Universidade Federal de São Carlos
Departamento de Computação

Projeto do AGV

Grupo do Controle
Tiago Bachiega de Almeida (628247)
Rodolfo Krambeck Asbahr(628042)

Disciplina
Sistemas de Integração e Automação Industrial

Professor
Orides Morandin Jr

O sistema de controle de um veículo auto-guiado, um AGV, deve permitir que ele consiga seguir trajetórias pré-definidas por uma linha, sem nunca sair de seu domínio. Um sistema desse tipo, embora possa ser implementado com um método de controle simples em versões de prototipação, necessita de técnicas de controle mais elaboradas para operar em escala industrial. O AGV possui diversas aplicações, como carregar carga de um lugar para outro em uma determinada instalação. Esta instalação pode ir desde um estoque de livros, com veículos menores, até containers de navios, que exige um veículo maior. Em ambos os casos é muito desejado que o veículo tenha precisão e consiga realizar sua rota de forma eficiente, minimizando ao máximo os erros que poderiam ocorrer no caminho, assim sendo, é desejável a implementação de métodos de controle mais sofisticados, que podem dar mais robustez à operação do veículo.

Sumário

1. Introdução

- Qual é o estado atua do sistema de controle do AGV?
- Quais são as melhorias de interesse e por que são necessárias?

No atual estado do AGV, no que diz respeito ao seu Sistema de Controle, temos um cenário onde há uma entrada de uma câmera apontada para a trilha que deve ser percorrida e enviando a imagem capturada para um código de Visão Computacional que computa o erro entre a trajetória obtida e a esperada. Este módulo que une a câmera e o código de Visão Computacional é executado dentro de um Raspberry Pi embarcada no veículo. Dentro da placa também será adicionado um cálculo de PID que processa o erro obtido com o intuito de enviar ao Arduino que opera as rodas Mecanum a velocidade desejada para cada roda, tentando corrigir o erro de trajetória encontrado.

Atualmente, somente o Arduino opera um PID Simples, para garantir uma consistência na velocidade de cada roda. Em uma aplicação industrial, porém, é necessário um Controle mais robusto, que suporte variações de velocidade, de massa colocada sobre o veículo, que consiga transitar em diferentes tipos de trajetórias, entre outros atributos. Com isso, é interessante o estudo e aplicação de técnicas mais sofistacadas com o PID Adaptativo[1][2] ou o Controle Fuzzy[7]. O intuito é que esse Controle mais adequado adicionado à Raspberry Pi de forma a processar melhor o erro, visto que o PID Simples do Arduino já opera bem para manter constante a velocidade das rodas, logo não necessita de mudanças.

Também se faz necessária a implementação de um Sistema de Controle que consiga se comunicar bem com a interface supervisória. Basicamente, Sistema Supervisório se trata de um sistema que consegue se comunicar com uma dada planta operante, no nosso caso o AGV, e consegue obter informações sobre o funcionamento e ainda enviar comandos ao sistema embarcado para alterar seu modo de operação caso necessário. Estando uma equipe trabalhando no Sistema Supervisório, deseja-se criar implementações do Sistema de Controle que podem ser facilmente manipuladas pala interface de supervisão.

As seguintes seções estão organizadas de forma a demonstrar o desenvolvimento do projeto desde a parte téorica da técnica de controle em questão até partes mais específicas de sua implementação. Inicialmente, haverá uma descrição geral do que são os controles adaptativos e quais são as particularidades do PID Adaptativo e do Controle Fuzzy. A seção referente à proposta conterá todo o projeto dos sistemas de controle apresentado, desde a descrição da lógica de funcionamento que adotamos para cada técnica à explicação dos diversos Diagramas UML utilizados para guiar e documentar o desenvolvimento. Por sua vez, na seção de Desenvolvimento, tem-se a completa descrição dos detalhes de implementação e uso dos sistemas contemplados, terminando com uma análise de resultados e uma conclusão sobre o projeto.

2. Sistemas de Controle Adaptativos

2.1. Sistemas de Controle Adaptativos e PID Adaptativo

- O que são sistemas de controle adaptativos e suas vantagens e desvantagens?
- O que é o PID Adaptativo?
- Quais as vantagens do PID Adaptativo para o AGV?

Sistemas de Controle Adaptativos, como o nome sugere, conseguem adaptar seus parâmetros de execução de forma a melhor operar sobre sua dada função. Este tipo de controle consegue perceber através de métodos matemáticos que seu desempenho atual não está conforme o desejado e consegue se ajustar para tentar minimizar sua taxa de mal funcionamento. A grande maioria dos sistemas de controle industriais são implementados com técnicas adaptativas justamente por serem mais robustos e ainda conseguirem tirar das mãos humanas a tarefa de regular precisamente seus parâmetros, conseguindo operar apenas com uma regulagem aproximada inicial. Apesar destas vantagens, possuem uma maior complexidade de implemetação e podem vir acompanhados de um maior custo computacional, exigindo sistemas mais poderosos para serem executados. No trabalho atual, iremos apresentar a implementação do PID Adaptativo e também sugerir a implementação do Controle Fuzzy, que também está sendo trabalhado.

O PID Adaptativo nada mais é do que uma variação do PID Simples[3] de forma que há a possibilidade de se modificar os parâmetros Kp, Ki e Kd. Em outra palavras, o sistema é capaz de perceber que os parâmetros de sintonia atualmente em uso não são bons o suficiente para se garantir o comportamento desejado e por isso busca outras possíveis sintonias que podem ter melhores resultados. Existem várias maneiras de se verificar se os parâmetros de sintonia estão ou não ruins, a maioria delas usando análises estatísticas. Da mesma forma, pode-se também implementar o método de escolha de novos Kp's, Ki's e Kd's de várias formas, desde a utilização de uma Tabela de Sintonia com parâmetros previamente inserida, que é o caso deste trabalho, como com a utilização de técnicas de Machine Learning.

No que diz respeito ao AGV, o PID Adaptativo como sistema de controle será útil de várias formas. A primeira aplicação é garantir que mesmo com variações de cargas o veículo não sairá de sua trajetória. A segunda aplicação implica que, como o próprio veículo agora vai corrigir seu caminho, é possível que ele consiga andar em qualquer trajetória contínua, incluindo curvas longas ou curtas, ou cruzamentos de linhas de trajetória, sem se desviar do objetivo. Assim sendo, pode-se dizer que com a adição de um método adaptativo de controle ele estará mais próximo de poder ser utilizado em indústrias.

2.2. Lógica Fuzzy e Controle Fuzzy

- O que é Lógica Fuzzy e como se aplica ao AGV?
- Quais são seus principais componentes e etapas?
- Quais as vantagens do Controle Fuzzy?

Sobre a Lógica Fuzzy[7], inicialmente foi proposta como um método de processameto de dados e sua base está no mapeamento não linear de um dado de entrada resultando em uma saída escalar, possuindo 4 etapas ou componentes principais: Fuzzificação, Regras, Motor de Inferência e Defuzzificação. Na lógica tradicional, podemos ter os valores 0 e 1 representando por exemplo Falso ou Verdadeiro. Em Fuzzy, podemos usar os valores intermediários a 0 e 1 para representar quão Falso ou quão Verdadeiro é um dado parâmetro, como exemplo, uma afirmação pode ser 0.3 x Falsa e 0.7 x Verdadeira. Para o caso do AGV, por exemplo, além de quantizar se o carrinho está deslocado em relação à linha ou não, podemos dizer o quão deslocado ele está, sabendo se ele está muito para a esquerda, pouco para a esquerda, muito pouco para a esquerda, entre outras classificações. Estas denominações farão mais sentido quando for apresentada a

implementação do Controle Fuzzy, na próxima etapa do projeto.

Os principais componentes e etapas da Lógica Fuzzy são:

- Fuzzificação: processo no qual um valor escalar e real é transformado em uma relação chamada Conjunto Fuzzy ou Fuzzy Set. A saída deste processo podem ser termos linguísticos, como Muito Longe, Pouco Longe, etc. Ela relaciona o valor de entrada com alguma característica que o representa e que será usada pelo Motor de Inferência.
- Regras: as regras são um conjunto de relações de formado IF-THEN que relaciona o Conjunto Fuzzy a uma determinada ação, também representada com um conjunto deste tipo. Por exemplo, IF MuitoLonge THEN MuitaInclinacao, indicando que se o AGV está muito desviado da reta, ele deve se inclinar bastante para se corrigir.
- Motor de Inferência: relaciona o Conjunto Fuzzy de entrada com as Regras para fazer um Conjunto Fuzzy de saída.
- Defuzzificação: transforma o Conjunto Fuzzy de saída do Motor de Inferência, que representa uma reação à entrada, em um valor real e escalar.

Entre as vantagens de se implementar um Controle Fuzzy é que ele é relativamente simples de ser implementado e consegue lidar muito bem com problemas onde estão envolvidos dados que exigem respostas de saída extremamente adequadas aos seus valores. Por exemplo, temos que o valor de inclinação A que o AGV deve buscar para um desvio X é adequado apenas para este X e não para um outro valor de inclinação Y. Mesmo que com o Fuzzy trabalhe em faixas de valores e respostas pré-definidas, pode-se adicionar quantos níveis de precisão desejar-se, embora esteja em jogo a complexidade final da implementação que pode exigir uma variedade gigantesca de regras devido ao número elevado de possibilidades de Conjunto Fuzzy. Assim, pode-se dizer que ele tem a sensibilidade necessária para se controlar o apontamento do AGV em relação à linha que deve seguir por conseguir medir quantitativamente de uma maneira até mais precisa que o PID Adaptativo, o quão deslocado o veículo está.

3. Objetivos

- O que o grupo de trabalho se propõe a fazer?

Os objetivos deste trabalho do grupo de Controle são estudar e implementar as Técnicas de Controle PID Adaptativo e Fuzzy para o AGV. Deve-se elaborar um projeto completo do desenvolvimento, desde a elaboração de diagramas até a implementação e calibragem dos Sistemas de Controle. Além disso, os sistemas de controle implementados devem conseguir conversar facilmente com o Sistema Supervisório e, para isso, serão feitas funções adequadas que permitam a edição dos parâmetros de operação do controle.

4. Proposta

4.1 Modelagem dos softwares

4.1.1 Considerações iniciais

- O que será adicionado?
- Que tipo de diagramas será utilizado para descrever o projeto?
- Como os diagramas foram ordenados para garantir uma sequência lógica?

Atualmente, a Raspberry Pi do AGV apenas recebe os dados do processamento de imagem. Os Softwares que serão adicionados são o Controle PID Adaptativo e, parcialmente, como poderá ser entendido adiante, o Controle Fuzzy para processar o erro enviado do código de visão computacional.

Para o projeto teórico, foi elaborada uma série de Diagramas UML para servir de base ao desenvolvimento do Software. No atual momento, o controle PID Adaptativo é o foco de trabalho, mas o projeto está sendo feito contemplando a ideia de que futuramente haverá também o Controle Fuzzy.

No que diz respeito à organização dos diagramas utilizados, buscou-se seguir uma sequência lógica que possibilitasse o melhor entendimento do sistema. Primeiramente, temos o Diagrama de Componentes que explicita como cada funcionalidade do sistema de controle irá se comunicar com as demais. Depois disso, tem-se o Diagrama de Atividades cuja função é mostrar o comportamento do controle de uma maneira procedural, quase a nível algorítmico, partindo de uma representação macro do sistema como um todo e descendo para a representação individual de alguns componentes mais elaborados. O Diagrama de Casos de Uso e suas descrições são extremamente simples pois não há muitos atores e casos de uso no sistema, portanto ele só explicita que haverá a Visão Computacional e o Controle de Rodas do Motor, que são componentes externos afetados pelo sistema. Tem-se o Diagrama de Sequência, mostrando o ciclo de execução do controle ao longo do tempo e, finalmente, os Diagramas de Classe, dando uma visão mais a nível de implementação do que existirá em cada componente, no que diz respeito a classes, atributos e funções.

4.1.2 Lógica do PID Adaptativo

- Como irá funcionar o PID Adaptativo? Qual é a ideia por trás dele?
- O que é o Decisor?
- O que é a Tabela de Sintonia?

O Controle com PID Adaptativo foi modelado com base em dois componentes principais, à parte do próprio PID, são eles o programa Decisor e a Tabela de Sintonia. A ideia central é que o Decisor verifique se o sistema está funcionando bem com os parâmetros atuais e, caso não, decida entre uma combinação de Kp, Ki e Kd mais ou menos agressiva que a em uso, que será obtida da Tabela de Sintonia, possuidora de várias combinações de Kp's, Ki's e Kd's. O ideal é que os parâmetros de sintonia sejam ordenados por agressividade de correção do sistema nesta tabela, ou seja, devem ser organizados em ordem crescente de Ki, que vai ser o principal responsável pela quantidade de desvio que o AGV vai tentar executar para voltar para a linha, já que ele representa o acúmulo dos erros obtidos da Visão Computacional com o tempo.

O programa Decisor é responsável por processar o erro de trajetória, utilizando-se de análises

estatísticas sobre os valores da Visão Computacional, gerar um índice para a Tabela de Sintonia e tentar recuperar dela um Kp, um Ki e um Kd que mais se adequem àquele caso atual do AGV. Ele é quem faz a parte adaptativa do sistema. Mais detalhadamente, o Decisor irá acumular valores vindos da Visão Computacional e calcular o desvio padrão e a média destes valores. Comparando os resultados destes cálculos com limites impostos pelo usuário, irá escolher entre manter a sintonia atual ou tentar a próxima na tabela, mais agressiva.

Por sua vez, a Tabela de Sintonia contém uma série de combinações de Kp's, Ki's e Kd's, ordenadas por agressividade, em relação ao Ki, que serão previamente determinadas com sintonias Zieger-Nichols e deve ser acessada pelo Decisor. O Decisor, então, irá enviar as novas constantes para o PID Adaptativo que passará a operar com novos parâmetros. No sistema, esta tabela é um arquivo CSV de três colunas onde cada coluna representa o Kp, o Ki e o Kd e cada linha uma nova sintonia, sempre mais agressiva que a sintonia da linha logo acima.

4.1.3 Lógica do Controle Fuzzy

- Como irá funcionar o Controle Fuzzy? Qual é a ideia por trás dele?
- Alternativa à implementação pelo grupo.

O grupo concluiu de estudos da implementação do Controle Fuzzy. De fato, não há como se distanciar da implementação padrão da Lógica Fuzzy, ou seja, com certeza devem ser utilizadas as 4 etapas ou componentes principais desta técnica. Porém, o que se precisa planejar é como será modelado nosso Conjunto Fuzzy e como serão as Regras do sistema, que consiste em um trabalho cujo desenvolvimento ainda será iniciado.

Como alternativa à implementação do Fuzzy, foi sugerido pelo Professor Orides que o grupo estudasse possíveis bibliotecas em Python que já implementam a técnica. Além do Python, estudou-se a implementação do MatLab pela *toolbox* Fuzzy Logic Toolbox[8]. O grupo trabalhou nos Diagramas UML incluindo a presença do Controle Fuzzy e sua lógica principal no sistema de controle. Além disso, também foi possível explorar, embora superficialmente, a *toolbox* do MatLab, que será apresentada posteriormente.

4.2 Diagramas de componentes

- O que este diagrama possui e como ele deve ser entendido?
- Como se dá a adição do Controle Fuzzy?
- Por quê a adição de um componente para o PID Simples?
- Resumindo, como os componentes operarão no sistema final?
- O que será apresentado a seguir?

O Diagrama da Figura 4.2.1 consiste no Diagrama de Componentes do sistema. Nele está explicitada a estrutura do sistema como um todo, incluindo as interfaces que cada componente possui ou necessita, com as informações e relações que são compartilhadas entre eles. É possível verificar que todas as três técnicas de controle implementadas, PID Simples, PID Adaptativo e Controle Fuzzy recebem como entrada o resultado do processamento da Visão Computacional. Pode-se observar também a conexão do PID Adaptativo com o Decisor, e deste com a Tabela de Sintonia que utiliza para buscar novos parâmetros para o PID Adaptativo. Também pode-se verificar que associado ao processamento bruto do Fuzzy estão as Regras que ele utilizará para inferência e que devem ser acessadas quando chegar na etapa onde é executado o Motor de Inferência.

