



CAMPUS SUZANO

**Adaptação do projeto físico e a programação da plataforma
Zumo para participação em competições de robótica**

ALUNO: Elton Cardoso do Nascimento

ORIENTADORA: Prof.^a Dra. Vera Lúcia da Silva

**Coorientadores: Prof. Esp. Raphael Antonio de Souza e
Prof. Me. Masamori Kashiwagi**

Dezembro, 2018

Dados Internacionais de Catalogação na Publicação (CIP)

Preparada pelo Serviço de Biblioteca e Informação do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo campus Suzano

Nascimento, Elton Cardoso do.

Adaptação do projeto físico e a programação da plataforma Zumo para participação em competições de robótica / Elton Cardoso do Nascimento. -- Suzano, 2018.

Trabalho de Conclusão de Curso (Técnico) -- Curso Técnico Integrado ao Ensino Médio em Automação Industrial do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo - Campus Suzano.

Orientadora: Profa. Dra. Vera Lúcia da Silva

Descritores: 1. Robótica. 2. Robô Zumo. 3. Competições de robótica. 4. Controle PID. 5. Algoritmo genético I. Título.

IFSP/SZN/SBI-33/18

RESUMO

Este trabalho apresenta continuação da implementação dos programas de controle e da adaptação de um robô, “Zumo Robot”, para competições de robótica na modalidade Busca e Resgate. O projeto do robô visa superar os desafios propostos por competições de robótica, como a OBR (Olimpíada Brasileira de Robótica). Foram desenvolvidos programas para o controle de variáveis, mapeamento robótico e análise de dados, utilizando o controlador PID (proporcional-integral-derivativo) e técnicas de computação evolutiva, como o algoritmo genético. Também ocorreram modificações na mecânica do robô, utilizando impressão 3D, na eletrônica com adaptação de sensores e atuadores e na busca de uma bateria adequada para alimentação da robô.

Palavras-chave: Robótica, *Robô Zumo*, *Competições de Robótica*, *Controle PID*, *Algoritmo Genético*.

ABSTRACT

This work presents a continuation of the implementation of control programs and the adaptation fo a robot “Zumo Robot”, for robotics competitions in serach and rescue mode. The robot projet aims to overcomo the challenges posed by robotics competitions, such as the OBR (“Olimpíada Brasileira de Robótica). Programs were developed for variable control, robotic mapping and data analisys, using PID (proportional-integral-derivative) controller and evolutionary computing techniques, such as the genetc algorithm. There were also modifications in robot mechanics, using 3D printing, in electronics with the adaptation of sensors and actuators and in the search foa a suitable battery to feed the robot.

Keywords: *Robotics, Zumo Robot, Robotics Competitions, PID Control, Genetic Algorithm.*

Elton Cardoso do Nascimento

**Adaptação do projeto físico e a programação da plataforma Zumo para
participação em competições de robótica**

Projeto Integrador do curso Técnico Integrado ao Ensino Médio em Automação Industrial
apresentado no Instituto Federal de Educação, Ciência e Tecnologia – Campus Suzano

Aprovado em: 20/12/ 18

Banca Examinadora



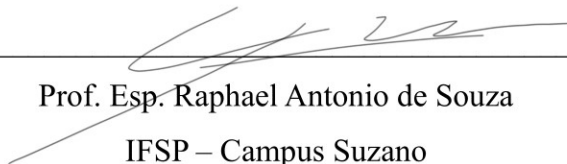
Prof.^a Dra. Vera Lúcia da Silva (Orientadora)

IFSP – Campus Suzano



Prof. Me. Alessandro Emilio Teruzzi

IFSP – Campus Suzano



Prof. Esp. Raphael Antonio de Souza

IFSP – Campus Suzano



Andreia de Almeida

IFSP – Campus Suzano

Agradecimentos

Agradeço a minha orientadora

Prof.^a Dr. Vera Lúcia da Silva

Agradeço a meus co-orientadores

Prof. Me. Masamori Kashiwagi

Prof. Esp Raphael Antonio de Souza

Agradeço a meus companheiros de equipe

Gabriel Freitas Yamamoto – Claudio Sasaki Calanca – Alvaro Coelho Jesus - Adriano de Oliveira
Lacerda – Gabriel Augusto Santos Santhiago – Ana Luiza Santos Coura

Agradeço aos servidores

Renato de Paula Cabral – Efraim Caetano dos Santos - Douglas

Agradeço a todos outros docentes

Que eventualmente me ajudaram nesse projeto

Adriel – Alessandro - Antonio M. - Antonio N. - Breno - Carlos – Cleide -Wagner

Agradeço a meus pais

Maria Luisa do Nascimento – Vasco Cardoso do Nascimento

SUMÁRIO

Sumário

RESUMO	2
ABSTRACT	3
1 - Capítulo 1: Introdução e Objetivos	9
1.1 - Introdução	9
1.2 - Objetivos	9
2 - Capítulo 2: Fundamentação Teórica	10
2.1 - Robótica, Robôs e Robôs móveis.	10
2.2 - Arduino	10
2.3 - Robô Zumo	11
2.4 - Controle PID	11
2.4.1 - Indicador de Desempenho	12
2.4.2 - Integral <i>WindUp</i>	12
2.5 - Sensores	12
2.5.1 - Sensor Ultrassônico HC-SR04	12
2.5.2 - Sensor Inerciais: Acelerômetro e Giroscópio	12
2.5.3 - Sensor <i>laser</i> ToF VL53L0X	13
2.5.4 - Encoder	13
2.6 - Algoritmo Genético	13
2.6.1 - Seleção por Roleta	14
2.6.2 - Cruzamento Uniforme	14
2.6.3 - Mutação Indutiva	14
3 - Capítulo 3: Competições de Robótica	15
3.1 - Competição Teórica: a OBR	15
3.2 - Competições Práticas: a OBR e o TRIF	15
3.2.1 - Divisão de salas	15
3.2.2 - Encruzilhadas e curvas	15
3.2.3 - Redutores de Velocidade	16
3.2.4 - GAPs	16
3.2.5 - Obstáculos	17
3.2.6 - Desafio Surpresa	17
3.2.7 - Rampa	17
3.2.8 - Sala 3	17
3.3 - Participações em Competições de Robótica	17
4 - Capítulo 4: Trabalho Anterior	18
4.1 - Alterações Físicas e Eletrônicas no Robô Zumo	18
4.2 - Programação anterior	18
4.2.1 - Sensores de Refletância	19
4.2.2 - Sensores inerciais	19
4.2.3 - Classes para atuadores	19
4.2.4 - Controle PD para a linha e GAPs	20

4.2.5 - Controle da movimentação	21
4.2.6 - Superando curvas	21
4.2.7 - Obstáculo	22
4.2.8 - Detecção da rampa	22
4.3 - Problemas encontrados	23
5 - Capítulo 5: Alterações Físicas no Robô	24
5.1 - Alimentação	24
5.2 - Rampa	24
5.3 - Sensores ToF	27
5.4 - Garras	27
5.5 - Remoção dos sensores ultrassônicos laterais	30
6 - Capítulo 6: Alterações no Código	31
6.1 - Alteração do Ambiente de Desenvolvimento e Organização do Código	31
6.2 - Sensores Inerciais e PID	31
6.2.1 - Tentativas de melhoria das leituras	31
6.2.2 - PIDs	34
6.3 - Outras Alterações	34
6.3.1 - Detecção de redutores	34
6.3.2 - Função Obstáculo	35
7 - Capítulo 7: Criando uma solução para a 3ª sala do percurso	36
7.1 - Detectando as Vítimas	36
7.2 - Detectando a Zona de Resgate	38
8 - Capítulo 8: Sintonizando o Controlador PID com Algoritmo Genético	45
8.1 - Definições do Algoritmo Genéticos	45
8.2 - Implementando para sintonizar o controlador	45
8.3 - Testes com a aproximação de uma função	46
8.4 - Testes com o pidGiro	48
9 - Capítulo 9: Resultados e Conclusão	51
9.1 - Resultados	51
9.2 - Conclusão	52
9.2.1 - Melhorar a superação de Obstáculos	52
9.2.2 - Melhoras a detecção de curvas	52
9.2.3 - Criar uma solução para a 3ª sala do percurso	52
9.2.4 - Estudar melhorias para o tratamento dos sinais dos sensores inerciais	52
9.2.5 - Estudar os sinais obtidos pelo sensor de cor	53
9.2.6 - Encontrar uma solução para superar o redutor, com alterações mecânicas no robô	53
9.2.7 - Utilizar técnicas de IA (algoritmo genético) para sintonizar o controlador PID	53
9.2.8 - Problemas detectados e possíveis soluções em trabalhos futuros	53
10 - Capítulo 10: Caráter Integrador do Projeto	55
Referências Bibliográficas	57
Apêndice A: Código Principal	59
Apêndice B: Código para Processamento dos Dados da Zona de Resgate	60
Apêndice C: Algoritmo Genético - Biblioteca	74
Cabeçalho (.h):	74
Implementação (.cpp)	75

Índice de figuras

Figura 1: Encruzilhadas Possíveis na OBR/TRIF - Fonte: OBR, 2017, p. 10.....	16
Figura 2: Gráfico - Acelerômetro na transição plano para rampa.....	23
Figura 3: Foto frontal da rampa impressa.....	25
Figura 4: Foto lateral da rampa impressa.....	26
Figura 5: Foto da rampa impressa.....	26
Figura 6: Alocação do sensor laser lateral.....	27
Figura 7: Foto da garra impressa.....	28
Figura 8: Foto da dilatação dos elásticos.....	29
Figura 9: Foto da vítima dentro da garra.....	29
Figura 10: Foto do pegador de vítimas novo.....	30
Figura 11: Gráfico das distâncias medidas x calculadas.....	33
Figura 12: Gráfico do teste do encouder experimental.....	33
Figura 13: Gráfico - Leituras da vítima pelo sensor de distância.....	36
Figura 14: Gráfico - Variações das leituras da vítima.....	37
Figura 15: Gráfico - Leituras e Variações da Vítima.....	37
Figura 16: Gráfico - Leituras da sala 3 - Forma Polar.....	38
Figura 17: Gráfico - Mapa da sala 3.....	39
Figura 18: Gráfico - Filtragem do mapa da sala 3.....	40
Figura 19: Gráfico - Geração de retas da sala 3.....	41
Figura 20: Gráfico - Pontos não utilizados da sala 3.....	42
Figura 21: Gráfico - Mapa 2.....	43
Figura 22: Gráfico - Mapa 3.....	43
Figura 23: Gráfico - Mapa 4.....	44
Figura 24: Gráfico - Melhores Aptidões: Aproximação da Função Quadrática.....	47
Figura 25: Gráfico - Melhores Genes: Aproximação da Função Quadrática.....	47
Figura 26: Gráfico - Melhores aptidões do algoritmo genético.....	49
Figura 27: Gráfico - Melhores genes do algoritmo genético.....	50

1 - Capítulo 1: Introdução e Objetivos

1.1 - Introdução

Este relatório foi escrito com o propósito de descrever o trabalho de complementação da pesquisa sobre robôs móveis e a participação em competições de robótica, realizada no grupo de robótica do IFSP Suzano durante o ano de 2017 e 2018. Este trabalho também será validado como projeto integrador do curso técnico integrado em automação industrial.

O relatório descreve o resultado da pesquisa. Na primeira parte contém um levantamento teórico das informações relevantes a esse trabalho. O capítulo 4 apresenta o levantamento dos resultados do trabalho anterior, as conclusões e a identificação de melhorias no projeto do robô.

No capítulo 5, descreve-se a proposta de alterações no robô, com objetivo de cumprir com as necessidades levantada.

No capítulo 6, as novas alterações na programação são mostradas e explicadas.

No capítulo 7, são apresentadas as soluções encontradas para superar a 3ª sala do percurso proposto pelas competições de robótico.

No 8º capítulo, as atividades realizadas em relação ao uso de técnicas de programação evolutiva, em específico o algoritmo genético, para sintonizar os algoritmos PIDs são expostas.

No 9º capítulo são discutidos os novos resultados obtidos, as conclusões e novas necessidades encontradas.

Por fim, é realizada uma reflexão sobre o caráter integrador do projeto no décimo e último capítulo.

1.2 - Objetivos

O projeto tem como objetivos a superação dos desafios propostos pela OBR; o aprendizado e aplicação de temas relevantes a robótica e a automação industrial, como o controle de processos e algoritmo genético, adquiridos por aulas ao decorrer do curso e pela própria pesquisa; concluir o trabalho já realizado sobre o robô Zumo

2 - Capítulo 2: Fundamentação Teórica

2.1 - Robótica, Robôs e Robôs móveis.

Define-se robótica como a “Ciência e técnica que envolve a criação, a construção e a utilização de robôs” (ROBÓTICA, 2017), sendo que “um robô é um manipulador reprogramável e multifuncional projetado para mover materiais, peças e ferramentas ou dispositivos especializados através de movimentos variáveis e programados para o desempenho de uma variedade de tarefas”¹(MCMILLAN, 1999, Section 9.123, tradução própria).

Segundo o Dr. Humberto Secchi (2012), os robôs podem ser classificados em três tipos: industriais, médicos e moveis; sendo para este relatório relevante apenas os robôs móveis. Robôs móveis são os robôs capazes de se transportar automaticamente, possuindo em si um sistema de localização.

Um exemplo de robô móvel é o robô seguidor de linha, o robô solicitado pela OBR e outras diversas competições. O robô deve seguir uma linha pré colocada em uma superfície. A OBR ainda, em sua competição, inclui obstáculos no meio do trajeto, como falhas na linha, redutores de velocidade, objetos, inclinações, entre outros.

2.2 - Arduino

Arduino é uma plataforma de desenvolvimento livre baseada em hardwares e softwares de uso livre. Possui uma IDE própria, e é amplamente utilizado por estudantes, artistas, hobistas, artistas, programadores e profissionais (ARDUINO, [2017]).

Possui vários tipos de placas, “as opções vão das mais comuns, como o Arduino Uno e suas 14 portas digitais e 6 analógicas, passando por placas com maior poder de processamento, como o Arduino Mega, com microcontrolador ATmega2560 e 54 portas digitais” (THOMSEN, 2014).

1 Traduzido do original: “A robot is a reprogrammable, multifunctional manipulator designed to move materials, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks”

2.3 - Robô Zumo

O Robô Zumo, robô utilizado como base para este projeto, é “uma plataforma robótica com esteira controlada por Arduino [...]. É incluso dois micromotores com redução, acompanhados de um par de esteiras de silicone, uma lâmina com formato de escavadeira de aço inoxidável, um conjunto de seis sensores de refletância com infravermelho para seguir linha ou para detecção de bordas, um buzzer para sons simples e música, um acelerômetro, giroscópio e magnetômetro de três eixos para detecção de impacto e orientação”² (POLOLU, [2017], tradução própria).

2.4 - Controle PID

Os controles PID são amplamente utilizados em indústrias para o controle de sistemas. É composto por três coeficientes: o proporcional (kp), o integral (ki) e o derivativo (kd) (NI, 2011).

Sua equação pode ser descrita como: $[kp \times e(t)] + [ki \times \int e(t) dt] + [kd \times \frac{de}{dt}]$. (UNIVERSITY OF MICHIGAN, [2017]). Sendo que $e(t)$ é o erro em função do tempo e t_x é o tempo atual.

Porém em um sistema discreto, a equação pode ser redefinida como

$[kp \times e(t_x)] + [ki \times \sum_{t=0}^{t_x} e(t) T] + [\frac{kd}{T} \times (e(t_x) - e(t_{x-1}))]$, sendo T o período de amostragem e t_{x-1} o tempo anterior (BOLTON, p. 440-442 ,1995). Em um sistema discreto com período de amostragem = 1, como no caso de robôs onde o período de amostragem é medido em nº de execuções do programa, pode ser redefinida como:

$$[kp \times e(t_x)] + [ki \times \sum_{k=t_0}^{t_x} e(k)] + [kd \times (e(t_x) - e(t_{x-1}))]$$

2 Traduzido do original: “an Arduino-controllable tracked robot platform [...]. It includes two micro metal gearmotors coupled to a pair of silicone tracks, a stainless steel bulldozer-style blade, an array of six infrared reflectance sensors for line following or edge detection, a buzzer for simple sounds and music, a 3-axis accelerometer, magnetometer, and gyro for detecting impacts and tracking orientation.”

2.4.1 - Indicador de Desempenho

Caso seja preciso comparar diferentes tipos de controles, é possível utilizar critérios de desempenhos, como a “Integral do erro quadrado multiplicado pelo tempo” (ITAE – equação I), em que erros negativos não anulam os positivos, dando maior peso a erros maiores e a erros posteriores (GARCIA, 2018). Dessa forma, os erros em momentos posteriores, em que o controlador já deveria ter ajustado a variável, tornam-se penalizados.

2.4.2 - Integral WindUp

O “integral *windup*” é um fenômeno que ocorre quando o controlador integral continua aumentando, mesmo que a saída do controlador esteja saturada, ou seja, tenha chegado a seu limite. A integral começa então a diminuir com a inversão do sinal do erro, porém demora, o que gera um sobressinal, levando a uma demora para atingir o *setpoint* (GARCIA, 2018).

2.5 - Sensores

2.5.1 - Sensor Ultrassônico HC-SR04

O sensor ultrassônico HC-SR04 é capaz de medir distâncias de 2 a 400 cm, a partir de um circuito transmissor, receptor e controlador (ELEC FREAKS, [2017]).

2.5.2 - Sensor Inerciais: Acelerômetro e Giroscópio

Sensores inerciais são sensores capazes de calcular a forças aplicadas sobre um objeto. Entre eles destacam-se o acelerômetro, que calculam a aceleração sobre o objeto, e o giroscópio, que calcula a velocidade de rotação de um objeto. Integrando a aceleração sobre o objeto, é possível saber a velocidade e o deslocamento do mesmo, porém sensores acelerômetros costumam ter muitos ruídos, necessitando de filtros, como o filtro de Kalman, e calibrações (ALMEIDA, 2014).

É possível também integrar a velocidade angular de um objeto para obter a quantidade de graus rotacionados.

2.5.3 - Sensor *laser* ToF VL53L0X

O sensor Time-of-Fly VL53L0X é um sensor de distância que (semelhante ao ultrassônico) mensura o tempo que um pulso laser demora para ser emitido e refletido de volta do sensor. Possui um ângulo de medida menor que os sensores ultrassônicos. Este modelo é capaz de fazer medidas de 5 a 120 cm (ADA, 2018).

2.5.4 - Encoder

Encoders são sensores capazes de obter o posicionamento de um motor, e dessa forma criar um controle em malha fechada desse. O encoder incremental é composto por um disco acoplado ao eixo do motor e que possui trilhas/ranhuras que modulam a luz emitida por um emissor, fazendo o detector gerar um sinal elétrico a partir dessas ranhuras. Entre os parâmetros de um encoder, se encontram a resolução, menor deslocamento detectado pelo sensor (CAPELLI, 2008).

2.6 - Algoritmo Genético

Algoritmos genéticos são programas que resolvem problemas a partir de um processo evolutivo, inspirado nas leis da evolução, em que a solução é desenvolvida. Começa a partir de uma população, um conjunto de possíveis soluções para o problema, que a partir de avaliações e operadores genéticos chegará a uma nova população que espera-se ser melhor do que a primeira (OBITKO, 1998).

A seguir descreve-se um pseudocódigo para um algoritmo genético: (OBITKO, 1998)

1. [Início] Gere uma população aleatória de n cromossomas (soluções adequadas para o problema)
2. [Adequação] Avalie a adequação³ $f(x)$ de cada cromossoma x da população
3. [Nova população] Crie uma nova população repetindo os passos seguintes até que a nova população esteja completa
4. [Seleção] Selecione de acordo com sua adequação (melhor adequação, mais chances de ser selecionado) dois cromossomas para serem os pais
5. [Cruzamento] Com a probabilidade de cruzamento, cruze os pais para formar a nova geração. Se não realizar cruzamento, a nova geração será uma cópia exata dos pais.

3 Também dito “aptidão”

6. [Mutaç o] Com a probabilidade de muta o, altere os cromossomas da nova gera o nos locus (posi o nos cromossomas).
7. [Aceita o] Coloque a nova descend ncia na nova popula o
8. [Substitua] Utilize a nova popula o gerada para a pr xima rodada do algoritmo
9. [Teste] Se a condi o final foi atingida, pare, e retorne a melhor solu o da popula o atual
10. [Repita] V  para o passo 2

Os genes de um algoritmo gen tico podem ser organizados de v rias formas, chamadas codifica es, sendo o mais comum o uso de codifica es bin rias, onde o indiv duo   representado por um n mero bin rio; tamb m existe a codifica o por valores, onde o indiv duo   representado por um conjunto de n meros reais (OBITKO, 1998).

A adequa o/avalia o deve ser feita de acordo com o problema em espec fico.

2.6.1 - Sele o por Roleta

Nesse tipo de sele o, a probabilidade de um indiv duo ser selecionado varia proporcionalmente a sua aptid o, como se estivesse em uma roleta (pizza), e cada indiv duo tivesse uma parcela proporcional a sua aptid o (POZO et al., 2017).

2.6.2 - Cruzamento Uniforme

Nesse cruzamento, copia-se aleatoriamente os genes dos pais para o filho (OBITKO, 1998).

2.6.3 - Muta o Indutiva

Nesse tipo de muta o, adiciona-se um valor gerado aleatoriamente ao gene codificado por valor. (OBITKO, 1998).

3 - Capítulo 3: Competições de Robótica

Neste capítulo serão descritos elementos das competições práticas de robótica, como a Olimpíada Brasileira de Robótica (OBR) e o Torneio de Robótica dos IFSP (TRIF), e das competições teóricas, como a OBR.

3.1 - Competição Teórica: a OBR

A modalidade teórica da OBR é realizada a partir de provas com conteúdos relacionados a robótica, sendo que a lista desses conteúdos podem ser encontrada no seu site (OBR, 2018).

