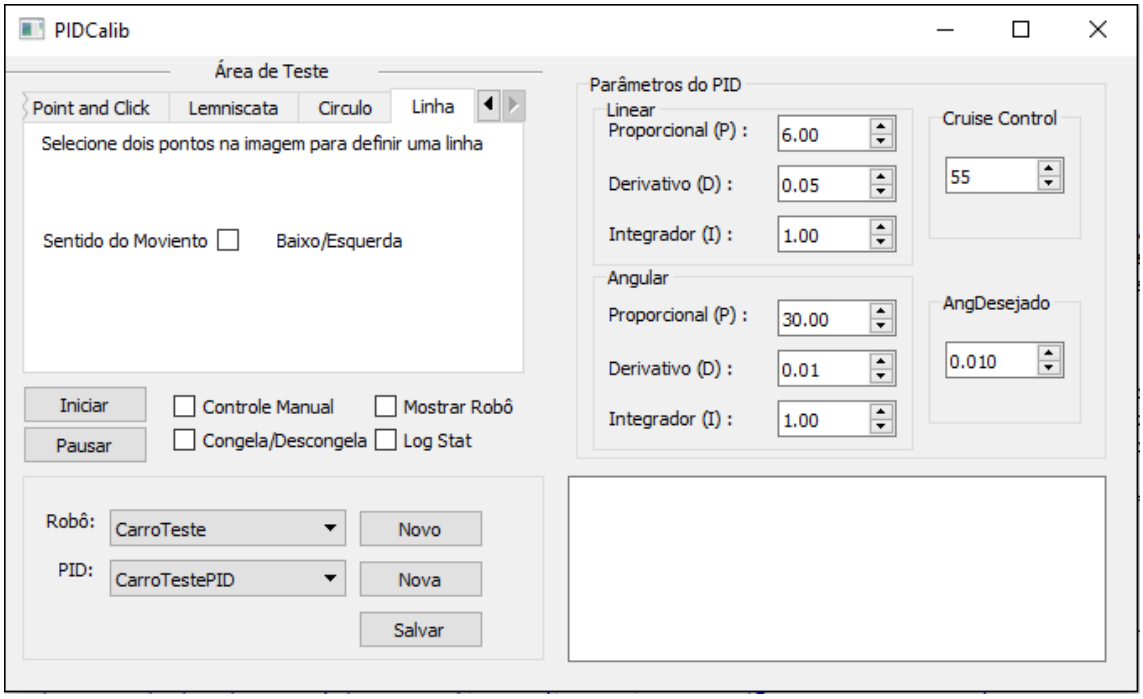
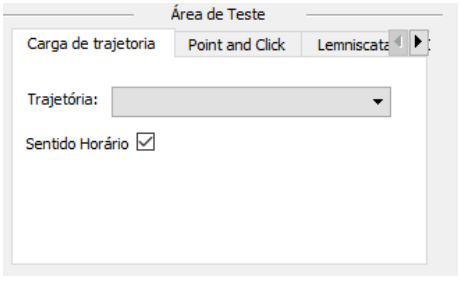


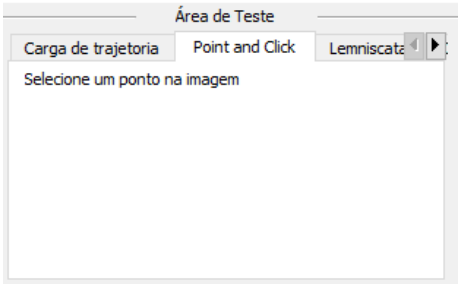
Figura 9.12: Tela de configuração/teste/criação de controles(PID).



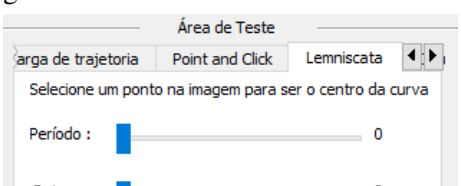
(a) Tela Completa

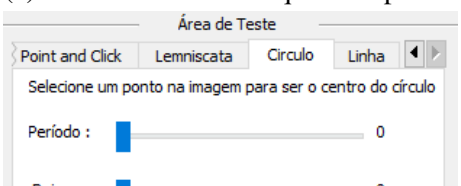


(b) Aba de teste com trajetória carregada.



