Robô seguidor de linha com controle automático via processamento de imagem

Luis Fernando Fontoura Spaniol
Marco Antônio Sandini Trentin (orientador)

Universidade Passo Fundo (UPF)
Passo Fundo – RS – Brasil

Instituto de Ciências Exatas e Geociências (ICEG)

128423@upf.br, trentin@upf.br

Abstract. *This article describes the construction and programming of a follower line robot. It was used some components like Raspberry, batteries and engines and webcam for the construction of the prototype. It also describes the use of image processing techniques and artificial intelligence to guide the prototype through the track.The track is defined by a black line in contrast to a white background. To align the prototype to the track, a proportional-integral-derivative controller (PID) was used. The research demonstrates how to integrate prototyping, image processing, artificial intelligence and looping control system for the robot follower line implementation with images captured by a Webcam.*

Resumo. *Este artigo descreve a construção e programação de um robô seguidor de linha. Foram utilizados alguns componentes como Raspberry, baterias e motores e webcam para a construção do protótipo. Descreve também a utilização de técnicas de processamento de imagem e inteligência artificial para guiar o protótipo através da pista, cujo trajeto é definido por uma linha preta em contraste a um fundo branco. Para alinhar o protótipo à pista foi utilizado um controlador proporcional-integral-derivativo (PID). A pesquisa demonstrou como unir prototipação, processamento de imagem, inteligência artificial e controle de sistemas em loop para a implementação de seguidor de linha a partir imagens capturadas por uma Webcam.*

1. Introdução

O advento da robótica possibilitou um grande aumento na automação de diversas tarefas e o aumento da produtividade humana. A robótica, inclusive, possibilitou que a humanidade partisse para a conquista de mundos inóspitos a ela, como por exemplo os *rovers Spirit* e *Opportunity* que são utilizados para a exploração do planeta Marte. A robótica é responsável desde desenvolver os carros autônomos até promover a inclusão digital, através de tecnologias como o *Arduino* e *Raspberry*.

Outro fator que impulsionou a popularização da robótica foi o maior poder de processamento das tecnologias embarcadas, aliado à adoção do modelo de desenvolvimento *Open Source*, que resultou em muitas tecnologias potencializadas. Essa integração entre as tecnologias abertas proporciona a facilidade de implementação e teste de modelos

computacionais. Uma das tecnologias que mais se desenvolveram foram o Processamento Gráfico e a Inteligência Artificial. Englobando essas duas áreas, existem algumas bibliotecas de código livre como, por exemplo, o OpenCV.

Entre as aplicações da robótica está a educativa, e uma das atividades que se destaca nessa área são as competições de robótica em âmbito educacional. Um exemplo disso é a Olimpíada Brasileira de Robótica (OBR). Um dos desafios comuns presentes em olimpíadas é o de construir robôs seguidores de linha. Esse desafio geralmente é implementado de duas formas: através de sensores de infravermelho ou de processamento de imagem. Neste trabalho será utilizado o modelo de processamento de imagem, utilizando para a sua implementação ferramentas de software aberto, dentre elas, a biblioteca OpenCV.

Os seguidores de linha são utilizados para a aprendizagem de modelos matemáticos. Dentre os modelos está o controle proporcional-integral-derivativo (PID), que é uma das técnicas de controle de processos mais utilizadas atualmente. Estima-se que mais de 90% dos controladores em *loop* empregam o PID. Ao longo do último meio século, a grande parte do esforço acadêmico e industrial nesta área tem se concentrado em melhorar o controle PID, principalmente nas áreas de regras de ajuste, esquemas de identificação e técnicas de adaptação [KNOSPE 2006].

Diante disso, esta pesquisa teve por objetivo construir um protótipo de seguidor de linha capaz de unir as técnicas de processamento de imagem com o controlador PID para o alinhamento perante uma linha que serve de guia para um robô. Além disso foi utilizado um classificador para a condição de parada do protótipo, a fim de avaliar se essas técnicas são adequadas para trabalharem em conjunto com um robô seguidor de linha.

2. Tecnologias empregadas

Para [Wurman et al. 2008], o sistema de gestão de armazéns da empresa Kiva[1], responsável pela busca e reposição de produtos em prateleiras por robôs, criou um novo paradigma para os armazéns de *pick-pack-and-ship*, que melhora significativamente a produtividade da empresa. O sistema Kiva usa prateleiras de armazenamento móveis que podem ser levantadas por pequenos robôs autônomos. Ao trazer o produto para o trabalhador, a produtividade é aumentada por um fator de dois ou mais, ao mesmo tempo em que melhora a responsabilidade e a flexibilidade. Uma instalação Kiva para um grande centro de distribuição pode exigir 500 ou mais veículos. Como tal, o sistema Kiva representa o primeiro sistema robótico autônomo de grande escala comercialmente disponível. A primeira instalação permanente de um sistema Kiva foi implantada no verão de 2006 conforme a figura 1.