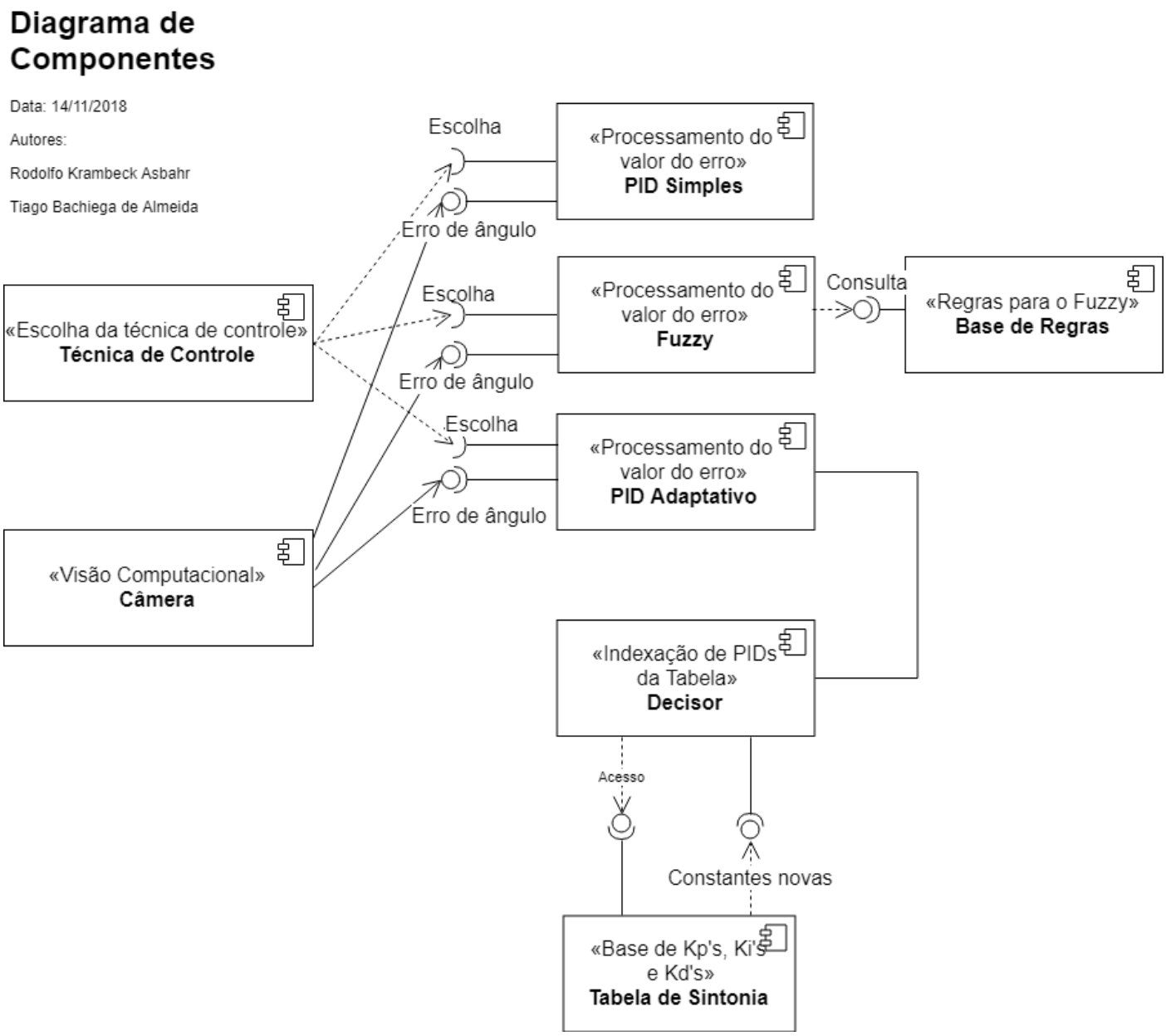
Com a adição do PID Simples e do Controle Fuzzy como opções de técnica de controle, tem-se que haverá um componente chamado Técnica de Controle que deve selecionar qual Técnica de Controle será executada. Ele consiste em um valor inteiro enviado pelo supervisório e é responsável por interromper a técnica atualmente em execução e iniciar a outra escolhida. Pode-se notar também a adição de um componente Base de Regras que é de onde o Fuzzy obtém as suas Regras para inferência.

Como o PID Simples já estava implementado no AGV, o grupo julgou uma boa ideia adicioná-lo como opção de técnica de controle, pois pode ser vantajoso para o usuário utilizar esta técnica para análises de sintonia, testes, entre outras aplicações.

Com o Diagrama de Componentes, tem-se uma visão geral de como os blocos mais gerais, de alto nível, do Sistema de Controle conversam entre si. Ou seja, pode-se verificar que o PID Simples apenas irá ler o valor de escolha do método de controle e, como será mais explorado adiante, se for o seu valor, será iniciado. Em seguida, obterá o resultado da Visão Computacional e realizará seu processamento, com o Kp, o Ki e o Kd pré-fixados. Da mesma maneira, o PID Adaptativo e o Controle Fuzzy são ativados de acordo com o valor da Escolha e lerão da Visão Computacional. O PID Adaptativo, por sua vez, irá interagir com o Decisor, que irá indexar a Tabela de Sintonia para adquirir novos Kp's, Ki's e Kd's. Já o Controle Fuzzy dependerá do componente Base de Regras, que será útil para o cálculo da inferência realizado internamente ao componente.

A seguir, será apresentado o Diagrama de Atividades. Este diagrama tem como principal função mostrar o processamento do Sistema de Controle de uma maneira procedural, deixando explícito quais ações o sistema toma e sob quais circunstâncias e demonstrando qual o caminho que cada técnica percorre desde a leitura da entrada até a emissão de seu resultado de saída.

Figura 4.2.1 – Diagrama de Componentes do Sistema



4.3 Descrição de atividades

- O que o Diagrama de Atividades do Projeto possui e como ele descreve o fluxo de comportamento do sistema para o PID Adaptativo e para o PID Simples?
- E para o Controle Fuzzy?
- Como o Diagrama de Atividades do Decisor descreve o fluxo de decisão de parâmetros de sintonia?
- Como o Diagrama de Atividades do PID Adaptativo descreve seu comportamento?
- Como o diagrama ilustra o sistema como um todo?
- O que será mostrado a seguir?

O Diagrama da Figura 4.3.1 o Diagrama de Atividades do Projeto. Este Diagrama consiste na representação do fluxo de atividades que será realizada pelo sistema em questão. Neste Diagrama, está explicitado que haverá uma escolha da Técnica de Controle utilizada e uma entrada do valor obtido do código de Visão Computacional. Com isso, caso seja selecionado o PID Adaptativo, será verificado se o erro obtido é tal que necessite da escolha de outros parâmetros de calibragem. Caso não seja, prossegue-se com o PID com a calibragem mais atual, caso contrário, o decisor deve buscar uma nova calibragem na Tabela de Sintonia. Caso seja escolhido o PID Simples como técnica de controle, o sistema deve ler o Kp, Ki e Kd inseridos pelo usuário e calcular a saída do PID.

Por sua vez, se for escolhido o Controle Fuzzy, a entrada do sistema irá passar pelo processo de Fuzzificação e serão gerados Conjuntos Fuzzy. Com a ajuda das Regras, o Motor de Inferência irá gerar Conjuntos Fuzzy de saída, que configuram a reação do sistema a partir da entrada. Estes Conjuntos Fuzzy de saída passam pelo processo de Defuzzificação e é obtida a saída do Controle Fuzzy.

No Diagrama de Atividades do Decisor, encontrado na Figura 4.3.2, temos que ele deve sempre acumular os erros obtidos da Visão Computacional e sempre perceber se o erro atual não está ultrapassando o limite estabelecido que diz até qual ponto os atuais parâmetros de calibragem funcionam. Caso o erro não esteja ultrapassando o limite, o programa continua. Caso contrário, o erro é transformado em índice da Tabela de Sintonia e um novo conjunto de parâmetros de calibragem é escolhido.

Para o Diagrama de Atividades do PID Adaptativo, na Figura 4.3.3, temos um modelo extremamente semelhante ao PID padrão. Ele é inicializado com um Kp, um Ki e um Kd. Com isso, são realizados os cálculos ganhos proporcionais, integrativos e derivativos, são somados os três e enviados o resultado para a saída.

Pode-se dizer que este diagrama é a base para o entendimento do que será o Sistema de Controle implementado no AGV. Observa-se que nele há um ponto inicial onde são lidas as entradas e analisado qual técnica de controle operará, já prevendo um sistema controlado por uma interface externa e com a possibilidade de ser escalável para quantas técnicas de controle forem desejadas. Assim sendo, o fluxo de execução prossegue para a técnica desejada pelo usuário, realizando suas etapas mais específicas até culminar no envio da saída para os motores.

A seguir, serão exibidos os Diagramas de Caso de Uso e sua descrição. Para o Sistema de Controle, ele é extremamente simples, não dizendo muito sobre o projeto em si. As causas disso serão explicadas na seção seguinte.