3.2 - Competições Práticas: a OBR e o TRIF

A OBR é uma das principais competições de robótica do país, sendo que algumas competições como o TRIF acabam se inspirando nela para criação de suas regras. Portanto, nessa seção serão descritos elementos da OBR, que são aproximadamente os mesmos elementos do TRIF.

As seções a seguir foram retiradas das regras da etapa regional da OBR de 2018.

3.2.1 - Divisão de salas

A OBR é dividida em 3 salas, as duas primeiras onde o robô deve seguir uma linha preta e superar alguns desafios descritos entre as seções 3.2.2 a 3.2.7, e a terceira sala onde o robô deve procurar por bolas de isopor para devolvê-las a um local específico. A seguir descrevem-se os desafios da sala 1 e 2 e a 3ª sala.

3.2.2 - Encruzilhadas e curvas

As curvas são curvas na linha, onde o robô deve girar e continuar seguindo-a. As encruzilhadas são encontros de linha, podendo ou não possuir marcadores verdes. Nelas o robô deve seguir para o como indicado na figura 1:

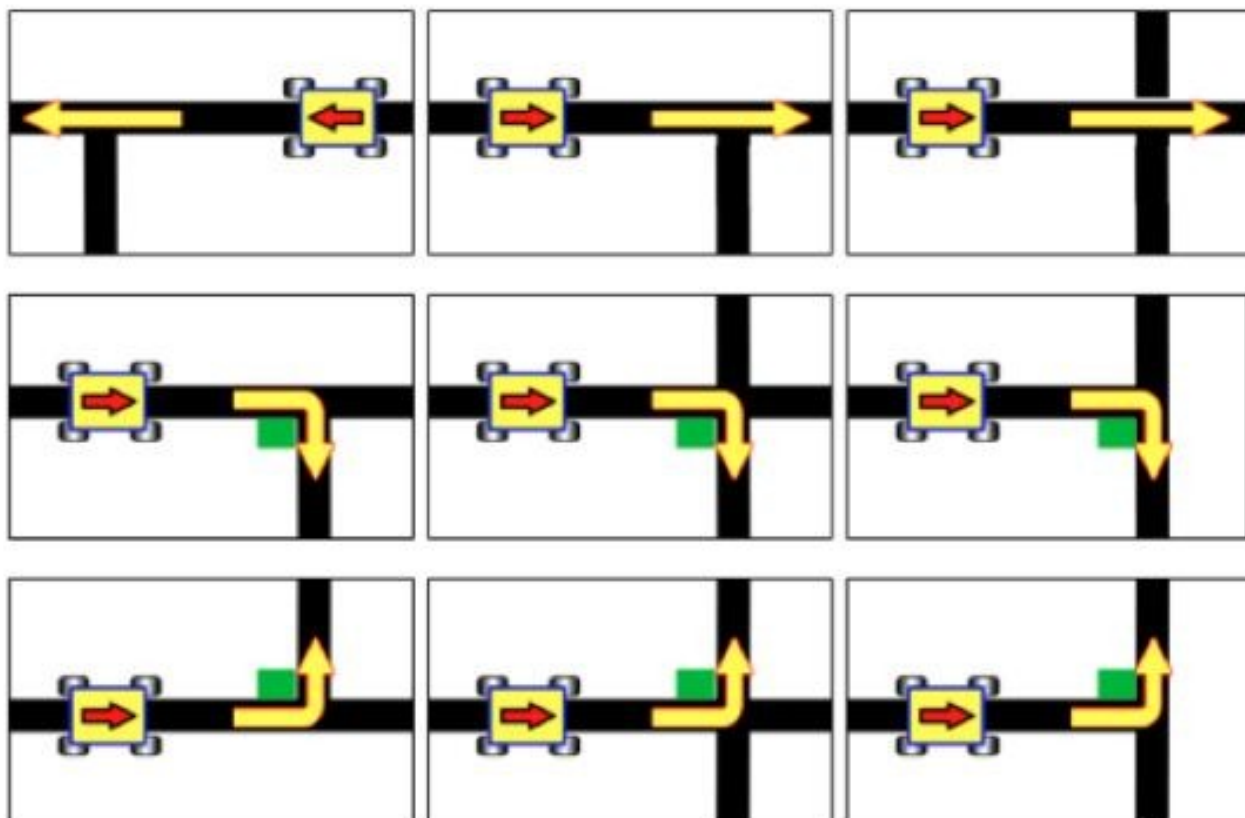


Figura 1: Encruzilhadas Possíveis na OBR/TRIF - Fonte: OBR, 2017, p. 10.

No ano do 2018, foi adicionada uma encruzilhada com dois verdes, na qual o robô deve fazer um retorno, girando 180°.

3.2.3 - Redutores de Velocidade

Redutores de velocidade são objetos cilíndricos, utiliza-se bastante lápis para criá-los, colocados no meio da linha, sendo que o robô deve subir sobre eles e ultrapassá-los.

3.2.4 - GAPs

GAPs (do inglês buracos) são falhas na linha, trechos onde a linha não existe, sendo que o robô deve seguir em frente quando encontra uma dessas falhas, até que ele retorne a linha.

3.2.5 - Obstáculos

Obstáculos são objetos cúbicos (normalmente utiliza-se caixas de leite) colocados no meio da linha. Quando o robô encontra um desses objetos, deve sair da linha e dar a volta no obstáculo, até retornar a linha no outro lado.

3.2.6 - Desafio Surpresa

O desafio surpresa é revelado alguns minutos antes do robô entrar no percurso. Ele é principalmente um desafio de programação, no qual a equipe deve resolvê-lo em pouco tempo.

Para alguns desafios, é necessário que o robô tenha algum tipo de sinalização, seja ela sonora, luminosa ou mecânica.

3.2.7 - Rampa

A rampa é um meio de transição entre as primeiras salas e a sala 3. Ela possui inclinação entre 10° e 20°, e pode possuir duas “paredes” em suas laterais.

O seu final e o início da sala 3 são marcados por uma fita adesiva cinza, normalmente uma “Silver Tape”.

3.2.8 - Sala 3

Nela, o robô deve procurar por bolas prateadas ou pretas, e colocá-las em uma área elevada com parede preta encontrada em um dos cantos da sala.

3.3 - Participações em Competições de Robótica

Houve participações no 1º e 2º TRIF, e na OBR – Modalidade Prática Regional de 2017 e OBR – Modalidade Teórica Nível 5 de 2017, como descrito no relatório anterior, e recentemente no 3º TRIF e na OBR Modalidade Prática Regional e Estadual de 2018, além da OBR Modalidade Teórica Nível 5 de 2018.

4 - Capítulo 4: Trabalho Anterior

Neste capítulo, será descrito o trabalho realizado anteriormente nos anos de 2016 e 2017, sendo que explicações mais aprofundadas podem ser encontradas no relatório, que poderá ser encontrado no site do Campus Suzano do IFSP Suzano.

4.1 - Alterações Físicas e Eletrônicas no Robô Zumo

Foram realizadas alterações significativas no Robô Zumo, entre elas a confecção de uma placa para utilização do Arduino Mega, pois o Zumo foi concebido para utilização apenas com o Arduino Uno, a construção de uma garra e alocação de sensores.

A garra foi primeiramente feita de isopor de alta densidade, para verificar seu funcionamento, e após impressa em PLA, utilizando uma impressora 3D. A garra possui dois eixos de liberdade, sendo movida a partir de dois servomotores.

Foram alocados no robô quatro sensores ultrassônicos, dois nas laterais, e dois na parte frontal, em alturas diferentes. Além desses, foi alocado um sensor de cor.

Foram também colocadas na parte frontal, rodas auxiliares, para facilitar a subida dos redutores, porém verificou-se também sua ineficácia e capacidade de atrapalhar a superação de curvas, por gerar um atrito com a superfície quando o robô tenta girar.

Por fim, foi colocado também um *buzzer*, com objetivo de superar o desafio surpresa da OBR.

4.2 - Programação anterior

O programa anterior, programado na “Arduino IDE”, implementa soluções para seguir a linha, desvio de obstáculos, curvas e a passagem pela rampa; além do tratamento dos sensores a partir de classes próprias.

4.2.1 - Sensores de Refletância

O robô Zumo possui 6 sensores de refletância, numerados de 0 a 5 da direita para a esquerda. Eles possuem uma biblioteca para leitura analógica dos sensores, e é criada uma variável de “corte” em que, caso algum sensor leia um valor maior que o “corte”, ele está em uma superfície preta ou verde, caso contrário, em uma superfície branca, digitalizando as leituras do sensor.

4.2.2 - Sensores inerciais

O robô Zumo possui também um acelerômetro (lê aceleração) de 3 eixos e um giroscópio (lê velocidade angular) de 3 eixos. Como o deslocamento é a segunda integral da aceleração, e o ângulo rotacionado a integral da velocidade angular, esses sensores forma utilizados para se obter essas grandezas. Porém antes foi preciso processar devidamente suas leituras.

Utilizou-se um filtro de Kalman, obtido a partir de uma biblioteca que o implementa (criada por Alex Vargaa Benamburg e disponível na internet no repositório “https://github.com/alexvargasbenamburg/arduino_FiltroKalman”), para reduzir o ruído. Foi também corrigido o eixo de referência das leituras, ou seja, fazendo com que as leituras com aceleração (ou velocidade angular) nula demonstrem isso, e não algum valor deslocado para cima ou para baixo. Para isso os sensores são calibrados várias vezes durante o programa, pegando a média de várias leituras com o robô parado, e subtraindo essa média das próximas leituras. Por fim, é determinado um mínimo, positivo e negativo, para as leituras; caso elas não ultrapassem esse mínimo, possuem valor “0”, evitando ruídos.

Todo esse processamento dos sensores, da leitura bruta até a leitura final (deslocamento/ângulo) é feita dentro de classes específicas para cada sensor

Como as leituras do deslocamento utilizam uma integral dupla, integram também duas vezes os erros de leituras, sendo assim uma medida mais imprecisa do que a de rotação.

4.2.3 - Classes para atuadores

Foram criadas classes para controlar o *buzzer* e motores da garra.

4.2.4 - Controle PD para a linha e GAPs

A programação para que o robô seja capaz de seguir a linha é desenvolvida a partir de um controlador proporcional derivativo, em que os seis sensores de refletância voltados para o chão na parte frontal do robô ($sensor_t^{0-5}$) são reunidos em uma única variável (posição_t) como a equação I:

$$I. \quad posição_t = -(sensor_t^0 \cdot 3) - (sensor_t^1 \cdot 2) - (sensor_t^2 \cdot 1) + (sensor_t^3 \cdot 3) + (sensor_t^4 \cdot 2) + (sensor_t^5 \cdot 1)$$

Como os sensores são posicionados com os sensores 0-2 no lado esquerdo e 3-5 no lado direito da linha, com um lado com peso negativo e outro com peso positivo, quando o robô se posiciona centralizado a linha (ou a qualquer padrão simétrico de reflexão), os dois lados possuem o mesmo valor e se anulam, com a variável “posição” recebendo um valor “0”.

Dessa forma, o setpoint (valor desejado pelo robô) torna-se 0, pois deseja-se que ele se centralize na linha. Portanto, o erro (e) pode ser calculado de acordo com a equação II:

$$II. \quad e_t = posição_t - setpoint = posição_t - 0 = posição_t \\ \therefore e_t = posição_t$$

Logo, a saída do controlador, chamada de “diferença” pode ser calculada como:

$$III. \quad diferença_t = kp \cdot e_t + kd \cdot (e_t - e_{t-1})$$

Sendo que os ganhos utilizados foram determinados empiricamente, e a saída do controlador é enviada aos motores, alterando suas velocidades:

$$IV. \quad velocidade \text{ motor direito}_t = velocidade - diferença_t \\ velocidade \text{ motor esquerdo}_t = velocidade + diferença_t$$

A “velocidade” é uma velocidade padrão determinada. Como a “posição”, logo o “erro” logo a “diferença” variam seu sinal de acordo com o lado que o robô se posiciona em relação a linha, isso leva os motores a ajustarem o robô movimento um motor mais rápido que o outro.

Como ao se posicionar sobre um GAPs, não existe nenhuma variação de reflexão em nenhum sensor, estando em um padrão uniforme e simétrico, a posição torna-se “0”, com o robô seguindo em frente com velocidades iguais nos dois sensores.

Esse método se mostrou muito eficaz para seguir a linha

4.2.5 - Controle da movimentação

Foram criadas funções para movimentar o robô por ângulos ou distâncias calculadas pelos sensores inerciais.

A função para giro recebe como parâmetro o grau desejado (positivo ou negativo), e gira o robô em uma velocidade padrão até que calcule-se que o robô já girou o desejado.

A função para deslocamento recebe como parâmetro a distância desejada, e movimenta o robô em uma velocidade padrão até que ele tenha se movimentado o desejado.

O controle utilizado se encaixa nos controladores ON-OFF, em que o atuador varia em uma potência ligada predeterminada e desligado. O método para giro se mostrou mais precisa que a para deslocamento, pois o cálculo do ângulo movimentado é mais preciso.

4.2.6 - Superando curvas

Para detectar curva, o robô verifica se os sensores de refletância 1 ou 4, sendo os que não estão nem na borda, nem no centro, lendo os marcadores verdes e fitas de curvas acentuadas; verificam uma superfície “preta”, sendo que os sensores não fazem distinção entre preto e verde, o robô inicia a função para superação de curva, para a esquerda ou direita dependendo do sensor acionado.

Ele então se movimenta para frente temporizadamente e, se detectar novamente o preto, ele executa a sequência para curva com marcador verde, pois entende-se que ele detectou primeiro o verde de depois a linha. Caso contrário, e os sensores centrais detectarem uma continuidade da linha, ele executa a sequência para as encruzilhadas em que deve seguir reto. Por fim, caso nenhuma das ocasiões anteriores, ele executa a sequência para curva comum.

Na curva com marcador verde, o robô movimenta-se mais para frente e gira 80° para o lado desejado. Depois executa o controle para seguir linha para se ajustar a linha. Na curva comum, retorna um pouco menos que a movimentação para frente realizada, e executa o controle para seguir linha até que o sensor do lado que detectou a linha termine de ler a cor preta. Como ele retorna

menos que o desejado, é útil para fazer pequenas correções a linha que se confundem com curvas e evitar que o robô fique preso em alguma parte do circuito. Na encruzilhada em que o robô deve seguir reto, ele simplesmente segue reto por um determinado período de tempo.

4.2.7 - Obstáculo

Para detectar os obstáculos, utiliza-se o sensor de distância frontal do robô. O método utilizado para desviar do obstáculo faz uma sequência de movimentos controlados, em deslocamento e giro, para se movimentar ao redor do mesmo. Para retornar a linha, o robô executa a mesma sequência para curvas com marcadores verdes.

4.2.8 - Detecção da rampa

Para detectar a rampa, aproveitou-se do fato dos sensores acelerômetros lerem também a aceleração gravitacional, e utilizou-se as leituras filtradas por média móvel do eixo X (que corta o robô de sua frente a sua traseira). Quando o robô entra na rampa, as leituras desse eixo sofrem um aumento, pois a aceleração gravitacional que antes se aplicava apenas no eixo Z (que corta o robô de cima para baixo, perpendicular a superfície da terra em áreas planas), em áreas inclinadas se aplica parte no eixo X e parte no eixo Z. O aumento no eixo Z pode se observar na figura 2:

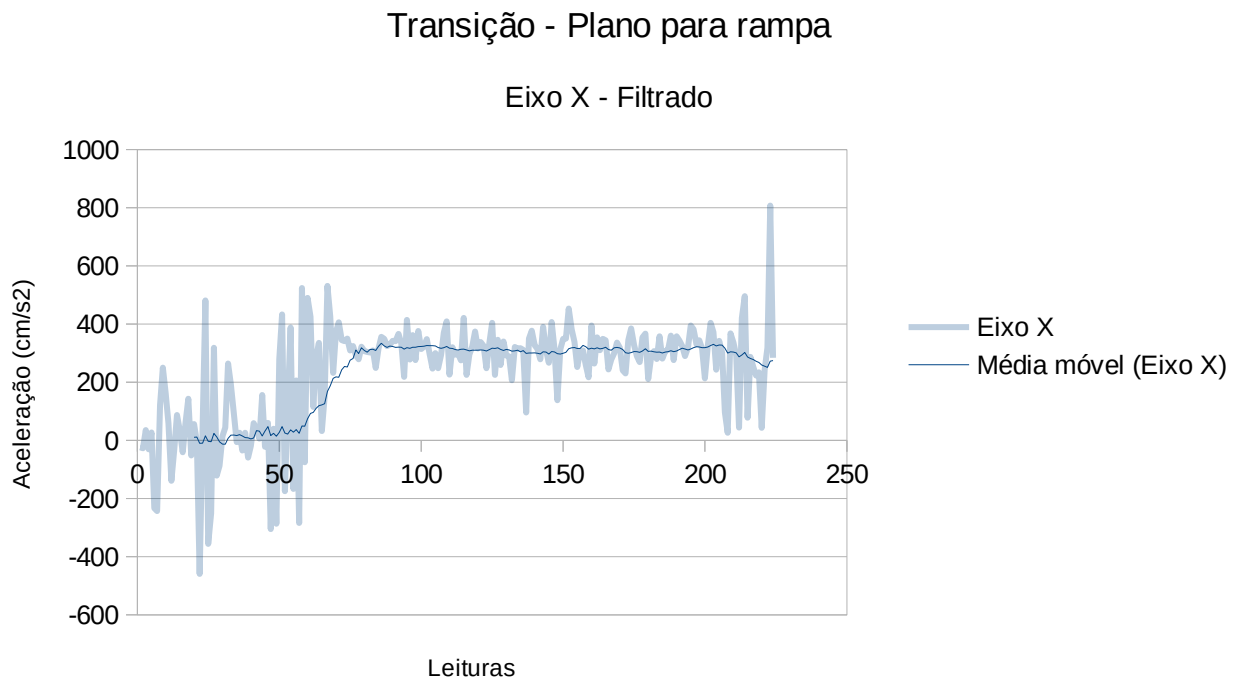


Figura 2: Gráfico - Acelerômetro na transição plano para rampa

Logo, quando as leituras estão acima de um limiar determinado, compreende-se que o robô está sobre a rampa. E quando as leituras voltam para baixo desse limiar, compreende-se que o robô entrou na sala 3, voltando a uma superfície plana.

4.3 - Problemas encontrados

Discutiu-se a necessidade de encontrar melhorias para os seguintes problemas:

- Melhorar a superação de obstáculos
- Melhorar a detecção de curvas, que possui movimentos temporizados
- Criar uma solução para a 3^a sala do percurso
- Estudar melhorias para o tratamento dos sinais dos sensores inerciais
- Estudar os sinais obtidos pelo sensor de cor
- Encontrar uma solução para superar o redutor, com alterações mecânicas no robô
- Utilizar técnicas de IA (algoritmo genético) para sintonizar o controlador PID

5 - Capítulo 5: Alterações Físicas no Robô

5.1 - Alimentação

A alimentação do robô mostrou-se ser um problema considerável. Anteriormente, utilizava-se quatro pilhas AA alcalinas, porém a duração da carga criava um grande gasto contínuo para alimentar o robô; e a curva de descarga característica dessas pilhas, em que a tensão varia excessivamente em relação a carga das pilhas, o que alterava grandemente a resposta dos motores a velocidade programada em relação a carga das pilhas. Portanto o uso de pilhas alcalinas tornou-se inaceitável.

Tentou-se então o uso de quatro pilhas AA de NiMh, sendo essas recarregáveis e com curva de descarga mais aceitável que as alcalinas. Porém, essas não foram capazes de suportar a corrente exigida para movimentar o robô e a garra, e suas tensões nominais de 1,2 V (menor que a de 1,5 V das alcalinas) deixaram a movimentação do robô mais vagarosa.

Após as pilhas de NiMh, tentou-se utilizar baterias portáteis para o carregamento de celulares. Mesmo possuindo uma capacidade de carga suficiente, e tensão nominal ideal (5 V) essas também não conseguiram suportar a corrente necessária para alimentar o robô.

Foi então experimentado uma bateria de Li-Po, com tensão nominal de 11,1 V (2 células) a 7,4 V (1 célula). Mesmo possuindo um tamanho extenso comparada com as pilhas, ter um risco de acidentes maior e precisar de um regulador de tensão para converter sua tensão para 6 V, mostrou-se ser a que mais se encaixa com as necessidades do robô, possuindo muita carga, capaz de alimentar o robô por um longo período, e capacidade de alimentar o robô e a garra.

Porém, o elevado custo dessa bateria impossibilitou seu uso. Foram então utilizadas quatro baterias de NiMh de 9 V. Possuindo uma capacidade de alimentar simultaneamente o robô e a garra, com quatro baterias em paralelo não é possível alimentar o robô por muito mais de uma hora, e o tamanho de quatro baterias compara-se a uma bateria de Li-Po.

5.2 - Rampa

Como a estrutura do robô Zumo mostrou-se anteriormente incapaz de ultrapassar os redutores de percurso, e as rodas auxiliares colocadas no robô não foram capazes de ajudá-lo, foi preciso a impressão de um tipo de rampa (figuras 3 - 5), que posicionada na parte frontal do robô, protege os sensores e faz com que o robô deslize sobre o redutor, até que a esteira consiga tocá-lo.