(c) Aba de teste com clique num ponto.





1ª - Área de Teste Esta região é formada pelas diversas abas vistas na figura 9.13, 2(dois) botões e 4(quatro) *checkboxes*. Esses 6 últimos itens tem "ação global" em relação aos testes, ou seja, ao serem clicados, não importa em qual aba de teste se esteja a ação deles será executada. Sendo assim, esses serão descritos primeiro para depois se falar sobre cada aba individualmente.

- Botão "Iniciar" → De maneira bastante intuitiva, esse botão ao ser clicado inicia/-reinicia um teste.

- Botão "Pausar" → Ao ser clicado durante a execução de um teste esse botão envia uma mensagem de parar ao robô usado no teste, zera o valor das componentes integral do controles (a fim de evitar que ao reiniciar um teste essa componente force o carro a dar uma acelerada brusca) e pausa o teste.

- *Checkbox* "Controle Manual" → Ao ser marcado, desabilita o controle automático do carro, ou seja, desliga os PID's e dá ao usuário a capacidade de controlar manualmente o robô utilizando as teclas (A, S, W e D) que correspondem respectivamente a "virar para esquerda". "dar ré". "andar pra frente" e "virar à direita".

- *Checkbox* "Mostrar Robô" → Assim como o *checkbox* "Mostrar Desenho Robô" mencionado na tela principal 9.2.0.1 tem a função de desenhar a(s) marcação(ões) de identificação do(s) carrinho(s) assim como sua direção e sentido no caso de ser marcado.

- *Checkbox* "Congela/Descongela" → Ao ser selecionado faz com que a imagem mostrada na tela de vídeo seja "Congelada" permitindo ao usuário fazer a análise de uma situação específica.

- *Checkbox* "Log Stat" → Esse checkbox ao ser marcado tem a finalidade de registrar num arquivo de texto (.txt) um log com as informações de todos os carrinhos na tela, como coordenadas, ângulo atual, ângulo desejado, velocidade linear atual e desejada, número do frame, se o carro foi ou não identificado pela parte de "visão" do programa, erros angular e linear etc, de forma a possibilitar os autores, com o auxílio de um script feito pelos mesmos em Matlab, plotarem automaticamente os gráficos necessários para fazer as análises do seguimento de trajetória, fps médio, controle angular, controle linear, cruise control etc observados na seção 7. Esse script pode ser visualizado no Apêndice 9.1.

Anteriormente mencionou-se que os botões e *checkboxes* descritos acima são de ação global nos testes, isso tendo em vista que os elementos internos de cada aba tem influência somente no teste ao qual ele se relaciona, ou seja, o *dropdown* observado na aba "Carga de Trajetória" não exerce nenhuma influência no teste da aba "Círculo", assim como o

checkbox de “Sentido Horário” da aba “Círculo” não influencia em nada o teste da aba “Lemniscata” ou na aba “Carga de Trajetória” ou em qualquer outra aba, diz-se então que estes elementos tem ação local no teste.

2ª - Área de Criação/Seleção de PID e/ou Robô Nesta área pode-se escolher o robô que será usado nos testes da configuração do PID usando o *dropdown* “Robô”. Além do robô, utilizando o *dropdown* “PID” escolhe-se qual a configuração de controle PID que estará sendo configurado, em que cada configuração é um conjunto dos ganhos para o controle Linear e Angular. Logo, pode-se ter no banco de dados uma série de configurações possíveis de PID para o caso de haver mais de uma estrutura de robô como por exemplo se tivesse um robô que ao invés de representar um carro representasse um ônibus.