Figura 4.3.1 – Diagrama de Atividades do Projeto

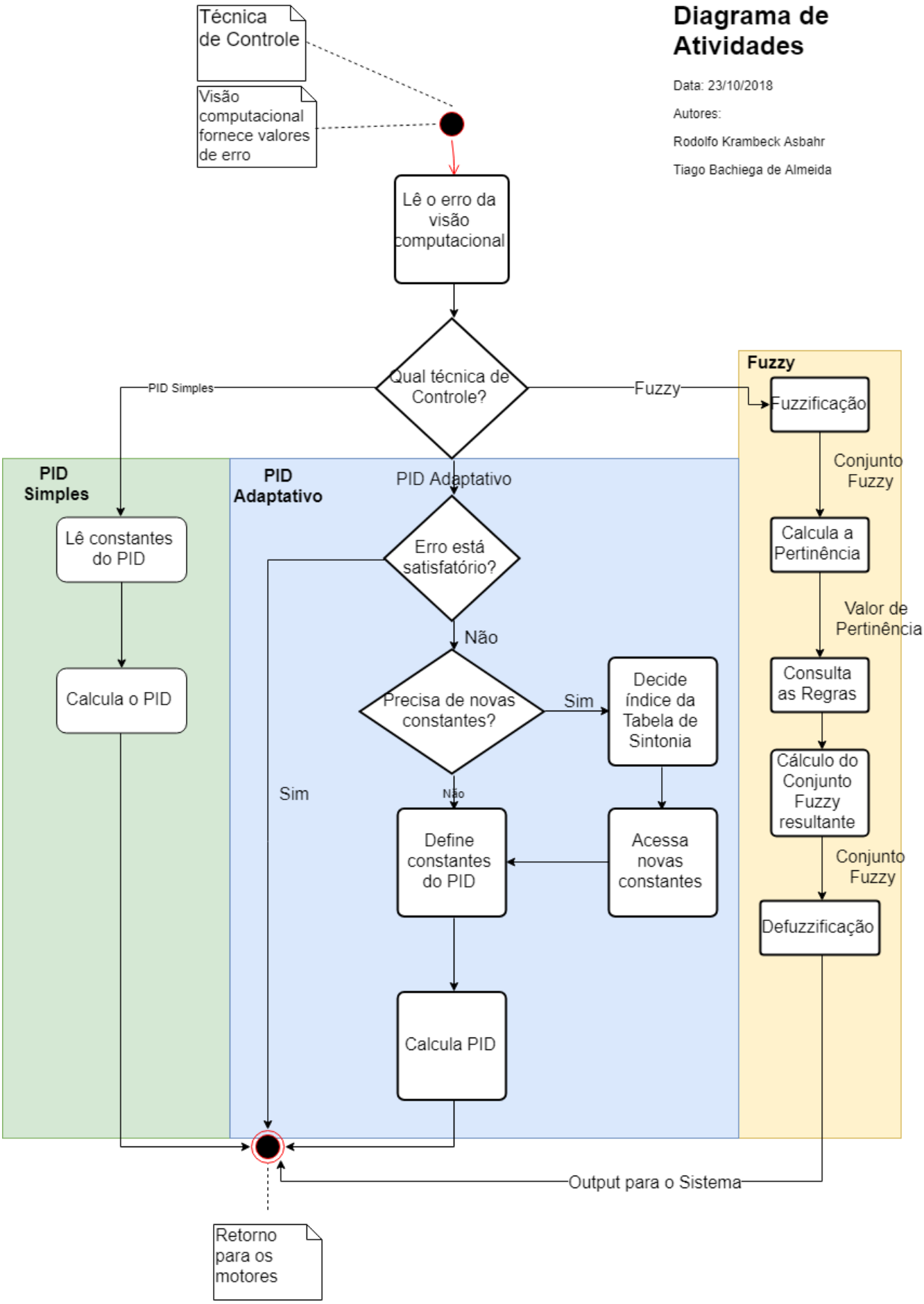


Figura 4.3.2 – Diagrama de Atividades do Decisor

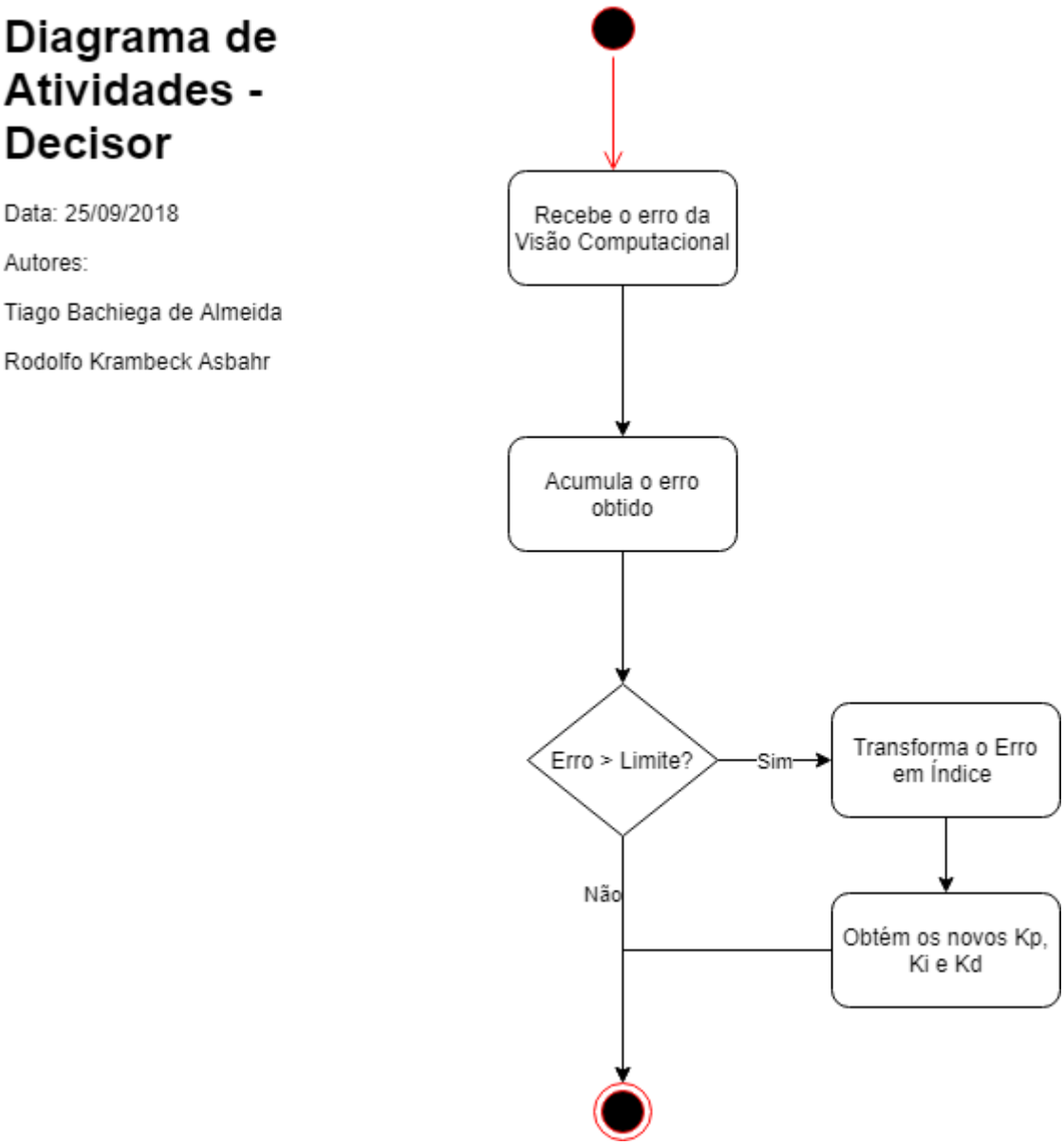


Figura 4.3.3 – Diagrama de Atividades do PID Adaptativo

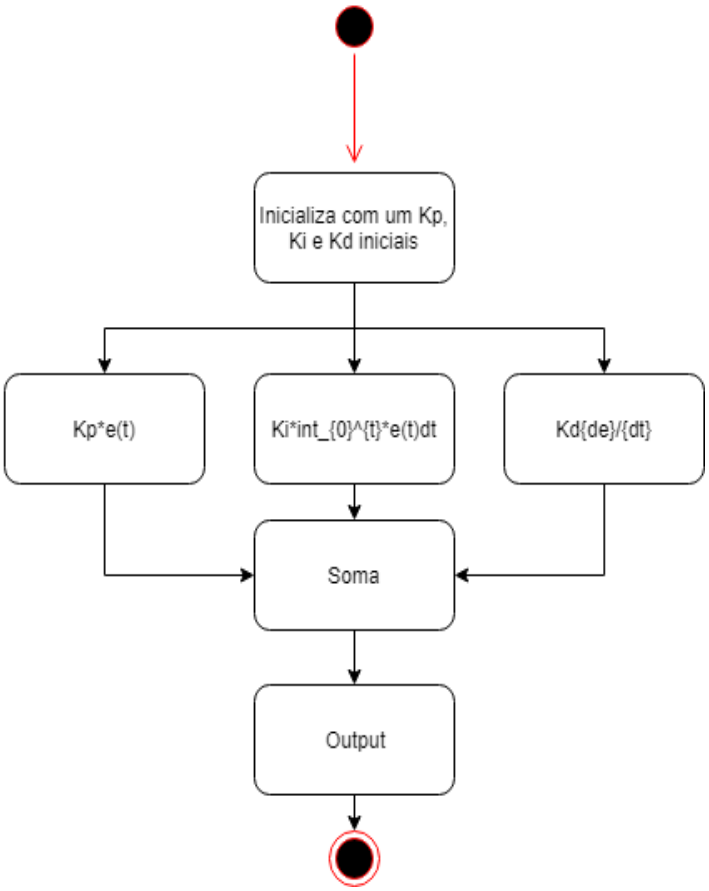
Diagrama de Atividades - PID

Data: 25/09/2018

Autores:

Tiago Bachiega de Almeida

Rodolfo Krambeck Asbahr



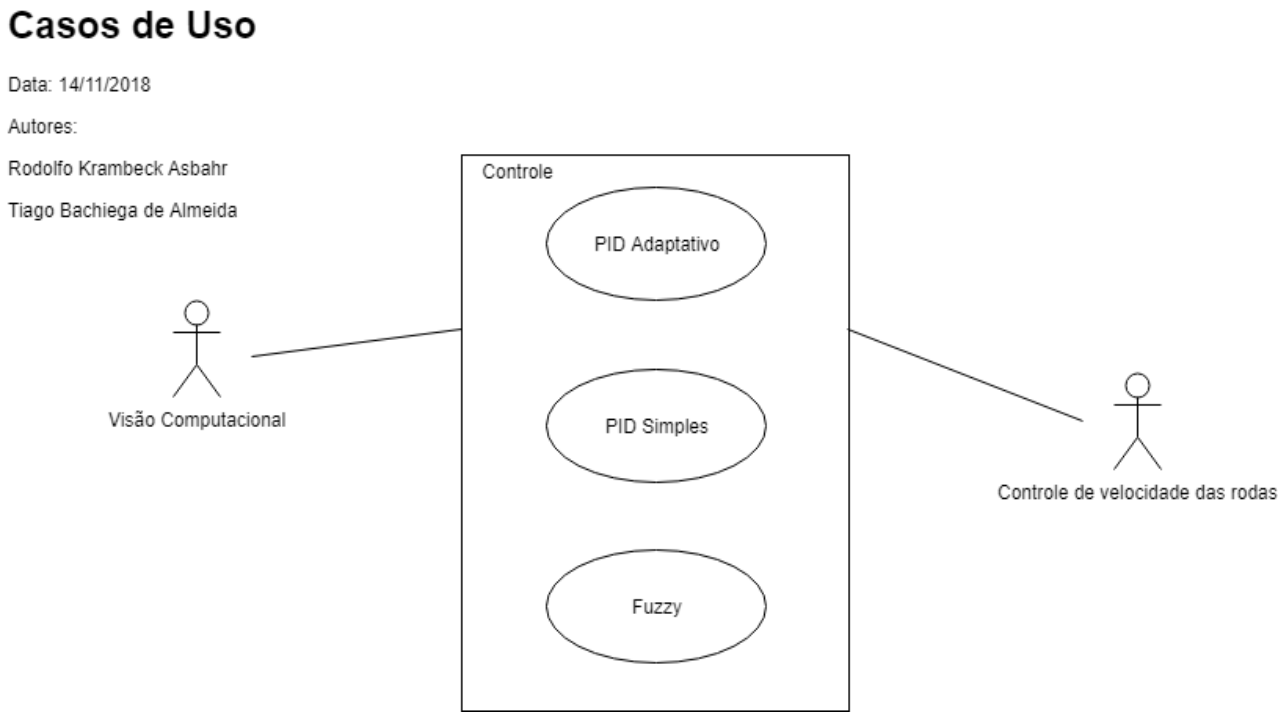
4.4 Diagramas de Casos de Uso

- O que este Diagrama indica e como ele foi adaptado para o contexto?
- Por que ele falha em descrever melhor o sistema?

Este Diagrama tem como objetivo especificar os usuários do sistema e as iterações deles com os componentes existentes. Como no sistema em questão os usuários acabam sendo outros módulos do próprio sistema, fizemos uma versão adaptada para o Diagrama de Casos de Uso, que pode ser visto na Figura 4.4.1. Ele apenas especifica que o Ator Visão Computacional irá interagir com o bloco do software de Controle, que contém o PID Simples, o PID Adaptativo e o Controle Fuzzy. Por sua vez, o Ator Controle de Velocidade das Rodas irá também interagir com estes blocos, pegando suas saídas.

A utilidade primária deste tipo de diagrama é explicitar como os Atores de um sistema interagem com seus componentes. No caso do Sistema de Controle, tanto os Atores quanto os componentes são extremamente simples e triviais. Assim, o diagrama não dá muitos detalhes a mais do que já foi explicitado, servindo apenas para registro de projeto.

Figura 4.4.1 – Diagrama de Casos de Uso



4.5 Descrições de Casos de Uso

- Qual é o fluxo principal e os alternativos do PID Simples?
- Qual é o fluxo principal e os alternativos do PID Adaptativo?
- Qual é o fluxo principal e os alternativos do Controle Fuzzy?

Caso de Uso (PID)

Ator Principal: Visão Computacional

Visão Geral:

O sistema receberá o valor do erro da Visão Computacional e o Kp, Ki e Kd informados pelo usuário. Com isso, irá calcular o PID e retornará o resultado de ganho obtido.

Pré-Condição:

A Visão Computacional deve fornecer um valor para o erro.

Pós-Condição:

Um valor deve ser enviado para as rodas do sistema.

Cenário de Sucesso Principal:

1. Escolhe como técnica o PID Simples.
2. Recebe o erro da Visão Computacional.
4. Recebe as constantes do cálculo do PID.
5. Calcula o PID.
6. Envia o retorno do PID para o controle de velocidade das rodas.

Fluxo Alternativo:

Não existem fluxos alternativos.

Caso de Uso (PID Adaptativo)

Ator Principal: Visão Computacional

Visão Geral:

O sistema receberá o valor do erro da Visão Computacional, verificará se está dentro dos limites desejados, assim irá conseguir calcular um índice para a Tabela de Sintonia e recuperará os valores adequados das constantes Kp, Ki e Kd. Por fim, calculará o PID com essas constantes obtidas e retornará o resultado para o controle de velocidade das rodas do sistema.

Pré-Condição:

A Visão Computacional deve fornecer um valor para o erro.

Pós-Condição:

Um valor deve ser enviado para as rodas do sistema.

Cenário de Sucesso Principal:

1. Escolhe como técnica o PID Adaptativo.
2. Recebe o erro da Visão Computacional.
3. Verifica se o erro está dentro dos limites desejados.
4. Calcula qual o índice de acesso da Tabela de Sintonia.
5. Acessa e recupera as constantes da Tabela de Sintonia.
6. Atualiza as constantes do cálculo do PID.
7. Calcula o PID.
8. Envia o retorno do PID para o controle de velocidade das rodas.

Fluxo Alternativo:

4a. Se o erro estiver dentro dos limites aceitáveis, pula para a etapa 7 com as constantes atuais

Caso de Uso(Controle Fuzzy)

Ator Principal: Visão Computacional

Visão Geral:

O sistema receberá o valor do erro da Visão Computacional e utilizará a Técnica Fuzzy para calcular os valores a serem enviados para o controle de velocidade das rodas.

Pré-Condição:

A Visão Computacional deve fornecer um valor para o erro.

Pós-Condição:

Um valor deve ser enviado para as rodas do sistema.

Cenário de Sucesso Principal:

1. Escolhe como técnica o Controle Fuzzy.
2. Recebe o erro da Visão Computacional.
3. O erro da Visão Computacional passa pelo processo de Fuzzificação.
4. O Conjunto Fuzzy gerado é enviado para o Motor de Inferência.
5. O Motor de Inferência infere um Conjunto Fuzzy de saída com base nas Regras.
6. O Conjunto Fuzzy de saída passa pelo processo de Defuzzificação.
7. Envia o retorno do Controle Fuzzy para o controle de velocidade das rodas.

Fluxos Alternativos:

5a. O Motor de Inferência pode não encontrar nenhuma regra, então retorna ao passo 1.

5b. O Motor de Inferência encontra mais de uma regra, então retorna ao passo 1.

4.6 Diagrama de sequência

- Como este Diagrama se relaciona com o Diagrama de Casos de Uso e o que ele mostra?
- O que o trecho do PID Simples indica?
- O que o trecho do PID Adaptativo indica?
- O que o trecho do Controle Fuzzy indica?
- Qual é a utilidade prática deste diagrama?
- O que virá a seguir?

O Diagrama de Sequência, presente nas Figuras 4.6.1 e 4.6.2, explicita as ações feitas em um caso de uso pelo sistema através do tempo. Nele é mostrado o que cada componente realiza, explicitando as entradas, saídas e como ele se encaixa na sequência de processamento do controle.

Caso o sistema escolhido seja o PID Simples, o sistema simplesmente deve ler o erro de saída da Visão Computacional, executar o PID com parâmetros de sintonia pré-definidos e enviar a saída para o controlador dos motores.

No caso da escolha feita for de usar o PID Adaptativo, será lido o erro da Visão Computacional e o Decisor verificará se é necessário mudar as constantes do PID. Se for necessário, calculará, através do erro, o índice de acesso à Tabela de Sintonia. Em seguida, a Tabela será acessada e os valores Kp, Ki e Kd serão recuperados e utilizados no cálculo do PID do controle. Após o cálculo, o valor do PID calculado será passado para o Controle de Velocidade das Rodas.

No caso da escolha ser o Controle Fuzzy, a saída da Visão Computacional irá passar pela etapa de Fuzzificação e será transformada em um Conjunto Fuzzy. Com isso, será feito o Cálculo da Pertinência, que consiste no Motor de Inferência consultar as Regras e gerar um Conjunto Fuzzy de saída. Este Conjunto Fuzzy de saída passará pela Defuzzificação e será transformado num valor escalar, que será enviado para o controlador das rodas.

Na prática, este diagrama consegue mostrar bem o que cada elemento do sistema utilizará dos elementos anteriores a ele na linha do tempo ou enviará para os que vierem a seguir no que diz respeito a parâmetros. Ele serve muito bem para mostrar como o sistema funcionará durante o tempo e do que cada parte dele necessitará ou gerará no seu intervalo de execução.

A seguir, serão apresentados os Diagramas de Classe. Agora que foi demonstrado como o sistema como um todo funcionará a partir de uma visão de mais alto nível, será necessário descer um pouco o nível, exibindo mais detalhes, pois está sendo alcançado o momento de implementação das técnicas de controle planejadas. Assim, se faz necessária a especificação de seus parâmetros e funções, além de como os componentes descritos anteriormente serão traduzidos em classes.

Figura 4.6.1 – Diagrama de Sequência – Primeira Parte

Diagrama de Sequência

Data: 14/11/2018

Autores:

Rodolfo Krambeck Asbahr

Tiago Bachiega de Almeida

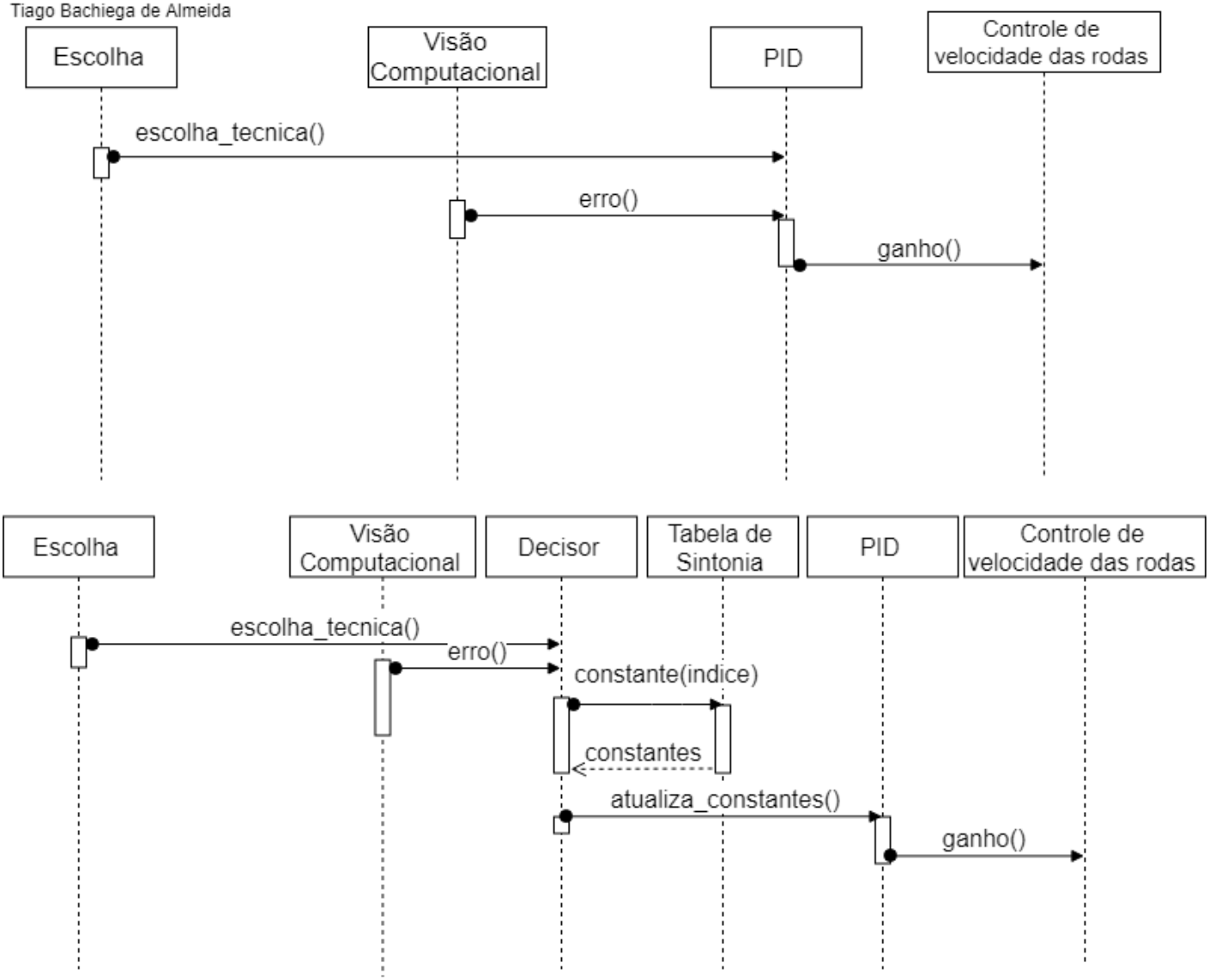
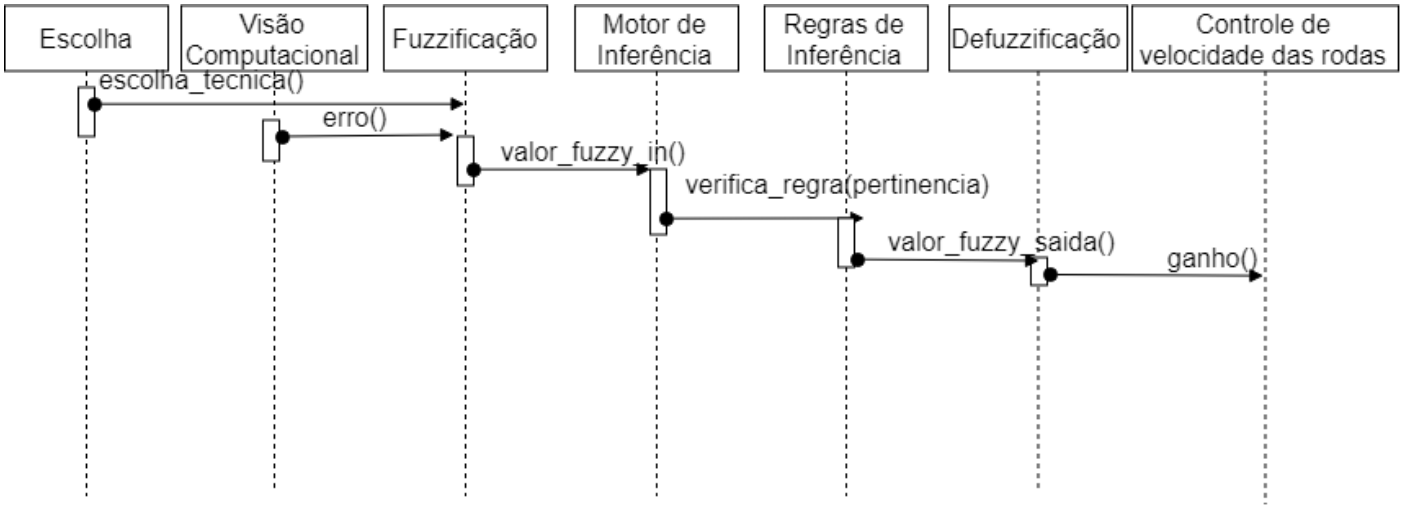


