MARVIN – Protótipo de Carro Autônomo para Ambiente Industrial

Arthur V. M. Sabatino, Gustavo de Oliveira Guidetti, Daniela Eloise Flôr, Eduardo H. M. Cruz

[1] Instituto Federal do Paraná (IFPR) – Campus Paranavaí
CEP 87703-536 – Paranavaí – PR – Brasil

sabatinoarthur@gmail.com, gustavooguidetti@gmail.com,

daniela.flor@ifpr.edu.br, eduardo.cruz@ifpr.edu.br

Resumo. *Quando se trata de agilidade no processo fabril, nos deparamos com um empecilho, trata-se do transporte de materiais entre os setores de uma indústria. Muitas vezes, este problema é resolvido através do transporte manual operado por funcionários, assim como também é frequente, dentro da área da automação, o uso de esteiras, dentre outros. Porém, são soluções incompletas e imprecisas. MARVIN: o robô de transporte para materiais em ambiente industrial, proposto nesta pesquisa, colabora para minimizar tais inconvenientes, visto que o robô deverá ser capaz de seguir uma rota pré-estabelecida de forma autônoma, sem qualquer intervenção humana.*

Abstract. *When it comes to agility in the manufacturing process, we are faced with an obstacle, it is the transport of materials between sectors of an industry. This problem is often solved through manual transportation operated by employees, as well as the use of conveyors, among others, within the automation area. However, they are incomplete and inaccurate solutions. MARVIN: the transport robot for materials in an industrial environment, proposed in this research, collaborates to minimize such inconveniences, since the robot must be able to follow a pre-established route autonomously, without any human intervention.*

1. Introdução

Em um ambiente industrial, o tempo e a eficiência do trabalho são duas coisas muito importantes quando se trata da possibilidade de aumento das margens de lucro. Por essa razão, a maioria das grandes indústrias, quando observadas, revelam um processo de produção na maioria das vezes mais automatizado. Uma corporação moderna que possui a presença de sistemas automatizados, indubitavelmente, será mais produtiva do que uma que dependa da mão de obra humana para a execução de serviços, o que implica em obsolência, imprecisão e desequilíbrio em termos de competição de mercado. [INDÚSTRIA 2018]

Com base nos conceitos físicos e matemáticos necessários para a construção de um robô integrado a uma placa Esp32 e sensores infravermelhos, o presente trabalho apresenta a produção de Marvin um protótipo de veículo autônomo voltado para ambiente industrial. Fazendo uso das propriedades matemáticas do PID (Proporcional, Integral

e Derivativa) para o controle direcional, de forma que o protótipo responda da melhor maneira possível e de forma autônoma, com base nos dados captados pelos sensores de luminosidade e interpretados pelo microcontrolador.

Neste artigo, iremos visualizar seções sobre: os trabalhos que serviram como inspiração e base de conhecimento na área da robótica autônoma (Seção 2), a criação do protótipo MARVIN assim como seu desenvolvimento (Seção 3), os resultados alcançados ao fim de todo o trabalho realizado (Seção 4), a conclusão que podemos tirar desta obra (Seção 5) e enfim agradecimentos aos colaboradores, que contribuiram para o desenvolvimento desse projeto. (Seção 6).

2. Trabalhos Relacionados

Muitos protótipos buscam um microcontrolador que melhor se adeque às funcionalidades exigidas, uma grande parte faz uso do Arduino Uno. O trabalho de [Pereira 2014], desenvolvido pela Universidade Estadual do Piaui (UESPI), é um deles. Fazendo uso do Arduino, o autor descreve como a aplicação do PID pode ser utilizada nas competições de Robôs seguidores de linha. São utilizados componentes de leitura de sinais infravermelho conhecido como QTR-8A. Os sensores fazem a leitura de sinais baseado em seu distanciamento perante a linha que o robô deve seguir, o que facilita a sua movimentação no decorrer do trajeto e permite a verificação da eficiência se comparado aos outros métodos utilizados em competições. Para gerar o ajuste do robô com base no PID, Pereira deixa que claro que o resultado foi obtido com base em tentativa e erro partindo de adaptações no modelo *On/Off*. A fórmula obtida para o gerenciamento da movimentação do robô pode ser definida por:

$$PID_{value} = Kp * P + Ki + I + Kd * D \tag{1}$$

O autor define a função de suas variáveis baseado nas propriedades matemáticas do PID. Sendo que Kp refere-se a ganho proporcional, Ki tem ligação direta com o ganho integrativo e Kd com o ganho derivativo. Além disso, P trata da correção proporcional ao erro. A variável I é a correção proporcional ao erro x tempo. Por último, a variável D, referente a correção proporcional à taxa de variação do erro, é útil se o erro está variando muito rápido. Esta taxa de variação deve ser reduzida para evitar oscilações.