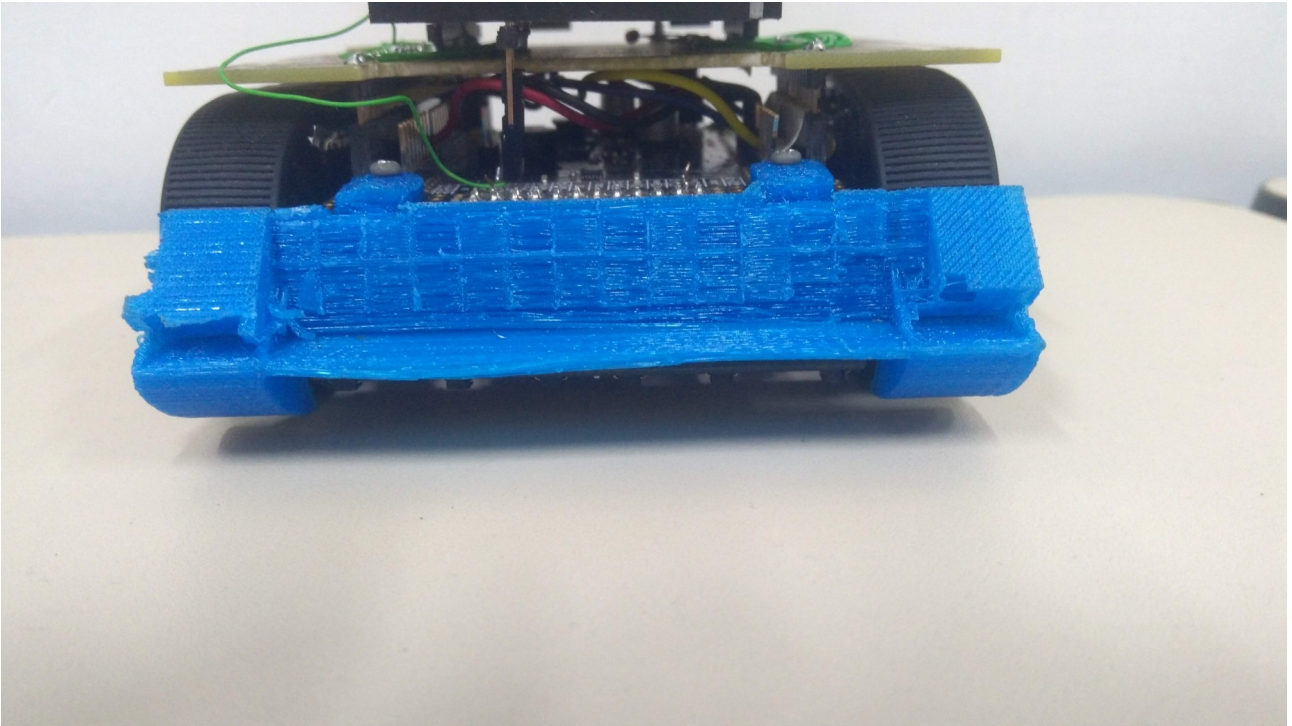


Figura 3: Foto frontal da rampa impressa

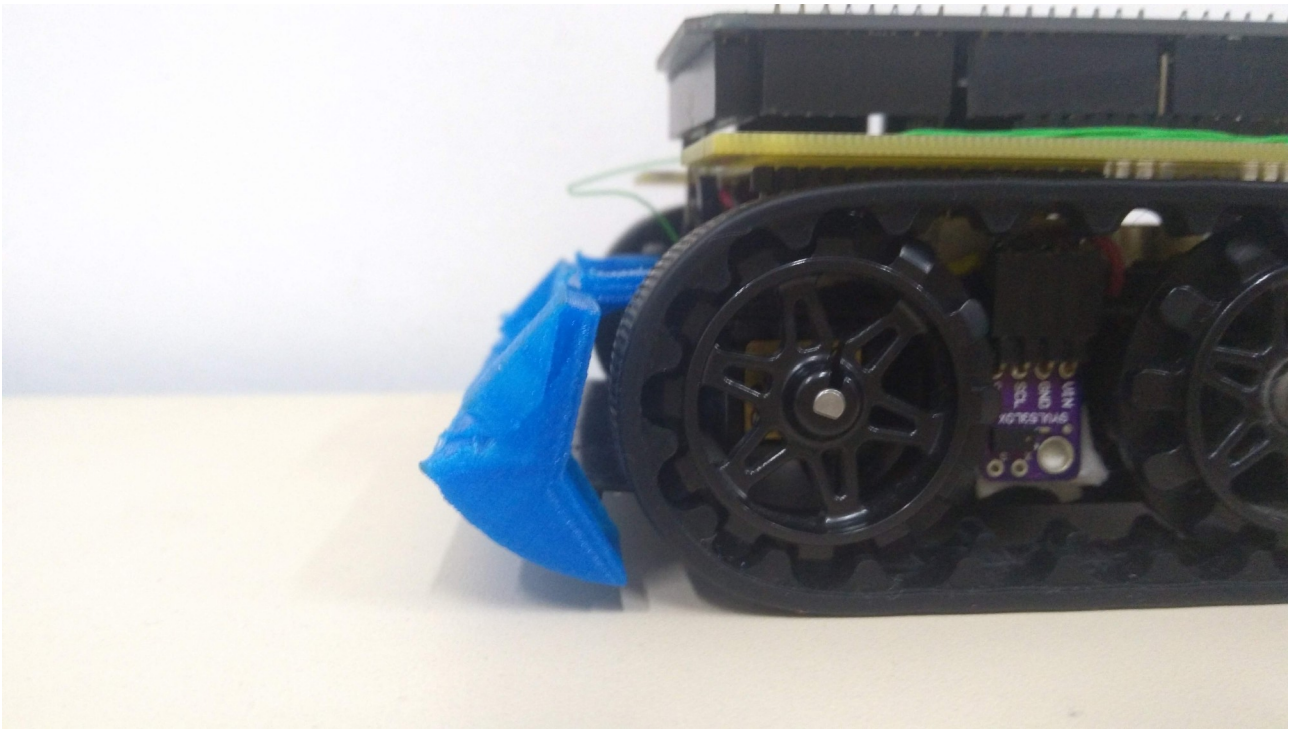


Figura 4: Foto lateral da rampa impressa

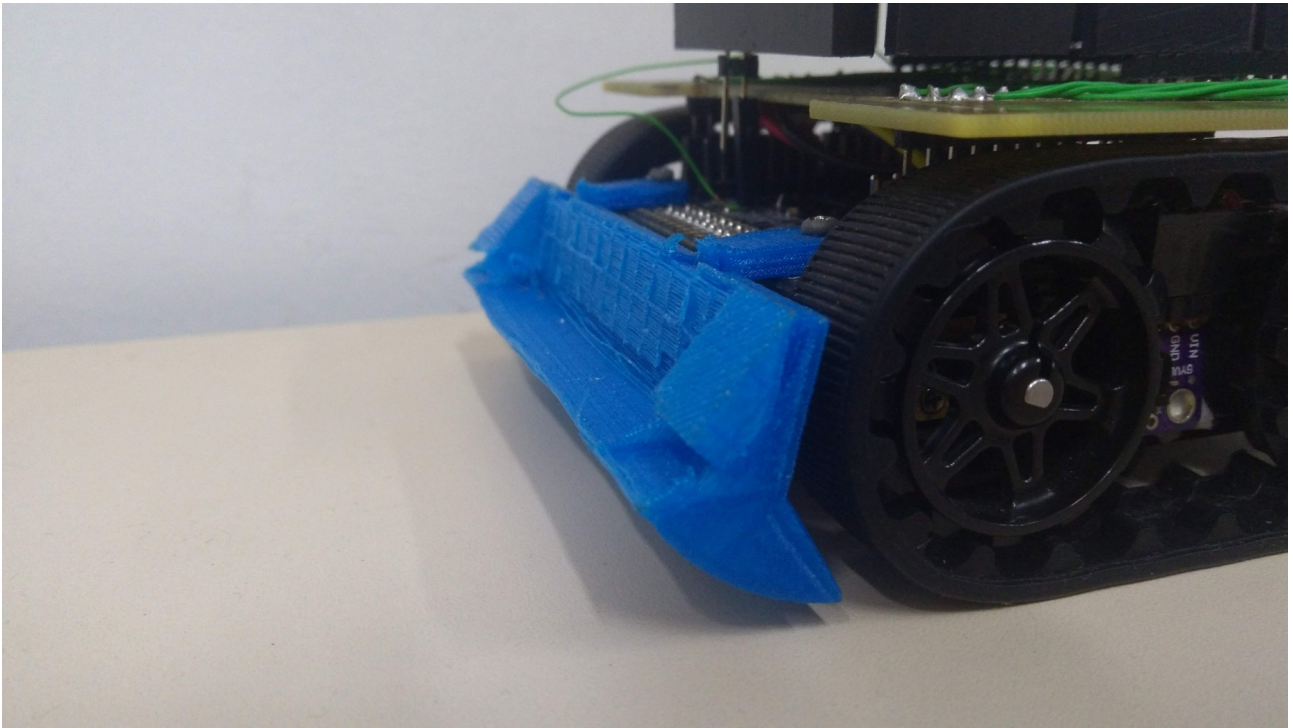


Figura 5: Foto da rampa impressa

5.3 - Sensores ToF

Ao tentar criar uma solução para a terceira sala do percurso, verificou-se que seria útil posicionar sensores de distância nas laterais do robô, em uma altura que fosse capaz de identificar as vítimas.

Como os sensores de distância ultrassônicos utilizados anteriormente (HC-SR04) tinham dimensões que os tornavam incapazes de serem colocados nessa altura, foram utilizados sensores ToF de laser. O modelo utilizado (VL53L0X) possui dimensões que o torna capaz de ser acomodado entre as rodas do robô, como na figura 6:

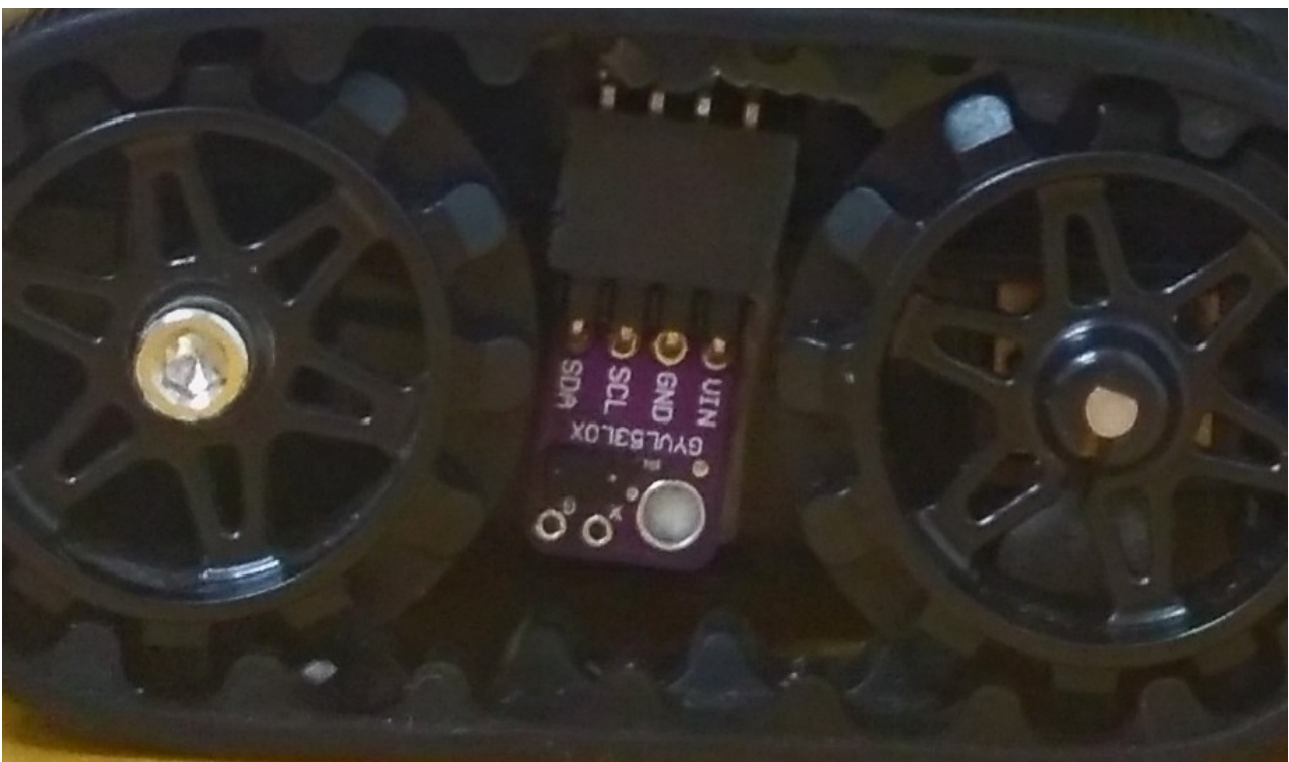


Figura 6: Alocação do sensor laser lateral

Foram colocados dois desses sensores, um em cada lateral, e para poder endereçar corretamente esses sensores que utilizam o protocolo I2C, foi preciso conectar, além das portas necessárias ao protocolo e a alimentação, uma porta digital para o endereçamento.

5.4 - Garras

Com objetivo de capturar as vítimas da sala 3, havia sido construída uma garra com dois eixos de liberdade. Porém, ela mostrou-se ineficiente, pois tendo uma haste muito longa, projetava-se

muito para a parte traseira do robô, causando um desbalanceamento em sua distribuição de peso. Para tentar resolver esse problema, projetou-se uma garra com três eixos de liberdade, como na figura x:

Porém, sua ponta demonstrou ser também ineficiente, pois para capturar uma vítima era necessário uma precisão muito grande no posicionamento do robô. A solução encontrada foi a criação de uma garra que utilizava elásticos para prender a vítima (figuras 7-9). Com isso, a garra era empurrada contra a superfície, e a vítima era comprimida contra os elásticos, que estendiam e possibilitavam a sua entrada. Como os elásticos voltam a sua posição inicial, a vítima fica presa entre eles. Para sua liberação a ponta é aberta utilizando o motor.

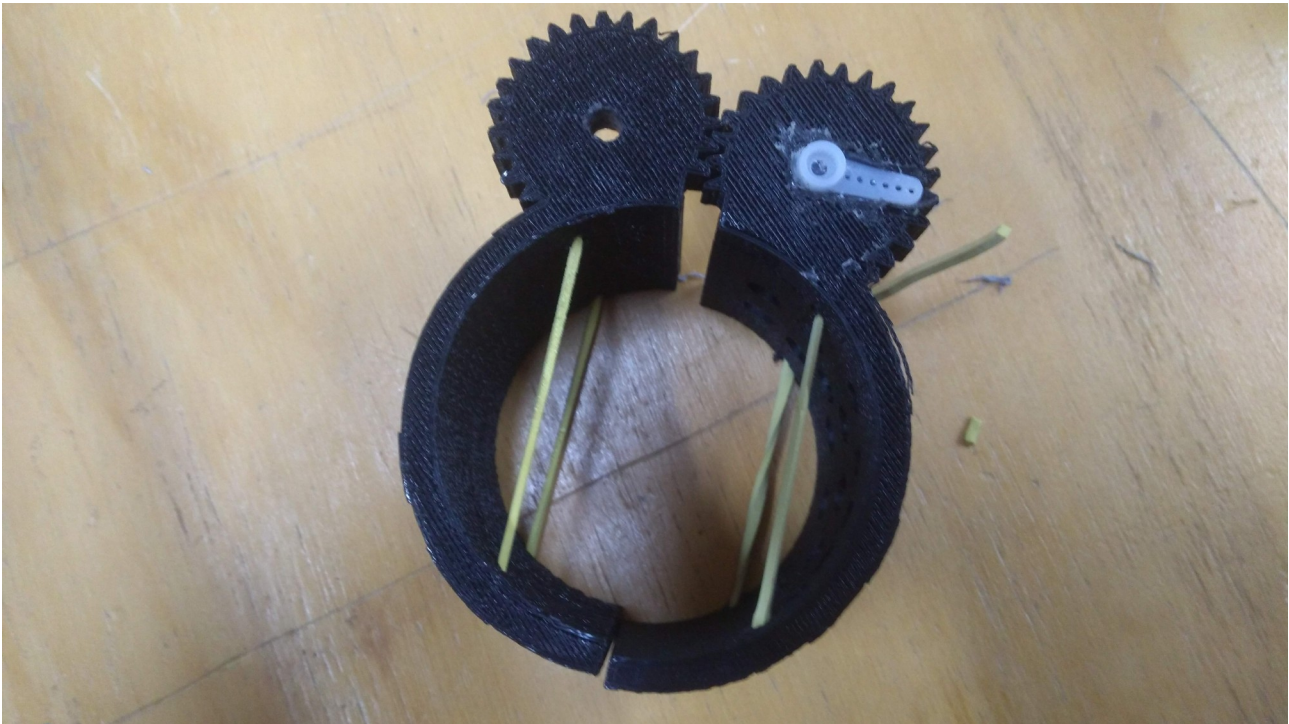


Figura 7: Foto da garra impressa



Figura 8: Foto da dilatação dos elásticos



Figura 9: Foto da vítima dentro da garra

Mas, esse modelo de garra também não era eficaz, pois o esforço sobre os motores para mantê-la na posição enquanto o robô andava pelas salas 1 e 2 do percurso criava um alto consumo de corrente, e seu peso impossibilitava posteriormente sua movimentação. Por fim, criou-se um

pegador de vítimas diferente (figura 10), que consistia em uma caixa que era posicionada na parte frontal do robô, o qual corria até encostar na parede da sala, pegando as vítimas sem necessidade de identificação. Após essa caixa era levantava e colocava as vítimas em uma outra caixa, que levanta e deposita as vítimas na sala de resgate.

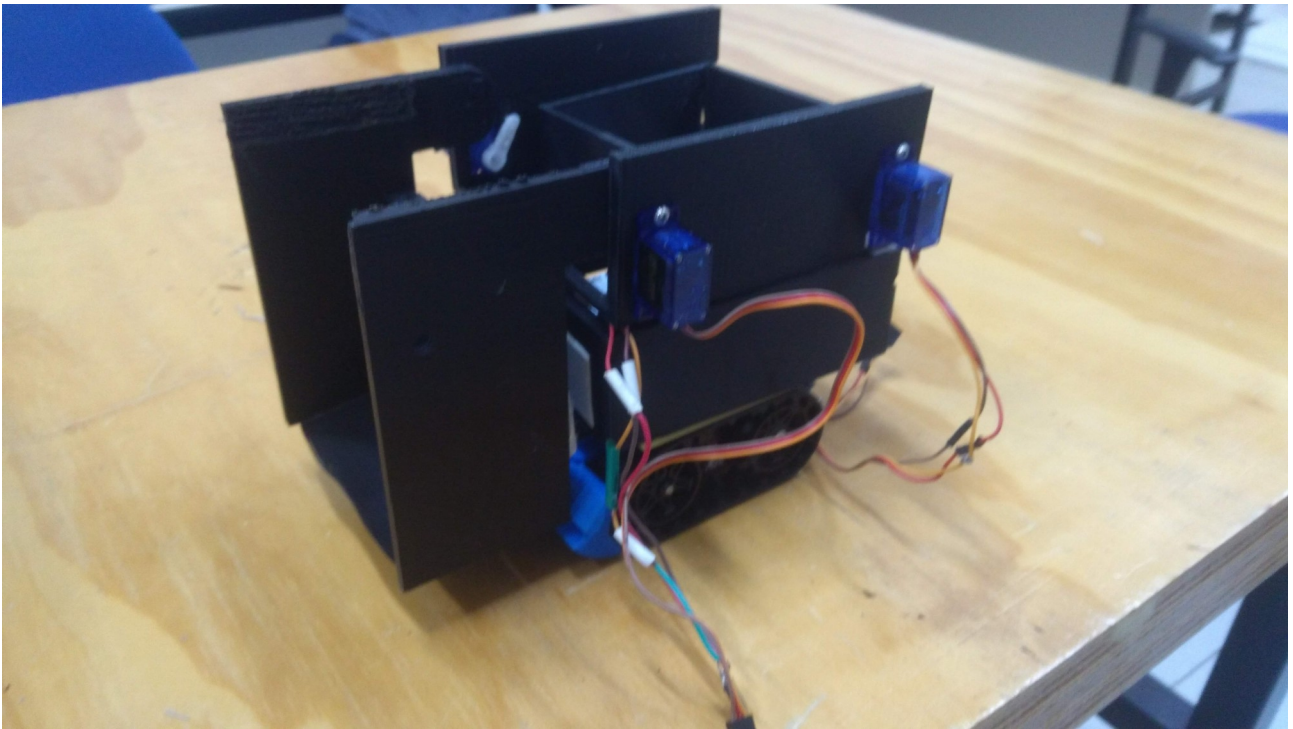


Figura 10: Foto do pegador de vítimas novo

Infelizmente, a necessidade de reconstruir os circuitos que eram acomodados na garra anterior impossibilitaram o seu teste, pois os mesmos não puderam ser concluídos a tempo.

5.5 - Remoção dos sensores ultrassônicos laterais

Com a inserção dos sensores laser nas laterais do robô, e com a impossibilidade de utilizar os sensores ultrassônicos laterais para identificar as vítimas, pois era preciso que eles não as detectassem, o que ocorria em alguns momentos, esses sensores tornaram-se inúteis para o robô, e foram então removidos.

6 - **Capítulo 6: Alterações no Código**

6.1 - Alteração do Ambiente de Desenvolvimento e Organização do Código

Visando uma maior organização das várias versões do código, e uma maior capacidade de programação conjunta do mesmo código, por conta do aumento da equipe, desejou-se o uso de um sistema de controle de versão, e uma IDE com esse sistema integrado.

Portanto, alterou-se a IDE da “Arduino IDE” para a “Visual Studio Community”, que integração com o GitHub. Para a compilação do código, utilizou-se a extensão “Arduino IDE for Visual Studio”.

Com o uso do GitHub, o código pode também ser disponibilizado virtualmente no repositório “Athenas”, disponível em: “<https://github.com/RoboticalFSPSuzano/Athenas>”.