Figura 4.6.1 – Diagrama de Sequência – Segunda Parte



4.7 Diagrama de Classes

- O que este diagrama indica e quais são seus principais componentes?
- Como este diagrama pode ser usado para entender o projeto?
- O que existe dentro da classe Decisor?
- O que existe dentro da classe Manipulador da Tabela?
- O que existe dentro da classe PID Simples?
- O que é a Tabela de Sintonia?
- O que é a classe Fuzzy?

Este Diagrama tem como objetivo mostrar quais são as classes do projeto e o que compõe as classes, quais são seus métodos e atributos. No diagrama presente nas Figuras 4.7.1 e 4.7.2, existem três classes principais, o Decisor, o Manipulador da Tabela e o PID Simples. Também foi incluída a classe Fuzzy embora ela não interaja com as classes anteriores, só consigo mesma, visto que se trata de um processamento extremamente procedural.

O Diagrama de Classes deve registrar exatamente quais atributos, funções e parâmetros existirão para cada classe do sistema. Ele é muito útil para a implementação de sistemas orientados a objetos, que é mais o caso do PID Adaptativo visto que o Controle Fuzzy possui só uma classe, sendo totalmente procedural. Com este diagrama fica muito mais intuitivo o que deve ser programado nas etapas futuras, onde será feita a implementação efetiva do Sistema de Controle.

Dentro da classe Decisor há os seguintes atributos:

- *erros*: que armazena uma quantidade de amostra de retornos da Visão Computacional para cálculos estatísticos.
- *amostra:* define quantas amostras de retornos de erros de ângulo da Visão Computacional devem ser obtidas para o cálculo estatístico.
- *constantes:* que armazena o atual Kp, Ki e Kd obtidos da Tabela de Sintonia.
- *idx:* que armazena o atual índice da Tabela de Sintonia.
- *limite_media:* que estabelece um limite máximo para a média dos erros a partir do qual precisa-se calcular outro Kp, Ki e Kd.
- *limite_desvio*: que estabelece um limite máximo para o desvio padrão a partir do qual precisa-se calcular outro Kp, Ki e Kd;
- *media:* que armazena a média.
- *desvio_padrao:* que armazena o desvio padrão.

Sobre as funções do Decisor, além de *setters* e *getters padrões,* tem as mais importantes como:

- *add_erro:* que armazena novos valores de erro de ângulo à lista *erros*.
- *computa_index:* que usando análise da média e do desvio padrão calcula um novo índice da Tabela de Sintonia.
- *get_constantes*: que lê os valores do índice atual da Tabela de Sintonia para a lista *constantes.*

A classe Manipulador da Tabela não contém parâmetros. Apesar disso, ela contém adaptações de *setters* e *getters* para a atualização e leitura do arquivo CSV que constitui a Tabela de Sintonia.

A classe PID Simples possui como atributos os seguintes:

- *kp, ki* e *kd:* os parâmetros de sintonia do PID.
- *sample_time:* sendo a amostragem de tempo usada para o cálculo da integral.
- *current_time*: que armazena o valor de tempo de execução atual.
- *last_time:* que armazena o último valor de tempo calculado.
- *windup_guard*: um parâmetro usado no cálculo da integral.

Entre suas funções, além dos *setters e getters,* as principais são:

- *update*: que efetivamente calcula o PID.
- *getOutput:* que retorna o valor de saída do PID.

Para poupar tempo, o PID Simples foi retirado de um repositório online, encontrado em [6]. Assim, o grupo pôde focar na implementação do Decisor e do Manipulador de Tabela, mais específicos ao PID Adaptativo.

Para a classe Fuzzy, embora o grupo ainda não saiba se realmente será feita a implementação, foram explicitadas algumas classes e parâmetros básicos. Os parâmetros são os seguintes:

- *input:* o valor da Visão Computacional.
- *fuzzy_in*: valor de entrada do processo de fuzzificação. Talvez não seja exatamente o valor da Visão Computacional, visto que ele pode precisar de algum pré-processamento.
- *fuzzy_out:* valor de saída do processo de fuzzificação. É um Conjunto Fuzzy.
- *defuzzy_out:* valor de saída do sistema de controle fuzzy, após o processo de defuzzificação.
- *pesos:* valores de peso que serão utilizados sobre os valores para calcular a transformação no processo de fuzzificação ou defuzzificação.

Entre as principais funções, além dos *setters* e *getters* padrão, tem-se as listas a seguir. Nota-se que muitas retornam um valor do tipo *float*, isso foi feito desta forma devido a implementações exemplos que o grupo obteve de artigos, em especial devido ao artigo[9]. Pode ser futuramente estes tipos sejam alterados por algum mais condizente com o cenário do projeto.

- *fuzzify:* realiza o processo de fuzzificação.
- *defuzzify:* realiza o processo de defuzzificação.
- *computa_inferencia:* relaciona o Conjuto Fuzzy de entrada com as Regras e produz um Conjunto Fuzzy de saída.
- *regra:* armazena a lógica que compõe as regras da Lógica Fuzzy.

Figura 4.7.1 – Diagrama de Classes do Sistemas – Primeira Parte

Diagrama de Classes

Data: 23/10/2018

Autores:

Tiago Bachiega de Almeida

Rodolfo Krambeck Asbahr

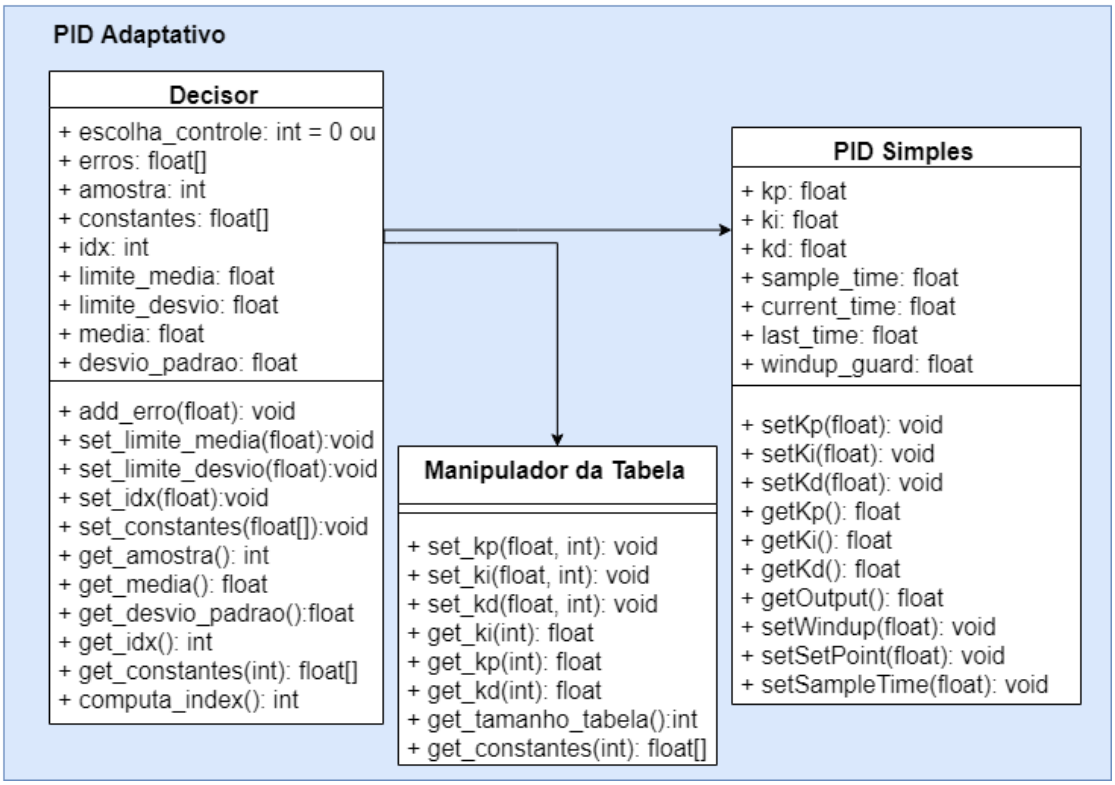
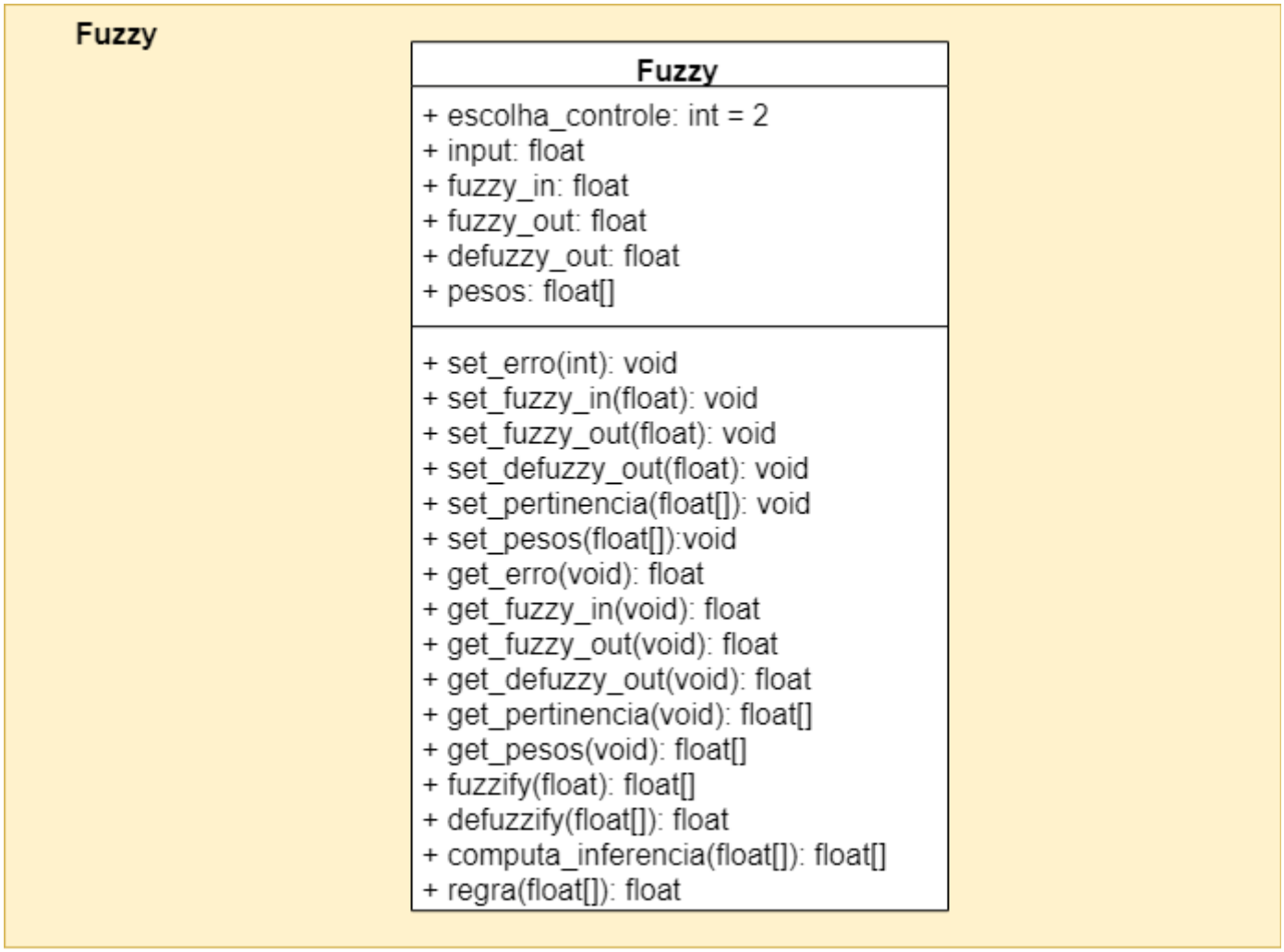


Figura 4.7.2 – Diagrama de Classes do Sistema- Segunda Parte



5. DESENOLVIMENTO

5.1 ESTRATÉGIAS UTILIZADAS

- Qual linguagem de programação foi utilizada e por quê?
- O que foi implementado?
- Como o código foi testado?

A ideia inicial era implementar o sistema em C#, assim ele conseguiria se comunicar com mais facilidade com Supervisório. Porém, devido a limitações do laboratório de desenvolvimento, utilizamos Python. Apesar de não ter sido a primeira opção, como o sistema rodará em uma RaspberryPi com Linux, é uma boa opção também.

Primeiro, foram implementadas todas as classes referentes ao PID Adaptativo, e consequentemente ao PID Simples, que está contido na sua versão adaptativa. Assim, tem-se as classes Decisor, que computa qual índice da Tabela de Sintonia será acessado a partir da análise estatística dos valores da Visão Computacional; o Manipulador de Tabela, que escreve e lê do arquivo CSV da Tabela de Sintonia, que será melhor explicada posteriormente; e o PID Simples, que calcula um PID a partir de parâmetros Kp, Ki e Kd. Também há o código PID Adaptativo, que realiza a união das classes descritas a fim de realizar o sistema adaptativo de controle. Além deles, há ainda o código de Controle Geral, que é responsável por executar as diferentes técnicas de controle implementadas. A ideia é que o Supervisório converse com este código e ele execute o controle com base no que foi ordenado. Além disso, também serão feitas funções que podem ser utilizadas para editar os parâmetros do sistema de controle através da interface Supervisória.