Ainda nessa área pode-se observar 3(três) botões. Ao ser clicado o botão “Novo” dispara a tela mencionada na parte 9.2.0.3. Já o “Nova” cria uma nova configuração de PID a ser “calibrada” (ajustar os ganhos dos PID’s Linear e Angular), esse botão tem um funcionamento idêntico ao botão “Novo” na tela 9.2.0.3. Por fim, o último botão “Salvar”, como o nome já deixa a entender, salva as alterações feitas nas configurações de PID, inclusive, caso uma configuração tenha sido criada, é esse botão que registra as informações dessa nova configuração no banco de dados.

3ª - Área de Parâmetros do PID Essa área foi dividida em 4(quatro) pequenas regiões:

- Linear: Nessa região encontram-se 3(três) *doublespinboxes*, um para o ganho (K_P) da parcela proporcional do controle PID linear, um para o ganho (K_I) da parte integral do mesmo PID e o último para o ganho (K_D) da fração derivativa do controle.
- Cruise Control: Nessa pequena região existe somente um *doublespinbox* onde pode-se variar o *setpoint* de velocidade linear do robô que está sendo usado.
- Angular: Essa região tem a mesma estrutura que a descrita na Linear, entretanto a variação dos valores dos *doublespinboxes* altera os ganhos do controle angular e não mais do linear.
- AngDesejado: Essa última região possui também um *doublespinbox* que altera o valor da constante K usado no cálculo do ângulo desejado para o objeto em questão. Esse ganho assim como sua função são melhor explicados na seção 4.2.

4ª - Área de Resultados/Log Dividida em 2(duas) regiões, essa área possui uma caixa de texto observada no canto direito inferior da tela em questão e uma tela de vídeo desconectada fisicamente da tela de configuração do PID.

Na caixa de texto observam-se algumas informações sobre o carro como por exemplo velocidade linear atual do carro, ângulo desejado para o objeto naquele *frame*, ângulo

atual do carro etc. Já na tela de vídeo, pode-se observar o vídeo capturado pela câmera, a movimentação do carro, e a trajetória desejada naquele instante para o carro, permitindo assim analisar se o comportamento do carro para os valores de ganhos dos PID‘s Linear e Angular estão bons ou se ainda precisam de alguma alteração.

Referências Bibliográficas

[1] BAK, RICHARD. *Henry and Edsel:The Creation of the Ford Empire*. 1 ed. New Jersey, John Wiley and Sons, 2003.

[2] "Problemas do Engarrafamento". Disponível em: <http://g1.globo.com/sao-paulo/noticia/2016/04/transito-em-sp-causa-317-km-de-filas-e-bate-recorde-de-lentidao-no-ano.html>. Acessado em 12 de Janeiro de 2017.

[3] "Acidentes de Trânsito". Disponível em: <http://www1.folha.uol.com.br/cotidiano/2016/05/1772858-brasil-e-o-quarto-pais-com-mais-mortes-no-transito-na-america-diz-oms.shtml>. Acessado em 12 de Janeiro de 2017.

[4] "Desperdício causado por engarrafamentos". Disponível em: <http://exame.abril.com.br/brasil/o-tamanho-do-transito-em-sp-e-o-que-a-cidade-perde-com-isso/>. Acessado em 12 de Janeiro de 2017.

[5] "Porque engarrafamentos existem". Disponível em: <http://m.gizmodo.uol.com.br/video-engarrafamentos-transito/>. Acessado em 05 de Janeiro de 2017.

[6] "Ted sobre carros autônomos". Disponível em: <https://www.youtube.com/watch?v=OlLFK8oSNEM&feature=youtu.be>. Acessado em 17 de Janeiro de 2017.

[7] "Beneficios dos carros autônomos". Disponível em: <http://www.inova.jor.br/2016/07/29/carros-autonomos/>. Acessado em 01 de Fevereiro de 2017.

[8] ULSOY, A. G., PENG, H., ÇAKMAKCI, M. *Automotive Control Systems*. teste, Cambridge University Press, 2012. Disponível em: <https://books.google.com.br/books?id=OK0y9pV1O7UC>.

[9] NAKAMURA, M., HANAWA, K., MATSUURA, K., KURAGAKI, S. "Intelligent cruise control system for moving body". 2000. Disponível em: <https://www.google.com/patents/US6044321>.