Conforme [Gurel 2017], no campo da robótica móvel, os robôs com rodas são amplamente utilizados devido suas vantagens sobre os com pernas. O controlador PID é uma estratégia de controle livre de contexto, que é frequentemente implementada no controle de baixo nível da hierarquia de controle robótico móvel, mais especificamente no auxílio de posicionamento de um robô enquanto se desloca. Embora os controladores PID sejam os controladores mais empregados na robótica, seu desempenho é muito dependente de quão bem os parâmetros do controlador PID serão ajustados.

[1]Kiva é subsidiária da Amazon responsável pela manufatura de sistemas robóticos Link: https://www.amazonrobotics.com/

Figura 1. Robô utilizados pela Amazon fabricados pela Kiva

Segundo [Mohamad 2015], o *Open Source Computer Vision Library* (OpenCV) é uma *Application Peripheral Interface* (API) desenvolvida pela Intel que pode ser usada para muitas aplicações de processamento de imagens e visão computacional. O OpenCV foi lançado oficialmente em 1999 e o projeto foi inicialmente uma iniciativa da Intel Research para promover aplicações intensivas de CPU. Parte de uma série de projetos, incluindo *Ray tracing* em tempo real e paredes de exibição 3D.

Para [Al Tahtawi et al. 2017], uma Máquina de Estado Finito (MEF) é uma metodologia de sistema de controle que descreve o comportamento de um sistema usando três elementos básicos: estado, evento e ação. O sistema se alterna linearmente, entre os três estados. Para que o sistema alterne é necessário um *input* ou evento. Um seguidor de linha pode ser implementado através de uma MEF com o controlador de tipo PID. O controlador é utilizado para se manter o alinhamento entre o seguidor e a linha. O resultado apresentado neste trabalho demonstra que o algoritmo projetado pode funcionar bem e pode ser usado como um algoritmo baseado nesse robô.

3. Modelo Proposto

Nesta seção serão descritos os detalhes da construção e programação do robô que foi criado para conhecer e avaliar a integração das tecnologias envolvidas, tanto de *software* quanto *hardware*.

3.1. Hardware utilizado

Para a construção do protótipo desta pesquisa foram utilizados vários componentes da microeletrônica e robótica, a saber: um *Raspberry Pi B 3*, *shield* ponte-H L298N, dois motores DC, *jumpers*, duas baterias, sendo uma modelo 25c com 1000 mAh com 2 células e saída nominal de 7,4 V, utilizada para fornecer energia para a *shield*, e outra com 3200 mAh com saídas USB de 5 V (*power bank*), para alimentar o *Raspberry*, além de uma *webcam* para se adquirir informação do meio.

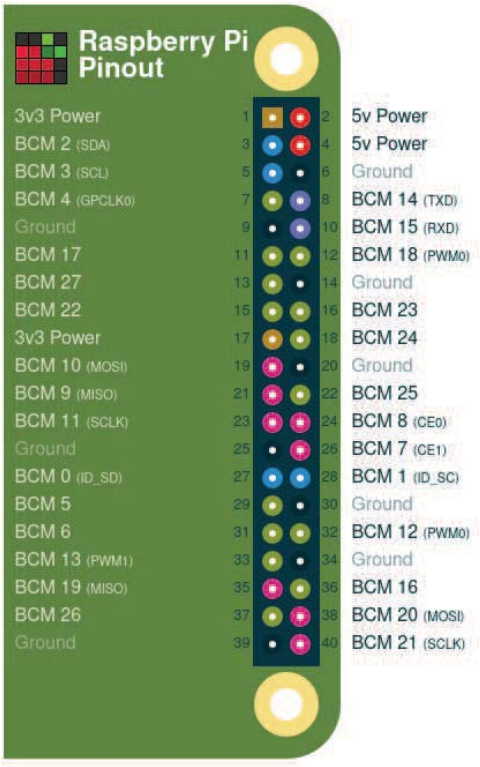

Figura 2. Pinagem do *Raspberry* pi 3 b

Decidiu-se pela utilização de um *Raspberry* pois ele fornece uma capacidade de processamento superior ao Arduíno, devido ao seu processador ARM de 1,2 GHz e 1 GiB de memória RAM, possibilitando assim a capacidade computacional necessária para a instalação de sistema operacional Linux e a utilização da linguagem Python para o desenvolvimento da aplicação do modelo proposto, além de possibilitar a conexão da *webcam* via USB.

Para o controle dos motores foi utilizado o *Raspberry* em modo BOARD (placa) onde, neste caso, os números dos pinos seguirão a marcação física, ao contrário do modo BCM (*Broadcom pin number*), onde os pinos seguirão a marcação conforme a sua função, como pode ser visto na Figura 2. Já a Figura 3 representa as conexões dos pinos à placa *shield* ponte-H L298N. Os pinos de *Pulse Width Modulation* (PWM) são representados pelos fios de cor rosa e os fios de cor azul representam os digitais. Os fios de cor preta representa o aterramento (GND) e o vermelho representa a entrada da alimentação a partir da bateria.