Além da alteração da IDE, por conta do tamanho do código, alterou-se também a forma como ele é organizado, agora dividido em 7 arquivos, um para implementação de bibliotecas, constantes e variáveis globais; para classe; para declaração de objetos; para funções não diretamente relacionadas aos desafios da OBR; para funções diretamente relacionadas a OBR; para o setup e para o loop.

6.2 - Sensores Inerciais e PID

6.2.1 - Tentativas de melhoria das leituras

Para melhorar as leituras do giroscópio e acelerômetro, que, principalmente o último, mostraram-se imprecisas, tentou-se uma análise de vídeos do robô em movimento, utilizando o programa “Tracker”, para verificar as relações entre as leituras dos sensores e os dados reais obtidos pelo vídeo.

Exemplificando, pode-se observar um gráfico com o deslocamento calculado pelo acelerômetro e pelo vídeo na figura 11:

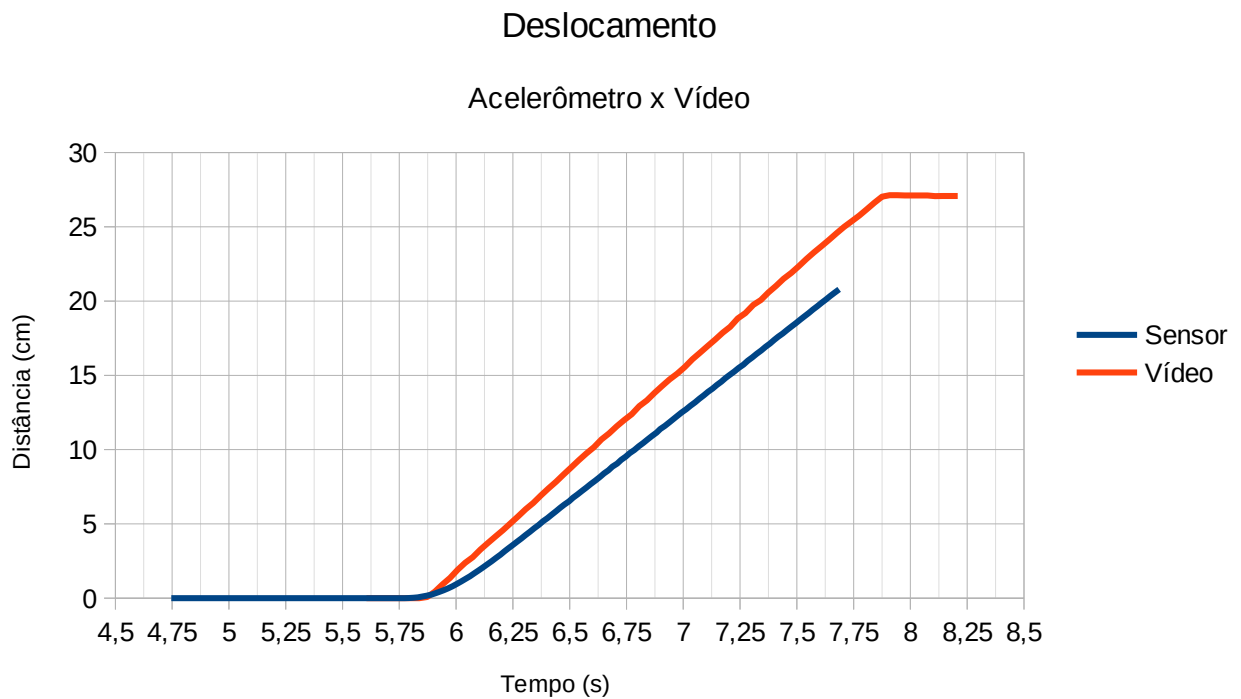


Figura 11: Gráfico das distâncias medidas x calculadas

Observa-se uma relação entre os dados, porém não foi possível encontrar uma forma eficaz de utilizar os vídeos para melhorar os sensores. Uma possível solução encontra-se no uso de encoders e filtros de fusão.

Outro teste para melhorar esses sensores foi a tentativa de utilizar as ranhuras das rodas do robô junto de um sensor de reflexão para montar um encoder. Dessa forma, a partir da variação das medidas do sensor, utilizando um valor como limiar entre leitura na ranhura e leitura fora da ranhura, foram digitalizadas essas leituras em dois níveis lógicos. Um teste movimentando o robô por 10 cm gerou o seguinte resultado (figura 12):

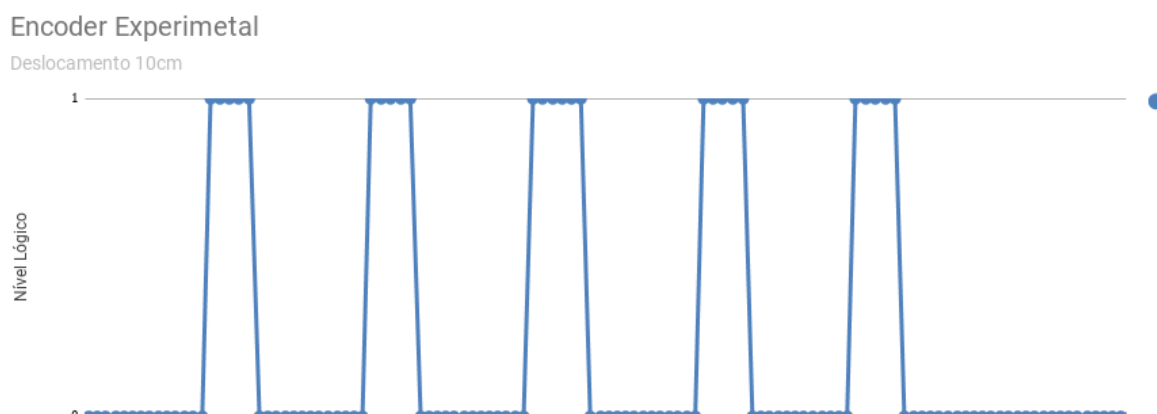


Figura 12: Gráfico do teste do encoder experimental

Observa-se o funcionamento adequado do projeto, porém as poucas ranhuras na roda do robô incapacita uma grande resolução, como observa-se ao gerar apenas 5 pulsos em 10 cm. Seria necessário trocar essas rodas por outras com mais ranhuras, que poderiam ser impressas na impressora 3D. Porém a substituição de uma parte mecânica do robô patrimoniado e em responsabilidade da instituição vai além do possível pelo projeto.

6.2.2 - PIDs

Utilizando controladores PID, as funções que anteriormente faziam o robô se locomover uma certa distância linear, ou girar um certo ângulo, que utilizavam um método ON-OFF, foram alteradas, criando-se as funções “pidDeslocamento” e “pidGiro”.

A função “pidDeslocamento” utiliza como variável do processo o deslocamento já realizado do robô, como set point o deslocamento desejado, e como variável manipulada a velocidade dos dois motores, alterando assim sua velocidade linear.

Já o “pidGiro” utiliza como variável do processo a rotação já realizado do robô, como set point a rotação desejado, e como variável manipulada a diferença de velocidade entre os motores, alterando assim sua velocidade angular.

Como o sensor acelerômetro demonstrou-se muito mais impreciso que o giroscópio, a função “pidDeslocamento” não teve uso, por não conseguir realizar seu papel.

6.3 - Outras Alterações

6.3.1 - Detecção de redutores

Com a construção da rampa para tornar possível a ultrapassagem do redutor, mostrou-se necessário programar uma detecção do mesmo, evitando que o robô confunda-o com uma curva, pois ao subi-lo os sensores de refletância ficam sem uma superfície para refletir a luz, e detectam como se todos estivessem vendo preto.

Essa programação foi feita verificando-se a aceleração sobre o robô, da mesma forma que a detecção de rampa, mas dessa vez verificando se existia uma aceleração sobre o eixo X no momento em que verificava o tipo de curva.

6.3.2 - Função Obstáculo

A função para desvio do obstáculo foi alterada, alterando-se os deslocamentos lineares controlados pelo acelerômetro por uma orientação em relação ao obstáculo utilizando os sensores de distância, verificando a se o robô já ultrapassou o obstáculo e é capaz de girar. Utiliza-se em conjunto a isso pequenos movimentos temporizados.

Para os giros, utilizou-se da função pidGiro.

7 - Capítulo 7: Criando uma solução para a 3ª sala do percurso

7.1 - Detectando as Vítimas

Para a detecção de vítimas, foram utilizados os sensores de distância laterais, primeiramente com testes os ultrassônicos junto dos lasers, após verificou-se a necessidade apenas com os lasers. Com o robô andando lateralmente a vítima, essa causa uma diferença no sensor de distância, que tem suas leituras filtradas por um filtro de média móvel para melhorar as leituras, como na figura 13:



Figura 13: Gráfico - Leituras da vítima pelo sensor de distância

Utilizando a variação dessas leituras, cria-se o gráfico 14, onde é possível observar um pico negativo seguido de um pico positivo. O ponto que é a raiz dessa curva entre esses dois picos representa então o ponto em que a vítima estava precisamente na frente do sensor. Para melhorar as leituras e diminuir a taxa de erro, utilizou-se um filtro do tipo média móvel para processar essas leituras, junto com a exclusão de valores muito discrepantes (valor absoluto maior que 0,5).

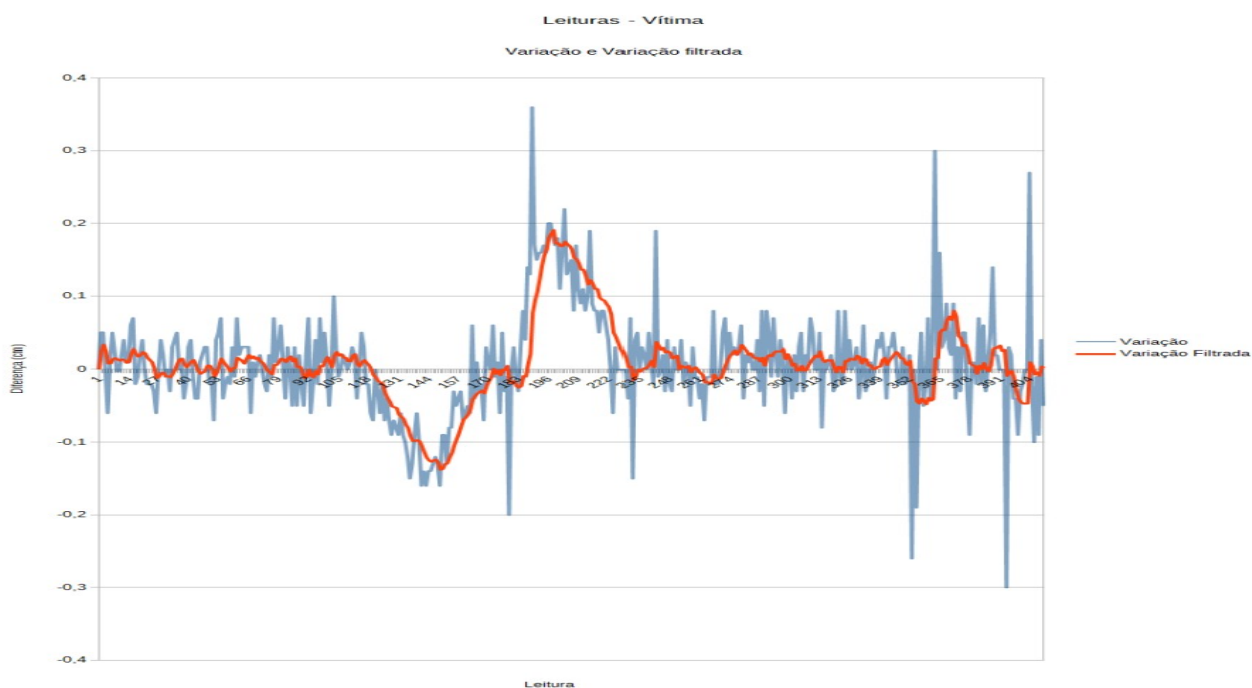


Figura 14: Gráfico - Variações das leituras da vítima

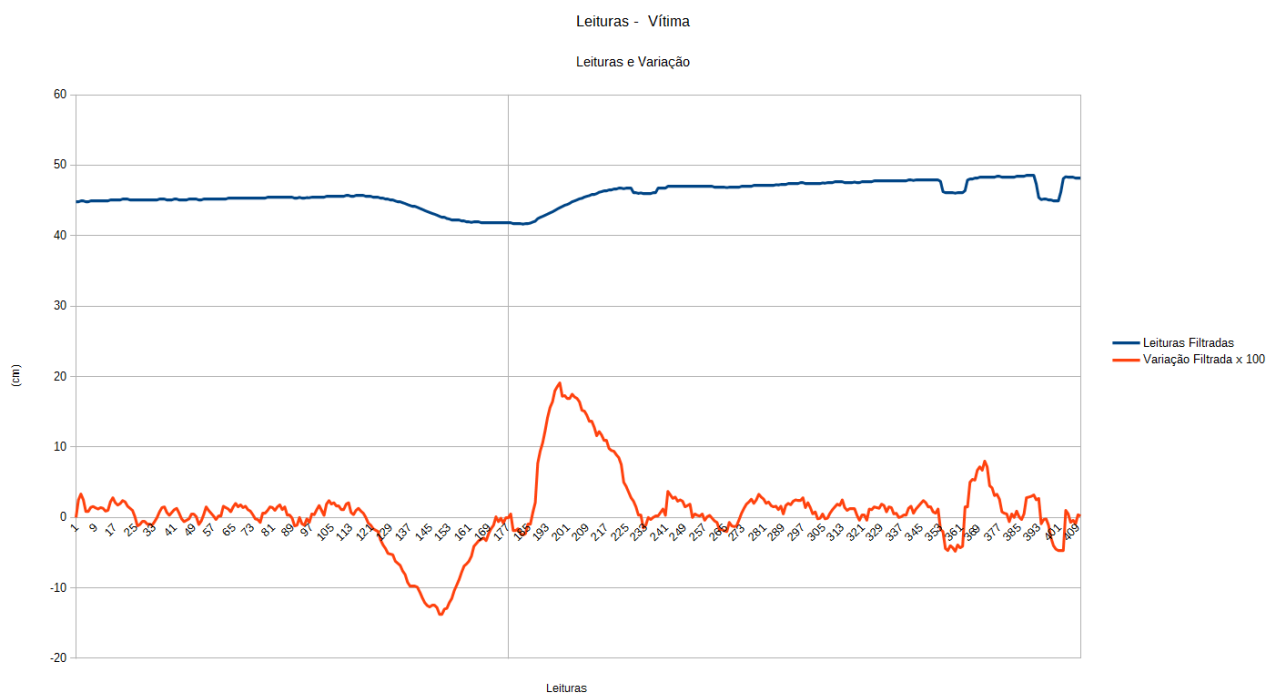


Figura 15: Gráfico - Leituras e Variações da Vítima

Na figura 15, verifica-se como a raiz da variação coincide com a vítima.

Em testes, o método descrito mostrou-se capaz de detectar vítimas, porém seu acerto varia com a distância entre o robô e a parede, e a vítima e a parede. É preciso também testes para verificar a influência da zona de resgate nas leituras do robô.

7.2 - Detectando a Zona de Resgate

Para detecção da zona de resgate, foi desenvolvido um algoritmo utilizando o giroscópio e os dois lasers laterais. O código encontra-se no apêndice B.

Primeiramente, cria-se um mapa da sala. Para isso, o robô é colocado no centro da sala, e começa a rotacionar em seu próprio eixo durante alguns segundos, e soma-se as leituras dos lasers no ângulo inteiro mais próximo ao medido pelo giroscópio, sendo que a leitura do laser direito é gravado no ângulo do esquerdo mais 180°; e grava-se também a quantidade de leituras em cada grau. Ao final da rotação, é feita uma média das leituras em cada grau, dividindo o somatório pelo número das leituras. A figura 16 contém um gráfico com um exemplo de leitura:

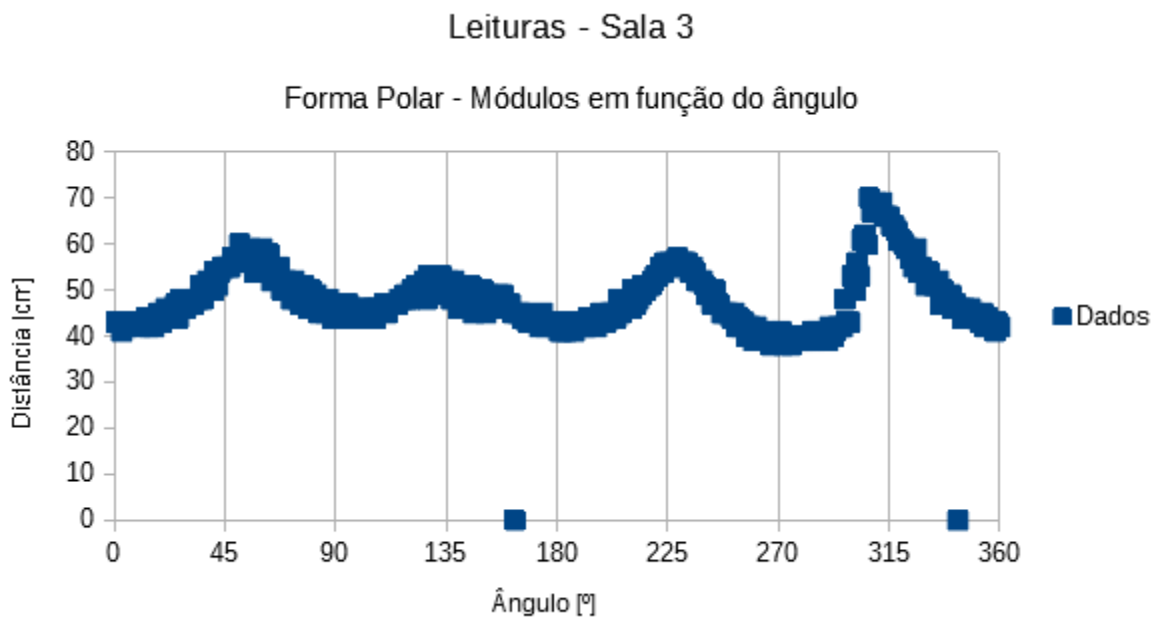


Figura 16: Gráfico - Leituras da sala 3 - Forma Polar

O código existente no apêndice contém apenas a parte do processamento dos dados, sendo programado para receber os dados emitidos pelo robô (linhas 158 – 161), retornar o ângulo da zona de resgate, e escrever outros dados em um arquivo de planilha. Após, utilizando o teorema de Pitágoras, calcula-se os pontos no eixo cartesiano (166 - 169):

$$P(x,y)=(\text{leitura} \cdot \cos(\hat{\text{ângulo}}), \text{leitura} \cdot \text{sen}(\hat{\text{ângulo}}))$$

Criando-se assim um mapa da sala como na figura 17:

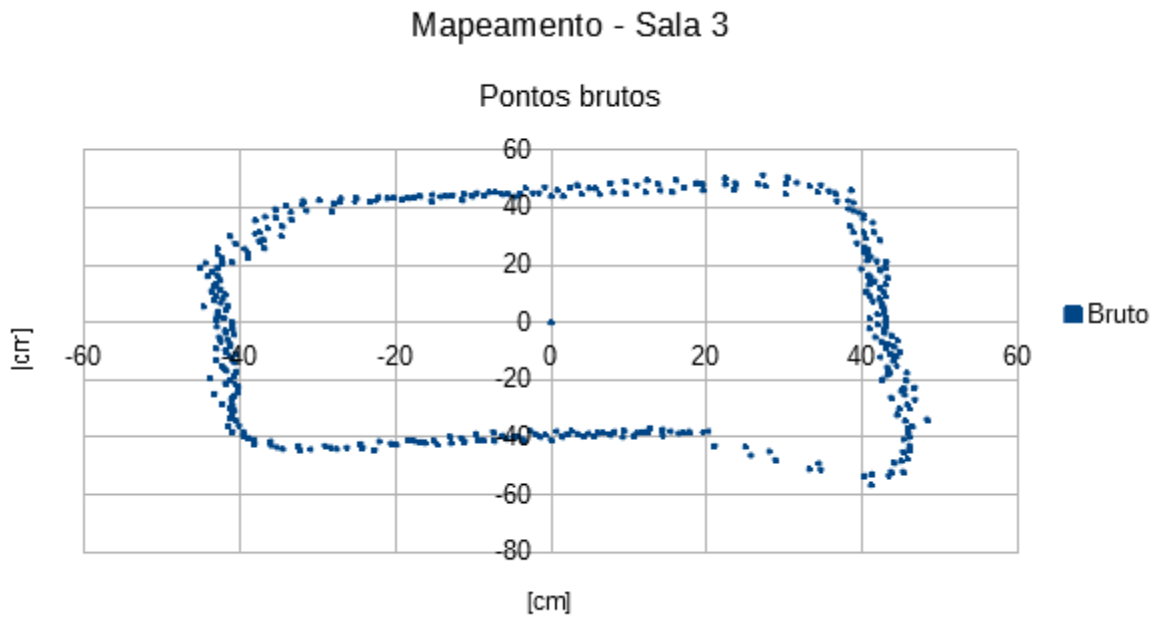


Figura 17: Gráfico - Mapa da sala 3

Porém, percebe-se que as leituras possuem um certo ruído, e para amenizá-lo utiliza-se um filtro do tipo média móvel (175 - 208), utilizando o ponto a ser filtrado e os pontos com até 5° (alterável pela variável nLeituraFiltro – linha 15) a mais ou a menos, criando-se assim uma versão melhor do mapa:

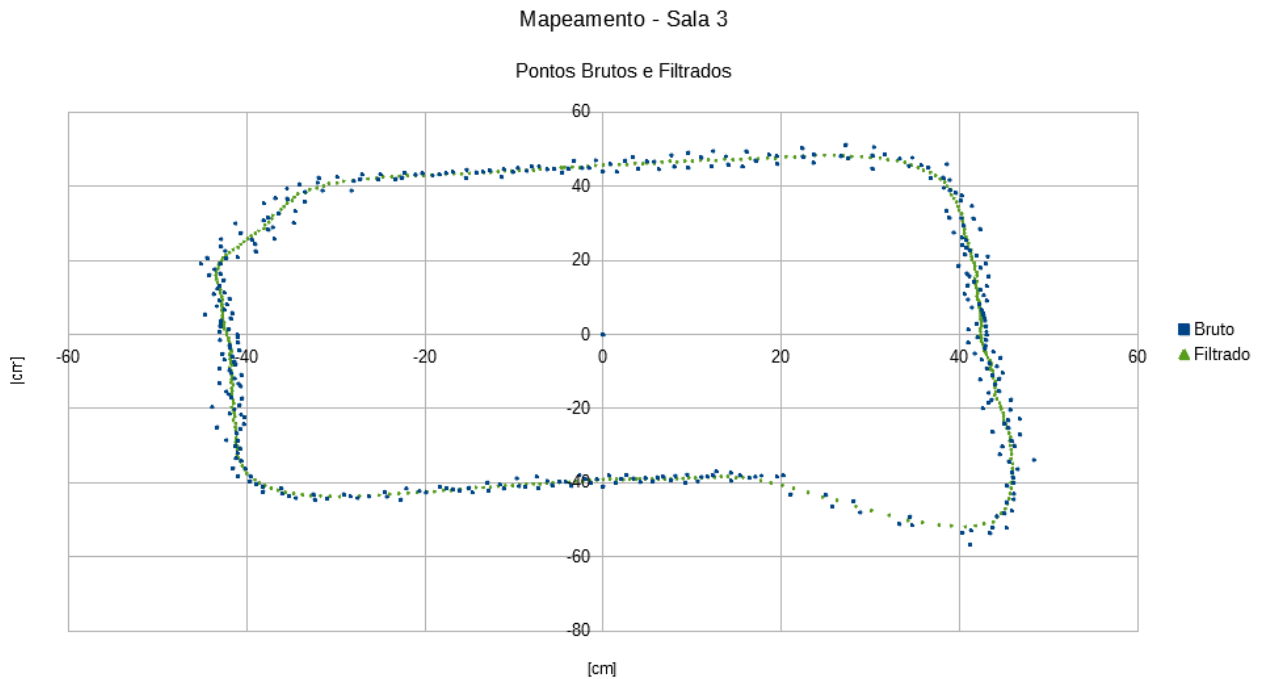


Figura 18: Gráfico - Filtragem do mapa da sala 3

Percebe-se na figura um canto que parece sair do quadrado, sendo esse formado pelas leituras da saída da sala, dois cantos que formam um ângulo de 90° , e um canto que aparenta ter uma linha na diagonal, sendo esse último a zona de resgate. Mas, criou-se um problema de difícil solução: como fazer um algoritmo capaz de encontrar esse canto?