O autor, com base nos resultados, nota que o controle PID mostrou-se mais eficiente, partindo dos dados que obteve durante a fase de testes. Enquanto o robô não tinha o controle de PID implementado, usando apenas do algoritmo de *On/Off*, notou-se que o nível de oscilação era muito alto, o que gerava movimentos muito bruscos e, consequentemente, um tempo muito alto para a conclusão do circuito, isso quando o mesmo conseguia êxito.

Pereira, em seu trabalho, afirma também que a pesquisa pode servir de base para estudantes dos cursos técnicos e das engenharias, para entender como funcionam as técnicas de controle de processos na prática.

Seguindo a mesma linha de trabalho, o artigo de [Caitité 2018], desenvolvido na Universidade Federal de Minas Gerais aborda também a aplicação de algoritmo controlador PID em robôs seguidores de linha. O trabalho de Catité buscou tornar seu protótipo o mais similar possível com a forma que os carros transitam pela cidade. Com base nisso

o robô nomeado como “Uaile”, simula algumas atividades cruciais de trânsito que são aplicadas nas cidades.

O trabalho do autor desenvolve um protótipo de seguidor de linha com boa resposta dinâmica, baseando-se nos dados de posição, velocidade dos motores e posicionamento na pista tendo como base para análise o desempenho do robô.

Catité traz em seu artigo questões de mecânica do robô. Dessa forma, instrui que ao montar o robô existam algumas variáveis a serem consideradas, sendo elas: massa, localização de componentes, estrutura e incluindo também as características gerais do protótipo. Com base nessas condições, o autor afirma que o ideal de montagem é que o robô possua a menor quantidade de massa possível, pois assim consegue vencer a inércia com mais facilidade e, consequentemente, forçar menos o uso dos motores.

Catité opta pelo uso do PID pelo fato do mesmo ser um algoritmo que gera robustez e maior estabilidade à movimentação de seu protótipo. O mesmo refere-se ainda ao PID como um algoritmo aplicado em malhas fechadas, ou seja, o sistema está continuamente alimentando os dados de entrada com os dados de saída.

O autor ainda explica que, a variável que está sendo controlada tem seu valor da saída constantemente comparada com seu valor de acerto, ou *set point*. Dessa forma, a diferença entre os valores é denominada erro $[e(t)]$, sendo que o principal objetivo do PID é minimizar esse erro utilizando seus três fatores: P, I e D.

Com base nos dados gerados por erro e acerto pelo robô o autor obteve a seguinte fórmula para seu PID:

$$PID = Kp * e(t) + Ki * \int e(t)dt + Kd * \frac{de(t)}{dt} \tag{2}$$

Onde o autor definiu que Kp, Ki e Kd, são constantes do algoritmo e são definidas no projeto.

Para os outros fatores Catité trás as seguintes definições:

- O fator P é diretamente proporcional ao erro, assim sendo, $P = Kp * e(t)$. Este valor fica responsável por controlar a velocidade do ajuste, ou seja, a magnitude da mudança necessária na quantidade física para atingir o ponto de ajuste.
- O fator I é proporcional à integral do erro, dessa forma é a propriedade que faz a soma do erro no determinado momento. Logo, $I = Ki * \int e(t)dt$. Com essa definição obtida o termo fica responsável por definir a velocidade de resposta do sistema.
- O fator D é proporcional à derivada, podendo ser definida então por:

$$D = Kd * \frac{de(t)}{dt} \tag{3}$$

Dessa forma, é responsável por gerir a variação física para o controle de direção quando o algoritmo se encontra próximo ao ponto de ajuste.

O protótipo de Catité possui, na parte dianteira, 6 sensores IR (infra-vermelho) um ao lado do outro, responsáveis por fazerem as leituras da pista e fornecerem os dados

para o PID. Dessa forma, ao implementar o algoritmo, o robô teve um ótimo desempenho com base no que esperava o autor. Uma vez que para o mesmo, em linhas retas, o robô deve-se manter a maior parte do tempo recebendo sinais nos sensores centrais.