A *shield* ponte-H L298N foi utilizada para conectar os motores em suas portas de *output* e controlar a corrente elétrica que passa para os motores, além de receber os comandos do *Raspberry* através dos pinos digitais. A bateria de 7,2 V foi usada para alimentar a *shield* ponte-H L298N que por sua vez alimenta os motores.

Para o comando do motores todos os pinos utilizados foram definidos como pinos de OUTPUT (saída). O controle de direção dos motores é controlada pelo pinos digitais. O controle de velocidade é feito pelos pinos PWM. A *webcam* foi conectada a uma das entradas USB do *Raspberry* e a entrada mini USB de alimentação foi ligada a bateria

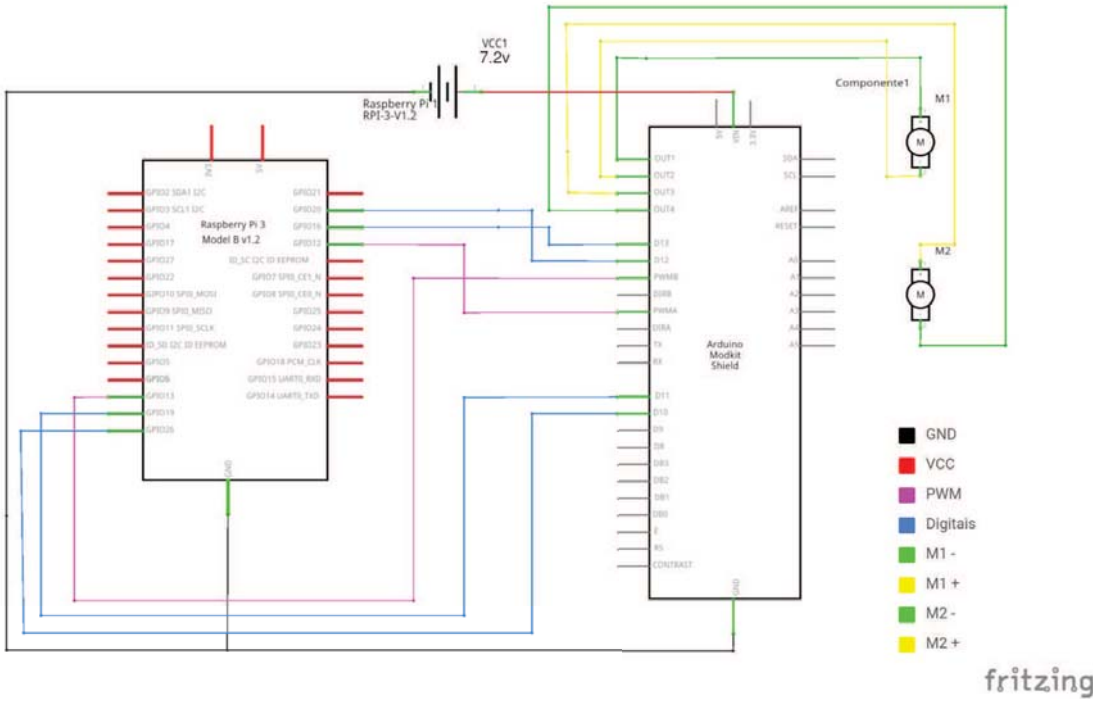

Figura 3. Esquemático de ligação *Raspberry-shield*

power bank de 3200 mAh através de um cabo USB.

Decidiu-se por utilizar os últimos pinos com numeração mais altas do *Raspberry* para se liberar espaço, a fim de conectar uma pequena tela que faz uso dos primeiros pinos como alimentação e um pequeno *jumper* HDMI, para ligar a saída HDMI do *Raspberry* na entrada HDMI da tela. Essa tela não foi usada diretamente neste projeto, apenas serviu para interação direta com o *Raspberry* e para pequenos ajustes diretos no robô. O protótipo final pode ser visto na Figura 4.

3.2. Desenvolvimento do Software

Nestas seção serão apresentados detalhes das configurações de *software* utilizadas neste projeto.

3.2.1. Preparando a plataforma

Foi instalado no *Raspberry* deste projeto o Sistema Operacional Ubuntu MATE. Para o desenvolvimento do software foi empregada a linguagem de programação Python na versão 2,7 e algumas bibliotecas, tais como a *General Purpose Input/Output* (GPIO), que é própria do *Raspberry*. Ela é uma biblioteca de propósito geral para o controle dos pinos de *input/output* do *Raspberry*. Também foi utilizada a biblioteca gráfica OpenCV. Ela é uma das bibliotecas disponíveis para a computação visual e foi utilizada para capturar imagens da *webcam* e processá-las. Já a biblioteca *Threading* foi utilizada para se obter paralelismo entre um identificador de imagem e o controle de movimentos.

