

CADERNO DE PROJETO DO ROBÔ E DA PROGRAMAÇÃO

Equipe ERA2-D2

Divinópolis/2024

SUMÁRIO

1. APRESENTAÇÃO DA EQUIPE.....	2
2. INFORMAÇÕES DA EQUIPE.....	4
3. OBJETIVOS DA EQUIPE COM O PROJETO DO ROBÔ.....	7
4. ORGANIZAÇÃO DA EQUIPE NO PROJETO DO ROBÔ.....	8
5. ESTRATÉGIAS DE ABORDAGENS DOS DESAFIOS PRÁTICOS.....	9
5.1 ESTRATÉGIA NO CAMPO DE PROVA.....	9
5.2 DESEMPENHO DO PROJETO.....	9
6. CRONOGRAMA DE TRABALHO.....	48
7. RESULTADOS COLHIDOS.....	51
8. ANEXO.....	52
9. APÊNDICE.....	52

LISTA DE FIGURAS

Figura 01 - Equipe ERA2-D2.....	5
Figura 02 - Design de PCB no software Proteus.....	12
Figura 03 - PCB após retirada da CNC.....	12
Figura 04 - Protótipo inicial do braço robótico.....	13
Figura 05 - Versão final do braço robótico.....	13
Figura 06 - Cálculos do sensor de cor.....	14
Figura 07 - Programa para calcular melhor resistência e diferença de tensão.....	15
Figura 08 - Sensor de cor.....	15
Figura 09 - Suporte das baterias.....	16
Figura 10 - Esquema do LM7805cv.....	16
Figura 11 - Inclusão das bibliotecas.....	17
Figura 12 - Leitura do valor salvo nos slots 5 e 6 da EEPROM.....	18
Figura 13 - Declaração dos Servomotores.....	18
Figura 14 - Movimentação dos Servomotores.....	18
Figura 15 - Declaração dos sensores ultrassônicos.....	19
Figura 16 - Medição dos sensores ultrassônicos.....	19
Figura 17 - Constantes e Variáveis.....	20
Figura 18 - Constantes e Variáveis.....	20
Figura 19 - Utilização do #define.....	21
Figura 20 - Função para parar o robô.....	21
Figura 21 - Função para o robô mover-se para frente.....	21
Figura 22 - Função para o robô mover-se para trás.....	22
Figura 23 - Função para o robô girar no próprio eixo para a direita.....	22
Figura 24 - Função para o robô girar no próprio eixo para a esquerda.....	22
Figura 25 - Função para leitura do sensor de cor.....	23
Figura 26 - Função para auxiliar na comparação da cor lida pelo sensor.....	23
Figura 27 - Função para colocar o braço robótico na posição inicial.....	24
Figura 28 - Função para pegar a árvore grande.....	24
Figura 29 - Função para pegar a árvore pequena.....	25
Figura 30 - Função para leitura do sensor e análise da árvore identificada.....	25
Figura 31 - Função para subir a rampa.....	25
Figura 32 - Função para aumentar a velocidade das rodas.....	26
Figura 33 - Função para o braço robótico ajustar-se para leitura de cor principal.....	26
Figura 35 - Recorte da parte inicial do loop.....	28
Figura 36 - Sensor acelerômetro e giroscópio MPU6050.....	29
Figura 37 - Motor N20 com encoder.....	30
Figura 37 - Motores instalados na PCB.....	30
Figura 38 - Sensor de distância a laser VL53L0XX.....	31
Figura 39 - Sensor de distância a laser instalado na garra.....	31
Figura 40 - Organizador de fios feito de garrafa PET.....	32
Figura 41 - Arduino MEGA 2560 modelado em 3D no software Fusion 360.....	32