Para testar o sistema do PID Adaptativo, utilizamos uma Planta de Primeira Ordem Discretizada pelo Método de Tustin[4][5]. Essa estratégia permite que o grupo tenha uma melhor visualização de como está ocorrendo a realimentação do sistema a partir dos valores do PID Adaptativo. Para simular a entrada da Visão Computacional, foram utilizados números aleatórios de -6 a 6, emulando inclinação tanto para a esquerda quanto para a direita da linha percorrida pelo AGV.

5.2 DECISOR

- O que é a classe Decisor?
- Como o Decisor escolhe o índice da Tabela de Sintonia?
- O que a média e o desvio padrão nos mostram?
- Como isso foi implementado no código com busca sucessiva?

Como dito anteriormente, o Decisor é o responsável por verificar se a média e o desvio padrão dos valores de entrada vindos da Visão Computacional estão dentro dos limites desejados para o atual índice da Tabela de Sintonia. Em outras palavras, verificar se o Kp, o Ki e o Kd atuais estão satisfatórios. Caso não, ele decide entre a próxima linha da tabela, com uma sintonia mais brusca, ou a anterior, com uma sintnia mais moderada. A Figura 5.2.1 descreve o Algoritmo Geral do Decisor. Embora ele seja uma classe, descrevemos um algoritmo contemplando o que suas principais funções farão no código.

Para escolher o índice da Tabela de Sintonia, inicializamos o controle pela primeira linha da tabela, de índice 1 (o por quê de não ser o índice 0 será explicado posteriormente). Assim, o Decisor irá utilizar sua função computa_index(), para verificar como está a média e o desvio padrão dos n valores da Visão Computacional acumulados, sendo n o número de amostras necessárias para se fazer esta operação, decidindo se deve incrementar decrementar o índice para a Tabela de Sintonia.

Podemos ter uma média pequena ou grande. Uma média pequena vai pode nos dizer duas coisas: 1. o AGV está oscilando para ambos os lados na mesma taxa de oscilação (o que pode ser muito ruim se a taxa for grande) ou 2. o AGV está virando para algum dos lados. O desvio padrão poderá nos dizer o quanto dispersão dos valores está sendo em relação à média. Ele vai ser muito útil porque podemos identificar os

dois casos de erro do PID: quando ele está oscilando muito, mas com a mesma taxa de oscilação (média pequena e desvio grande), que devemos corrigir pois não queremos um movimento de zigue-zague; e quando ele está tentando pra algum dos lados, se desviando da linha, mas a taxa de correção está muito branda (média grande e desvio pequeno), pois não queremos que ele demore para virar para a direção certa.

A Figura 5.2.2 exibe a principal função do Decisor, *computa_index()*. Utilizando biblioteca *numpy*, inicialmente calculamos a média e o desvio padrão dos valores do *array erros*. A seguir, verificamos as condições de média e desvio padrão acima descritas com base em valores limites inseridos manualmente. Caso haja uma média pequena e um desvio padrão grande, entendemos que é devido aos parâmetros de correção estarem sendo muito agressivos, fazendo o AGV girar muito para ser corrigido, então decrementamos o índice para um jogo de constantes mais brando. Caso contrário, entendemos que é necessária uma maior agressividade na correção e incrementamos o índice, exigindo valores de constantes mais agressivos. Este tipo de correção é uma busca sucessiva (rampa) na tabela, mas poderia ser implementado de diversas outras formas, como através de aproximações sucessivas (ponto médio) onde, ao invés de se incrementar ou decrementar os valores, busca-se em posições mais extremas da tabela até o índice convergir para um ponto médio entre estas posições extremas que tenha constantes mais ideais.

O código completo do Decisor encontra-se no Anexo 2.

Figura 5.2.1 – Algoritmo Geral do Decisor

```
Decisor:
        Lê da Visão Computacional
        Gera uma entrada para o Decisor
        Acumula o valor de entrada

        Se valor de entrada acumulados > amostra:
                Verifica a média dos valores de entrada
                Verifica o desvio padrão dos valores de entrada

                Caso média pequena e desvio grande:
                        Diminui a agressividade da correção
                Caso média grande e desvio pequeno:
                        Aumenta a agressividade da correção
                Caso nenhum dos anteriores:
                        Mantém a agressividade da correção

                Obtém os valores da Tabela de Sintonização
                Envia os novos Kp, Ki e Kd
```

Figura 5.2.2 – Função *computa_index()*

```
#Usa media e desvio padrao para calcular o indice da Tabela
def computa_index(self):
    #calcula a media
    self.media = np.mean(self.erros)
    #calcula o desvio padrao
    self.desvio_padrao = np.std(self.erros)
    #calcula o valor absoluto da media
    media_abs = abs(self.media)

    #Estamos realizando uma busca sucessiva (rampa).
    #Existe a alternativa de busca de aproximacao sucessiva (ponto medio)
    #verifica as condicoes e edita o indice
    if media_abs < self.limite_media and self.desvio_padrao > self.limite_desvio:
        self.idx = self.idx - 1
    elif media_abs > self.limite_media and self.desvio_padrao < self.limite_desvio:
        self.idx = self.idx + 1

    return self.idx
```

5.3 Manipulador da Tabela e Tabela de Sintonia

- O que é o Manipulador de Tabela e como ele funciona?
- O que é a Tabela de Sintonia e como ela está organizada?

O Manipulador de Tabela nada mais é que uma classe com funções que vão fazer a interface para a Tabela de Sintonia, que é um arquivo *CSV*. Para tanto, usamos a biblioteca de python chamada *csv*, que possui funcionalidades específicas para lidar com este tipo de arquivo. Esta classe possui apenas *getters* e *setters* adaptados a ler e escrever no atributo *tabela* desta classe, que é uma lista. Note que para escrever esta lista em um arquivo *.csv* de fato, deve-se usar a função *escreve_tabela()*, assim como para ler deve-se usar a função *le_tabela()*. A Figura 5.3.1 mostra um exemplo de *setter* e as Figuras 5.3.2 e 5.3.3 mostram dois exemplos de *getter* desta classe.

Figura 5.3.1 – Exemplo de *setter*

```
def set_kp(self, kp, idx):
    #Nao permite tentar acesso a posicao inexistente
    if idx >= len(self.tabela):
        print("Nao existe elemento nesta posicao da tabela")
    #Nao permite a edicao do primeiro e ultimo elementos pois sao elementos de controle
            elif self.tabela[idx][0] != 0.0 and self.tabela[idx][0] != float('Inf'):
                    #edita a o parametro
                    self.tabela[idx][0] = kp
    else:
        print("Voce esta tentando editar uma posicao da tabela que nao existe!")
```

Figura 5.3.2 – Exemplo de *getter* de parâmetro

```
def get_kp(self, idx):
    #le o elemento
    kp = self.tabela[idx][0]
    return  kp
```

Figura 5.3.3 – Exemplo de *getter* da lista de constantes

```
def get_constantes(self, idx):
    #cria uma lista de constantes
    constantes=[]
    #adiciona os parametros na lista
    constantes.append(self.get_kp(idx))
    constantes.append(self.get_ki(idx))
    constantes.append(self.get_kd(idx))
    return constantes
```

A Tabela de Sintonia é um arquivo *CSV* que possui *n* linhas e 3 colunas, cada linha sendo um jogo de Kp, Ki e Kd e cada coluna representando um destes parâmetros, nesta ordem. A primeira linha deve ter o Kp, o Ki e o Kd com valor 0, servindo apenas para indicação de início da tabela. A última linha deve ter o Kp, o Ki e o Kd com valores INF, representação do Python para infinito, servindo para indicação de fim da tabela. Caso essas duas regras não sejam obedecidas, o programa pode tentar acessar regiões inexistentes do arquivo. A Tabela 5.3.1 mostra um exemplo de Tabela de Sintonia. Note que os valores são imaginários, sem nenhum compromisso com a fidelidade na ordenação de agressividade das correções do PID que utilizar os valores da tabela. Foi feita apenas para mostrar seu formato.

O código completo do Manipulador de Tabela se encontra no Apêndice 1.

Tabela 5.3.1 – Exemplo de Tabela de Sintonia

0	0	0
10.0	0.0	0.2
8.0	0.1	0.1
11.0	0.2	0.3
9.0	0.8	0.3
INF	INF	INF

5.4 PID Simples

- Qual a sua utilidade dentro do PID Adaptativo?
- Como o grupo procedeu para a implementação do PID Simples?

O PID Simples será utilizado dentro do código do PID Adaptativo. Isto acontece porque, na nossa implementação, o PID Adaptativo sempre irá chamar um PID Simples, mas com novos parâmetros de sintonia, se for o caso.

Como já foi implementado um PID Simples para o projeto do AGV, o grupo decidiu não fazer a implementação novamente, então optamos por utilizar o código disponibilizado no repositório disponível em [6]. Assim, foi possível acelerar o desenvolvimento e focar em partes do projeto que ainda não haviam sido desenvolvidas.

5.5 PID Adaptativo

- Em que se consiste o PID Adaptativo?
- Como é o algoritmo?
- Como é o código?

O código do PID Adaptativo consiste em integrar as classes já implementadas, sendo ela o Decisor, o Manipulador da Tabela de Sintonia e o próprio PID. O PID Adaptativo se comunica diretamente com o Decisor, com a leitura da Visão Computacional e o PID. Através do Decisor, o Manipulador é acessado para usar a Tabela de Sintonia.

O algoritmo é apresentado na figura 5.5.1. O PID Adaptativo recebe um valor da Visão Computacional e o envia para o Decisor. Em seguida, se há leituras suficientes, o Decisor tenta atualizar as constantes do PID e, em seguida, o PID Adaptativo calcula o PID novo usando o valor lido junto com um feedback calculado. Esse feedback é atualizado com o novo valor calculado do PID.

Figura 5.5.1 – Algoritmo do PID Adaptativo.

```
PID Adaptativo:
Inicializa variáveis necessárias
Obtém um erro da Visão Computacional
Adiciona o erro ao Decisor

Se o número de amostras é suficiente:
        Verifica se são necessários novos parâmetros do PID
        Obtém os parâmetros
        Atualiza os parâmetros do PID

Calcula o PID com os novos parâmetros
Obtém o output
Coloca o output na planta de teste

Repete tudo
```

As linhas de código do PID Adaptativo não se estendem muito mais que o algoritmo para fazer suas tarefas, como mostrado nas Figuras 5.5.2, 5.5.3, 5.5.4 e 5.5.5. A Figura 5.5.2 apresenta o começo do código, onde o Decisor recebe o valor da Visão Computacional, que para testes é um valor aleatório de -6 a 6.

Figura 5.5.2 – Trecho de adição do valor ao Decisor.

```
last_output = output

time.sleep(sample_time)

#adiciona o erro no decisor
decisor.add_erro(feedback)

#soma uma leitura
qnt_leituras = qnt_leituras + 1
```

A figura 5.5.3 apresenta a parte onde é verificado se foram recebidos valores o suficiente para tentar atualizar as constantes do PID, se foi, o Decisor atualizará as constantes do PID da classe do PID. Devemos

nos lembrar que a classe Decisor sempre armazena os *n* últimos valores da Visão Computacional, sendo *n* o número de amostras. Quando ele entra neste trecho, chama *computa_index()* e atualiza o Kp, o Ki e o Kd.

Figura 5.5.3 – Trecho de atualização das constantes do PID.

```
#caso tenha o numero de amostras, deve verificar
#a performance
if qnt_leituras >= num_amostras:
    qnt_leituras = 0

    #calcula o novo indice se precisar
    index = decisor.computa_index()

    #obtem as constantes e atribui ao PID Simples
    constantes = decisor.get_constantes(manipulador, index)
    PID.setKp(float(constantes[0]))
    PID.setKi(float(constantes[1]))
    PID.setKd(float(constantes[2]))
```

A figura 5.5.4 apresenta o trecho de código do cálculo do PID, quando o código for colocado para uso no AGV, será usado o valor retornado da Visão Computacional no lugar do número inteiro gerado por *np.random.randint(-6,6)*. Esse valor aleatório foi usado para teste apenas, simulando uma discrepância entre o valor de feedback do PID e o que realmente foi lido da Visão Computacional.

Figura 5.5.4 – Trecho de cálculo do PID.

```
#coloca o feedback da planta + a leitura da
#   visao computacional (valor aleatorio)
PID.update(feedback + np.random.randint(-6,6))
#pega o output
output = PID.getOutput()
```

A figura 5.5.5 apresenta a parte do código onde o valor de feedback é atualizado. Para essa atualização, são usados o valor de feedback anterior e o valor de saída do PID. Note que foi utilizada para teste uma planta de primeira ordem discretizada pelo método de Tustin. Ela recebe os valores de saída do PID e simula um efeito em planta do ganho retornado pelo controle. Isso nos permitiu uma maior fidelidade de teste para o sistema.

O código completo do PID Adaptativo com a planta discretizada se encontra no Apêndice 1.

Figura 5.5.5 – Trecho de atualização do valor do feedback.

```
# Atualizacao da planta
# Planta de primeira ordem discretizada pelo metodo de Tustin
last_feedback = feedback
feedback = last_feedback*((2*tau - sample_time)/(2*tau + sample_time))
    +output*((sample_time)/(2*tau + sample_time))+last_output*((sample_time)/(2*tau + sample_time))
```

5.6 Controle Fuzzy

- O que é o método Mamdani?
- O que o MatLab oferece?
- O que o Python oferece?

O método de controle fuzzy mais comum é o Mamdani[10]. Esse método consiste em 4 etapas: a fuzzificação da entrada, onde se obtém um valor nominal e um valor de pertinência para a entrada; a obtenção das conclusões das regras, essa etapa é a aplicação das regras definidas dentro do método; agregação das conclusões de cada regra feita na etapa anterior, para isso são usados operadores de implicação, que são parecidos com os operadores lógicos mas podem ser implementados de formas diferentes; e, por último, a defuzzificação do valor resultante, o valor até então é fuzzy, então para usar no sistema de controle é necessário voltar para ser um valor determinístico, sendo o método de defuzzificação mais utilizado é o de centroide. Caso seja necessário implementar o método, é recomendado estudar como cada etapa é estruturada e calculada.

O MatLab oferece uma toolbox chamada Fuzzy Logic Toolbox. A Figura 5.6.1 mostra a tela inicial da toolbox, onde o método Mamdani já está implementado, se fazendo necessário só modificar valores das funções de pertinência e seus intervalos, como mostrado na Figura 5.6.2. Neste ambiente também é possível editar como são usados os operadores AND, OR e de implicação, assim como o método de agregação das regras e a defuzzificação.

Como pode ser visto na figura 5.6.2, é possível selecionar a função que se quer editar clicando nela, assim ela ficará destacada em vermelha, podendo ser input ou output. Em seguida pode-se mudar seu nome, e os parâmetros que caracterizam como ela é, além de poder escolher se ela é triangular, gaussiana ou outra função conhecida. Nessa mesma janela é possível mudar o intervalo que a função abrange, assim como o intervalo que o gráfico mostra a função.

A figura 5.6.3 mostra a janela das funções de defuzzificação, a janela é organizada da mesma forma que a fuzzificação. A figura 5.6.4 é a janela de regras, onde pode-se criar, editar e apagar as regras do método, sendo possível escolher qual a função de fuzzificação e qual a função de defuzzificação serão utilizadas.

A sequência de imagens 5.6.5, 5.6.6 e 5.6.7 ilustram a visualização da agregação das regras e como cada valor de entrada deixa a agregação.

Figura 5.6.1 – Janela inicial da toolbox.