Mais especificamente, a biblioteca GPIO foi utilizada para definir o modo de op-

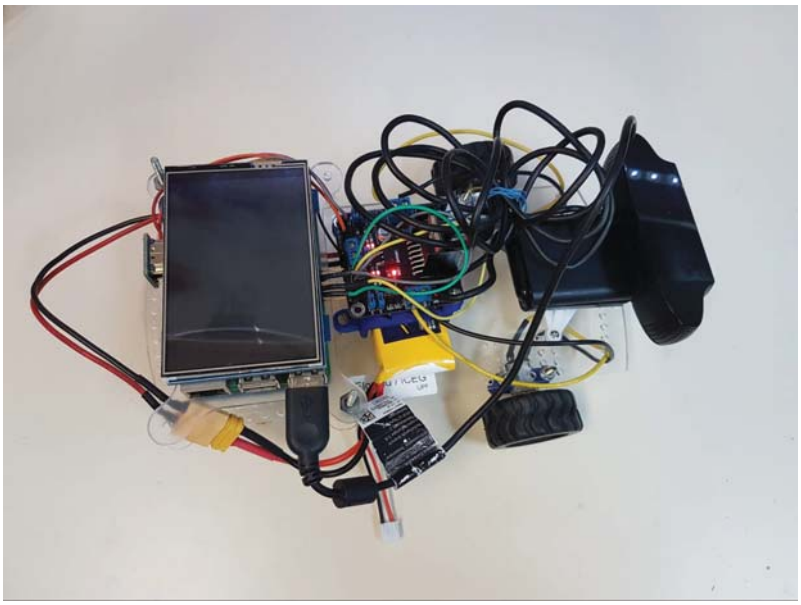

Figura 4. Protótipo da pesquisa do trabalho

eração dos pinos do *Raspberry* para o modo de BOARD, sendo que os pinos 32, 36, 38, 33, 35 e 37 foram configurados para o modo de OUTPUT. Para os pinos 32 e 33 foi indicado como função de PWM, sendo definido que eles começarão com 0% da capacidade. Os pinos 36 e 37 foram alterados para o valor HIGH. Como eles são digitais, os mesmos foram definidos como "ligados", ou seja, nível de saídas 1. Já os pinos 35 e 38 foram alterados para LOW, fixando as suas saídas em 0.

A seguir serão descritos os detalhes da implementação e funcionamento do software construído para seguir linhas usando processamento gráfico e PID.

3.2.2. Processamento gráfico

Para tratar do *stream* de vídeo capturado, se fez uso de algumas funções da biblioteca OpenCV. Foi utilizada a função de *VideoCapture* para receber essa *stream* de vídeo da câmera selecionada. Foi utilizada a função *read* para se ter um *frame* do fluxo de vídeo da câmera. Ela retorna duas informações: o *grab*, que serve de controle para sistemas multi-câmeras e o *frame*. Neste trabalho não é feito uso do *grab*. Após, é necessário definir o formato e a resolução desejada, sendo que neste trabalho foi utilizada 16:9 de formato e 1920x1080 de resolução de imagem.

Dentre as técnicas de processamento de imagem empregadas neste trabalho pode-se citar o *Gaussian blur*, que consiste em desfocar a imagem, de modo a diminuir a quantidade de imperfeições nela presentes. Decidiu-se pelo uso dessa técnica devido ao fato de que toda imagem obtida por câmeras fotográficas ser limitada quanto à resolução e estar sujeita a imperfeições, causadas pela difração e pelo ruído de *Poisson*. Essa degradação ocorre em muitas aplicações, como microscopia de fluorescência ou astronomia, devido a várias restrições físicas (por exemplo, fonte de luz de baixa intensidade, tempo de exposição curto). Os valores referentes à quantidade de desfoque necessário ao projeto foi calculada com o uso do algoritmo *Gaussian kernel* [Li et al. 2017].