Para isso, foi utilizado o seguinte método:

Considerando que o robô estava alinhado as paredes, com suas laterais paralelas as paredes da sala, utiliza-se os pontos com até 10 cm (variável `distMaxPonto` – linha 14) de distância dos pontos nos ângulos de 0, 90, 180 e 270; para criar, usando regressões lineares, quatro retas que representam as paredes da sala (220 - 252).

Depois, verifica-se os pontos com distância as retas menores a 1,5 cm (`distMax` - 13), e utiliza-se esses pontos para ajustar as retas e recalcular os pontos próximos; esse processo se repete várias vezes (261 - 291).

Por fim, é obtido retas que se ajustam as paredes da sala, como na figura 19:

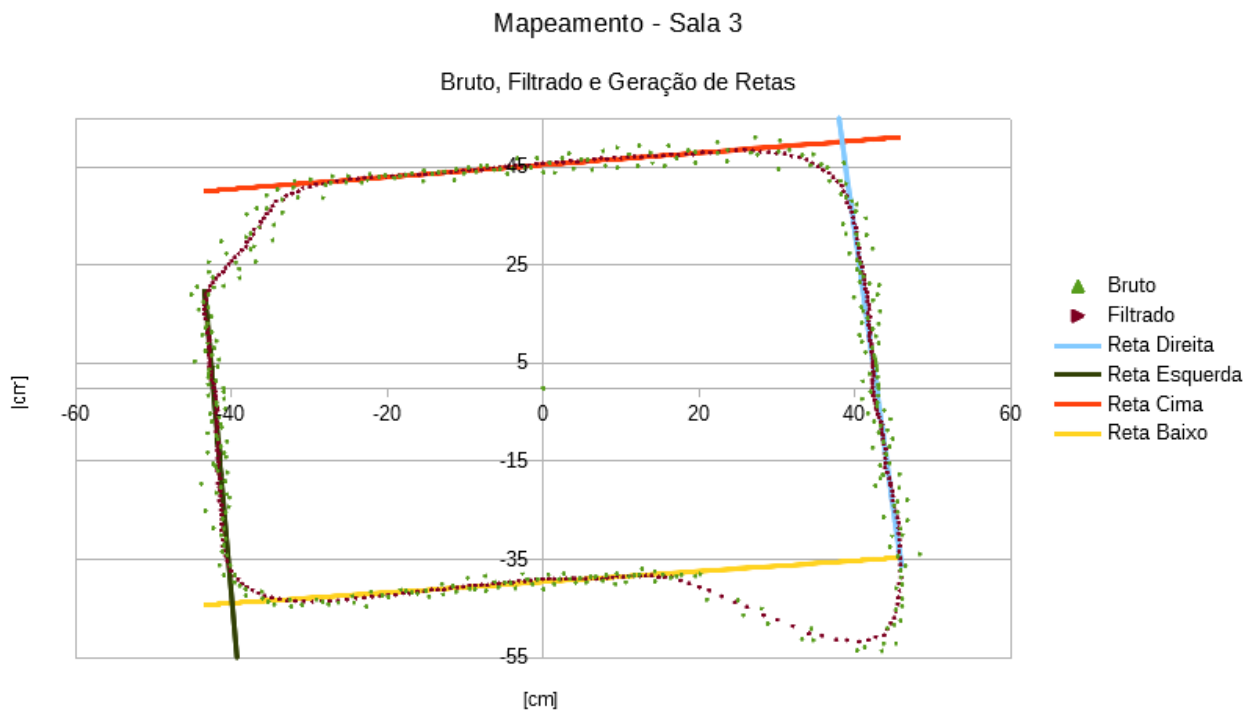


Figura 19: Gráfico - Geração de retas da sala 3

Porém, a real função dessas retas é remover do conjunto de pontos os pontos das paredes, removendo todos os pontos utilizados previamente para criar ou ajustar as retas. (gravados no vetor “usado” - 210). Ainda é preciso remover os pontos da saída da sala, para isso verifica-se quais pontos estão fora do quadrilátero delimitado pelas quatro retas (322 – 401) e considera-os “usados”. Restam então apenas os pontos existentes na figura 20:

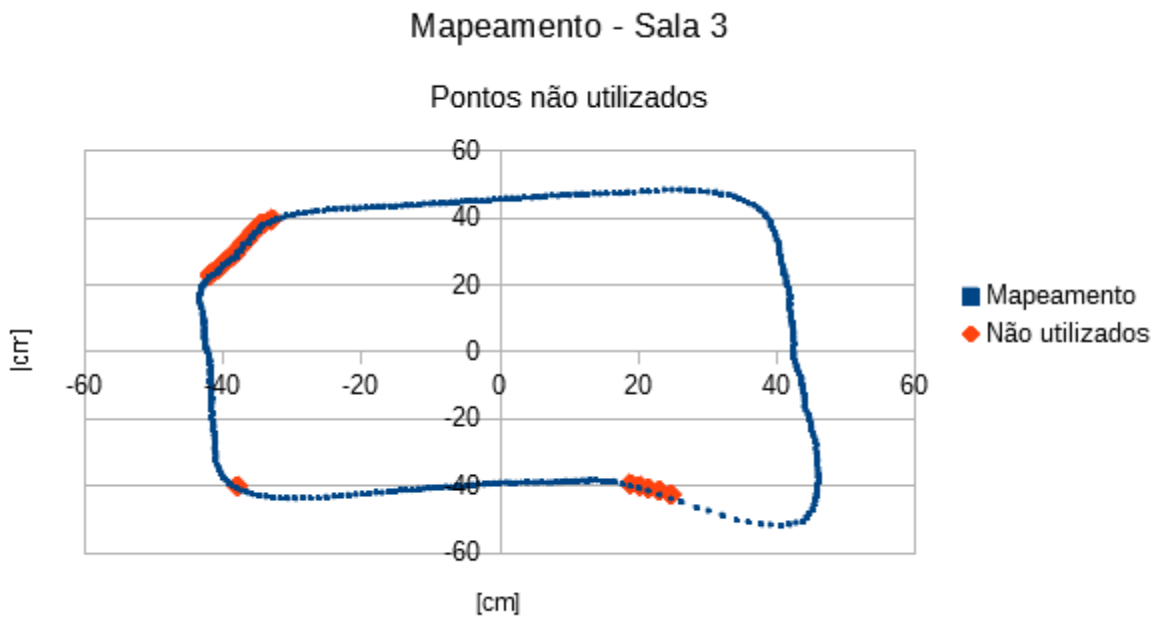


Figura 20: Gráfico - Pontos não utilizados da sala 3

Em seguida, verifica-se a quantidade de pontos restantes em cada conjunto que contém os pontos vizinhos, que possuem diferença de 1° entre eles (416 - 435). Como a zona de resgate é o canto que mais afasta-se das retas, é também o que possui mais pontos. Finalmente, calcula-se o ângulo médio do conjunto com mais pontos (439 - 457). No exemplo das figuras anteriores, o ângulo calculado foi 140° , coincidindo com o local da zona de resgate.

Verificou-se que o algoritmo é capaz de obter corretamente o ângulo da zona de resgate nos quatro testes realizados, porém ao analisar os pontos e retas gerados dos outros testes (figuras 21-23), observa-se que as retas dependem das leituras próximas aos cruzamentos entre os pontos e os eixos, ocasionando em retas longe do ideal em leituras “ruins” próximas a esses locais.

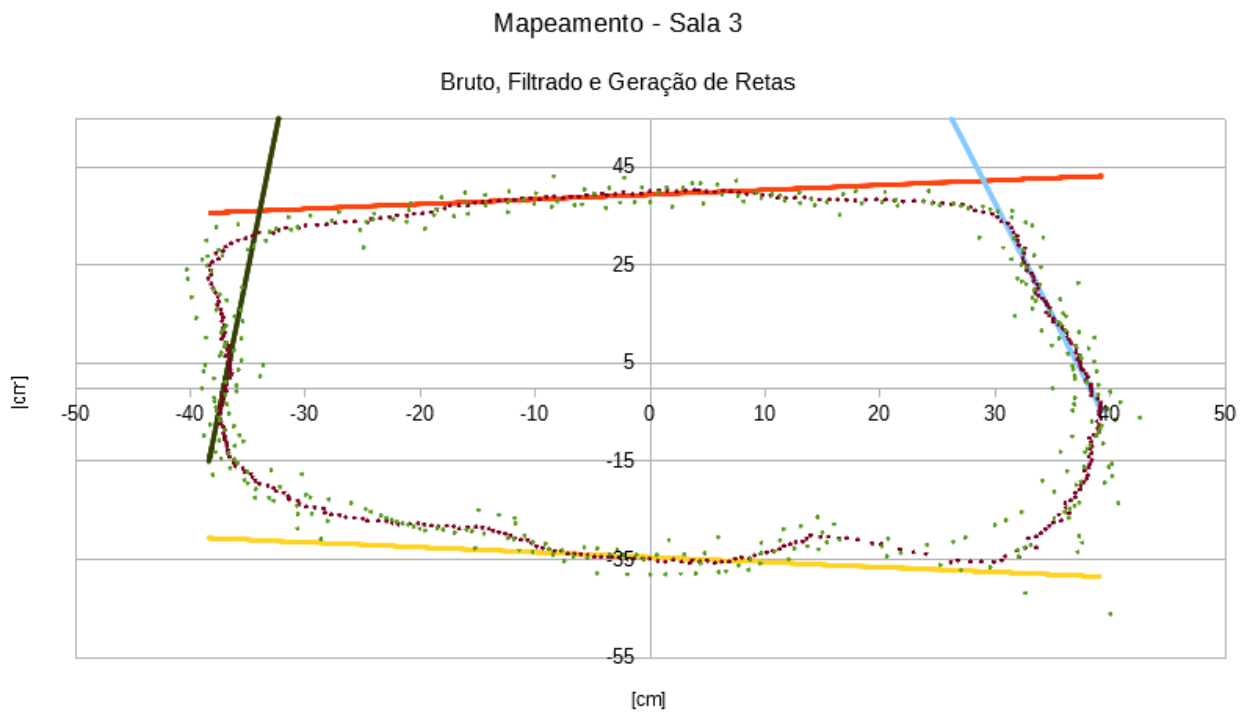


Figura 21: Gráfico - Mapa 2

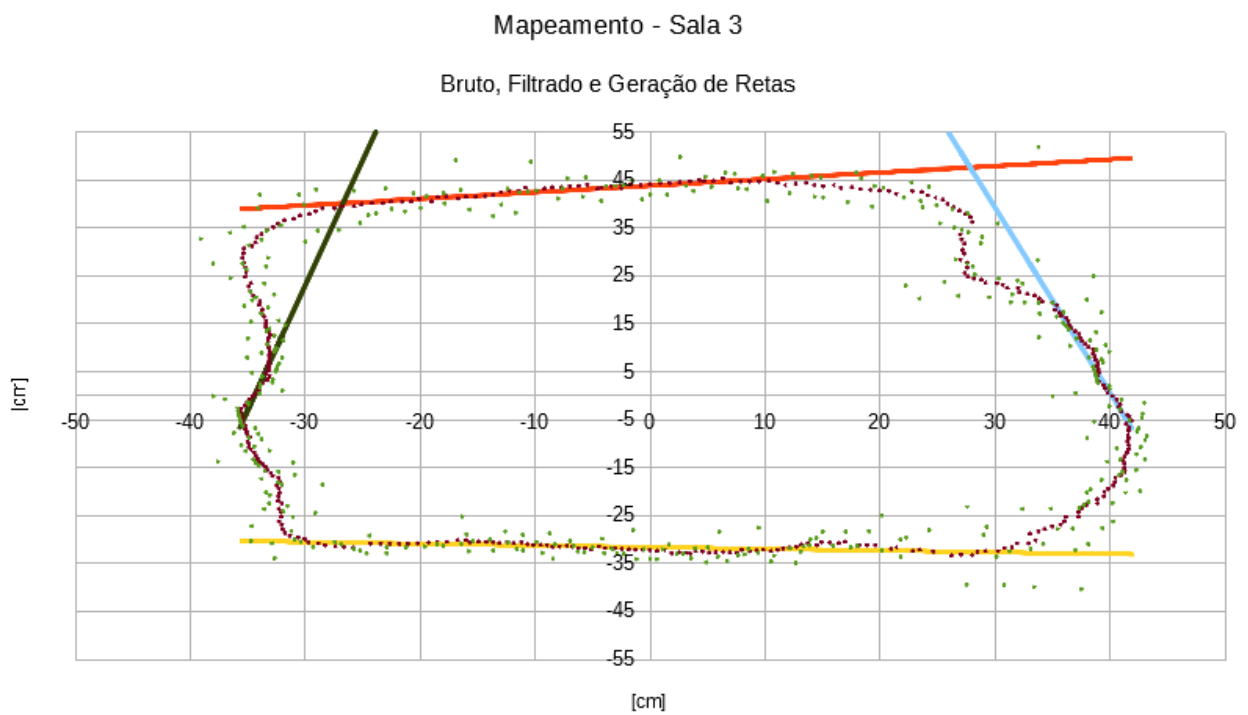


Figura 22: Gráfico - Mapa 3

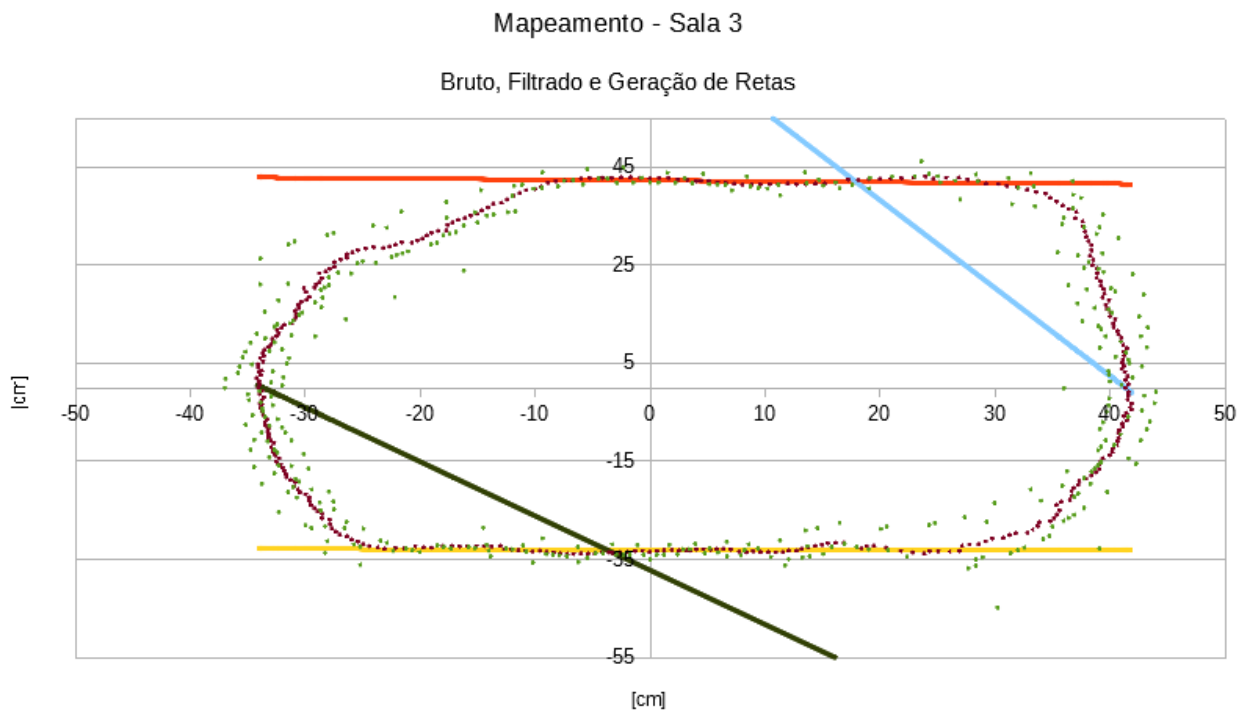


Figura 23: Gráfico - Mapa 4

Percebe-se que as outras três leituras também não geraram mapas muito próximos do retângulo que deveria ter sido. Possíveis causas para isso são a velocidade de rotação do robô e sua alimentação, com níveis de tensão menores o robô tende a sair de seu centro de rotação, ter rotações desiguais e em alguns momentos a parar a rotação. Esse fenômeno aparece principalmente em leituras seguidas uma da outra, com o segundo mapa sempre mais deformado, por conta da resistência interna da bateria.

Existe também o problema da memória do Arduino Mega, que é insuficiente para executar o programa. Para realizar os testes foi utilizado as médias das leituras e ângulos enviados ao computador via comunicação serial, que eram então enviados ao programa sendo executado no computador. Além desses problemas, o algoritmo apresentado somente funciona se o robô é colocado devidamente orientado na sala, e sem nenhuma vítima, sendo que o robô deve capturar todas elas antes de detectar a zona de resgate.

8 - Capítulo 8: Sintonizando o Controlador PID com Algoritmo Genético

Entre as necessidades levantadas, estava a de concluir o trabalho de sintonia do algoritmo PID utilizando um algoritmo genético. Criou-se então uma classe para trabalhar com esse algoritmo (apêndice C), e ela foi testada na sintonização do pidGiro (apêndice D).

8.1 - Definições do Algoritmo Genéticos

A biblioteca implementa um algoritmo genético com codificação por valores, a mutação implementada foi a indutiva, possuindo a possibilidade de ajustar a probabilidade de mutação e valor máximo da mutação no método “setMutacao” (linha 29), e sendo implementada no método “mutacao” (35). O cruzamento foi uniforme, implementada no método “cruzamento”. A seleção utilizada foi a por roleta, implementada no método “selecao” (38). A classe utiliza também a struct “Individuo”, que contém os genes e as aptidões. Originalmente, a biblioteca foi codificada para ser executada em ambientes Windows, utilizando alocação dinâmica, e após adaptada para ser executada pelo Arduino, que não possui a classe “Vector” utilizada no Windows, portanto a quantidade de genes deve ser ajustada manualmente na estrutura, e a classe utiliza dois vetores de indivíduos com o tamanho da população, que devem ser enviados no construtor do objeto.

Por último, a avaliação funciona enviando para o método “setAvaliacao” (28) um ponteiro para uma função que deve receber o vetor de genes e retornar a aptidão, e se o algoritmo deve maximizar ou minimizar as aptidões.

8.2 - Implementando para sintonizar o controlador

Na sintonia do controlador, utilizou-se três genes por indivíduo, sendo cada gene uma constante do algoritmo PID (k_p , k_i , k_d). A avaliação utilizada foi o indicador de desempenho ITSE (integral do erro ao quadrado vezes o tempo, equação I). Porém, como são feitas medidas discretas do erro, e a memória do Arduino é limitada, esse indicador foi ajustado para a equação II:

$$I - ITSE = \int_0^T e^2(t) \cdot t \, dt$$

$$II - \sum_{i=0}^T e_i^2 \cdot t_i$$

Mantendo assim os objetivos do indicador de atribuir mais valor a erros mais tardios no tempo e a erros maiores.