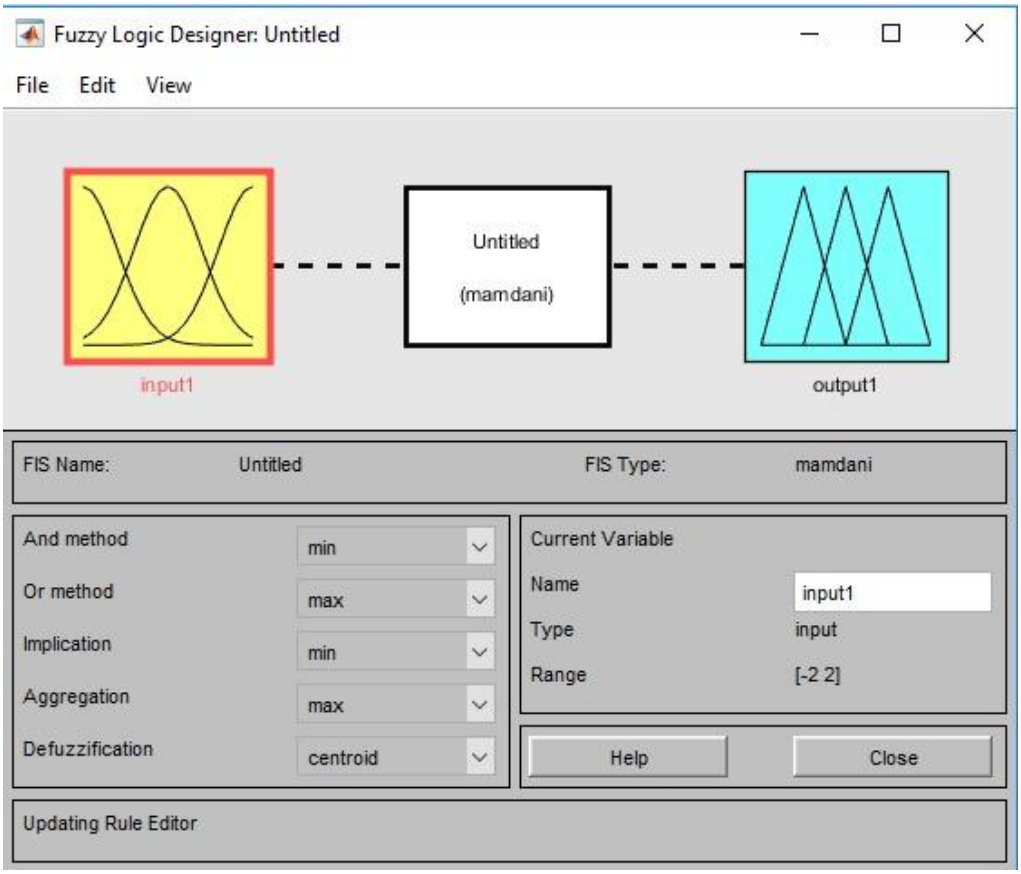


Figura 5.6.2 – Janela das funções de fuzzificação.

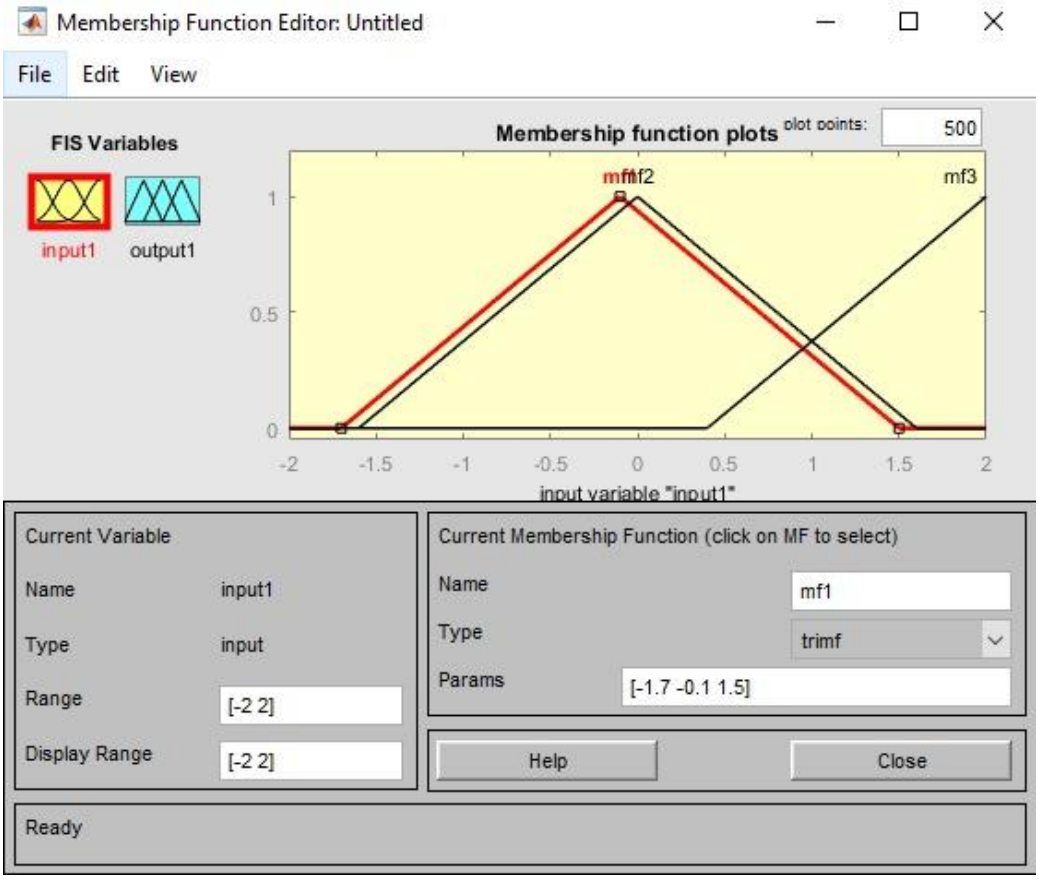


Figura 5.6.3 – Janela das funções de defuzzificação.

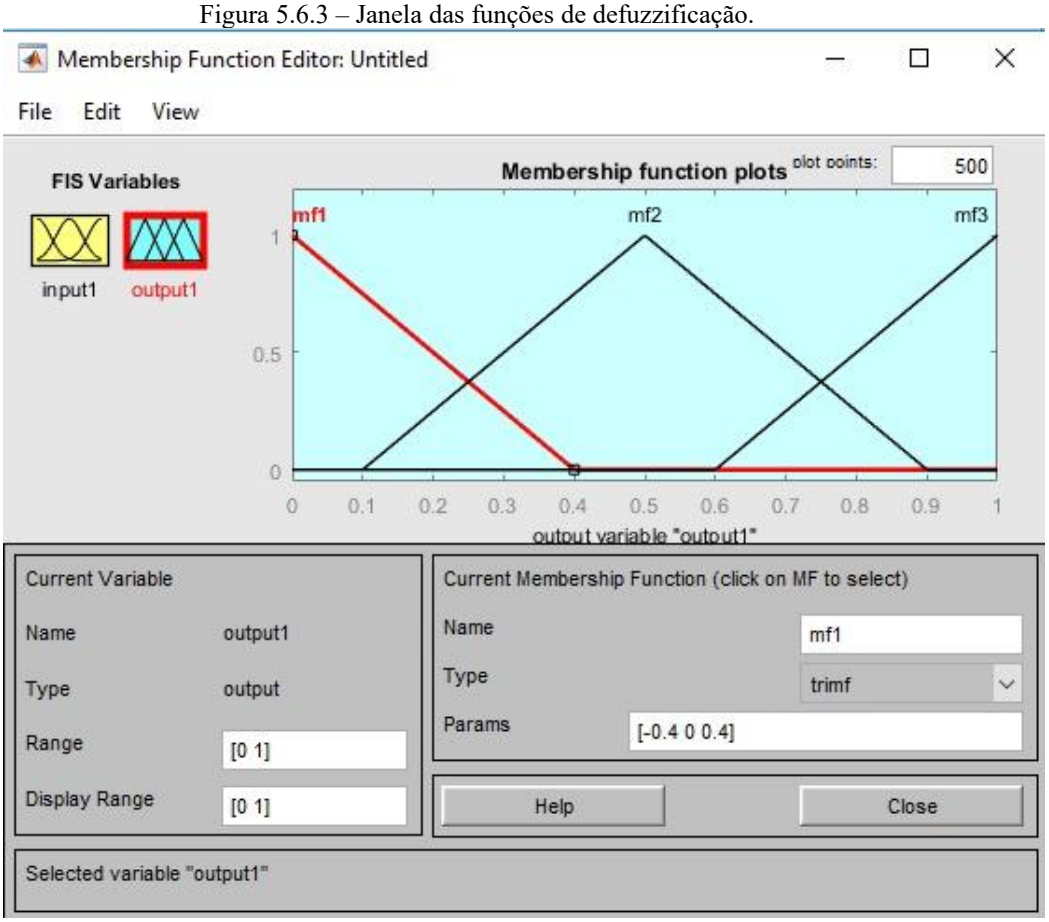


Figura 5.6.4 – Janela das regras do método.

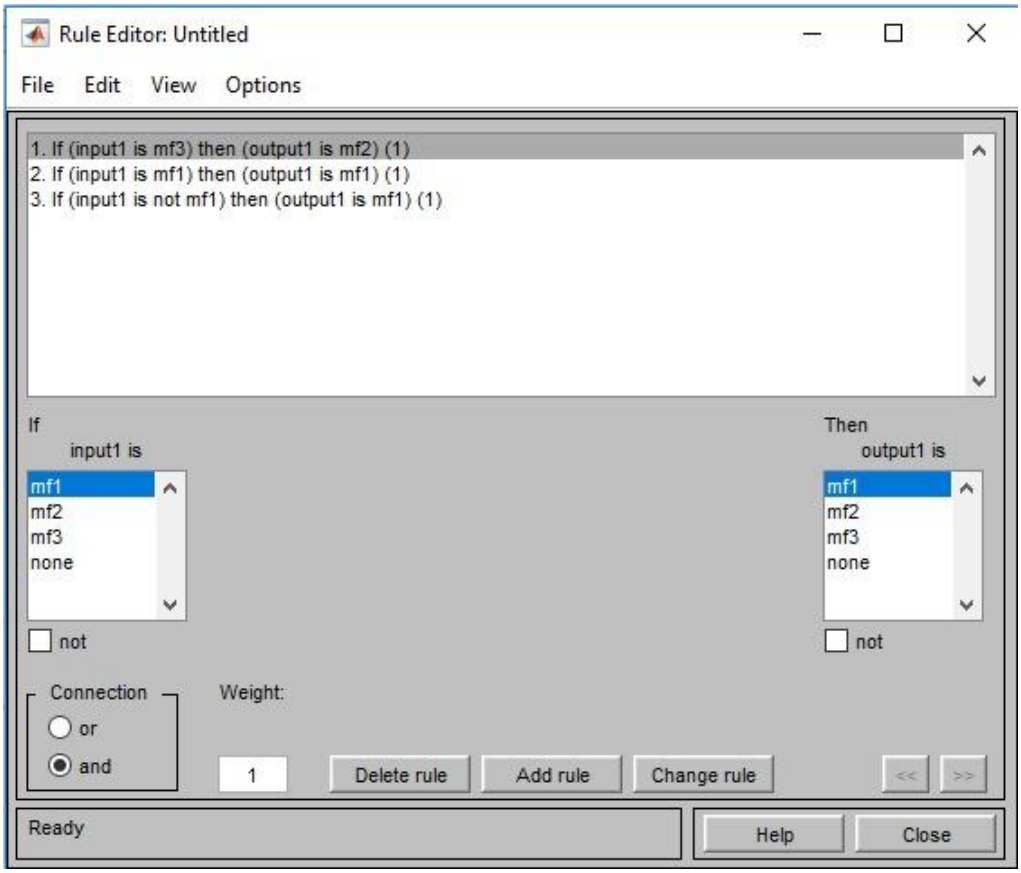


Figura 5.6.5 – Janela da visualização da agregação das regras em um primeiro momento.

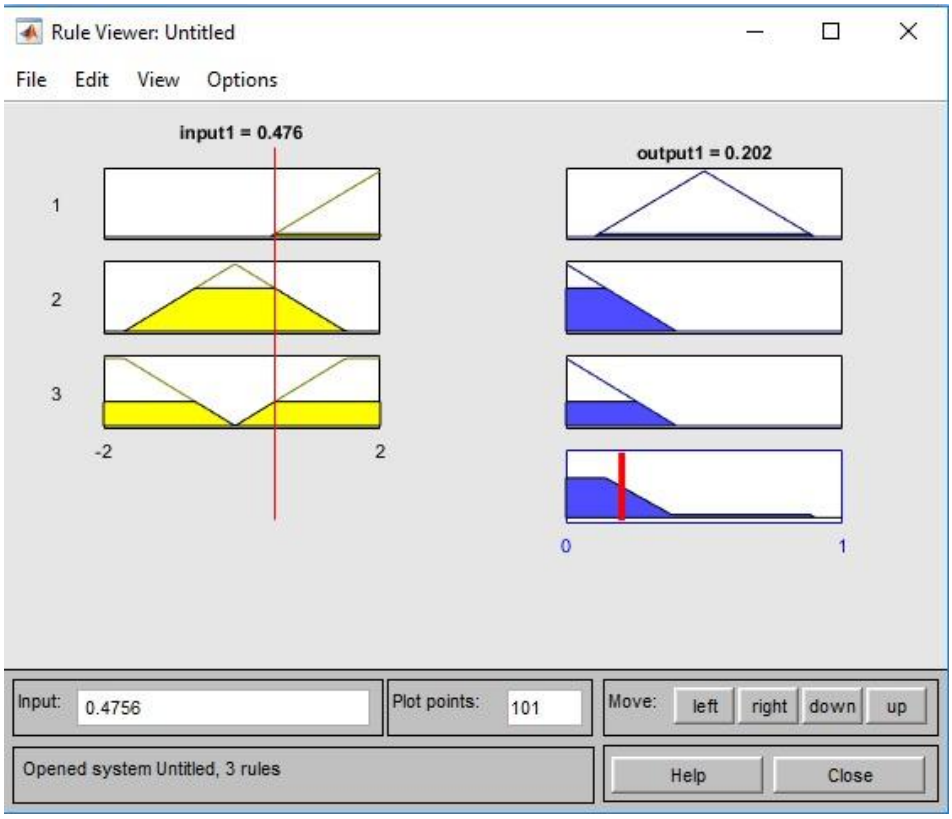


Figura 5.6.6 – Janela da visualização da agregação das regras, depois de arrastar o ponto de entrada.