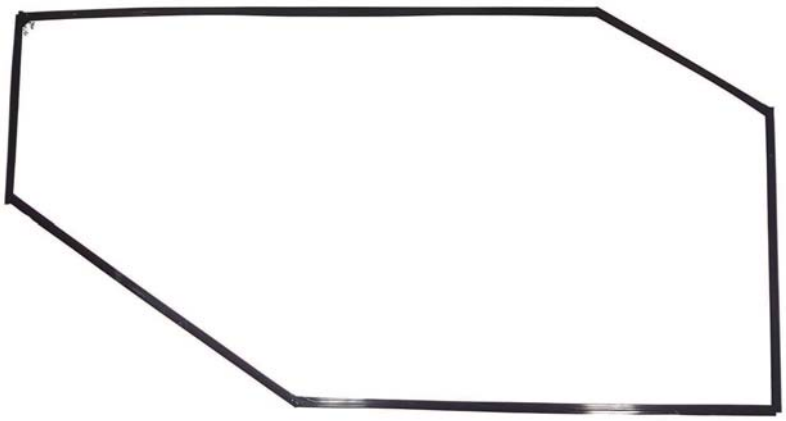

Figura 5. Pista de teste do protótipo nessa pesquisa

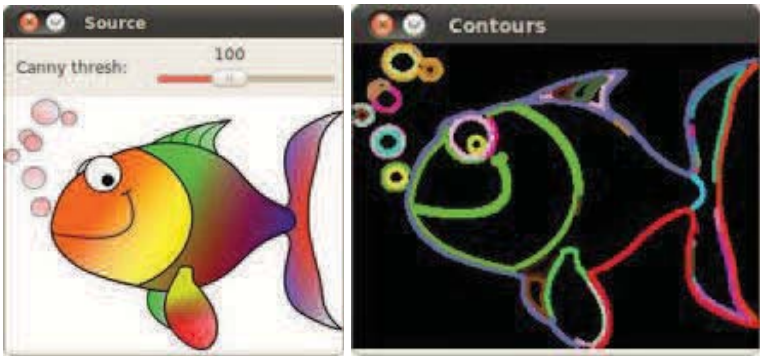

Figura 6. Exemplo de *findContours* e seu desenho

Uma vez que nos testes desta pesquisa, a linha é da cor preta e o entorno branco, conforme a pista representada pela figura 5, foi possível fazer algumas otimizações, sendo uma delas a de deixar o *frame* preto e branco. Essa otimização possibilita o aumento de *Gaussian blur*. Assim, antes de se passar por um processo de reconhecimento, realiza-se essa etapa, a fim de auxiliar na remoção do conteúdo de alta frequência, tais como as bordas da imagem e ruídos.

Para passar o *frame* pelo processo de reconhecimento, é necessário que o conteúdo do *frame* que está em *Red*, *Green*, *Blue*, seja convertido para binário. Essa modificação é responsável pela transcrição do formato pixel para um formato de vetor. A função responsável por essa transformação é a função de *threshold*. Para localizar e capturar a linha no chão pela *webcam* foi utilizada a função de *findContours*, que é responsável por retornar os contornos e a sua hierarquia. A *findContours* se utiliza de um seguidor de borda que consiste em derivar a imagem em plano de fundo identificado pelo valor 0, e a linha correspondendo ao valor 1 [Suzuki and Abe 1984]. É possível ver um exemplo disso na figura 5, que apresenta uma imagem em seu estado padrão (esquerda) e apresenta o resultado da aplicação da função *findContours* (direita), sendo que para esse trabalho somente é necessário os contornos.

Com o retorno da *findContours* é necessário calcular o ponto central pertencente a

$$C_x = \frac{M_{10}}{M_{00}} C_y = \frac{M_{01}}{M_{00}}$$

Figura 7. Formula para o calculo do ponto central utilizando a *moments*

linha. Isso é feito pela função *moments*, que calcula os pontos de borda da linha e retorna uma classe contendo os pontos e a área da linha. Para tal, aplica-se a fórmula conforme a figura 7, onde M01 e M00 representam os pontos extremos no eixo X e M01 representa o ponto do eixo Y. Esse ponto central é o local por onde o protótipo deverá passar para estar alinhado com a linha. Definido o ponto, o próximo passo é gerar um controlador para alinhar com a linha. Neste trabalho foi definido que se utilizará o controlador de tipo PID com metodologia Ziegler Nichols que, segundo [Kagueyama 2011], é um método heurístico de sintonizar o controlador, e ainda é amplamente aplicado até hoje.

3.2.3. Controle

O resultado da subtração do *setpoint* (ponto de alinhamento mútuo entre o protótipo e a linha) pela posição da *moments* é o erro relativo. Se o erro é 0 significa que o carrinho está sobre a linha, caso contrário ele estará fora da linha. O controlador PID, segundo [Kagueyama 2011], funciona quando uma ação provoca uma reação em um sistema, proporcional ao erro presente, ou seja, quando a ação de controle é proporcional, e a relação entre a saída do controlador u(t) e o sinal de erro atuante e(t), que é a entrada do controlador, é dada por um ganho simples, que é o Kp, que funciona essencialmente como um amplificador com um ganho ajustável, fórmula essa apresentada na figura 8. Um aumento da ação proporcional geralmente ocasiona melhora na precisão do sistema em malha fechada (sistema em *loop*). A ação integral proporciona melhoria na precisão do sistema em regime permanente, pois ela atua de acordo com uma taxa proporcional ao erro atuante, ou mais especificamente, seu sinal de controle é um sinal proporcional à integral do erro. A ação derivativa consiste na aplicação de um sinal de controle proporcional à derivada do sinal de erro, que é tida como antecipatória, podendo proporcionar melhora na velocidade do sistema em malha fechada, e só apresenta influência em condições transitórias.