Ao analisar os dados do robô por conta de ter o PID e consequentemente o auto ajuste, em 100% do tempo o robô manteve-se recebendo sinais em seus 3 sensores centrais. O mesmo, quando aplicado em uma pista que possuía curvas, manteve o rendimento, sendo que só foi necessário utilizar dos sensores mais extremos por poucos segundos para que logo pudesse se ajustar. O que trouxe mais estabilidade, velocidade e consequentemente um movimento mais suave e com erros mínimos.

3. Trabalho Desempenhado

Para o desenvolvimento do Marvin, como foi chamado o protótipo, procurou-se encontrar em algumas frentes de trabalho existentes algo que tivessem alguma similaridade com o que iria ser produzido nesse trabalho. Dessa forma, encontrou-se nos robôs seguidores de linha a similaridade e foi decidido que esta seria a base para a produção do protótipo. Os robôs seguidores de linha são uma categoria de robôs caracterizados por seguir uma linha que demarca seu trajeto.

Para a produção do protótipo, tendo em vista a vasta quantidade de microcontroladores no mercado, optou-se pelo uso do ESP32, que vem sendo muito utilizado na produção de protótipos que contemplam a temática da *IoT(Internet of Things)*.

3.1. Construção do Chassi

Para a construção do chassi, optou-se pelo uso do papelão, pelo baixo custo, leveza, facilidade de manipulação e disponibilidade, além do fato dos professores a que temos convívio na instituição de ensino a qual estamos vinculados (Instituto Federal do Paraná - Paranavaí) utilizarem de uma metodologia de ensino conhecida como Metodologia *STEAM (Science, Technology, Engineering, Arts and Mathematics)*. Basicamente, a ideia dessa metodologia é instruir os estudantes nas variadas áreas do conhecimento existente, além de trabalhar conceitos fundamentais que preparam o mesmo para o mercado de trabalho e a vida em cidadania. Os componentes eletrônicos que compõem o protótipo foram fixadas no papelão. Dentre os componentes fixados, cabe aqui citar: Esp 32(microcontrolador), motores, sensores infravermelho(TCRT5000) e Ponte H.

O módulo TCRT5000 é principalmente aplicado no desenvolvimento de projetos de “robôs seguidores de linha”. O sensor possui um LED que emite luz infravermelha e um fototransistor sensível a essa luz, onde é possível fazer a detecção de uma linha por meio da variação de luz infravermelha refletida pela superfície onde está a linha. Para um melhor desempenho com relação a leitura dos dados vindos do infravermelho, foram utilizados 6 sensores. Os mesmos foram fixados bem próximo um do outro, para que assim a leitura da linha seja a melhor e mais exata possível.

Com isso, foi possível montar o chassi do protótipo que é representado pela Figura 1. Na foto, também pode-se notar a presença de um módulo STEP-DOWN, que atua regulando a tensão que sai da fonte de alimentação do protótipo e vai direto para as entradas de alimentação do ESP32 e da Ponte H que controla os motores. Na Figura 1, é possível notar também todos os *jumpers* que ligam os sensores às portas de entrada do ESP32,

juntamente com as duas *protoboards* utilizadas para facilitar o manuseio das pinagens do microcontrolador.

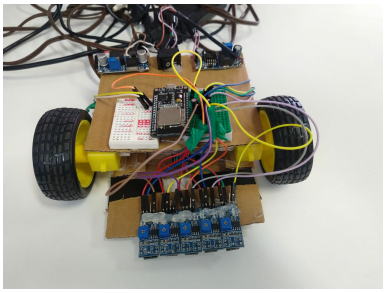

Figura 1. Atual Estrutura do Robô Fonte: Autor

3.2. Criação de um Algoritmo Base

A ideia básica de um robô seguidor de linha é que o robô consiga completar um trajeto seguindo uma linha branca sem que saia da sua rota ou se perca e acabe sem direção. A criação do primeiro algoritmo foi resultado de muitos testes e ajustes. Sua formulação se deu em etapas conforme a evolução que Marvin tinha nos testes impostos.

Nos primeiros testes, o protótipo tinha apenas de seguir uma linha reta sem que perdesse a sua direção Para essa etapa proposta o protótipo se saiu muito bem, porém passou a se apresentar de forma instável conforme foram adicionadas as funcionalidades para que o mesmo pudesse fazer curvas, fossem elas abertas ou fechadas, encruzilhadas, dentre outros.