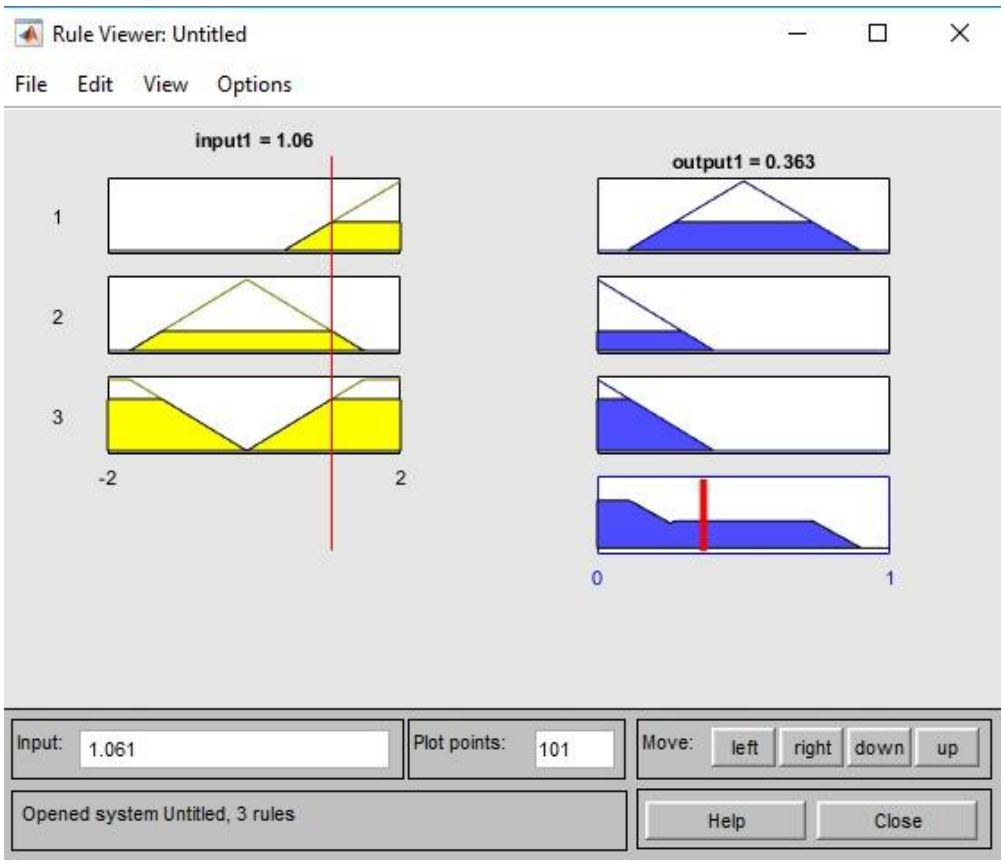
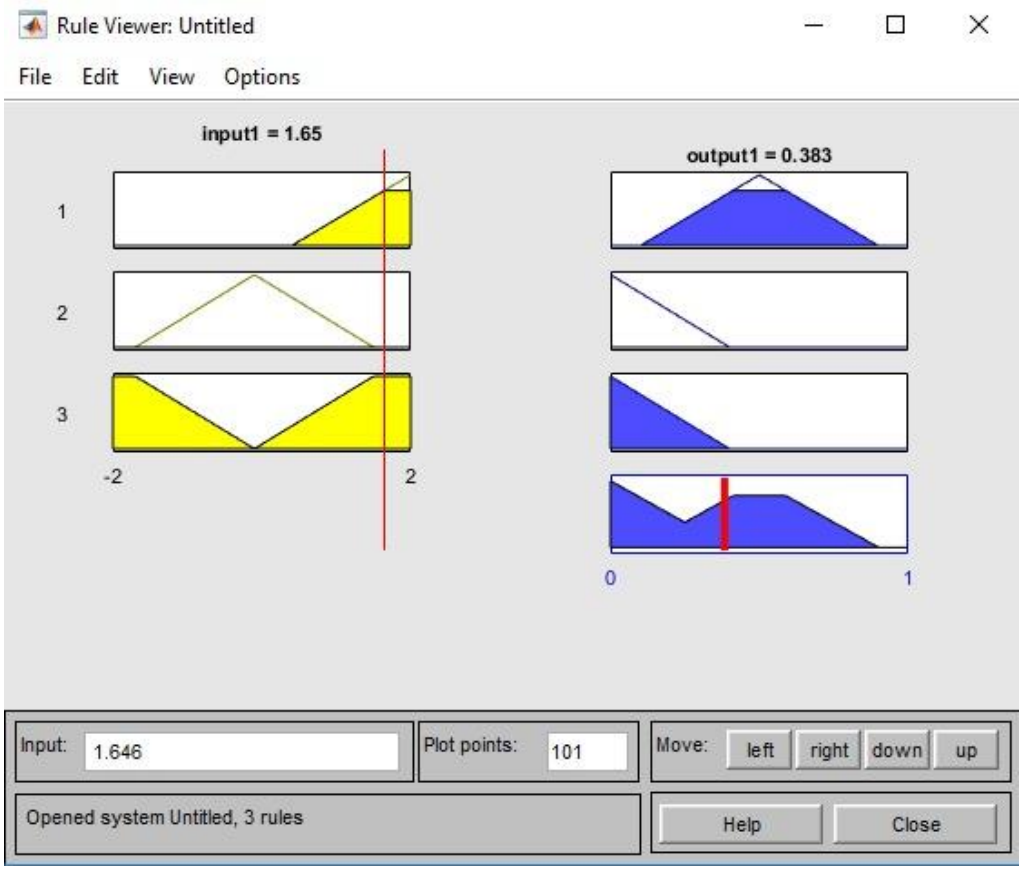


Figura 5.6.7 – Janela da visualização da agregação das regras em momento final.



Devido a dificuldades de instalação e dependências de pacotes, o estudo da biblioteca scikit-fuzzy do Python foi superficial. Só foi constatado que o pacote contém os métodos de defuzzificação, métodos de funções de pertinência e os operadores de agregação e implicação. Mais informações sobre o pacote pode ser encontrado na documentação do pacote, localizado em [11].

5.7 Controle Geral

- O que é o controle geral?
- Como é implementado?

O controle geral é o código que controla qual a forma de controle será utilizado pelo AGV. Essa escolha é feita pelo usuário através da interface com o Supervisório, que também deve configurar os parâmetros condizentes à implementação de controle escolhida. Os parâmetros que o grupo trabalhou para serem editáveis estão explicitados na seção posterior. Através do controle geral, são atualizadas as variáveis parametrizadas para cada controle. O controle geral implementa o que é apresentado na Figura 4.3.1.

A implementação consiste em um laço que continua enquanto não for dada uma ordem de parada. Dentro há a verificação da escolha do controle através de uma variável seletora, *seletorControle*, que pode ser de parada do AGV ou para selecionar o controle a ser executado. A figura 5.7.1 apresenta a forma que foi implementado o código de escolha da forma de controle do AGV.

Nota-se, pelo código a seguir, trechos comentados com TODO. Estes trechos especificam onde o grupo de Controle idealizou a inserção de códigos relacionados ao Supervisório. Nestes trechos, o Supervisório deve configurar o que foi especificado como parâmetros em seu sistema para as respectivas variáveis do controle. Por exemplo, imagina-se que haverão 3 campos para que o usuário insira o Kp, Ki e Kd desejados para o PID Simples. Deve haver neste código o processamento que obtém estes Kp, Ki e Kd da interface e os atribui aos parâmetros do PID Simples.

Observa-se também que o PID Adaptativo foi modularizado para uma função que recebe somente o nome do arquivo *.csv* da Tabela de Sintonia utilizada. Não há grandes diferenças entre o código que está dentro do Controle Geral e aquele usado para teste, da Seção 5.5. Apesar disso, decidiu-se manter o código do PID Adaptativo de testes e o de Controle Geral, que agora pode ser também encontrado no Apêndice 1.

Figura 5.7.1 – Código de escolha de forma de controle.

```
#Se for 0 nao roda nenhum controle
while seletorControle!=0:
    # TODO: Ler o tipo de controle pela interface
    if seletorControle == 1:
        print("Executando o PID Simples")
        # TODO: Ler Kp, Ki e Kd da interface
        # TODO: Ler sample time
        # TODO: Ler SetPoint
        output = PIDSimples(kp, ki, kd)
    if seletorControle == 2:
        print("Executando o PID Adaptativo")
        # TODO: Ler o nome da tabela da interface
        # TODO: Ler numero de amostras
        # TODO: Ler o sample time
        # TODO: Ler o limite de desvio padroa
        # TODO: Ler o limite da media
        # TODO: Ler SetPoint
        # TODO: Ler indice inicial da tabela
        output = PIDAdaptativo(nome_da_tabela)
    if seletorControle == 3:
        print("Executando o Controle Fuzzy")
        output = ControleFuzzy()
    print(output)
return
```

5.8 Interface com o Supervisório

- Quais são as variáveis parametrizadas?
- Como serão definidas pelo usuário?

As variáveis parametrizadas são todas aquelas que não possuem um sistema de escolha automática. Por exemplo, as constantes Kp, Ki e Kd são decididas automaticamente pelo PID Adaptativo, mas o os limites de média e desvio padrão são decididos pelo usuário, nesse estado de implementação atual. Foi feita então uma lista de variáveis parametrizadas pelo usuário no Supervisório elencando qual função permite sua edição. O objeto desta lista é facilitar o desenvolvimento do Supervisório, deixando registrado quais parâmetros precisam ser editáveis pela interface e quais funções utilizar para editá-los.

As variáveis parametrizáveis são as seguintes:

Decisor:

- ❑ Quantidade de amostras: *set_amostra(amostra), get_amostra( )*
- ❑ Limite da média: *set_limite_media(limite_media), get_limite_media( )*
- ❑ Limite do desvio padrão: *set_limite_desvio(desvio_padrao), get_limite_desvio( )*
- ❑ Índice da Tabela (para casos de teste onde queira manipular o comportamento do AGV): *set_index(idx), get_index( )*

Manipulador da Tabela:

- ❑ Nome da Tabela: *set_nome_tabela(nome_da_tabela), get_nome_tabela( )*
- ❑ Inicializar Tabela: *cria_tabela( )*

- Limpar Tabela: *limpa_tabela( )*
- Setar/Adicionar Parâmetros: *cria_tabela( ) se não criada, le_tabela( ), set_kp(kp, idx) ou set_ki(ki, idx) ou set_kd(kd, idx) ou set_parametros(kp, ki, kd, idx) ou add_parametros(kp, ki, kd), escreve_tabela*
- Ler Parâmetros: *le_tabela( ), get_kp(idx) ou get_ki(idx) ou get_kd(idx) ou get_constantes(idx)*
- PID:
- Sample Time: *setSampleTime(sampleTime), getSampleTime( )*
- Kp (para o PID Simples): *setKp(kp), getKp( )*
- Ki (para o PID Simples): *setKi(ki), getKi( )*
- Kd (para o PID Simples): *setKd(kd), getKd( )*
- Setpoint: *setSetPoint(setPoint), getSetPoint( )*
- Controle geral:
- Escolha do método de controle: *0 – Nenhum, 1 – PID Simples, 2 – PID Adaptativo e 3 – Controle Fuzzy*

Atualmente, como não há a interface com o Sistema Supervisório, essas variáveis são inicializadas e mantidas no próprio código do controle geral, como mostrado na figura 5.8.1.

Figura 5.8.1 – Código de escolha de forma de controle.

```
#Parametros inicias
qnt_leituras = 0
num_amostras = decisor.get_amostra() #numero de amostras
decisor.set_limite_desvio(3.5)
decisor.set_limite_media(1)
saida_visao = 5.0 # saida de angulo visao computacional
sample_time  = 0.0
PID.setSetPoint(1.0)
output = 0.0
kp = 1.0
ki = 0.5
kd = 0.2
nome_da_tabela = "tabela_de_sintonia.csv"

#feedback inicial como numero aleatorio entre -6 e 6
#isso vai simular o erro de angulo da visao
saida_visao = np.random.randint(-6,6)
PID.setSampleTime(1.0)
```

Em etapas posteriores do projeto, quando houver a interface como Sistema Supervisório, a atribuição de valores a essas variáveis parametrizadas será feito a partir do Sistema Supervisório que chamará funções de atribuição de valores a essas variáveis. Essa chamada poderá ser feita de várias formas, botões, caixas de texto em que se insere o valor desejado, barra de intervalo de valores, etc. A forma de implementação da interface deverá ser discutida entre os grupos de controle e do Sistema Supervisório para que sejam respeitados as restrições de cada valor. Essas restrições podem ir desde verificações não feitas dentro do código do controle até formas de deixar a manipulação dessas variáveis mais intuitivo para o usuário.

6. Análise dos resultados

- Como está o registro do planejamento do projeto?
- Quais são os resultados de testes da implementação?
- Quais sãos os resultados da adição do Controle Geral e do refinamento do código pensando na adição do Sistema Supervisório no cenário total do projeto?

O projeto de software do Controle Adaptativo se encontra devidamente documentado. Graças à criação e constante atualização dos diagramas, além de se terem disponíveis modelos fiéis do que foi usado para a implementação, também se conseguiu realizar a implementação de um código mais robusto, conciso e organizado. Isso é muito útil para futuros trabalhos no projeto, já que se deixa um ponto de partida bem formalizado e documentado para melhorias em trabalhos posteriores, como a adição efetiva do Controle Fuzzy, que não foi finalizada neste trabalho.

Para testar a implementação do PID Adaptativo sem colocá-la no AGV, utilizamos uma planta de primeira ordem discretizada pelo método de Tustin, que simula um feedback para o PID. Somamos a este feedback um valor aleatório de -6 a 6, simulando uma entrada da Visão Computacional. A meta era que, dado um set point desejado, o PID iria fazer com que o feedback tentasse se manter no set point. Foi testado o código com vários exemplos de set point e o controle se mostrou muito eficaz tentando manter o feedback em uma região próxima à esperada, mesmo com a adição de valores aleatórios em cena. Futuramente, ele será testado no AGV, com valores vindos da Visão Computacional.

Sobre o Controle Fuzzy, não se tem resultados a não ser a documentação e alguns estudos feitos no MatLab, conforme foi discutido em seções anteriores. Felizmente, pôde-se montar um ponto de partida bem consolidado para a implementação desta técnica para futuras equipes.

Com o código do Controle Geral criado e os parâmetros de controle editáveis registrados, agora se está mais perto de se ter um sistema que consiga se comunicar com um Supervisório e onde podem ser executadas diversas implementações de controles diferentes, inclusive outros que não o PID e o Fuzzy. Foi possível com poucas alterações criar um *software* de controle facilmente comunicativo com uma interface de controle externa, mostrando que a edição dos parâmetros de controle utilizados pelas implementações foi implementada de forma bastante modularizada. Também foi aprimorada a documentação do código como um todo de forma a se facilitar o trabalho de futuras equipes que atuarão sobre o controle do AGV.

7. Conclusão

- Como o desenvolvimento atual afeta o projeto?
- Os resultados obtidos até então são satisfatórios?
- Limitações do Sistema e ideias de trabalhos futuros.

Quando foi iniciado o projeto, o AGV possuía um sistema de controle com PID Simples. Para determinados casos, de fato ele funciona muito bem, principalmente em testes em laboratório. Porém, para que o AGV chegue mais perto de poder operar em escala industrial é necessário que nele esteja implementado um sistema de controle mais adequado, configurando um controle adaptativo. Com isso em mente, foi iniciado o projeto de controle deste trabalho, que explorou a implementação do PID Adaptativo e fundamentou a base para o Controle Fuzzy, disponibilizando um sistema de controle que, com poucas alterações em seu estado atual, já pode ser implementado no AGV e posto para operar.

Atualmente, se tem um PID Adaptativo que foi testado com uma planta discretizada pelo método de Tustin e constatou-se o seu perfeito funcionamento. Apesar disso, é necessária a implementação no AGV recebendo a saída do código de Visão Computacional e com uma Tabela de Sintonia devidamente sintonizada para constatar se o controle do PID Adaptativo realmente funciona na prática.

O sistema de Controle Geral e seus códigos dependentes foram refinados para permitir uma futura comunicação com o Sistema Supervisório. Com isso, espera-se que o grupo responsável por tal sistema consiga mais facilmente implementar a interface responsável pela customização do controle e sua eventual execução.

Estudou-se a implementação teórica do Controle Fuzzy e um pouco da utilização da *toolbox* de MatLab Fuzzy Logic Control, e ele realmente é um sistema de controle que pode se provar muito útil ao AGV. Assim sendo, o grupo ~~investigará~~ iniciou a investigação de implementações já existentes desta técnica e métodos de se aplicá-la ao sistema embarcado atual do veículo. ~~Também será sopesada as vantagens de se implementar manualmente o Controle Fuzzy ao invés de se usar uma implementação já existente.~~

A primeira limitação do Sistema de Controle diz respeito ao modo de busca de novas constantes de sintonia pelo Decisor do PID Adaptativo. Com certeza a busca sequencial que foi feita, elemento por elemento consecutivo, não é a mais otimizada. Portanto, é uma boa ideia se pensar e desenvolver métodos mais sofisticados para que o Decisor encontre outro Kp, Ki e Kd. Além disso, o PID Adaptativo ainda não é capaz de modificar apenas um de seus parâmetros de sintonia, sempre modificando os três. Pode ser muito vantajosa a implementação de maneiras do sistema saber qual constante ele precisa atualizar e alterar apenas esta.

No que diz respeito à possibilidade de implementação em hardware, muito possivelmente o PID Adaptativo consegue ser executado em uma Raspberry Pi enviando o resultado para o Arduino sem grandes problemas de performance. Por outro lado, pode ser que o Controle Fuzzy não tenha um bom desempenho por exigir uma quantidade muito maior de cálculos durante o tempo de execução. Existem então algumas alternativas a serem sopesadas, entre elas, a implementação do sistema de controle em outra linguagem de programação, como o C, que é conhecido por ser altamente eficiente.

Para trabalhos futuros, recomenda-se que seja seguido o trabalho atual exatamente do Controle Fuzzy, mas em outra linguagem de programação, embora hajam melhorias a serem feitas no PID Adaptativo. Isso se deve à probabilidade de que, se o Controle Fuzzy funcionar corretamente, dificilmente a performance do PID Adaptativo será melhor que a dele. Unindo os fatos de que a implementação do Controle Fuzzy ou utilização da Fuzzy Logic Control não é muito complicada e de que é um sistema altamente eficiente, pode ser que este seja um ponto de partida muito interessante para trabalhos futuros.