Conforme descrito por [Gurel 2017], o desempenho do PID é muito dependente de quão bem os parâmetros do controlador serão ajustados. Neste trabalho uma das principais dificuldades em se utilizar o PID foi a de ajustar seus parâmetros (Kp, Kd, Ki), pois, como citado por [Ogata et al. 2003], o método Ziegler-Nichols consiste em analisar o comportamento do processo e realizar o ajuste por meio de fórmulas pré-estabelecidas. Como o método Ziegler-Nichols é baseado em tentativa e erro, demanda-se um grande trabalho para se ajustar as constantes PID para se ter um desempenho satisfatório.

Porém, a saída do PID não é diretamente compatível com PWM da biblioteca GPIO, então se faz necessário uma conversão. O PWM é controlado através da função *ChangeDutyCycle* da biblioteca GPIO, que é responsável pela alteração do fluxo de energia dos motores e pelo controle da velocidade, através de parâmetros da porcentagem de potência que será destinada aos motores. O PID no protótipo varia entre -300 e 300, sendo

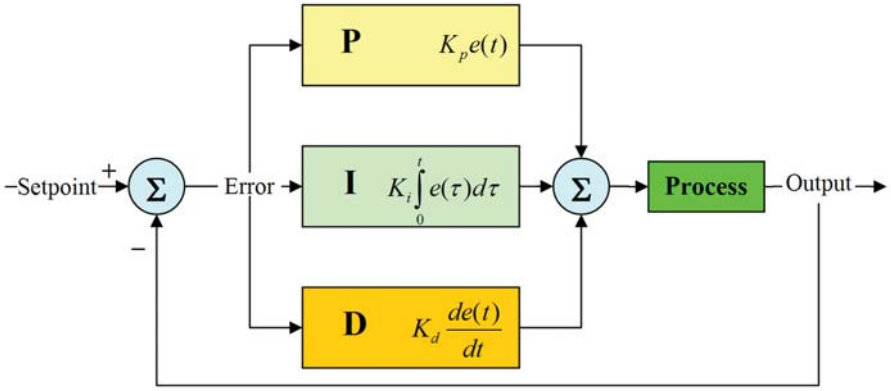

Figura 8. Esquematico de funcionamento do PID

que 0 é o centro da linha. Porém devido a característica da biblioteca GPIO a potência que é estabelecida para o PWM é em formato de percentual. Com isso é necessário a adequação da saída PID para percentual.

Devido ao PID ter características somatórias é iminente que esse controle tente acelerar o PWM de algum motor a mais de 100%. Quando isso ocorrer é necessário que se faça um truncamento nesse valor limitando-o em 100%. Esse efeito do PID também pode ser visto quando se atingir 0% em algum motor. Há a tendência que o motor ganhe valores negativos, para se corrigir isso qualquer valor negativo é igualado a 0% com isso desligando o motor.

3.2.4. Detector de placa de Pare

Também foi implementado um classificador de imagem baseado em técnicas de inteligência artificial. Foi utilizado um classificador em cascata. Para [Viola and Jones 2001], a técnica de *Boosted Cascade* é um método eficaz de detecção de objetos. É uma abordagem baseada em *Machine Learning*, em que uma função *cascade* é treinada com o uso de muitas imagens positivas e negativas. É então usado para detectar objetos em outras imagens. O classificador é exportado em um formato XML.

O objetivo de usar um classificador foi o de encontrar uma placa *STOP*(americana), cujo tamanho real presente em ruas tem as dimensões de 24 x 24 cm. Porém esse formato é grande para ser aplicado na pista, com isso foi diminuído para 5,5 x 5,5 cm. O classificador para esse tipo de identificador é comum na Internet, com isso não foi necessário o treinamento da rede. O classificador utilizado neste projeto foi obtido de [Fizette 2017][2], e é composto por 19 camadas. As imagens de treinamento contém imagens positivas, que servem para a identificação da placa e imagens negativas que servem para evitar falso positivo.