Essa inconsistência no desempenho apresentada pelo protótipo se dava pelo algoritmo utilizado no primeiro momento, conhecido como *"On/Off"*. O modelo de *"On/Off"* tem a característica de fazer movimentos sem ajustes, baseado em movimentos bruscos ao ver a linha, ou seja, quando o sensor da direita vê a linha branca o mesmo vira bruscamente para a direita. A mesma condição é válida para o sensor do lado inverso, onde o robô só anda para frente se estiver 100% centralizado.

Com este algoritmo, são realizados movimentos muito bruscos e a cada leitura realizada o protótipo se perde facilmente de sua rota. Outro ponto negativo desse modelo é a baixa autonomia e grande quantidade de tentativa e erro, uma vez que cada falha do robô é necessário alterar os valores referentes à potência do motor e testar novamente.

3.3. Restruturação do Algoritmo Base: Implementação do PID e Máquina de Estados Finitos

Após constatado a inviabilidade do primeiro código, por conta da forma que o protótipo se locomovia, passou-se então a procurar formas alternativas para tornar seus movimentos mais sutis, para que o robô pudesse seguir seu trajeto sem que houvesse falhas durante o percurso. Com base em algumas leituras sobre robôs seguidores de linha, vários relatos foram encontrados sobre o uso da ferramenta matemática PID (controlador proporcional integral derivativo), que usa como base seu funcionamento os princípios matemáticos de proporcionalidade, integral e derivada. Controladores PID são controlados pela seguinte equação:

$$u(t) = k_p * e(t) + k_i * \int_0^t e(t)dt + k_d \frac{de(t)}{dt} \quad (4)$$

Tem-se, por definição, que u(t) é o sinal de saída, usado para controlar o motor; e(t) é referente ao erro atual; kp, ki e kd dão permissão para atribuir pesos para o erro atual, erro passado (integral) e previsão de erro futuro (derivada).

Ao fazermos os primeiros testes, notou-se que, para o problema de ajuste de movimento, a variável mais importante era a que se referia ao erro atual. Sendo assim, configurou-se os parâmetros ki e kd como 0 (zero) e assim obteve-se um bom controle do robô.

Mesmo com essa melhora realizada pelo controle PID, ainda ocorria do robô se perder, pelo fato de não saber onde estava em relação a linha. Dessa forma, a segunda implementação que foi adicionada ao código foi um algoritmo de Máquina de Estados Finitos (modelo de código utilizado para formular inteligências artificiais). Essa funcionalidade foi adicionada para que o protótipo ficasse de certa forma mais inteligente e autônomo.

Para o algoritmo da Máquina de Estados, foram criados os seguintes estados para marcar a posição do protótipo com relação a linha que o mesmo deve seguir (A Figura 2 apresenta de forma ilustrativa o fluxo da máquina de estados implementada):

POS CENTER O robô está aproximadamente no centro da trajetória).
POS LEFT O robô está um pouco à esquerda da trajetória).
POS FULL LEFT O robô saiu completamente da trajetória pelo lado esquerdo. Logo, deve ignorar o PID e forçar uma curva à direita).
POS RIGHT O robô está um pouco à direita da trajetória).
POS FULL RIGHT O robô saiu completamente da trajetória pelo lado direito.

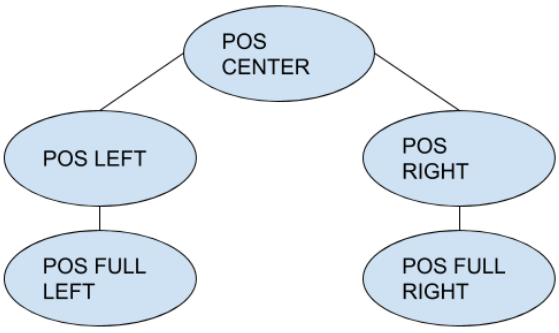

Figura 2. Fluxo Máquina de Estados

3.4. IOT: Uso do Protocolo MQTT

A ideia do uso por trás da Internet das coisas tem como fundamentação a nova dimensão propiciada pela Internet, que possibilita a comunicação a qualquer tempo e em qualquer lugar. Isso não é válido apenas para a comunicação entre pessoas, mas também na

comunicação entre objetos eletrônicos utilizando como meio protocolos de comunicação e a internet. [Diniz 2006]

O protocolo MQTT é um dos protocolos de rede para troca de mensagens utilizados na *IoT*, sendo um dos mais recomendado para a comunicação de forma otimizada em redes TCP/IP de baixa confiabilidade, baixa banda e alta latência. O protocolo foi desenvolvimento para exigir poucos recursos de hardware e para rodar em dispositivos embarcados mais simples.