Figura 42 - Servomotor S9g modelado em 3D no software Fusion 360.....	33
Figura 43 - Robô completo modelado em 3D no software Fusion 360.....	33
Figura 44 - Inclusão das bibliotecas.....	34
Figura 45 - Double no código.....	35
Figura 46 - Sintaxe, INT 4 e 5, função chamada, borda de mudança (CHANGE).....	37
Figura 47 - Funções chamadas pelas interrupções, somam um pulso.....	37
Figura 48 - Função responsável por calcular o PID e ajustar os motores.....	38
Figura 49 - Discretização de uma equação PID.....	41
Figura 50 - Função para ajustar a altura de leitura das árvores.....	43
Figura 51 - Função para utilização do sensor laser.....	43
Figura 52 - Função para teste de balanceamento dos motores.....	43
Figura 53 - Função para abaixar a rampa.....	44
Figura 54 - Função de controle dos motores em linha reta.....	44
Figura 55 - Função de controle dos motores em curvas no próprio eixo.....	45
Figura 56 - Função de controle da garra.....	47
Figura 57 - Parte do Setup referente ao encoder.....	48
Figura 58 - Inicialização do Laser no Setup.....	48
Figura 59 - Parte inicial do novo loop.....	49

LISTA DE TABELAS

Tabela 01 - Informações da equipe ERA2-D2.....	7
Tabela 02 - Patrocinadores da equipe ERA2-D2.....	8
Tabela 03 - Integrantes da equipe ERA2-D2.....	9
Tabela 04 - Redes Sociais da equipe ERA2-D2.....	10
Tabela 05 - Organização da Equipe ERA2-D2 no projeto do robô.....	11
Tabela 06 -Tamanhos e tipos de variáveis.....	21
Tabela 07 - Tamanhos e tipos de variáveis double.....	36
Tabela 08 - Interrupções associadas a pinos em diferentes versões de Arduino.....	37
Tabela 09 - Cronograma FEVEREIRO à MAIO de 2024.....	51
Tabela 10 - Cronograma JUNHO à SETEMBRO de 2024.....	52
Tabela 11 - Cronograma SETEMBRO à DEZEMBRO de 2024.....	53

1. APRESENTAÇÃO DA EQUIPE

O ERA2-D2 é uma equipe de robótica do ensino médio do CEFET-MG de Divinópolis, Minas Gerais. A equipe foi criada no início do ano de 2023 pelo grupo Estudos de Robótica e Automação (ERA), visando participar da categoria High do Torneio Brasil de Robótica (TBR). Após a formação do grupo, os membros se reuniram para definir o nome, e assim surgiu o "ERA2-D2". A escolha foi inspirada pelo fato de fazerem parte do ERA e pela paixão da maioria da equipe por Star Wars, sendo uma homenagem a um dos personagens mais queridos da saga, o R2-D2.

No ano passado, o grupo participou do torneio regional Belo Horizonte/MG, mesmo não tendo o resultado esperado, a experiência de competir pela primeira vez foi muito importante para os membros. Ao longo deste ano, a equipe passou por algumas mudanças, como a entrada de novos integrantes e a troca de uniformes.

Na etapa regional de Belo Horizonte em 2024, o ERA2-D2 teve um desempenho incrível, conquistando o primeiro lugar e garantindo a classificação para a etapa nacional do TBR. Agora, a equipe segue motivada e trabalha com dedicação para representar o CEFET-MG e Divinópolis na competição nacional.

Figura 01 - Equipe ERA2-D2.



Fonte: Equipe ERA2-D2.

2. INFORMAÇÕES DA EQUIPE

Tabela 01 - Informações da equipe ERA2-D2.