O classificador foi integrado ao projeto através da aplicação de *Threads* a fim de se ter um paralelismo de funções conforme demonstrado pelo diagrama de atividades na Figura 9. Para configurar a *Thread* auxiliar é utilizada a função *CascadeClassifier* para

[2]disponivel em https://github.com/cfizette/road-sign-cascades

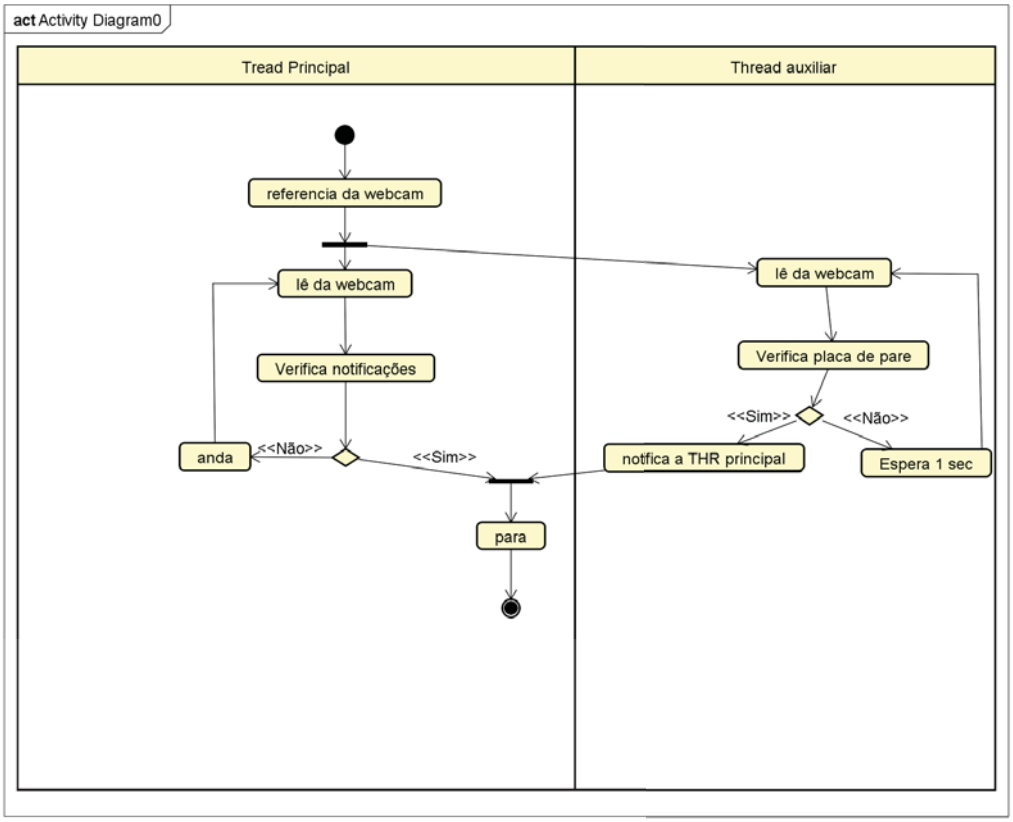

Figura 9. Diagrama de atividades da aplicação

importar o classificador. Após importar o classificador, repete-se o passo de ler o *frame* da *webcam* e aplicar o *Gaussian blur*. Em seguida é usada a função *detectMultiScale* que retornará a posição X e Y do da placa de trânsito. Caso não seja identificada placa, retornará X=0 e Y=0 e será colocada em espera por um segundo, posteriormente volta a repetir o processo. Se a placa for encontrada, é notificada a *thread* principal e finalizada a *thread* secundária.

4. Resultados e discussão

O comportamento do protótipo é variante no início, quando detecta a linha preta, mas tende a se estabilizar. Isso acontece devido ao efeito do processo de autocorreção do PID conforme o decorrer do tempo, apresentado pela figura 10. Devido ao método Ziegler-Nichols ser baseado em heurística, quando se altera a pista são necessários os ajustes nos parâmetros do PID (Kp, Ki, Kd). Uma saída para esse problema seria, como [Gurel 2017] que propõem a utilização de um algoritmo *Q-Learning* para o ajuste adaptativo do PID.

Um das dificuldades encontradas nesse projeto foi de que o *Raspberry* não tem poder de processamento suficiente para executar o classificador em cascata em tempo real. Isso impossibilita que se modele um classificador mais preciso. Devido a essa característica o processamento é realizado em segundo plano. Também percebeu-se que, enquanto a realização de alguns testes, houve superaquecimento do *Raspberry*. Tal fato ocorreu pois no protótipo havia apenas um sistema de refrigeração passiva, na forma de *Heatsinks*. Outra constatação foi oriunda da pequena diferença de rotação de um motor