O protocolo se caracteriza por possuir um servidor conhecido como *broker*, que é responsável por receber, enfileirar e disparar as mensagens recebidas dos *publishers* (quem envia as mensagens) para os *subscribers* (quem recebe as mensagens). [Borba 2018]

Usando como auxílio o protocolo MQTT, foi possível melhorar o desempenho do robô e o desenvolvimento do código final, uma vez que foi possível ter maior entendimento de quais pontos Marvin apresentava falhas para que pudessem ser corrigidas.

3.5. O código Final

Com base na definição da fórmula base de PID para o protótipo e também na implementação do algoritmo da Máquina de Estados ao protótipo, foi possível verificar os resultados dos testes aplicados em Marvin.

Para o controle PID funcionar, é necessário transformar o valor retornado pelos sensores em uma medida que se refere a distância que o robô se encontra do centro da linha da trajetória. Para tal, utilizamos a biblioteca QTRSensors[1]. Com esta biblioteca configurada, as portas dos sensores infravermelhos retornam um valor que representa o quão distante do centro da trajetória o robô está.

O metódo *"Setup()"*, é responsável por iniciar e configurar os componentes e portas que são utilizados no protótipo. Dentro do mesmo, foi implementada a função de calibragem. Esse processo leva cerca de 8 segundos e é realizado pelo comando "qtr.calibrate()". Essa função utiliza as propriedades da biblioteca *QTRSensors* baseado nos dados retornados pelas leituras que os sensores estão fazendo da pista. O algoritmo de calibragem dos sensores foi implementado da seguinte maneira:

```
1  void setup() {
2  ...
3    qtr.setTypeAnalog();
4    qtr.setSensorPins((const uint8_t[]){esquerda_e,esquerda,centro,
       direita, direita_d}, SensorCount);
5    delay(500);
6    pinMode(2, OUTPUT);
7    digitalWrite(2, HIGH);
8
9    for (uint16_t i = 0; i < 8000; i++) {
10     qtr.calibrate();
11   }
12
13   digitalWrite(2, LOW);
14
```

[1]Disponível no site https://www.arduinolibraries.info/libraries/qtr-sensors

```
  for (uint8_t i = 0; i < SensorCount; i++) {
    Serial.print(qtr.calibrationOn.minimum[i]);
    Serial.print(' ');
  }

  Serial.println();

  for (uint8_t i = 0; i < SensorCount; i++) {
    Serial.print(qtr.calibrationOn.maximum[i]);
    Serial.print(' ');
  }
}
```

Para a aplicação do PID juntamente com o algoritmo de máquina de estados foi implementada uma função chamada de *"void line_follow()"*. Essa função foi criada com função de encapsular e guardar todas as configurações e procedimento que o robô deve fazer ao seguir a linha.

Quando o método de seguir linha é chamado, o robô faz as leituras e, baseado no PID, transforma esses valores em estimativa de erro. O valor de erro é gerado dentro da variável "error", que faz uso da função *"qtr.readLineWhite()"*, função essa que utiliza da biblioteca *QTRSensor* para transformar os valores recebidos pelo sensores em distância. Dessa forma, o valor retornado em *"error"* é aplicado na variável *"direction"*, que é responsável pelo cálculo de auto-ajuste do robô quanto à aceleração dos motores e movimentação.

Com base nos valores retornados, o processamento da máquina de estados é iniciado, onde foi definido os estados que são usados pelo robô durante a execução do algoritmo. Dessa forma, os estados vão acompanhando e sendo trocados conforme a posição que o robô toma ao decorrer do circuito.