Referências bibliográficas

[1] OSÓRIO, Luís et al. A Comparison of Adaptive PID Methodologies Controlling a DC Motor With a Varying Load. Disponível em: <http://home.isr.uc.pt/~rui/publications/etfa2013_3_web.pdf>. Acesso em: 27 set. 2018.

[2] ANNASWAMY, Anuradha. Adaptive Control: Adaptive PID Control. Disponível em: <http://aaclab.mit.edu/material/lect/lecture7.pdf>. Acesso em: 22 out. 2018.

[3] JOHAN ASTROM, Karl. PID Control. 2002. Disponível em: <https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch6.pdf>. Acesso em: 22 out. 2018.

[4] NEVES, Felipe. Método de Tustin. Disponível em: <https://www.embarcados.com.br/metodo-de-tustin/>. Acesso em: 22 out. 2018.

[5] MORTIZ, Guilherme Luiz. Projeto de controladores. 2014. Disponível em: <http://paginapessoal.utfpr.edu.br/moritz/disciplinas-anteriores/2014-01/et77h-controle-2/aulas/09_ProjetodeControladores.pdf/at_download/file>. Acesso em: 22 out. 2018

[6] IVPID. Ivmech Mechatronics Innovation Ltd., 2016. Disponível em: <https://github.com/ivmech/ivPID>. Acesso em: 22 out. 2018.

[7] DENONCOURT, Frank**.** Introduction to Fuzzy Logic. 2013. Disponível em: <http://aisii.azc.uam.mx/mcbc/Cursos/IntCompt/Lectura15.pdf>. Acesso em: 18 nov. 2018.

[8] FUZZY Logic Toolbox: Design and simulate fuzzy logic systems. Disponível em: <https://www.mathworks.com/products/fuzzy-logic.html>. Acesso em: 19 nov. 2018.

[9] SINGHALA, P.; SHAH, D. N.; PATEL, B. Temperature Control using Fuzzy Logic. 2014. 10 p. Artigo (Artigo em Instrumentação e Controle)- Sarvajanik College of Engineering and Technology Surat, Gujarat, India, 2014. Disponível em: <https://arxiv.org/ftp/arxiv/papers/1402/1402.3654.pdf>. Acesso em: 19 nov. 2018.

[10] ALONSO, Sanjay Krishnankutty. Mamdani's Fuzzy Inference Method. 1. 2018. Disponível em: <http://www.dma.fi.upm.es/recursos/aplicaciones/logica_borrosa/web/fuzzy_inferencia/main_en.htm>. Acesso em: 03 dez. 2018.

[11] SCIKIT: Fuzzy. Version 0.2. [S.l.: s.n.], 2018. Disponível em: <https://pythonhosted.org/scikit-fuzzy/>. Acesso em: 03 dez. 2018.

Apêndice 1 – Códigos

Código 1 – Classe Decisor

```
import numpy as np

class Decisor:
        erros = []
        amostra = 10
        constantes = []
        idx = 0
        #valores iniciais. O usuario deve selecionar
        #o valor que melhor se encaixa
        limite_media = 5
        limite_desvio = 0
        media = 0
        desvio_padrao = 0

        #Adiciona valores de erro no vetor de erros
        def add_erro(self, erro):
                #se exisitir mais que o numero de amostras de erros
                #vai remover o ultimo inserido antes de adicionar o novo
                if len(self.erros) >= self.amostra:
                        self.erros.pop(0)
                self.erros.append(erro)

        def set_amostra(self, amostra):
                self.amostra = amostra

        def set_limite_media(self, limite):
                self.limite_media = limite

        def set_limite_desvio(self, limite):
                self.limite_desvio = limite

        def set_index(self, idx):
                self.idx = idx

        def get_index(self):
                return self.idx

        def get_amostra(self):
                return self.amostra

        def get_media(self):
                return self.media

        def get_desvio_padrao(self):
                return self.desvio_padrao

        def get_limite_media(self):
                return self.limite_media


        def get_limite_desvio(self):
                return self.limite_desvio

        def set_constantes(self,kp, ki, kd):
```

```
                self.constantes[:] = []
                self.constantes.append(kp)
                self.constantes.append(ki)
                self.constantes.append(kd)

        #Usa media e desvio padrao para calcular o indice da Tabela
        def computa_index(self):
                #calcula a media
                self.media = np.mean(self.erros)
                #calcula o desvio padrao
                self.desvio_padrao = np.std(self.erros)
                #calcula o valor absoluto da media
                media_abs = abs(self.media)

                #Estamos realizando uma busca sucessiva (rampa).
                #Existe a alternativa de busca de aproximacao sucessiva (ponto medio)
                #verifica as condicoes e edita o indice
                if media_abs < self.limite_media and self.desvio_padrao > self.limite_desvio:
                        self.idx = self.idx - 1
                elif media_abs > self.limite_media and self.desvio_padrao < self.limite_desvio:
                        self.idx = self.idx + 1

                return self.idx

        #usa o manipulador de tabelas para a acessar as contantes
        def get_constantes(self, manipulador, idx):
                self.constantes = manipulador.get_constantes(idx)
                #primeira linha e invalida, entao pega a proxima
                if self.constantes[0] == 0.0:
                        self.idx = idx + 1
                        self.constantes = manipulador.get_constantes(self.idx)
                #ultima linha e invalida, entao pega a anterior
                if self.constantes[0] == float('Inf'):
                        self.idx = idx - 1
                        self.constantes = manipulador.get_constantes(self.idx)
                return self.constantes
```

Código 2 – Classe Manipulador de Tabela

```
import csv
from operator import itemgetter

#A tabela esta sendo ordenada pelo Ki, pois a agressividade da
#correcao do angulo esta relacionada a somatoria dos erros acumulado
class ManipuladorDaTabela:

        tabela = list()
        nome_da_tabela = "tabela_de_sintonia.csv"

        #Cria um arquivo .csv com o nome da tabela especificado
        def criar_tabela(self):
                f = open(nome_da_tabela, "w+")
                f.close()

        #Abre o arquivo especificado em nome_da_tabela e le seu
        #conteudo numa lista
        def le_tabela(self):
                #abre um indicador para o csv
                reader = csv.reader(open(nome_da_tabela))
                #tranforma os elementos da tabela em uma lista
                tabela_str = list(reader)

                #Configura o primeiro e o ultimo elementos da tabela, que devem
                #ser como especificados a seguir
                if len(tabela_str) == 0:
                        self.tabela.append([0,0,0])
                        self.tabela.append([float('Inf'),float('Inf'),float('Inf')])

                for i in range(len(tabela_str)):
                        row = list()
                        for j in range(len(tabela_str[i])):
                                row.append(float(tabela_str[i][j]))
                        self.tabela.append(row)

        #Escreve dados na tabela especificada em nome_da_tabela
        def escreve_tabela(self):
                #Nao fizemos um sistema de desempate, para o caso onde tenham
                #dois ou mais Kis iguais na tabela
                self.tabela.sort(key=itemgetter(1), reverse=False)
                writer = csv.writer(open(nome_da_tabela, "w"))
                writer.writerows(self.tabela)

        def limpa_tabela(self):
                f = open(nome_da_tabela, "w+")
                f.close()

        def set_kp(self, kp, idx):
                #Nao permite tentar acesso a posicao inexistente
                if idx >= len(self.tabela):
                        print("Nao existe elemento nesta posicao da tabela")
                #Nao permite a edicao do primeiro e ultimo elementos pois sao elementos de
controle
                elif self.tabela[idx][0] != 0.0 and self.tabela[idx][0] != float('Inf'):
                        #edita a o parametro
                        self.tabela[idx][0] = kp
                else:
                        print("Voce esta tentando editar uma posicao da tabela que nao existe!")

        def set_ki(self, ki, idx):
```

```
                if idx >= len(self.tabela):
                        print("Nao existe elemento nesta posicao da tabela")
                elif self.tabela[idx][0] != 0.0 and self.tabela[idx][0] != float('Inf'):
                        #edita a o parametro
                        self.tabela[idx][1] = ki
                else:
                        print("Voce esta tentando editar uma posicao da tabela que nao existe!")

        def set_kd(self, kd, idx):
                if idx >= len(self.tabela):
                        print("Nao existe elemento nesta posicao da tabela")
                elif self.tabela[idx][0] != 0.0 and self.tabela[idx][0] != float('Inf'):
                        #edita a o parametro
                        self.tabela[idx][2] = kd
                else:
                        print("Voce esta tentando editar uma posicao da tabela que nao existe!")

        def set_parametros(self, kp, ki, kd, idx):
                if idx >= len(self.tabela):
                        print("Nao existe elemento nesta posicao da tabela")
                elif self.tabela[idx][0] != 0.0 and self.tabela[idx][0] != float('Inf'):
                        #edita a o parametro
                        self.tabela[idx][0] = kp
                        self.tabela[idx][1] = ki
                        self.tabela[idx][2] = kd
                else:
                        print("Voce esta tentando editar uma posicao da tabela que nao existe!")

        def set_nome_tabela(nome_da_tabela):
                self.nome_da_tabela = nome_da_tabela

        def add_parametros(self, kp, ki, kd):
                self.tabela.append([kp, ki, kd])

        def get_kp(self, idx):
                #le o elemento
                kp = self.tabela[idx][0]
                return  kp

        def get_ki(self, idx):
                ki = self.tabela[idx][1]
                return  ki

        def get_kd(self, idx):
                kd = self.tabela[idx][2]
                return kd

        def get_tamanho_tabela(self):
                print(len(self.tabela))
                return len(self.tabela)

        def get_nome_tabela():
                return nome_da_tabela

        #sempre vai recalibrar o kp, ki e kd. Nunca um individualmente
        #se quisermos verificar individualmente, devemos ver outros parametros
        #que sao o erro, overshooting, etc, para cada entrada degrau, comparando
        #sempre com o anterior
        def get_constantes(self, idx):
                #cria uma lista de constantes
                constantes=[]
```

```
                #adiciona os parametros na lista
                constantes.append(self.get_kp(idx))
                constantes.append(self.get_ki(idx))
                constantes.append(self.get_kd(idx))
                return constantes
```

Código 3 – PID Adaptativo

```
from manipulador_da_tabela import ManipuladorDaTabela

from decisor import Decisor
from PID import PID
import time
import numpy as np

decisor = Decisor()
manipulador = ManipuladorDaTabela()
PID = PID()

qnt_leituras = 0 #quantas leituras da visao
num_amostras = decisor.get_amostra() #numero de amostras
e = 0 #erro
decisor.set_limite_desvio(3.5)
decisor.set_limite_media(1)
get_erro_angulo_visao = 5.0 #erro inicial da visao
sample_time  = 0.0
tau = 5
PID.setSetPoint(1.0)
feedback = 0.0
output = 0.0
last_output = 0.0
last_feedback = 0.0

#feedback inicial como numero aleatorio entre -6 e 6
#isso vai simular o erro de angulo da visao
feedback = np.random.randint(-6,6)
PID.setSampleTime(1.0)

while(1):

        last_output = output

        sample_time = PID.getSampleTime()
        time.sleep(sample_time)

        #adiciona o erro no decisor
        decisor.add_erro(feedback)

        #soma uma leitura
        qnt_leituras = qnt_leituras + 1

        #caso tenha o numero de amostras, deve verificar
        #a performance
        if qnt_leituras >= num_amostras:
                qnt_leituras = 0

                #calcula o novo indice se precisar
                index = decisor.computa_index()

                #obtem as constantes e atribui ao PID Simples
                manipulador.le_tabela("tabela_de_sintonia.csv")
                constantes = decisor.get_constantes(manipulador, index)
                PID.setKp(float(constantes[0]))
                PID.setKi(float(constantes[1]))
                PID.setKd(float(constantes[2]))

        #coloca o feedback da planta + um valor aleatorio da visao no PID
        PID.update(feedback + np.random.randint(-6,6))
        #pega o output
```

```
        output = PID.getOutput()

        #Exibe os resultados
        print("SETPOINT: " + str(PID.SetPoint))
        print("Output do PID: " + str(output))

        # Atualizacao da planta
    # Planta de primeira ordem discretizada pelo metodo de Tustin
        last_feedback = feedback
        feedback = last_feedback*((2*tau - sample_time)/(2*tau + sample_time))
+output*((sample_time)/(2*tau + sample_time))+last_output*((sample_time)/(2*tau + sample_time))
        print ("FEEDBACK: " + str(feedback))
```

Código 4 – Controle Geral

```
from manipulador_da_tabela import ManipuladorDaTabela

from decisor import Decisor
from PID import PID
import time
import numpy as np

#Seleciona qual metodo de controle sera utilizado
# 0 - Nenhum controle
# 1 - PID Simples
# 2 - PID Adaptativo
# 3 - Fuzzy
seletorControle = 3

#Criacao dos objetos da classe
decisor = Decisor()
manipulador = ManipuladorDaTabela()
PID = PID()

#Parametros inicias
qnt_leituras = 0 #quantidade de amostras
num_amostras = decisor.get_amostra() #numero de amostras
decisor.set_limite_desvio(3.5)
decisor.set_limite_media(1)
saida_visao = 5.0 # saida de angulo visao computacional
sample_time  = 0.0
PID.setSetPoint(1.0)
output = 0.0
kp = 1.0
ki = 0.5
kd = 0.2
nome_da_tabela = "tabela_de_sintonia.csv"

#feedback inicial como numero aleatorio entre -6 e 6
#isso vai simular o erro de angulo da visao
saida_visao = np.random.randint(-6,6)
PID.setSampleTime(1.0)

#Calcula um PID simples com os valores kp, ki e kd
#e retorna o ganho
def PIDSimples(kp, ki, kd):

          #Simula a leitura da visao
          saida_visao = np.random.randint(-6,6)

          PID.setKp(float(kp))
          PID.setKi(float(ki))
          PID.setKd(float(kd))

          #Calcula o output
          PID.update(saida_visao)

          #pega o output
          output = PID.getOutput()

          return output

#Calcula o PID Adaptativo com base numa tabela de
#sintonia e retorna o ganhos
def PIDAdaptativo(nome_da_tabela):
```

```
        global qnt_leituras

        #Simula a leitura da visao
        saida_visao = np.random.randint(-6,6)

        sample_time = PID.getSampleTime()
        time.sleep(sample_time)

        #adiciona o erro no decisor
        decisor.add_erro(saida_visao)

        #soma uma leitura
        qnt_leituras = qnt_leituras + 1

        #caso tenha o numero de amostras, deve verificar
        #a performance
        if qnt_leituras >= num_amostras:
                qnt_leituras = 0

                #calcula o novo indice se precisar
                index = decisor.computa_index()

                #obtem as constantes e atribui ao PID Simples
                manipulador.le_tabela(nome_da_tabela)
                constantes = decisor.get_constantes(manipulador, index)
                PID.setKp(float(constantes[0]))
                PID.setKi(float(constantes[1]))
                PID.setKd(float(constantes[2]))

        #Calcula o output
        PID.update(saida_visao)

        #pega o output
        output = PID.getOutput()

        return output

#Calcula o ganho com um Controle Fuzzy
def ControleFuzzy():
        print("Ainda nao implementado")
        return -1

def main():

        #Saida do controle
        output = 0.0

        #Se for 0 nao roda nenhum controle
        while seletorControle!=0:
                # TODO: Ler o tipo de controle pela interface
                if seletorControle == 1:
                        print("Executando o PID Simples")
                        # TODO: Ler Kp, Ki e Kd da interface
                        # TODO: Ler sample time
                        # TODO: Ler SetPoint
                        output = PIDSimples(kp, ki, kd)
                if seletorControle == 2:
                        print("Executando o PID Adaptativo")
                        # TODO: Ler o nome da tabela da interface
                        # TODO: Ler numero de amostras
                        # TODO: Ler o sample time
                        # TODO: Ler o limite de desvio padroa
```

```
                        # TODO: Ler o limite da media
                        # TODO: Ler SetPoint
                        # TODO: Ler indice inicial da tabela
                        output = PIDAdaptativo(nome_da_tabela)
                if seletorControle == 3:
                        print("Executando o Controle Fuzzy")
                        output = ControleFuzzy()
                print(output)
        return

if __name__ == "__main__":
        main()
```