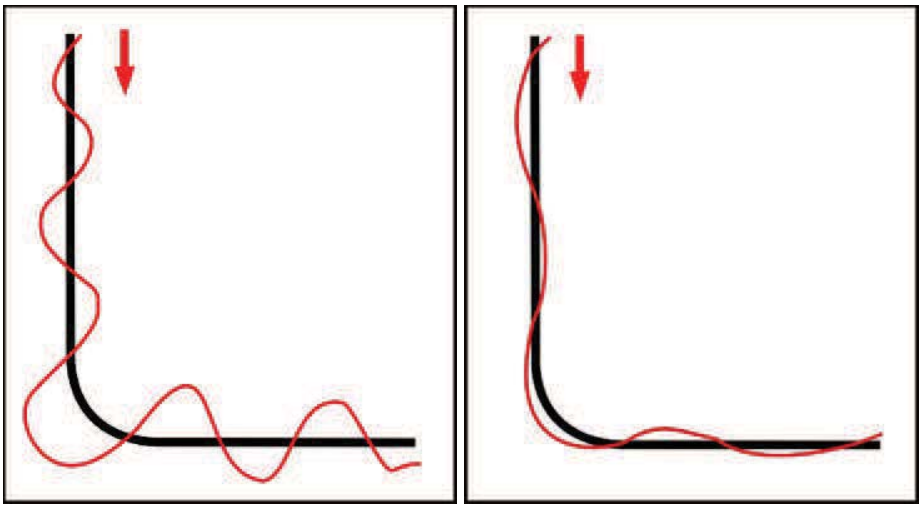

Figura 10. Comportamento do protótipo com controle tradicional(esquerda) e com PID(direita)

para o outro, apesar de serem do mesmo modelo, fazendo com que o protótipo tenha a tendência de sair da linha. Esse fato é delegado à operação do PID para a correção.

Foram realizadas comparações do seguidor de linha usando o processamento de imagem, que foi o protótipo desta pesquisa, juntamente com outro robô que utiliza um modelo que faz uso de sensores infravermelho. O modelo de sensores é limitado em expansão, necessitando adicionar um novo sensor para se ter mais funcionalidades, como quando se deseja adicionar um sistema anti-colisão. No robô com sensores de infravermelho é necessário que se adicione um novo sensor como por exemplo um sonar, enquanto para o processamento de imagem é apenas necessário que se implemente essa função em *software*.

Umas das desvantagens do modelo de processamento de imagem é o seu custo, em comparação com o modelo de sensores de infravermelho. Isso porque seu principal sensor é uma *webcam* que, diferentemente dos sensor de infravermelho, tem um custo bem maior. Outro componente que aumenta o custo neste projeto é o *Raspberry Pi*. Normalmente seguidores de linha utilizando sensores de infravermelho são implementados com a placa *Arduino*, bem mais barata que um *Raspberry Pi*. Já seguidores de linha com processamento de imagem, exigem uma placa de maior poder de processamento e, portanto, são implementados com *Raspberry Pi* ou similares.

5. Conclusão

Este trabalho teve por objetivo implementar um seguidor de linha utilizando PID e processamento gráfico. Os resultados obtidos puderam evidenciar as dificuldades encontradas para o desenvolvimento desse tipo de aplicação, principalmente em relação ao ajuste de PID. Porém essa técnica de controle é eficiente para esse tipo de aplicação.

A aplicação de seguidores de linha tem sido cada vez mais utilizada na educação, seja para atividades educacionais ou recreativa na escola, como também para competir, sendo muito utilizada na Olimpíada Mundial de Robótica e na Olimpíada Brasileira de Robótica. Além disso, essa aplicação possibilita obter um conhecimento básico sobre o

processamento de imagem e sobre o funcionamento de técnicas de inteligência artificial em embarcados, tais como o *boost cascade*

Como trabalhos futuros sugere-se a implementação de um sistema de multi-robôs similares ao da empresa Kiva, onde há uma comunicação entre os robôs, possibilitando a troca de informações sobre obstruções de tráfico e cálculo de rota se utilizando múltiplas linhas de diferentes cores. Outra possibilidade como trabalho futuro é a de utilizar detector de objetos como o YOLO[3] para o controle de tráfego e desvio de obstáculos.

Agradecimentos

Grupo de Pesquisa em Cultura Digital - por fornecer os materiais

Referências

Al Tahtawi, A., Somantri, Y., and Haritman, E. (2017). Design and implementation of pid control-based fsm algorithm on line following robot. *Jurnal Teknologi Rekayasa*, 1:23.

Fizette, C. (2017). road-sign-cascades.

Gurel, C. S. (2017). Q-learning for adaptive pid control of a line follower mobile robot.

Kagueyama, C. A. (2011). Sintonia do controlador pid: Método de ziegler nichols modificado. Monografia apresentada ao curso de Engenharia Elétrica da Universidade Estadual de Londrina.

KNOSPE (2006). C. pid control. *IEEE Control Systems Magazine.*, pages 30–31.

Li, J., Xue, F., and Blu, T. (2017). Gaussian blur estimation for photon-limited images.

Mohamad, M. (2015). A review on opencv. page 1.

Ogata, K. et al. (2003). Engenharia de controle moderno. *São Paulo: Pearson*, 4:788.

Suzuki, S. and Abe, K. (1984). Topological structural analysis of digital images by border following. 30.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *IEEE Conf Comput Vis Pattern Recognit*, 1:I–511.

Wurman, P., D'Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29:9–20.

[3] Yolo é um projeto de *Real-Time Object Detection* link: https://pjreddie.com/darknet/yolo/