Ao final de tudo, a variável *"direction"* é aplicada, uma vez que a mesma contém o controle de direção de Marvin. Esta direção é então transformada em potência para os motores. Por exemplo, se precisamos virar para a direita, a potência do motor esquerdo é configurada para ser maior que a potência do motor direito. A diferença entre as potências dos motores é proporcional a quanto o robô precisa virar para corrigir sua rota.

```
void line_follow(){
  unsigned long time;

  int left, right, center, left_left, right_right;


  time = millis();

  error = 2000 - qtr.readLineWhite(sensorValues);


    direction = (int)((-255.0f*(float)(error)) / MAXIMO);

  //PRETO VALORES PROXIMOS DE 1
  if(error < 2000){

  }
```

```
  dprint(" , ");
  dprint(direction);
  dprint(" , ");
  dprint(error);
  dprint(" , ");

  //print ("estado:", state, "esquerda:", left, "direita:", right , "
   error:", error)

   switch (state) {

    case POS_CENTER: { //iniciando curva a direita
      if (error > 500) {
        state = POS_RIGHT;
      }
      else if (error < 50){
        state = POS_LEFT;
      }
      break;
    }

    case POS_LEFT:{
      if (error < -1500){
        state = POS_FULL_LEFT;
      }
    else if (error <= 500){
        state = POS_CENTER;
      }
      break;
    }

    case POS_FULL_LEFT:{
      direction = 255;
      if (error < 50){
        state = POS_LEFT;
      }
      break;
    }

    case POS_RIGHT: {
      if (error > 1500) {
        state = POS_FULL_RIGHT;
      }
      else if (error <= 500) {
        state = POS_CENTER;
      }
      break;
    }

    case POS_FULL_RIGHT:{
      direction = -255;
      if (error <= 1500) {
        state = POS_RIGHT;
      }
    }
  }
```

```
73
74    if (direction > 255)
75      direction = 255;
76    else if (direction < -255)
77      direction = -255;
78
79       if (direction < 0){   //# queremos virar a esquerda
80        direction = -direction ;
81        if (direction < 5){
82          //com 5 funciona
83          eng_left.set_boost(1.0);
84          eng_right.set_boost(1.0);
85        }
86        else{
87          eng_left.set_boost(0.9);
88          eng_right.set_boost(0.9);
89        }
90        eng_left.set_speed(255 - direction);
91        eng_right.set_speed(255);
92      }
93      else{ //# queremos virar a direita
94        //com 5 funciona
95         if (direction < 5){
96          eng_left.set_boost(1.0);
97          eng_right.set_boost(1.0);
98        }
99        else{
100         eng_left.set_boost(0.9);
101         eng_right.set_boost(0.9);
102       }
103
104       eng_right.set_speed(255 - direction);
105
106        eng_left.set_speed(255);
107     }
108  }
109 }
```

Para melhor organização e seguindo a forma de funcionamento da IDE Arduino, existe uma função *"void loop()"*, função essa que é chamada continuamente pelo micro-controlador. O *"void loop()"* é a função principal de funcionamento do algoritmo, ou seja, enquanto o protótipo estiver ligado após executar a função de *setup()* ficará em repetição infinita executando essa rotina.

Dessa forma, dentro da função *loop()*, foi encapsulada as rotinas principais que são executadas durante todo o funcionamento do robô. A primeira chamada realizada é a função *"line_follow()"*, método que faz o cálculo do PID e o tratamento da máquina de estados no funcionamento do protótipo permitindo então que o robô se movimente com base no auto-ajuste implementado.

Na sequência, existem alguns trechos de código, que estão comentados, e que são ativados quando há a necessidade de fazer a inspeção do que vem sendo retornado pelo robô, utilizando como intermediário o protocolo MQTT que quando configurado envia as mensagens de retorno ao dispositivo vinculado, seja ele computador e *smartphones* ou

tablets.

```
void loop() {
  digitalWrite(2, HIGH);
  /*
  if (!client.connected()) {
    reconnect();
  }*/

  //client.loop();

  line_follow();
  Serial.println(error);


  /*char tempString[8];
  char tempString2[8];

  dtostrf(direction, 1, 2, tempString);
  dtostrf(error, 1, 2, tempString2);

  client.publish("esp32/pid", tempString);
  client.publish("esp32/error", tempString2)
}
```

O código fonte de Marvin com estas configurações apresentou melhor rendimento para o robô, de forma que toda a interface de controle para o PID ficou mais prática e manteve a robustez de todo algoritmo. Isto traz ao protótipo maior autonomia em seu auto-ajuste enquanto seguindo o circuito.