8.3 - Testes com a aproximação de uma função

Para fazer primeiros testes do algoritmo genético, foi tentada uma aproximação para a equação I:

$$I. \quad 5x^2 + 4x + 3$$

Em que os genes representaram os três coeficientes da função (a, b e c), e a aptidão calculada foi gerada pela equação II:

$$II. \quad \sum_{i=-10}^{10} [(5x^2 + 4x + 3) - (ax^2 + bx + c)]^2$$

Como pode-se observar, quanto mais a aproximação se distanciar da função, maior será a aptidão, logo o objetivo do algoritmo foi o de minimizar as aptidões.

Em um teste com 460 gerações de 50 indivíduos, com os genes variando de -100 a 100, e 5% de mutação com valores entre -10 e 10, a evolução das aptidões geradas foi (figura 24):

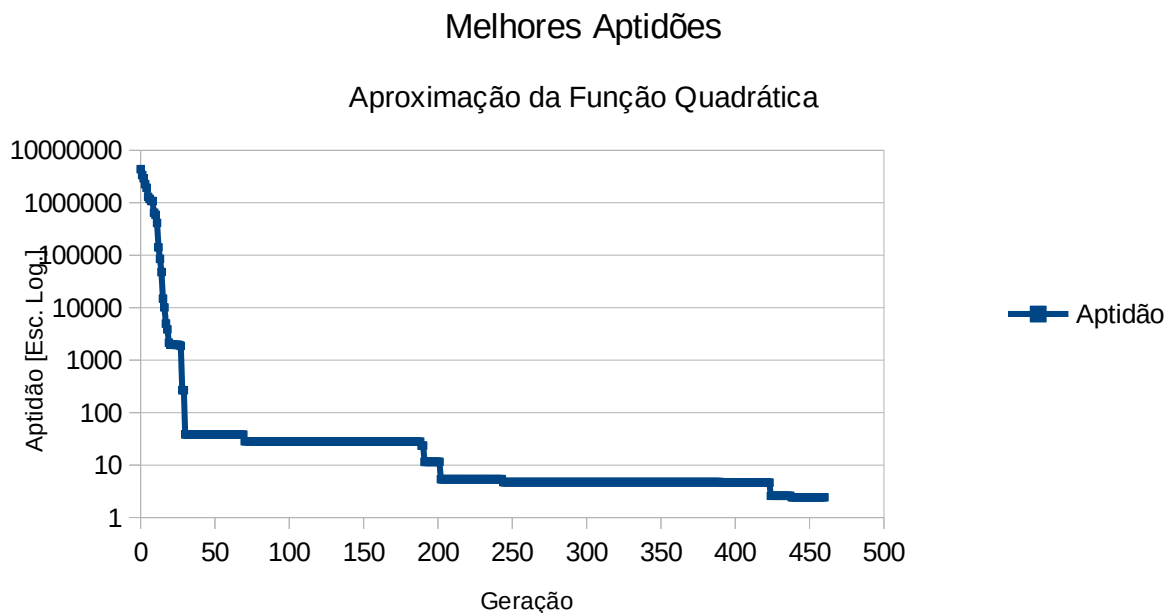


Figura 24: Gráfico - Melhores Aptidões: Aproximação da Função Quadrática

É notável a imensa redução da aptidão, como o desejado. Além disso, observa-se que essa redução pode ter sido muito rápida, possivelmente encontrando máximos locais; e que ela ocorreu em “passos”, com quedas correspondentes a mudanças bruscas nos genes, como observa-se na figura 25:

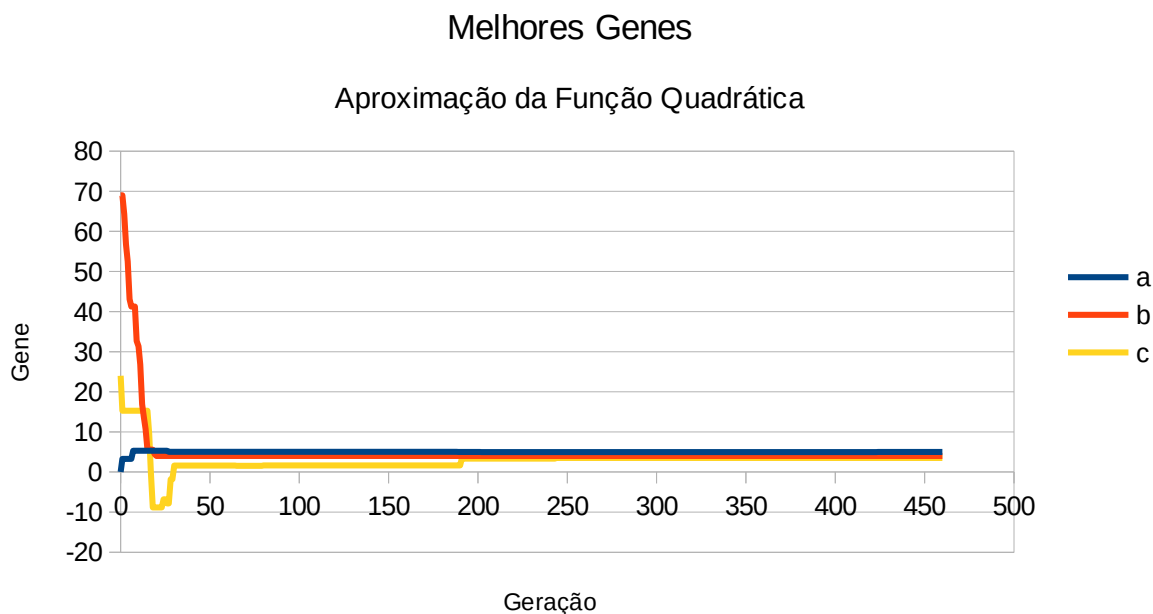


Figura 25: Gráfico - Melhores Genes: Aproximação da Função Quadrática

Nela, percebe-se também que o coeficiente “c” teve uma maior demora em se ajustar, provavelmente pois possui um menor “peso” na função, por ser o coeficiente que acompanha a incógnita com menor expoente (x^0).

Por fim, os resultados gerados foram $a = 4,993$, $b = 4,009$, $c = 3,503$; com o coeficiente c o mais longe do ideal, provavelmente pelo motivo já citado. O coeficiente “a” teve um erro (valor ideal – valor encontrado) menor que 2 já na segunda geração, e um valor menor que 1 na sétima geração; já o coeficiente “b” teve um erro menor que 2 a partir da 15^a geração, e menor que 1 na 19^a; o coeficiente “c” teve um erro maior, sendo menor que 2 a partir da 30^a geração, e menor que 1 a partir da 191^a (coincidindo com as variações bruscas no gráfico).

Como já dito, quanto maior o peso/grau da incógnita que acompanhava o coeficiente, mais rápido foi seu ajuste. E determinou-se que o algoritmo genético estava operante para os testes com o algoritmo PID.

8.4 - Testes com o pidGiro

O algoritmo foi testado no pidGiro, com a aptidão calculada enquanto o robô fazia uma rotação de 45°. Como os dados eram enviados ao computador via comunicação serial utilizando um cabo, os indivíduos foram alternados, um girando 45° e outro girando -45°, evitando assim que o cabo enrolasse. Foram feitas 11 gerações de 35 indivíduos, com genes iniciando entre 50 e -50, probabilidade de mutação de 10% e com valores entre -10 e 10.

Com as 11 gerações, observou-se uma redução da aptidão, o desejado, como pode se observar na figura 26:

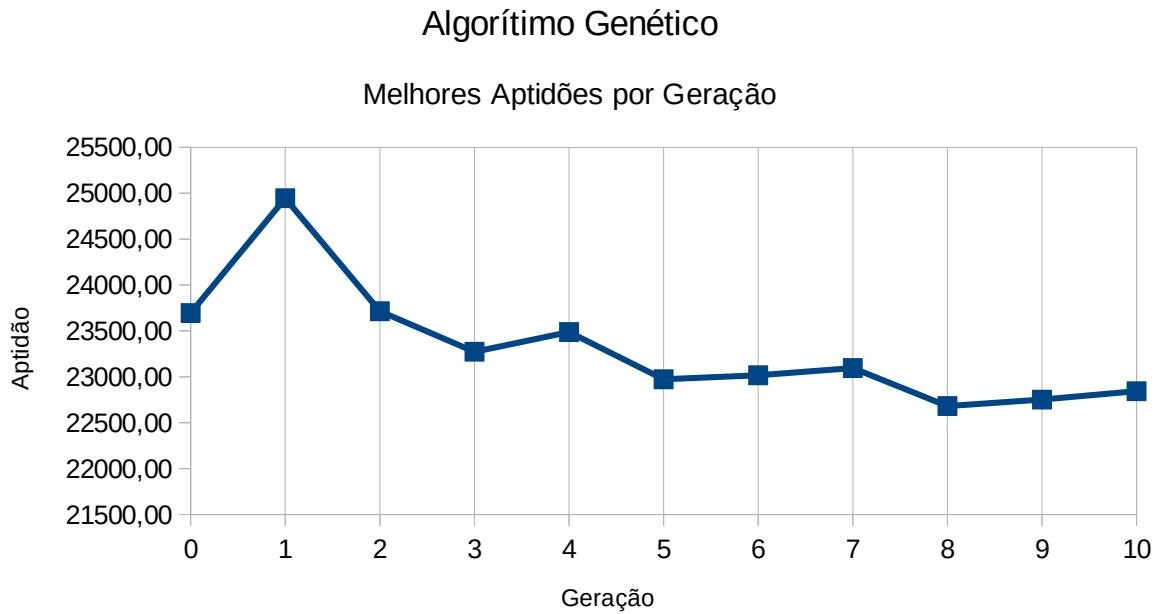


Figura 26: Gráfico - Melhores aptidões do algoritmo genético

Observa-se uma redução da aptidão seguida de estabilizações. Ao observar as alterações dos genes (figura 27), percebe-se que essas estabilizações se devem ao mantimento de um indivíduo melhor nas gerações:

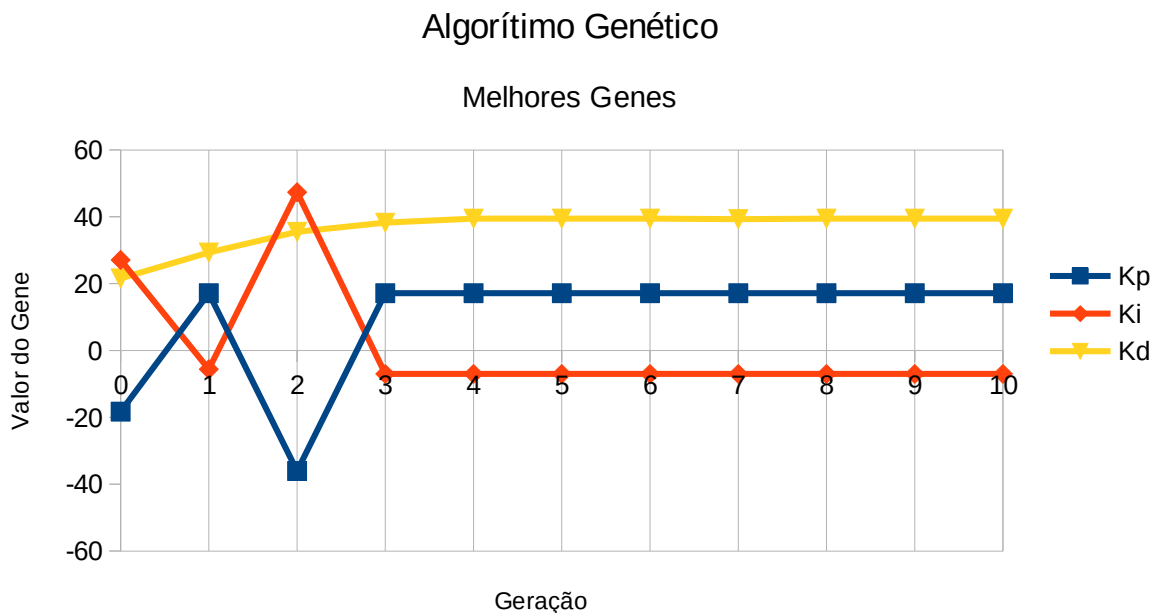


Figura 27: Gráfico - Melhores genes do algoritmo genético

Observa-se também que o algoritmo encontrou um resultado muito rapidamente, sendo esse possivelmente um máximo local, e não o global esperado. Além disso, a constante “ki” obteve um valor negativo, o que não deveria ocorrer com o algoritmo PID, sendo que esses valores foram permitidos no algoritmo para verificar sua potencialidade de encontrar os valores corretos. No final, os valores gerados foram $(k_p, k_i, k_d) = (17,13; -7,02; 39,46)$, mas no uso para as competições a constante “ki” não foi utilizada, e portanto o controlador integral. E como o controlador integral, tendo a constante negativa, tinha um caráter de anular as ações proporcionais, a constante proporcional foi reduzida para “10,13”.

Não foram geradas tantas gerações quanto no outro teste pois os testes com o robô são mais lentos e necessitam de monitoramento para evitar que o cabo continue conectado e que o robô não caia.

9 - Capítulo 9: Resultados e Conclusão

9.1 - Resultados

Nas competições de robótica participadas, a equipe conseguiu uma colocação de 3º lugar no 3º TRIF, e 2º lugar na etapa regional da OBR, conseguindo pela primeira vez no campus uma medalha nessa competição e a classificação para a etapa estadual, em que foi conseguida uma classificação de 41º lugar. Foi também submetido um artigo intitulado “Método para detecção de rampa e vítimas no desafio proposto pela olimpíada brasileira de robótica” para publicação na Mostra Nacional de Robótica de 2018 (aguardando avaliação).

Ocorreram problemas com a alimentação do robô, que chegaram a incapacitar a participação em algumas rodadas. A duração, tamanho, peso e corrente de descarga forma fatores relevantes no problema de alimentação.

A rampa demonstrou-se eficiente para a superação dos redutores, porém a detecção de redutores mostrou-se imprecisa em alguns casos. A programação para o desvio de obstáculos mostrou-se eficaz, porém com erro eventuais decorrentes de falhas do giroscópio e da temporização de movimentos.

Ainda ocorreram falhas na detecção das curvas, que não foram solucionadas. A execução das curvas demonstrou-se um pouco instáveis por conta das alterações feitas para detectar a curva retorno e redutor; sendo que a detecção do redutor precisa de ajustes na quantidade de aceleração que corresponde ao redutor para evitar falhas.

As soluções encontradas para a 3ª sala do percurso mostraram-se promissoras, porém os poucos testes realizados não mostram se são suficientes, e a aplicação do método para detectar a zona de resgate demonstrou-se impossível utilizando o Arduino Mega.

Os métodos para melhorar os sinais dos sensores inerciais não geraram resultados.

Testes realizados em diversos casos mostram a necessidade de obter dados do robô de forma mais prática, o que não ocorre com o cabo USB, pois é incômodo e altera a performance do robô por causa do tempo necessária para a comunicação.

Por fim, o algoritmo genético precisa de mais testes com alterações de parâmetros do algoritmo e do próprio PID para sintonizá-lo corretamente. Acredita-se que o valor obtido foi negativo por conta da saturação do atuador, fenômeno que ocorre quando a saída do controle integral supera o máximo do atuador, o chamado integral WindUp.

9.2 - Conclusão

Analisando as melhorias propostas no relatório anterior, observa-se as soluções encontradas para cada problema, e ao final novos problemas e soluções propostas:

9.2.1 - Melhorar a superação de Obstáculos

Foi possível graças as melhorias feitas com o giroscópio e sensores de distância, sendo um método satisfatório, mas que ainda apresenta falhas devido a movimentos temporizados e falhas no giroscópio.

9.2.2 - Melhoras a detecção de curvas

Nada foi feito em relação a esse problema, pois os pequenos movimentos temporizados que variam de acordo com a tensão do robô aparentemente não possuem substituição possível com o robô atual.

9.2.3 - Criar uma solução para a 3ª sala do percurso

Ocorreram avanços significativos nesse quesito, porém ocorre a falta de memória que dificulta a implementação prática do método de detecção da zona de resgate. Esse método e o de detecção de vítimas precisam de mais testes e de aplicações mais práticas. Ainda há o problema de obter dados do robô de forma prática.

9.2.4 - Estudar melhorias para o tratamento dos sinais dos sensores inerciais

Houve o estudo, porém não encontrou-se nenhuma melhoria possível

9.2.5 - Estudar os sinais obtidos pelo sensor de cor

Com sua remoção, tornou-se desnecessário.

9.2.6 - Encontrar uma solução para superar o redutor, com alterações mecânicas no robô

Problema detectado desde 2016, teve pela primeira vez uma solução eficaz com a rampa implementada na parte frontal do robô.

9.2.7 - Utilizar técnicas de IA (algoritmo genético) para sintonizar o controlador PID

O algoritmo genético foi implementado e testado, porém a sintonia do controlador PID não foi efetuada com sucesso.

9.2.8 - Problemas detectados e possíveis soluções em trabalhos futuros

O problema dos movimentos temporizados, decorrente da imprecisão dos motores, por conta de fatores ambientais e do nível de tensão variável do robô, é um impasse que atrapalha diversas partes do algoritmo. Uma possível solução teria sido os movimentos lineares controlados, porém o sensor acelerômetro demonstrou não ter precisão o suficiente para isso. O uso de encoders nos motores para controlar o deslocamento e velocidade do robô pode ser uma solução, e talvez o uso de filtros de fusão entre eles e o acelerômetro.

O uso de encoders pode também solucionar a imprecisão do giroscópio, controlando-se o giro do robô, talvez também com filtros de fusão.

As únicas soluções encontradas, para a falta de memória do controlador, foram criar outro algoritmo, ou trocar o controlador por outro com mais memória, sendo possível também utilizar em paralelo o arduino mega e outro controlador, como um Raspberry PI.

A dificuldade em alimentar o robô pode ser solucionada por uma bateria de Li Po, como dito anteriormente, porém seu custo é o maior problema, sua massa e volume podem ser contornados com a arquitetura do robô. Em relação a garra, precisa-se definir com um maior planejamento uma

estrutura, verificando-se se é necessário desistir definitivamente da estrutura do robô Zumo de base, por conta de suas limitações mecânicas.

A detecção de curvas, é dependente da velocidade dos motores, já discutida a possível solução, e do tom de verde, sendo que uma melhoria seria o uso de sensores de cor, que dever ser pequeno e possível de ser alocado na parte frontal do robô (talvez o TCS34725 ou APDS9960), alterando-se a estrutura da rampa.

O algoritmo genético precisa de mais testes para sintonizar corretamente o controlador, que precisa ser reformulado.

Em relação a própria atividade do desenvolvimento da pesquisa, a necessidade de obter dados do robô de forma prática e sem atrapalhar sua performance pode ser solucionada com o uso de um cartão SD, que é prático e possui um protocolo de comunicação mais rápido que o serial. É preciso também repensar a metodologia dos testes, para apurar de forma estatisticamente melhor a performance do robô, e a forma de registro das atividades e testes realizados.

10 - Capítulo 10: Caráter Integrador do Projeto

Sendo uma das premissas do projeto integrador a associação de conhecimentos de áreas diversas em um único projeto, acredita-se que o projeto realizado teve esse princípio atingido.

Primeiramente, observa-se os conhecimentos da matéria de Programação necessários para esse projeto, sendo essa a área base que o permeia inteiramente, desde seus conceitos básicos de programação C e Arduino até a exploração de conceitos além da disciplina, como orientação a objetos com C++ e uso de ferramentas como controle de versão.

Na área de Matemática, essa é mais uma vez uma área muito utilizada, com conceitos básicos aritméticos, de geometria analítica, funções, probabilidade, estatística, números complexos, e conceitos além do curso como cálculo integral e derivativo.

As áreas técnicas de Controle de Processos, com o controle PID e ON-OFF; de Processos Industriais, com o conhecimento sobre o processo de impressão 3D; Desenho Técnico, com o conhecimento sobre softwares de modelagem 3D; Instrumentação, com os conhecimentos sobre sensores, como o encoder, e suas características, como a precisão; de Eletricidade e Eletrônica Industrial, com os conhecimentos necessários sobre circuitos eletrônicos; e Eletrônica Digital, com seus operadores lógicos.

Já nas ciências da natureza, a Biologia encontra-se na capacidade de detectar semelhanças conceituais entre os algoritmos genéticos e a teoria sintética da evolução; e a Física com suas grandezas, encontradas principalmente nos sensores inerciais.

A Educação Tecnológica sendo vital no processamento dos dados obtidos através de planilhas eletrônicas; e a Língua Inglesa necessária nas leituras de textos nessa língua; além da produção de textos em Língua Portuguesa.

A articulação do projeto com a Epistemologia, que leva a questionamentos acerca do próprio método utilizado e sua legitimidade (colocar a bateria e motores como culpada quase sempre tornaria a pesquisa infalsificável?); e a capacidade de articular os efeitos dos conhecimentos pesquisados com o trabalho e o futuro da sociedade.

Além da expansão do conhecimento a áreas além do curso, como a computação evolutiva, processamento de dados, cálculo e mapeamento robótico.

Sendo seus métodos a articulação entre ensino (o que foi aprendido), pesquisa (o que foi procurado e desenvolvido), e extensão (o que foi mostrado a comunidade), acredita-se também que foram atingidos esses métodos. E sendo o objetivo a formação da capacidade de intervir na

sociedade, encontra-se aplicações dos conhecimentos obtidos nas mais diversas áreas, como a proposta pela OBR do resgate de vítimas em situações de risco.

Referências Bibliográficas

Robótica. **Dicionário Michaelis On-line**. Disponível em <<http://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/rob%C3%B4/>>. Acesso em 29 nov. 2017.

MCMILLAN, Gregory K. **Process/Industrial Instruments and Controls Handbook**. 5. ed. New York: Mcgraw-hill Education, 1999.