[10] FRANKLIN, G. F., POWELL, J. D., EMAMI-NAEINI, A. *Sistemas de Controle para Engenharia - 6ed*. teste, Bookman Editora, 2013. Disponível em: <https://books.google.com.br/books?id=WB84AgAAQBAJ>.

[11] DE WIT, C. C., SICILIANO, B., BASTIN, G. *Theory of Robot Control*. 12345, Springer London, 2012. Disponível em: <https://books.google.com.br/books?id=5NnUBwAAQBAJ>.

[12] GONZÁLEZ JIMÉNEZ, J Y OLLERO BATURONE, A. *Estimación de la posição de un robot móvil*. 1234, Asociación Española de Informática y Automática, 1996.

[13] BRADSKI, G., KAEBLER, A. *Learning Opencv*. 1 ed. New York, O'Reilly Media, 2008.

[14] BALLARD, D. H., BROWN, C. M. *Computer Vision*. New York, New Jersey: Prentice Hall, 1982.

[15] LUCCHESE, L., MITRA, S. K. "Color image segmentation: A state-of-the-art survey", 2001.

[16] ITOH, S., MATSUDA, I. "Segmentation of colour still images using Voronoi diagrams", *Proc. Of the 8th European signal processing conference (eusipco-96)*, 1996.

[17] SAPIRO, G. "Color snakes", .

[18] LOWE, D. "Distinctive image features from scale-invariant keypoints", *International Journal of Computer Vision*, v. 60, 2, pp. pp. 91–110, 2004.

[19] BAY, H., TUYTELAARS, T., VAN GOOL, L. "SURF : Speeded-Up Robust Features". In: *European Conference on Computer Vision (ECCV)*, 2006.

[20] HARRIS, C., STEPHENS, M. "A combined corner and edge detection". In: *Proc. of the Fourth Alvey Vision Conference*, pp. pp. 147–151, 1988.

[21] KADIR, T., BRADY, M. . "Saliency, scale and image description", *International Journal of Computer Vision*, v. 45, 2, pp. pp. 83–105, 2001.

[22] ITTI, L., KOCH, C. "Computational modeling of visual attention", *Nature Reviews Neuroscience*, v. 2, 3, pp. pp.194–203, 2001.

[23] ITTI, L., KOCH, C., NIEBUR, E. "A model of saliency-based visual attention for rapid scene analysis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 20, 11, pp. pp. 1254–1259., 1998.

[24] BUTKO, N. J., ZHANG, L., COTTRELL, G. W., MOVELLAN, J. R. "Visual Salience Model for Robot Cameras", *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. pp. 2398–2403, 2008.

[25] RUSU, R. B., BLODOW, N., MARTON, Z. C., BEETZ, M. "Close-range scene segmentation and reconstruction of 3D point cloud maps for mobile manipulation in domestic environments", *IEEE/RSJ Intelligent Robots and Systems (IROS)*, pp. pp. 1–6, 2009.

[26] FILLIAT, D., BATTESTI, E., BAZEILLE, S., DUCEUX, G., GEPPERTH, A., HARRATH, L., JEBARI, I., PEREIRA, R., TAPUS, A., MEYER, C., IENG, S., BENOSMAN, R., CIZERON, E., MAMANNA, J.-C., POTHIER, B. "Rgbd object recognition and visual texture classification for indoor semantic mapping." In: *Proceedings of the 4th International Conference on Technologies for Practical Robot Applications (TePRA)*, 2012.

[27] SMITH, B. "Object-Oriented Programming". In: *Advanced ActionScript 3: Design Patterns*, pp. 1–23, Berkeley, CA, Apress, 2015. ISBN: 978-1-4842-0671-3. doi: 10.1007/978-1-4842-0671-3_1. Disponível em: <http://dx.doi.org/10.1007/978-1-4842-0671-3_1>.

[28] DAGUM, L., MENON, R. "OpenMP: an industry standard API for shared-memory programming", *IEEE Computational Science and Engineering*, v. 5, n. 1, pp. 46–55, 1998. ISSN: 1070-9924.