NOME DA EQUIPE	ERA2-D2
INSTITUIÇÃO DE ORIGEM	CEFET-MG
ENDEREÇO DA INSTITUIÇÃO	R. ÁLVARES DE AZEVEDO, 400 - BELA VISTA, DIVINÓPOLIS - MG, 35503-822
MUNICÍPIO E ESTADO	DIVINÓPOLIS - MINAS GERAIS
CATEGORIA TBR	HIGH
TÉCNICO	JOÃO LUIZ DE SOUSA VIEIRA
IDADE TÉCNICO	25 ANOS
FORMAÇÃO TÉCNICO	TÉCNICO EM MECATRÔNICA
MENTOR	DIÊGO FERNANDES DA CRUZ
IDADE MENTOR	37 ANOS
FORMAÇÃO MENTOR	MESTRE EM ENGENHARIA MECÂNICA
COLABORADORES	CEFET-MG, SEBRAE-MG E ACID DIVINÓPOLIS

Fonte: Equipe ERA2-D2.

Tabela 02 - Patrocinadores da equipe ERA2-D2.

PATROCINADORES
AUTO ELÉTRICA DO VAGUINHO.
AUTO ELÉTRICA JB.
DISTRIBUIDORA GONTIJO.
FÁTIMA DESIGNS.
GONÇALVES E NIZA ADVOGADOS.
GRÁFICA GL.
GRUPO PLANEJAR.
HARPIA HARPYJA.
MAC SUPERMERCADOS.
SICOOB DIVICRED.
SUPERMERCADO KIT 10.
SUPERMERCADO OLÍMPIO ROCHA.
USELIGAS.

Fonte: Equipe ERA2-D2.

Tabela 03 - Integrantes da equipe ERA2-D2.

INTEGRANTES DA EQUIPE		
Nomes	Idade	Série
Ana Clara Monteiro Caetano	17	2º Ensino Médio
Ana Clara Siqueira	17	2º Ensino Médio
Davi Faria de Sousa Guimarães	18	2º Ensino Médio
Gustavo Campos da Silva Silveira	17	2º Ensino Médio
Isabella Olímpio Rocha	15	1º Ensino Médio
João Carlos Santos	16	2º Ensino Médio
Lucas Gabriel Soares Borges	17	2º Ensino Médio
Nina Lage Motta	17	3º Ensino Médio
Pedro Henrique Carlos de Souza	17	2º Ensino Médio
Suellen de Faria Silvério	15	1º Ensino Médio

Fonte: Equipe ERA2-D2.

Tabela 04 - Redes Sociais da equipe ERA2-D2.

Redes Sociais da Equipe ERA2-D2			
 TIKTOK (@ERA2D2)	 INSTAGRAM (@ERA2D2)	 BLUESKY (@ERA2D2)	 YOUTUBE (@ERA2D2)
 @ERA2D2	 @ERA2D2	 @ERA2D2	 @ERA2D2

Fonte: Equipe ERA2-D2.

3. OBJETIVOS DA EQUIPE COM O PROJETO DO ROBÔ

Para nossa equipe, o TBR é uma plataforma fundamental para o desenvolvimento de habilidades técnicas e práticas. Participar do torneio nos proporciona a oportunidade de aplicar conhecimentos teóricos em projetos reais e aprimorar nossa capacidade técnica e de trabalho em equipe.

Nosso principal objetivo é o avanço técnico no desenvolvimento do robô. Queremos aprofundar nosso conhecimento em robótica, programação e eletrônica, aplicando esses conhecimentos diretamente na construção e programação do robô. A competição nos oferece a chance de testar e ajustar nossas soluções para resolver problemas reais, como os desafios propostos no torneio.

Além disso, buscamos obter uma experiência prática valiosa. O TBR nos permite trabalhar em um projeto de robótica do início ao fim, o que nos ajuda a entender melhor o ciclo completo de desenvolvimento e a coordenação de tarefas, resolução de problemas e comunicação efetiva.

Ao final da temporada, pretendemos alcançar uma boa colocação na etapa nacional do TBR, além de finalizar nossa plataforma de monitoramento aquático com sucesso, apresentando resultados relevantes para a gestão de recursos hídricos.