CAPELLI, Alexandre. **Automação Industrial: Controle do movimento e processos contínuos**. 2. ed. São Paulo: Érica, 2008.

ARDUINO. **Introduction: What is Arduino?** Disponível em: <<https://www.arduino.cc/en/Guide/Introduction>>. Acesso em 29 nov. 2017.

THOMSEN, Adilson. **O que é Arduino**. Disponível em: <<https://www.filipeflop.com/blog/o-que-e-arduino/>>; Acesso em 29 nov. 2017.

POLOLU. **Zumo Robot for Arduino, v1.2 (Assembled with 75:1 HP Motors)**. Disponível em: <<https://www.pololu.com/product/2510>>. Acesso em 29 nov. 2017.

BOLTON, W. **Engenharia de Controle**. São Paulo: Macgraw-hill Ltda., 1995.

UNIVERSITY OF MICHIGAN. **Controls Tutorials for Matlab® & Simulink®: Introduction: PID Controller Design**. Disponível em: <<http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion=ControlPID>>. Acesso em 30 nov. 2017.

OBR. **Regras e Instruções: Provas Regionais/Estaduais: Modalidade Prática**. Disponível em: <http://www.obr.org.br/wp-content/uploads/2013/04/Manual_Regras_Pratica_2017_v2_Mai_2017_publicado.pdf>. Acesso em 30 nov. 2017.

OBR. **Manual de Regras e Instruções: Provas Regionais/Estaduais: Versão 1.0 – Março de 2018**. Disponível em:

<http://www.obr.org.br/wp-content/uploads/2018/03/OBR2018_MP_ManualRegrasRegional_v1Mar.pdf>. Acesso em 02 dez. 2018.

OBR. **Modalidade Teórica: Como Funciona**. Disponível em: <<http://www.obr.org.br/como-funciona-modalidade-teorica/>>. Acesso em 30 nov. 2017.

ELECK FREAKS. **Datasheet: Ultrasonic Ranging Module HC-SR04**. Disponível em: <http://img.filipeflop.com/files/download/Datasheet_HCSR04.pdf>. Acesso em 02 dez. 2017.

ADA, Lady [Limor Fried]. **Adafruit VL530LX Time of Flight Micro-LIDAR Distance Sensor Breakout**. 2018. Disponível em: <<https://cdn-learn.adafruit.com/download/pdf/adafruit-vl5310x-micro-lidar-distance-sensor-breakout.pdf?timestamp=1542823307>>. Acesso em 21 nov. 2018.

ALMEIDA, Vinicius Maurício de. **Laboratório Imobilis: Sensores Inerciais**. Disponível em: <<http://www.decom.ufop.br/imobilis/sensores-inerciais/>>. Acessos em 02 dez. 2017.

OBITKO, Marek; SLAVÍK, Pavel. **Uma introdução aos Algoritmos Genéticos com Java applets**. 1998. Traduzido por Hermelindo Pinheiro Manoel. Disponível em: <<http://www.obitko.com/tutorials/genetic-algorithms/portuguese/index.php>>. Acesso em: 04 dez. 2017.

MIRANDA, Marcio Nunes de. **Algoritmos Genéticos: Fundamentos e Aplicações**. Disponível em: <<http://www.nce.ufrj.br/GINAPE/VIDA/alggenet.htm>>. Acesso em: 05 dez. 2017

GARCIA, Claudio. **Controle de processos industriais: Estratégias convencionais: Volume 1** [livro eletrônico]. São Paulo: Blucher, 2018. 600 p.

POZO, Aurora et al. **Computação Evolutiva**. ???. Disponível em: <<http://www.inf.ufpr.br/aurora/tutoriais/Ceapostila.pdf>>. Acesso em: 01 dez. 2018.

Apêndice A: Código Principal

O código principal do programa final criado encontra-se no repositório “Athena” no GitHub, disponível no endereço: “<https://github.com/RoboticaIFSPSuzano/Athenas>”, na aplicação “AthenaArduino”.

Apêndice B: Código para Processamento dos Dados da Zona de Resgate

O código também encontra-se no repositório “Athena” no GitHub (<https://github.com/RoboticalFSPSuzano/Athenas>), na aplicação “Mapeamento”

```
1. #include "pch.h"
2. #include <iostream>
3. #include <math.h>
4. #include <stdlib.h>
5. #include <stdio.h>
6. #include "Planilha.h"
7.
8. #define CE 0
9. #define CD 1
10. #define BE 2
11. #define BD 3
12.
13. float distMax = 1.5;
14. float distMaxPonto = 10 ;
15. int nLeituraFiltro = 10;
16.
17. using namespace std;
18.
19. class Ponto
20. {
21.     private:
22.         int x,
23.         y;
24.
25.     public:
26.
27.         void set(float x, float y)
28.         {
29.             this->x = int(x * 1000);
30.             this->y = int(y * 1000);
31.         }
32.
33.         void setX(float x)
```

```

34.         {
35.             this->x = int(x * 1000);
36.         }
37.
38.         void setY(float y)
39.         {
40.             this->y = int(y * 1000);
41.         }
42.
43.         float getX()
44.         {
45.             return float(x) / 1000;
46.         }
47.
48.         float getY()
49.         {
50.             return float(y) / 1000;
51.         }
52. };
53.
54. class Reta
55. {
56. public:
57.     float a,
58.         b,
59.         sXY,
60.         sX,
61.         sY,
62.         sX2,
63.         sY2,
64.         pearson,
65.         raiz;
66.
67.     int n;
68.
69.     void reset()
70.     {
71.         a = 0;
72.         b = 0;

```

```

73.         sXY = 0;
74.         sX = 0;
75.         sY = 0;
76.         sX2 = 0;
77.         sY2 = 0;
78.         pearson = 0;
79.
80.         n = int(0);
81.     }
82.
83.     void inserir(Ponto a)
84.     {
85.         n += 1;
86.
87.         sXY += a.getX()* a.getY();
88.         sX += a.getX();
89.         sY += a.getY();
90.         sX2 += pow(a.getX(), 2);
91.         sY2 += pow(a.getY(), 2);
92.     }
93.
94.     void calcular()
95.     {
96.         a = ((n*sXY) - (sX*sY)) / ((n*sX2) - (pow(sX, 2)));
97.         b = (sY - (a*sX)) / n;
98.         pearson = ((n*sXY) - (sX*sY)) / sqrt((n*sX2 - pow(sX, 2))*(n*sY2 - pow(sY, 2)));
99.     }
100.
101.     void calcularRaiz()
102.     {
103.         raiz = -b / a;
104.     }
105.
106.     Ponto cruzamento(Reta *r)
107.     {
108.         Ponto c;
109.
110.         c.setX((this->b - r->b) / (r->a - this->a));
111.         c.setY((a*c.getX() + b));

```

```

112.
113.         return c;
114.     }
115.
116.     /*float custo(Ponto a)
117.     {
118.         float tXY = sXY + (a.x*a.y),
119.             tX = sX + a.x,
120.             tY = sY + a.y,
121.             tX2 = sX2 + pow(a.x, 2),
122.             tY2 = sY2 + pow(a.y, 2);
123.
124.         int t = n + 1;
125.
126.         pearson = ((n*sXY) - (sX*sY)) / sqrt((n*sX2 - pow(sX, 2))*(n*sY2 - pow(sY, 2)));
127.
128.         float tPearson = ((t*tXY) - (tX*tY)) / sqrt((t*tX2 - pow(tX, 2))*(t*tY2 - pow(tY, 2)));
129.
130.         return tPearson - pearson;
131.     }*/
132.
133.};
134.
135.float distancia(Ponto a, Ponto b)
136.{
137.    return sqrt(pow(a.getX()- b.getX(), 2) + pow(a.getY() - b.getY(), 2));
138.}
139.
140.float distancia(Ponto ponto, Reta reta)
141.{
142.    return abs((reta.a*ponto.getX()) - ponto.getY() + reta.b) / sqrt(pow(reta.a, 2) + 1);
143.}
144.
145.int main()
146.{
147.    char nome[10];
148.    nome[0] = 'Z';
149.    nome[1] =    '.';
150.    nome[2] = 'c';

```



```

151.     nome[3] = 's';
152.     nome[4] = 'v';
153.     Planilha planilha(nome);
154.
155.     float leitura[360]; //Armazena as leituras do laser
156.
157.     //Recebe as leituras
158.     for (int i = 0; i < 360; i++)
159.     {
160.         cin >> leitura[i];
161.     }
162.
163.     Ponto retangular[360]; //Armazena os pontos cartesianos do mapa
164.
165.     //Calcula os pontos (polar -> retangular)
166.     for (int i = 0; i < 360; i++)
167.     {
168.         retangular[i].set( leitura[i] * cos(i*0.0174533), leitura[i] * sin(i*0.0174533));
169.     }
170.
171.     int nLeitura;
172.
173.     Ponto retangularF[360]; //Armazena os pontos filtrados da sala
174.
175.     //Filtra o mapa utilizando média móvel
176.     for (int i = 0; i < 360; i++)
177.     {
178.         int indice = i - (nLeituraFiltro / 2);
179.
180.         nLeitura = 0;
181.
182.         if (indice < 0)
183.         {
184.             indice += 360;
185.         }
186.
187.         retangularF[i].set(0, 0);
188.
189.         for (int j = 0; j < nLeituraFiltro+1; j++)

```

```

190.         {
191.             if (retangular[indice].getX() != 0 && retangular[indice].getY() != 0)
192.                 {
193.                     retangularF[i].set(retangular[indice].getX() + retangularF[i].getX(),
retangular[indice].getY() + retangularF[i].getY());
194.
195.                     nLeitura += 1;
196.                 }
197.
198.
199.             indice += 1;
200.
201.             if (indice > 359)
202.                 {
203.                     indice = 0;
204.                 }
205.         }
206.
207.         retangularF[i].set(retangularF[i].getX() / nLeitura, retangularF[i].getY() / nLeitura);
208.     }
209.
210.     int usado[360]; //Indica se o ponto já foi utilizado para alguma reta
211.
212.     Reta reta[4]; //Armazena as 4 retas
213.
214.     //Inicia as retas
215.     reta[0].reset();
216.     reta[1].reset();
217.     reta[2].reset();
218.     reta[3].reset();
219.
220.     //Verifica quais pontos pertencem as retas
221.     for (int i = 0; i < 360; i++)
222.     {
223.         usado[i] = 0;
224.
225.         if (distancia(retangularF[0], retangularF[i]) <= distMaxPonto) //Verifica se o ponto está
próximo do ponto 0º
226.             {

```

```

227.             reta[0].inserir(retangularF[i]);
228.
229.             usado[i] = 1;
230.         }
231.
232.         if (distancia(retangularF[180], retangularF[i]) <= distMaxPonto) //Verifica se o ponto está
próximo do ponto 180°
233.         {
234.             reta[1].inserir(retangularF[i]);
235.
236.             usado[i] = 1;
237.         }
238.
239.         if (distancia(retangularF[90], retangularF[i]) <= distMaxPonto) //Verifica se o ponto está
próximo do ponto 90°
240.         {
241.             reta[2].inserir(retangularF[i]);
242.
243.             usado[i] = 1;
244.         }
245.
246.         if (distancia(retangularF[270], retangularF[i]) <= distMaxPonto) //Verifica se o ponto está
próximo do ponto 270°
247.         {
248.             reta[3].inserir(retangularF[i]);
249.
250.             usado[i] = 1;
251.         }
252.     }
253.
254.     //Calcula os coeficientes das retas
255.     for (int i = 0; i < 4; i++)
256.     {
257.         reta[i].calcular();
258.     }
259.
260.     //Reajusta as retas adicionando os pontos próximos a elas
261.     for (int j = 0; j < 10; j++)
262.     {

```

```

263.         for (int i = 0; i < 360; i++)
264.         {
265.             if (distancia(retangularF[i], reta[0]) < distMax && usado[i] == 0) //Verifica se o
                ponto está próximo da reta
266.             {
267.                 reta[0].inserir(retangularF[i]); //Insere o ponto na reta
268.                 usado[i] = 1; //Indica que o ponto foi utilizado
269.                 reta[0].calcular(); //Reajusta a reta
270.             }
271.             if (distancia(retangularF[i], reta[1]) < distMax && usado[i] == 0)
272.             {
273.                 reta[1].inserir(retangularF[i]);
274.                 usado[i] = 1;
275.                 reta[1].calcular();
276.             }
277.             if (distancia(retangularF[i], reta[2]) < distMax && usado[i] == 0)
278.             {
279.                 reta[2].inserir(retangularF[i]);
280.                 usado[i] = 1;
281.                 reta[2].calcular();
282.             }
283.             if (distancia(retangularF[i], reta[3]) < distMax && usado[i] == 0)
284.             {
285.                 reta[3].inserir(retangularF[i]);
286.                 usado[i] = 1;
287.                 reta[3].calcular();
288.             }
289.         }
290.     }
291. }
292.
293.     int cima, baixo, esq, dir; //Armazenam quais retas estão em quais lados do quadrilátero
294.
295.     dir = 0;
296.     esq = 1;
297.     cima = 2;
298.     baixo = 3;
299.
300.     //Recalcula as retas e calcula as raizes

```

```

301.     for (int i = 0; i<4; i++)
302.     {
303.         reta[i].calcular();
304.         reta[i].calcularRaiz();
305.     }
306.
307.
308.     Ponto cruzamento[4]; //Armazenam os cruzamentos entre as retas
309.
310.     //Calcula os cruzamentos entre as retas
311.     cruzamento[CE] = reta[cima].cruzamento(&reta[esq]);
312.     cruzamento[CD] = reta[cima].cruzamento(&reta[dir]);
313.     cruzamento[BE] = reta[baixo].cruzamento(&reta[esq]);
314.     cruzamento[BD] = reta[baixo].cruzamento(&reta[dir]);
315.
316.     Reta r; //segmento de reta auxiliar
317.
318.     //Inicia o segmento de reta auxiliar
319.     r.a = 0;
320.
321.
322.     //Remove os pontos fora do quadrilátero
323.     for (int i = 0; i < 360; i++)
324.     {
325.         if (usado[i] == 0)
326.         {
327.
328.             //Insere no segmento de reta o coeficiente do ponto analisado
329.             r.b = retangularF[i].getY();
330.
331.             //Segmento de reta que parte do ponto e segue ao infinito para a esquerda OU
direita
332.
333.             //Pontos de encontro entre as retas e o segmento
334.             Ponto encontro[4];
335.
336.             //Calcula os cruzamentos entre as retas e o segmento
337.             for (int j = 0; j < 4; j++)
338.             {

```

```

339.             encontro[j] = reta[j].cruzamento(&r);
340.         }
341.
342.         int vezes = 0; //Armazena quantas vezes o segmento cruza o quadrilátero
343.
344.         //Verifica quantas vezes o segmento de reta cruza o quadrilátero
345.
346.         //Verifica se cruza com a reta de cima
347.         if (encontro[cima].getX() > cruzamento[CE].getX() && encontro[cima].getX() <
cruzamento[CD].getX())
348.         {
349.             vezes += 1;
350.         }
351.
352.         //Verifica se cruza com a reta esquerda
353.         if (encontro[esq].getX() > retangularF[i].getX())
354.         {
355.             if (cruzamento[CE].getX() > cruzamento[BE].getX())
356.             {
357.                 if (encontro[esq].getX() > cruzamento[BE].getX() &&
encontro[esq].getX() < cruzamento[CE].getX())
358.                 {
359.                     vezes += 1;
360.                 }
361.             }
362.             else
363.             {
364.                 if (encontro[esq].getX() > cruzamento[CE].getX() &&
encontro[esq].getX() < cruzamento[BE].getX())
365.                 {
366.                     vezes += 1;
367.                 }
368.             }
369.         }
370.
371.         //Verifica se cruza com a reta de baixo
372.         if (encontro[baixo].getX() > cruzamento[BE].getX() && encontro[baixo].getX() <
cruzamento[BD].getX())
373.         {

```

```

374.             vezes += 1;
375.         }
376.
377.         //Verifica se cruza com a reta direita
378.         if (encontro[dir].getX() >= retangularF[i].getX())
379.         {
380.             if (cruzamento[CD].getX() > cruzamento[BD].getX())
381.             {
382.                 if (encontro[dir].getX() < cruzamento[BD].getX() &&
encontro[dir].getX() > cruzamento[CD].getX())
383.                 {
384.                     vezes += 1;
385.                 }
386.             }
387.             else
388.             {
389.                 if (encontro[dir].getX() > cruzamento[CD].getX() &&
encontro[dir].getX() < cruzamento[BD].getX())
390.                 {
391.                     vezes += 1;
392.                 }
393.             }
394.         }
395.         //Se não cruzar uma vez e apenas uma = fora do quadrilátero
396.         if (vezes != 1)
397.         {
398.             usado[i] = 1;
399.         }
400.     }
401. }
402.
403. int cluster[5]; //Angulo médio dos conjuntos de pontos
404. int nPonto[5]; //N de pontos nos conjuntos
405. int anterior = -1; //Indice do ponto anterior
406.
407. int indice = 0; //Indice do conjunto de pontos atual
408.
409. //Reseta os conjuntos
410. for (int i = 0; i < 5; i++)

```

```

411.     {
412.         nPonto[i] = 0;
413.         cluster[i] = 0;
414.     }
415.
416.     //Forma os conjuntos
417.     for (int i = 0; i < 360; i++)
418.     {
419.         if (usado[i] == 0) //Verifica se o ponto deve pertencer a algum conjunto
420.         {
421.             if (i - anterior < 2 || anterior == -1) //Verifica se o ponto pertence ao conjunto
anterior
422.                 {
423.                     cluster[indice] += i;
424.
425.                     nPonto[indice] += 1;
426.
427.                     anterior = i;
428.                 }
429.             else //Inicia um novo conjunto
430.             {
431.                 anterior = i;
432.                 indice += 1;
433.             }
434.         }
435.     }
436.
437.     int maior = 0; //Armazena o indice do conjunto com mais pontos
438.
439.     //Calcula os angulos médios
440.     for (int i = 0; i < 5; i++)
441.     {
442.         if (nPonto[i] != 0)
443.         {
444.             cluster[i] /= nPonto[i];
445.
446.             //cout << cluster[i] << " " << nPonto[i];
447.         }
448.     }

```



```

449.
450. //Verifica qual o maior conjunto
451. for (int i = 0; i < 5; i++)
452. {
453.     if (nPonto[i] > nPonto[maior])
454.     {
455.         maior = i;
456.     }
457. }
458.
459. for (int i = 0; i < 360; i++)
460. {
461.     planilha.escrever(retangular[i].getX());
462.     planilha.escrever(retangular[i].getY());
463.     planilha.escrever(retangularF[i].getX());
464.     planilha.escrever(retangularF[i].getY());
465.
466.     if (i <= 3)
467.     {
468.         planilha.escrever(reta[i].a);
469.         planilha.escrever(reta[i].b);
470.     }
471.     else
472.     {
473.         planilha.novaColuna();
474.         planilha.novaColuna();
475.     }
476.
477.     if (usado[i] == 0)
478.     {
479.         planilha.escrever(retangularF[i].getX());
480.         planilha.escrever(retangularF[i].getY());
481.     }
482.     else
483.     {
484.         planilha.novaColuna();
485.         planilha.novaColuna();
486.     }
487.

```

```
488.         if (i == 0)
489.             {
490.                 planilha.escrever(cluster[maior]);
491.             }
492.
493.         planilha.novaLinha();
494.     }
495.
496.     cout << cluster[maior]; //Imprime o angulo do maior conjunto = conjunto da zona de resgate
497.     return 0;
498. }
```

Apêndice C: Algoritmo Genético - Biblioteca

Cabeçalho (.h):

```
1. #ifndef GENETICOREAL_H
2. #define GENETICOREAL_H
3.
4. #include <Arduino.h>
5.
6. struct Indivuido
7. {
8.     double gene[3];
9.     double aptidao,
10.     normAptidao;
11. };
12.
13. typedef double (*FuncaoAvaliacao) (double*);
14.
15. /*struct Indivuido
16. {
17.     float gene[];
18.     double aptidao,
19.     normAptidao;
20. };*/
21.
22. class GeneticoReal
23. {
24. public:
25.     GeneticoReal(int nIndivuido, int nGene, double maxGene, double minGene, Indivuido *indivuido,
Indivuido *novaGeracao);
26.
27.     void evoluir(int vezes);
28.     void setAvaliacao(FuncaoAvaliacao funcao, bool min);
29.     void setMutacao(int probMut, double maxMut, double minMut);
30.     void setExportacao();
31.
32.     Indivuido melhorIndivuido();
33.
34. protected:
35.     void mutacao(int nIndivuido);
36.     void ordenar();
37.     void normalizarAptidao();
38.     int selecao();
39.
40.     Indivuido cruzamento(int ind1, int ind2);
41.
42.     double (*avaliar)(double*);
43.
44.     Indivuido *indivuido;
45.     Indivuido *novaGeracao;
46.
47.
48.     double maxMut,
```

```

49.     minMut,
50.     maxGene,
51.     minGene;
52.
53.     int nIndividuo,
54.     nGeracao,
55.     nGene,
56.     probMut,
57.     exportar;
58.
59.     bool min;
60. };
61.
62. #endif
63.