[29] POLOLU. "Pololu 3pi Robot". Disponível em: <https://www.pololu.com/product/975>. Acessado em 27 de janeiro de 2017.

[30] THURSKÝ, B., GAŠPAR, G. "Using Pololu's 3pi robot in the education process", .

[31] "Xbee". Disponível em: <https://www.digi.com/lp/xbee>.

[32] SPARKFUN. "SparkFun XBee Explorer Regulated". . Disponível em: <https://www.sparkfun.com/products/11373>. Acessado em 27 de janeiro de 2017.

[33] SPARKFUN. "SparkFun XBee Explorer Dongle". . Disponível em: <https://www.sparkfun.com/products/11697>. Acessado em 27 de janeiro de 2017.

[34] HIRZEL, T. "Arduino Tutorials - PWM". Disponível em: <https://www.arduino.cc/en/Tutorial/PWM>. Acessado em 07 de fevereiro de 2017.

[35] MICROSOFT. "Microsoft LifeCam HD3000". Disponível em: <https://www.microsoft.com/accessories/en-us/products/webcams/lifecam-hd-3000/t3h-00011#specsColumns-testCarousel>. Acessado em 27 de janeiro de 2017.

[36] GENE F.FRANKLIN, J. DAVID POWELL AND ABBAS EMAMI-NAEINI. *Feedback Control of Dynamic Systems*. 4 ed. New Jersey, Prentice Hall, 2002.

[37] "O Controle PID de Forma Simples e Descomplicada". Disponível em: <https://www.citisystems.com.br/controle-pid/>. Acessado em 14 de Janeiro de 2017.

[38] "Acidentes com Cruise Control". Disponível em: <http://www.snopes.com/autos/techno/cruise.asp>. Acessado em 12 de Janeiro de 2017.

[39] E.L. LEHMANN, G. C. *Theory of Point Estimation*. teste, Springer New York, 2003.

[40] WEISSTEIN, E. W. "Cassini Ovals". Disponível em: <http://mathworld.wolfram.com/CassiniOvals.html>.

[41] "Trabalho sobre Medição de radioatividade por imagem". Disponível em: <http://carpedien.ien.gov.br:8080/browse?type=author&value=MONTEIRO%2C+Jos%C3%A9+Guilherme+Tavares>. Acessado em 04 de Janeiro de 2017.

[42] "About QT". Disponível em: <https://www.qt.io/>. Acessado em 10 de Fevereiro de 2017.

[43] "About SQLite". Disponível em: <https://sqlite.org/about.html>. Acessado em 23 de Janeiro de 2017.

[44] MATHWORKS. "Add MATLAB data to database tables". https://www.mathworks.com/help/database/ug/insert.html, 2017. [Online; acessado em 23-Janeiro-2017].

[45] "About SVN". Disponível em: <http://svnbook.red-bean.com/>. Acessado em 03 de Fevereiro de 2017.

[46] "About Tortoise". Disponível em: <https://tortoisesvn.net/>. Acessado em 03 de Fevereiro de 2017.

[47] NONAKA, I., TAKEUCHI, H. *The Knowledge Creating Company*. teste, Oxford University Press, 1995. Disponível em: <https://books.google.com.br/books?id=OK0y9pV1O7UC>.

[48] “About SCRUM2”. Disponível em: <http://scrummethodology.com/>. Acessado em 04 de Fevereiro de 2017.

[49] “About Trello”. Disponível em: <https://trello.com/>. Acessado em 04 de Fevereiro de 2017.

[50] WALDNER, J. B. “Principles of Computer Integrated Manufacturing”. 1992.

[51] “About HSV”. Disponível em: <http://www.tech-faq.com/hsv.html>. Acessado em 12 de Janeiro de 2017.

[52] “SQLite Manager”. Disponível em: <https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/>. Acessado em 24 de janeiro de 2017.

[53] FALUDI, R. *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*. 1234, O’Reilly Media, 2010. Disponível em: <https://books.google.com.br/books?id=xMC69vQJLZIC>.