4. ORGANIZAÇÃO DA EQUIPE NO PROJETO DO ROBÔ

A equipe é composta por dez membros, e embora todos participem das diversas fases do projeto do robô, cada um lidera uma área específica: Davi, Lucas, Pedro e Suellen. A seguir, descrevemos as funções principais e responsabilidades de cada um:

Davi lidera a programação do robô. Ele é responsável por escrever e testar o código que controla as operações do robô, garantindo que todas as funcionalidades estejam implementadas e funcionando conforme o planejado. Embora todos participem no desenvolvimento e ajustes do *software*, Davi coordena essa parte crucial do projeto.

Lucas é o líder no planejamento do robô. Ele elabora os projetos e especificações técnicas, definindo as características, funcionalidades e requisitos do robô. Além disso, Lucas coordena as atividades e prazos, garantindo que todos os membros estejam alinhados com o cronograma e os objetivos estabelecidos. Todos colaboram no planejamento, mas ele supervisiona esta área.

Pedro é o responsável pela montagem do robô. Ele lida com a construção física e a integração dos componentes, desde a estrutura até os sistemas eletrônicos e mecânicos. Pedro assegura que todos os componentes estejam montados corretamente e que o robô seja funcional e seguro para operar. Apesar de todos ajudarem na montagem, Pedro lidera essa etapa.

Suellen coordena a documentação e os relatórios do projeto. Ela registra o progresso, cria relatórios detalhados e mantém toda a documentação atualizada, incluindo manuais de operação, guias de montagem e registros de testes. Suellen garante que a documentação seja completa e acessível, com todos participando da coleta de informações e relatórios, mas ela lidera essa função.

Cada membro contribui significativamente para o sucesso do projeto, cobrindo todas as etapas de desenvolvimento – projetar, construir, programar e testar – com suas respectivas lideranças. A comunicação e a colaboração entre os membros são essenciais para manter todos informados sobre o progresso e os desafios, facilitando a resolução de problemas e a execução eficiente das tarefas.

Essa estrutura organiza a equipe de maneira a aproveitar ao máximo as habilidades individuais, promovendo um ambiente de trabalho produtivo e harmonioso, essencial para o sucesso no TBR.

Tabela 05 - Organização da Equipe ERA2-D2 no projeto do robô.

MEMBRO	FUNÇÃO
Davi Faria de Sousa Guimarães	Programador
Lucas Gabriel Soares Borges	Planejador
Pedro Henrique Carlos de Souza	Construtor
Suellen de Faria Silvério	Relatora

Fonte: Equipe ERA2-D2.

5. ESTRATÉGIAS DE ABORDAGENS DOS DESAFIOS PRÁTICOS

5.1 ESTRATÉGIA NO CAMPO DE PROVA

Nossa estratégia envolve a realização das missões principais, priorizando a missão 2, para ter uma maior pontuação e ser mais eficiente. O robô sairá da base e pegará a primeira árvore disponível, que pode ser grande ou pequena, e a colocará na área de reflorestamento correspondente. Caso seja uma árvore grande, ela será plantada na área vermelha. Em seguida, o robô vai pegar a segunda árvore que será posicionada no local correto. Após finalizar, o robô subirá a primeira rampa para acessar o sistema de coleta e tratamento de água.

Ao chegar, o robô identificará a cor indicada na placa e, com base nessa leitura, derrubará os poluentes de mesma cor, garantindo que não sejam derrubados poluentes de cores diferentes, evitando assim penalizações. Concluída a neutralização, o robô subirá a segunda rampa para retornar à missão de reflorestamento e finalizar o plantio das árvores restantes.

Na retomada da missão 1, o robô seguirá para a próxima área de reflorestamento, onde pegará a primeira árvore disponível e a colocará no local correto. Se for uma árvore grande, ela será plantada na área amarela. Em seguida, posicionará a segunda árvore. Após concluir essa etapa, o robô se deslocará para a última área de reflorestamento, no lado oposto, e repetirá o processo, garantindo que a árvore grande seja plantada na área verde e a pequena na área branca. Totalizando 500 pontos. Essa abordagem foi planejada pela equipe visando a maior eficiência.