```

Implementação (.cpp)

```

1. #include "GeneticoReal.h"
2.
3. GeneticoReal::GeneticoReal(int nIndividuo, int nGene, double maxGene, double minGene, Individuo
   *individuo, Individuo *novaGeracao)
4. {
5.     this->nIndividuo = nIndividuo;
6.     this->nGene = nGene;
7.     this->maxGene = maxGene;
8.     this->minGene = minGene;
9.     this->individuo = individuo;
10.    this->novaGeracao = novaGeracao;
11.
12.    nGeracao = 0;
13.    probMut = 50;
14.    maxMut = 0.01;
15.    minMut = -0.01;
16.    exportar = 0;
17.
18.    bool min = 0;
19.
20.    for(int i = 0; i<nIndividuo; i++)
21.        {
22.            for(int j = 0; j<nGene; j++)
23.                {
24.                    double num = double(random(minGene*100,
maxGene*100))/100.0;

```

```

25.
26.             individuo[i].gene[j] = num;
27.         }
28.     }
29. }
30.
31. void GeneticoReal::setAvaliacao(FuncaoAvaliacao funcao, bool min)
32. {
33.     this->min = min;
34.     this->avaliar = funcao;
35. }
36.
37. void GeneticoReal::evoluir(int vezes)
38. {
39.     for(int i = 0; i<=vezes; i++)
40.     {
41.         for(int j = 0; j<nIndividuo; j++)
42.         {
43.             individuo[j].aptidao = avaliar(individuo[j].gene);
44.         }
45.         ordenar();
46.
47.         normalizarAptidao();
48.
49.
50.         if(i != (vezes))
51.         {
52.             if(exportar)
53.             {
54.                 Serial.println();
55.
56.                 char identificador[15] = "Geracao";
57.
58.                 char numero[10];
59.
60.                 itoa(nGeracao, numero, 10);
61.
62.                 strcat(identificador, numero);
63.

```

```

64.                                     Serial.println(identificador);
65.                                     }
66.                                 for(int j = 0; j<nIndividuo; j++)
67.                                     {
68.
69.                                         int ind1 = selecao();
70.                                         int ind2 = selecao();
71.
72.                                         novaGeracao[j] = cruzamento(ind1,
ind2);
73.
74.                                         if(exportar)
75.                                             {
76.                                                 for(int k = 0;
k<nGene; k++)
77.                                                     {
78.
79.                                     Serial.print(individuo[j].gene[k]);
80.
81.                                     Serial.print(";");
82.
83.                                     Serial.print(individuo[j].aptidao);
84.
85.                                     Serial.print(";");
86.
87.                                     Serial.println(individuo[j].normAptidao);
88.
89.                                     }
90.
91.                                     }
92.
93.                                     for(int i = 0; i<nIndividuo; i++)
94.                                         {
95.                                             for(int j = 0; j<nGene; j++)
96.                                                 {
97.
98.                                                     individuo[i].gene[j]
= novaGeracao[i].gene[j];
99.
100.                                                 }
101.                                         }

```

```

94.                                individuo[i].aptidao           =
    novaGeracao[i].aptidao;
95.                                individuo[i].normAptidao       =
    novaGeracao[i].normAptidao;
96.                                }
97.
98.                                nGeracao += 1;
99.
100.                               for(int j = 0; j<nIndividuo; j++)
101.                               {
102.                                   mutacao(j);
103.                               }
104.
105.                               }
106.                               }
107. }
108.
109.void GeneticoReal::ordenar()
110. {
111.     Individuo cesta;
112.
113.     for(int i = 0; i<nIndividuo; i++)
114.     {
115.         for(int j = i; j<nIndividuo; j++)
116.         {
117.             if(individuo[j].aptidao < individuo[i].aptidao)
118.             {
119.                 cesta = individuo[j];
120.
121.                 individuo[j] = individuo[i];
122.
123.                 individuo[i] = cesta;
124.             }
125.         }
126.     }
127. }
128.
129.void GeneticoReal::normalizarAptidao()
130. {

```

```

131.         double maxApt = individuo[0].aptidao,
132.                minApt = individuo[0].aptidao;
133.
134.         for(int i=0; i<nIndividuo; i++)
135.             {
136.                 if(individuo[i].aptidao > maxApt)
137.                     {
138.                         maxApt = individuo[i].aptidao;
139.                     }
140.                 if(individuo[i].aptidao < minApt)
141.                     {
142.                         minApt = individuo[i].aptidao;
143.                     }
144.             }
145.
146.         for(int i = 0; i<nIndividuo; i++)
147.             {
148.                 individuo[i].normAptidao = ((individuo[i].aptidao - minApt)/(maxApt-
149.                 minApt))*100; //Colocar a aptidão em um valor entre 0.0 e 100.0
150.
151.                 if(min)
152.                     {
153.                         individuo[i].normAptidao = 100-
154.                         individuo[i].normAptidao;
155.                     }
156.             }
157. void GeneticoReal::mutacao(int nIndividuo)
158. {
159.     int prob;
160.
161.     for(int i = 0; i<nGene; i++)
162.         {
163.             prob = random(101);
164.
165.             if(prob < probMut)
166.                 {

```



```

167.         double    num    =    double(random(minMut*100,
           maxMut*100))/100.0;
168.
169.         individuo[nIndividuo].gene[i] += num;
170.
171.         if(individuo[nIndividuo].gene[i] > maxGene)
172.             {
173.                 individuo[nIndividuo].gene[i] =
           maxGene;
174.             }
175.         else if(individuo[nIndividuo].gene[i] < minGene)
176.             {
177.                 individuo[nIndividuo].gene[i] =
           minGene;
178.             }
179.         }
180.     }
181. }
182.
183. void GeneticoReal::setMutacao(int probMut, double maxMut, double minMut)
184. {
185.     this->probMut = probMut;
186.     this->maxMut = maxMut;
187.     this->minMut = minMut;
188. }
189.
190. Individuo GeneticoReal::cruzamento(int ind1, int ind2)
191. {
192.     int num;
193.
194.     Individuo novoIndividuo;
195.
196.     for(int i = 0; i<nGene; i++)
197.         {
198.             num = random(2);
199.
200.             if(num == 1)
201.                 {
202.                     novoIndividuo.gene[i] = individuo[ind1].gene[i];

```

```

203.         }
204.     else
205.     {
206.         novoIndividuo.gene[i] = individuo[ind2].gene[i];
207.     }
208. }
209.
210.     return novoIndividuo;
211. }
212.
213. int GeneticoReal::selecao()
214. {
215.     double maxApt;
216.     int indice;
217.
218.     for(int i = 0; i<5; i++)
219.     {
220.         int num = random(nIndividuo+1);
221.
222.         if(i == 0)
223.         {
224.             maxApt = individuo[num].normAptidao;
225.             indice = num;
226.         }
227.         else if(individuo[num].normAptidao>maxApt)
228.         {
229.             maxApt = individuo[num].normAptidao;
230.             indice = num;
231.         }
232.     }
233.
234.
235.     return indice;
236.
237.     /*int num = -50 + rand() % 151; //Gera um número entre 0.0 e 100.0
238.
239.     for(int i = 0; i< nIndividuo; i++)
240.     {
241.         if(num<individuo[i].normAptidao)

```

```
242.         {
243.             return i;
244.         }
245.     }
246.     return nIndividuo-1;*/
247. }
248.
249.Individuo GeneticoReal::melhorIndividuo()
250. {
251.     return individuo[0];
252. }
253.
254.void GeneticoReal::setExportacao()
255. {
256.     exportar = 1;
257.}
```

Apêndice D: Código Algoritmo Genético

```
1. #include <ZumoMotors.h> //Motores
2. #include <Pushbutton.h> //Botão
3. #include <Wire.h>
4. #include <L3G.h> //Giroscópio
5. #include <FiltroKalman.h> //Filtro de Kalman
6.
7. #include <GeneticoReal.h>
8.
9. //Fim da inclusão de bibliotecas-----
10.
11. #define nMedidaAcel 20
12. #define eX 1
13. #define eY 2
14. #define eZ 3
15.
16. //Declaração de Variáveis -----
17.
18.
19. //Fim da declaração de variáveis-----
20.
21. //Declaração de Classes -----
22.
23. class Giroscopio : public L3G //Controla o sensor giroscópio
24. {
25.     public:
26.         struct Dados
27.         {
28.             float x,
29.             y,
30.             z;
31.         } velocidade, grau, velocidadeAnt , ganho, velocidadeBruta, ganhoBruto, maximo, minimo;
32.
33.         unsigned long tempo; //Tempo da última leitura
34.         float deltaT; //Tempo entre duas leituras
35.
```

```

36. //Filtros de Kalman
37. FiltroKalman1D filtroX,
38.     filtroY,
39.     filtroZ;
40.
41. Giroscopio() //Inicia os filtros e os ganhos
42. {
43.     //Suave/Rápido
44.     filtroX.init(0.0009, 0.15);
45.     filtroY.init(0.0009, 0.15);
46.     filtroZ.init(0.0009, 0.15);
47.
48.     ganho.x = 0;
49.     ganho.y = 0;
50.     ganho.z = 0;
51.
52.     ganhoBruto.x = 0;
53.     ganhoBruto.y = 0;
54.     ganhoBruto.z = 0;
55. }
56.
57. void calibrar() //Calibra os três eixos
58. {
59.     Dados somatorio,
60.     somatorioBruto;
61.
62.     somatorio.x = 0;
63.     somatorio.y = 0;
64.     somatorio.z = 0;
65.
66.     somatorioBruto.x = 0;
67.     somatorioBruto.y = 0;
68.     somatorioBruto.z = 0;
69.
70.     for(int i = 0; i<100; i++)
71.     {
72.         leitura();
73.
74.         somatorio.x+= velocidade.x;

```

```
75. somatorio.y+= velocidade.y;
76. somatorio.z+= velocidade.z;
77.
78. somatorioBruto.x+= velocidadeBruta.x;
79. somatorioBruto.y+= velocidadeBruta.y;
80. somatorioBruto.z+= velocidadeBruta.z;
81.
82. if(i == 0)
83.     {
84.         maximo.x = velocidade.x;
85.         maximo.y = velocidade.y;
86.         maximo.z = velocidade.z;
87.
88.         minimo.x = velocidade.x;
89.         minimo.y = velocidade.y;
90.         minimo.z = velocidade.z;
91.     }
92. else
93.     {
94.         if(velocidade.x > maximo.x)
95.             {
96.                 maximo.x = velocidade.x;
97.             }
98.         if(velocidade.x < minimo.x)
99.             {
100.                minimo.x = velocidade.x;
101.            }
102.
103.         if(velocidade.y > maximo.y)
104.             {
105.                 maximo.y = velocidade.y;
106.             }
107.         if(velocidade.y < minimo.y)
108.             {
109.                 minimo.y = velocidade.y;
110.            }
111.
112.         if(velocidade.z > maximo.z)
113.             {
```

```

114.         maximo.z = velocidade.z;
115.     }
116.     if(velocidade.z < minimo.z)
117.     {
118.         minimo.z = velocidade.z;
119.     }
120.
121.     }
122.
123.     }
124.
125.     ganho.x += somatorio.x/100;
126.     ganho.y += somatorio.y/100;
127.     ganho.z += somatorio.z/100;
128.
129.     ganhoBruto.x += somatorioBruto.x/100;
130.     ganhoBruto.y += somatorioBruto.y/100;
131.     ganhoBruto.z += somatorioBruto.z/100;
132.
133.     maximo.x -= ganho.x;
134.     maximo.y -= ganho.y;
135.     maximo.z -= ganho.z;
136.
137.     minimo.x -= ganho.x;
138.     minimo.y -= ganho.y;
139.     minimo.z -= ganho.z;
140. }
141.
142. void calibrar(int eixo) //Calibra apenas o eixo selecionado
143. {
144.
145.     float somatorio = 0,
146.         somatorioBruto = 0;
147.
148.
149.     switch(eixo)
150.     {
151.         case 1:
152.             for(int i = 0; i<100; i++)

```

```

153.     {
154.         leitura(eixo);
155.
156.         somatorio+= velocidade.x;
157.
158.         somatorioBruto+= velocidadeBruta.x;
159.
160.         if(i == 0)
161.             {
162.                 maximo.x = velocidade.x;
163.                 minimo.x = velocidade.x;
164.             }
165.         else
166.             {
167.                 if(velocidade.x > maximo.x)
168.                     {
169.                         maximo.x = velocidade.x;
170.                     }
171.                 if(velocidade.x < minimo.x)
172.                     {
173.                         minimo.x = velocidade.x;
174.                     }
175.             }
176.         }
177.
178.         ganho.x += somatorio/100;
179.
180.         ganhoBruto.x += somatorioBruto/100;
181.
182.         maximo.x -= ganho.x;
183.         minimo.x -= ganho.x;
184.     break;
185.
186. case 2:
187.     for(int i = 0; i<100; i++)
188.         {
189.             leitura(eixo);
190.
191.             somatorio+= velocidade.y;

```



```

192.
193.         somatorioBruto+= velocidadeBruta.y;
194.
195.         if(i == 0)
196.         {
197.             maximo.y = velocidade.y;
198.             minimo.y = velocidade.y;
199.         }
200.         else
201.         {
202.             if(velocidade.y > maximo.y)
203.             {
204.                 maximo.y = velocidade.y;
205.             }
206.             if(velocidade.y < minimo.y)
207.             {
208.                 minimo.y = velocidade.y;
209.             }
210.         }
211.     }
212.
213.     ganho.y += somatorio/100;
214.
215.     ganhoBruto.y += somatorioBruto/100;
216.
217.     maximo.y -= ganho.y;
218.     minimo.y -= ganho.y;
219.     break;
220.
221.     case 3:
222.         for(int i = 0; i<100; i++)
223.         {
224.             leitura(eixo);
225.
226.             somatorio+= velocidade.z;
227.
228.             somatorioBruto+= velocidadeBruta.z;
229.
230.             if(i == 0)

```

```

231.     {
232.         maximo.z = velocidade.z;
233.         minimo.z = velocidade.z;
234.     }
235.     else
236.     {
237.         if(velocidade.z > maximo.z)
238.         {
239.             maximo.z = velocidade.z;
240.         }
241.         if(velocidade.z < minimo.z)
242.         {
243.             minimo.z = velocidade.z;
244.         }
245.     }
246. }
247.
248.     ganho.z += somatorio/100;
249.
250.     ganhoBruto.z += somatorioBruto/100;
251.
252.     maximo.z -= ganho.z;
253.     minimo.z -= ganho.z;
254.     break;
255. }
256. }
257.
258.
259. void leitura() //Mensura a aceleração do robô em m/s2 nos três eixos, filtrando utilizando um filtro
    de Kalman
260.     {
261.
262.     read();
263.
264.     velocidadeBruta.x = ((g.x * 8.75)/1000);
265.     velocidadeBruta.y = ((g.y * 8.75)/1000);
266.     velocidadeBruta.z = ((g.z * 8.75)/1000);
267.
268.     Vec1f entradaX(velocidadeBruta.x);

```

```

269.     Vec1f entradaY(velocidadeBruta.y);
270.     Vec1f entradaZ(velocidadeBruta.z);
271.
272.     filtroX.update(entradaX);
273.     filtroY.update(entradaY);
274.     filtroZ.update(entradaZ);
275.
276.     Vec1f saidaX = filtroX.getEstimation();
277.     Vec1f saidaY = filtroY.getEstimation();
278.     Vec1f saidaZ = filtroZ.getEstimation();
279.
280.     velocidade.x = saidaX[0] - ganho.x;
281.     velocidade.y = saidaY[0] - ganho.y;
282.     velocidade.z = saidaZ[0] - ganho.z;
283.
284.     if(velocidade.x <= maximo.x && velocidade.x >= minimo.x)
285.     {
286.         velocidade.x = 0;
287.     }
288.
289.     if(velocidade.y <= maximo.y && velocidade.y >= minimo.y)
290.     {
291.         velocidade.y = 0;
292.     }
293.
294.     if(velocidade.z <= maximo.x && velocidade.z >= minimo.y)
295.     {
296.         velocidade.z = 0;
297.     }
298.
299. }
300.
301.
302. void leitura(int eixo) //Mensura a aceleração apenas no eixo selecionado, com filtro de Kalman
303. {
304.     read();
305.
306.     Vec1f entrada,
307.         saida;

```

```

308.
309.
310.  switch(eixo)
311.    {
312.      case 1:
313.        velocidadeBruta.x = ((g.x * 8.75)/1000);
314.
315.        entrada = velocidadeBruta.x *0.8;
316.
317.        filtroX.update(entrada);
318.
319.        saida = filtroX.getEstimation();
320.
321.        velocidade.x = saida[0] - ganho.x;
322.
323.        if(velocidade.x <= maximo.x && velocidade.x > 0.0)
324.          {
325.            velocidade.x = 0;
326.          }
327.
328.        if(velocidade.x >= minimo.x && velocidade.x < 0.0)
329.          {
330.            velocidade.x = 0;
331.          }
332.      break;
333.
334.      case 2:
335.        velocidadeBruta.y = ((g.y * 8.75)/1000);
336.
337.        entrada = velocidadeBruta.y;
338.
339.        filtroY.update(entrada);
340.
341.        saida = filtroY.getEstimation();
342.
343.        velocidade.y = saida[0] - ganho.y;
344.
345.        if(velocidade.y <= maximo.y && velocidade.y > 0.0)
346.          {

```

```

347.         velocidade.y = 0;
348.     }
349.
350.     if(velocidade.y >= minimo.y && velocidade.y < 0.0)
351.     {
352.         velocidade.y = 0;
353.     }
354.     break;
355.
356.     case 3:
357.         velocidadeBruta.z = ((g.z * 8.75)/1000);
358.
359.         entrada = velocidadeBruta.z*1.15;
360.
361.         filtroZ.update(entrada);
362.
363.         saida = filtroZ.getEstimation();
364.
365.         velocidade.z = saida[0] - ganho.z;
366.
367.         if(velocidade.z <= maximo.z && velocidade.z > 0.0)
368.         {
369.             velocidade.z = 0;
370.         }
371.
372.         if(velocidade.z >= minimo.z && velocidade.z < 0.0)
373.         {
374.             velocidade.z = 0;
375.         }
376.     break;
377.
378.     default:
379.     break;
380. }
381. }
382.
383.
384. void leituraBruta() //Faz a leitura sem filtragem dos três eixos
385.     {

```

```

386.     read();
387.
388.     velocidade.x = ((g.z * 8.75)/1000) - ganhoBruto.x;
389.     velocidade.y = ((g.z * 8.75)/1000) - ganhoBruto.y;
390.     velocidade.z = ((g.z * 8.75)/1000) - ganhoBruto.z;
391. }
392.
393. void reset() //Reseta todas as estruturas de dados
394. {
395.     grau.z = 0;
396.
397.     velocidadeAnt.z = 0;
398.
399.     velocidade.z = 0;
400.
401.     tempo = 0;
402.
403.     deltaT = 0;
404. }
405.
406. void medirGrau() //Mensura a quantidade de graus percorridos
407. {
408.     if(tempo == 0)
409.     {
410.         tempo = millis();
411.     }
412.
413.     //Define todos os pontos anteriores
414.     velocidadeAnt.z = velocidade.z;
415.
416.     leitura(eZ);
417.
418.     //Descobre o tempo percorrido entre as duas leituras
419.     deltaT = (float(millis() - tempo))/1000;
420.
421.     //Integra a velocidade angular para obter os graus percorridos
422.     grau.z += ((velocidade.z + velocidadeAnt.z) * deltaT)/2;
423.
424.     tempo = millis();

```

```

425.
426.     if(grau.z > 360)
427.     {
428.         grau.z -= 360;
429.     }
430.     if(grau.z < -360)
431.     {
432.         grau.z += 360;
433.     }
434.     }
435.};
436.
437.
438.//Sensores inerciais
439.Giroscopio giroscopio;
440.
441.//Classes do robô zumo
442.ZumoMotors motores; //Motores
443.Pushbutton botao(ZUMO_BUTTON); //Botão
444.
445.//Fim da declaração de objetos-----
446.
447.//Declaração de Funções -----
448.
449.void setup()
450. {
451.     motores.flipLeftMotor(true);
452.
453.     Serial.begin(250000);
454.
455.     Wire.begin();
456.
457.     //Inicia o giroscópio
458.     giroscopio.init();
459.     giroscopio.enableDefault();
460.
461. }
462.
463.

```

```

464.void para(int tempo)
465. {
466.   if(tempo == 0)
467.     {
468.       motores.setSpeeds(0, 0);
469.       botao.waitForButton();
470.     }
471.   else
472.     {
473.       motores.setSpeeds(0, 0);
474.       delay(tempo);
475.     }
476. }
477.
478.
479.Individuo individuo[35]; //Vetor para armazenamento dos indivíduos
480.Individuo novaGeracao[35]; //Vetor para gerar a nova geração
481.
482.GeneticoReal genetico(35, 3, 50, -50, individuo, novaGeracao); //N indivíduo, gene, máximo (gene),
    mínimo (gene)
483.
484.int grau = 45;
485.
486.double pidGiroAvaliacao(double kp, double ki, double kd) //Avalia o indivíduo
487. {
488.   float erro = 0,
489.     erroAnt = 0,
490.     integral = 0;
491.
492.   int diferenca,
493.     velocidadeEsq,
494.     velocidadeDir,
495.     velocidade,
496.     resetIntegral = 0;
497.
498.   double somatorio = 0; //Aptidão
499.
500.   //Calibra o giroscópio
501.   giroscopio.reset();

```



```

502.
503. para(500);
504.
505. giroscopio.calibrar(eZ);
506.
507. unsigned long tempo = millis();
508.
509. do
510. {
511.   giroscopio.medirGrau();
512.
513.   erroAnt = erro;
514.
515.   erro = grau - giroscopio.grau.z;
516.
517.   if(resetIntegral == 200);
518.   {
519.     resetIntegral = 0;
520.     integral = 0;
521.   }
522.
523.   integral += erro;
524.
525.   resetIntegral +=1;
526.
527.   velocidade = int( (kp*erro)+(ki*integral)+(kd*(erro-erroAnt)) );
528.
529.   velocidadeEsq = -velocidade;
530.   velocidadeDir = + velocidade;
531.
532.   motores.setSpeeds(velocidadeEsq, velocidadeDir);
533.
534.   somatorio += (erro*erro) * (millis()/1000); //Somatório (erro quadro * tempo)
535. } while((millis()-tempo)<4000);
536.
537. return somatorio;
538. }
539.
540.double avaliar(double *individuo)

```