4. Resultados

O objetivo de fazer Marvin seguir seu trajeto sem falha foi alcançado. De forma que os resultados que o mesmo obteve são animadores para que em trabalhos futuros possa ser utilizado o que foi implementado como base. Após a fase de testes, baseada não apenas em tentativa e erro, mas também em como o protótipo se adaptava a cada nova alteração em seu algoritmo, trajetória e luminosidade, o mesmo passou a cada vez mais se sobressair e alcançar eficiência com relação ao seu propósito inicial.

Com base nos resultados obtidos nos testes, a autonomia que foi dada ao protótipo com a implementação do PID e máquina de Estados fez com que ele conseguisse fazer ajustes em segmentos ou curvas em seu trajeto da forma mais sutil possível. Este feito é comemorável tendo em vista o fato de que se o mesmo estivesse transportando uma carga, o risco dela ser avariada durante o transporte seria mínimo.

O auto-ajuste que foi permitido ao protótipo com o algoritmo de PID, que fez com que o mesmo tenha um rendimento elevado em retas, oportunidade em que o mesmo mantém-se a maior parte do tempo no estado “POS_CENTER”, ou seja, mantém-se a maior parte do tempo alinhando e no centro da pista. Da mesma maneira, quando o protótipo se aproxima de curvas acentuadas rapidamente entra nos Estados “POS_FULL_LEFT” e “POS_FULL_RIGTH”, conseguindo se alinhar utilizando os dois sensores mais próximos do sensor central, logo não se distanciando tanto do centro, o que mantém a boa performance e velocidade no trajeto.

Com o uso do MQTT, foi possível gerar uma comunicação do microcontrolador com o celular/computador. Assim, foi possível manter relatórios de posicionamento e localização do protótipo sem uso de cabos. Essa estratégia permitiu saber a causa do erro ou em que status da máquina de estados o mesmo se encontrava, facilitando o controle e gerência do protótipo.

5. Conclusão

Este artigo apresentou o desenvolvimento de um protótipo de um robô autônomo voltado para o ambiente industrial. Durante o desenvolvimento, foram implementados algoritmos de processamento lógico juntamente com princípios matemáticos de PID para ajustar os movimentos do robô enquanto o mesmo executava o trajeto pré-definido.

Pode-se concluir que a aplicação do algoritmo baseado no uso do PID em sistemas embarcados de robôs autônomos sobressai-se sobre outros algoritmos, apresentando robustez e melhora no movimento que o robô faz durante o percurso. Movimentos mais sutis e eficientes, além de agilidade do protótipo, foram garantidos por essa pesquisa, de forma que tal aplicação, indubitavelmente, servirá como base para que trabalhos futuros, na mesma linha de pesquisa, possam ser prospectados. Considerando uma aplicação em uma indústria, os testes sugerem melhor rendimento uma vez que o robô de transporte poderia fazer várias viagens o mais rápido possível, alcançando maior produtividade.

Adicionalmente, conclui-se que o uso da máquina de estados, juntamente com o protocolo MQTT, possibilita maior autonomia ao robô uma vez que o mesmo consegue guardar as informações de estados que se encontra e por meio do protocolo de comunicação manter uma base do que o mesmo vem lendo nos sensores para monitoramento de seu funcionamento.

6. Agradecimentos

Os autores agradecem O CNPq e o IFPR pelo apoio em forma de bolsa de iniciação científica.

Referências

Borba, B. d. (2018). *Serviço de descoberta para o protocolo MQTT em um sistema de monitoramento de grupos motor-gerador baseado em internet das coisas*. PhD thesis.

Caitité, V. G. R. (2018). Robô seguidor de linha empregando controle pid. Disponivel em: <http://sistemaolimpo.org/midias/uploads/c9265a4069821ae565485c39b07d0902.pdf>. Acesso em: 15/07/2020.

Diniz, E. H. (2006). Internet das coisas. *GV executivo*, 5(1):59.

INDÚSTRIA, P. D. (2018). Robotista: a profissão chave da indústria 4.0 - agência cni de notícias. Disponivel em:https://noticias.portaldaindustria.com.br/noticias/educacao/robotista-a-profissao-chave-da-industria-40/. (Accessed on 10/06/2020).

Pereira, L. M. M. (2014). Robô de competição categoria seguidor de linha utilizando algoritimo pid e plataforma arduino. *Universidade Estadual do Piauí*. Acesso em: 10/07/2020.