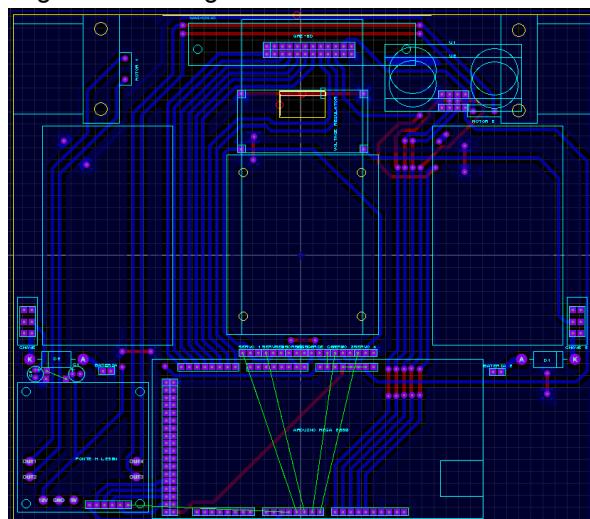
5.2 DESEMPENHO DO PROJETO

Durante esta temporada do TBR, a equipe enfrentou desafios que exigiram habilidades técnicas e estratégicas para utilizar os recursos disponíveis de melhor forma e garantir o melhor desempenho do robô. Com base no entendimento das missões no Campo de Provas e nas limitações técnicas, decisões fundamentais foram tomadas para orientar o projeto, a construção e a programação do robô.

O projeto foi desenvolvido em etapas, com uma abordagem inovadora que integrou o próprio chassi à parte elétrica do robô. A fabricação da PCB (Placa de Circuito Impresso) foi realizada no software Proteus, com componentes modelados manualmente, resultando em um mês de trabalho dedicado. Uma nova metodologia

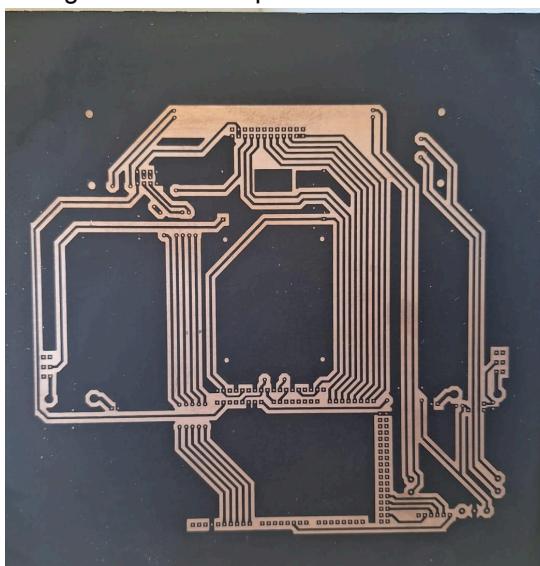
de produção foi adotada, utilizando tinta *spray* e uma CNC a laser para remoção precisa da tinta, substituindo o método tradicional de prensa térmica. Após a corrosão do cobre com Percloreto de Ferro (FeCl_3), as placas foram cortadas, lixadas e perfuradas para acomodar os componentes eletrônicos. Esse processo garantiu uma PCB funcional e sem erros.

Figura 02 - Design de PCB no software Proteus.



Fonte: Equipe ERA2-D2.

Figura 03 - PCB após retirada da CNC.

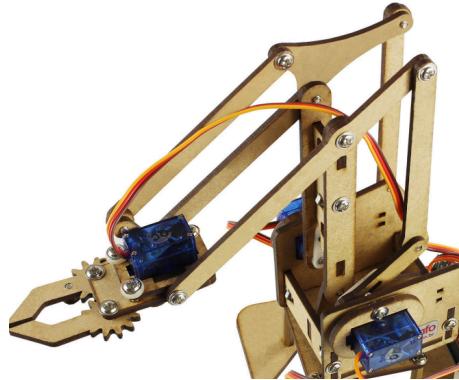


Fonte: Equipe ERA2-D2.

Um dos principais desafios técnicos foi o sistema de seguimento de linha, essencial para a execução autônoma das missões. Inicialmente, foram desenvolvidos sete códigos diferentes, incluindo um de calibração automática. No entanto, devido à baixa precisão e velocidade da estratégia inicial, optou-se por basear as missões em tempos pré-definidos, o que minimizou erros e aumentou o desempenho geral.

Na construção, a equipe decidiu utilizar exclusivamente impressão 3D para os suportes dos componentes, priorizando precisão e versatilidade. O braço robótico, inspirado em um modelo do site Thingiverse, foi completamente redesenrado para solucionar problemas de resistência mecânica e mobilidade, resultando em uma redução de peso de 18,75% e maior robustez estrutural. Servomotores SG90 foram instalados para garantir movimentos precisos em várias direções, operando com tensão ideal de 6V para oferecer força e segurança.

Figura 04 - Protótipo inicial do braço robótico.



Fonte: Matheus Gebert Straub, 2016.

Figura 05 - Versão final do braço robótico.



Fonte: Equipe ERA2-D2.

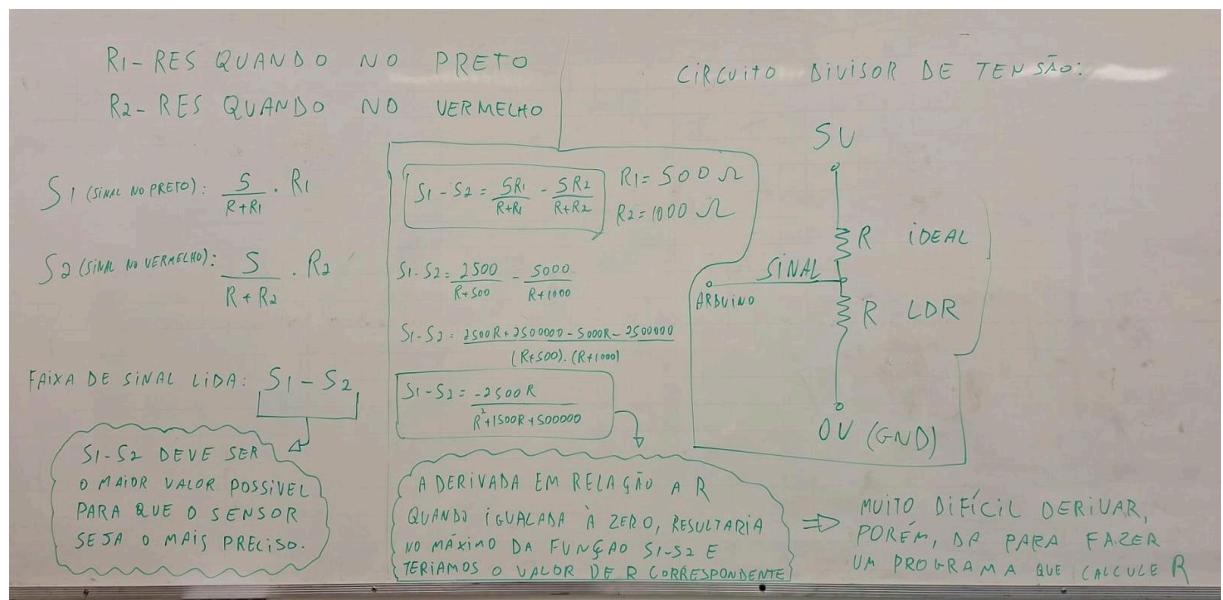
Outro destaque do projeto foi o sensor de Cor que funciona em 5v e identifica objetos vermelhos e pretos, sendo útil para diversas missões que envolvem diferenciamento de cor, podendo ser reprogramado para leitura de diversas cores.

Para fazer o sensor de cor, um LDR (*Light Dependent Resistor*, em português, Resistor Dependente de Luz) foi utilizado. Esse componente varia sua resistência de maneira inversamente proporcional à luminosidade, ou seja, mais claro menor resistência e, mais escuro, maior resistência. A partir da Lei de Ohm e da Lei de Kirchhoff das Tensões, é possível fazer um divisor de tensão para captar

as diferenças de potencial sobre o LDR e transformá-las em uma leitura analógica pelo arduino.

Com pouca luminosidade, ou seja, quando lido sobre uma superfície preta, o LDR apresentou uma resistência de 741Ω , enquanto que na cor vermelha, a resistência apresentada foi de 151Ω . Uma vez calculados S1 e S2, que são, respectivamente, os valores de tensão calculados nas superfícies preta e vermelha, pode-se calcular o maior valor de S1 - S2 a fim de obter a resistência ideal que gera tal diferença. A equação foi desenvolvida, obtendo o seguinte resultado:

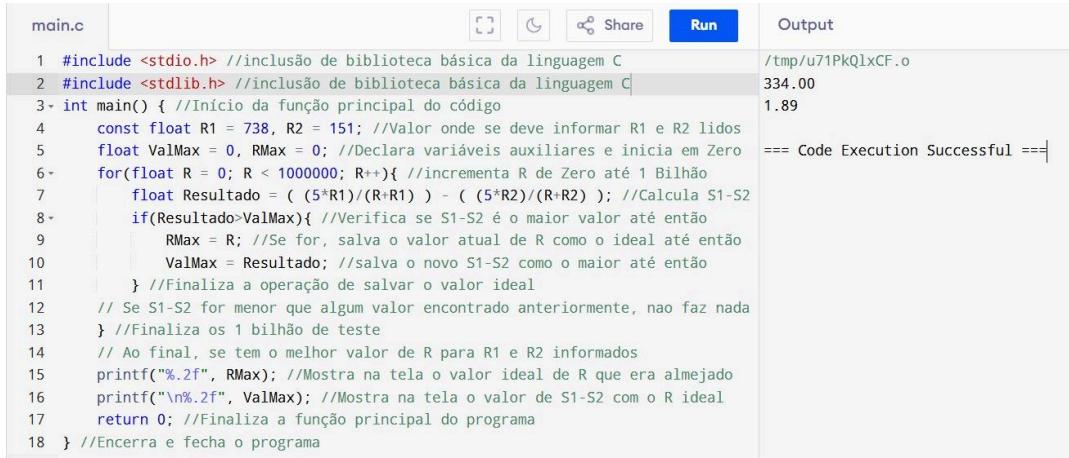
Figura 06 - Cálculos do sensor de cor.



Fonte: Equipe ERA2-D2.

Uma vez encontrada a equação em função de R, bastaria derivar em relação a R e igualar o resultado à Zero que se obteria o valor de R ideal, capaz de gerar a maior diferença possível entre S1 e S2. Como não se possui conhecimento suficiente em Cálculo, optou-se por fazer um programa utilizando a linguagem C para calcular, através do método de força bruta, o valor de resistência ideal. O programa informou, para os valores de R1 e R2 inicialmente mensurados, o valor de R ideal. Utilizou-se, então, o resistor de 330Ω , por ser o valor comercial mais próximo.

Figura 07 - Programa para calcular melhor resistência e diferença de tensão.



```

main.c
1 #include <stdio.h> //inclusão de biblioteca básica da linguagem C
2 #include <stdlib.h> //inclusão de biblioteca básica da linguagem C
3 int main() { //Início da função principal do código
4     const float R1 = 738, R2 = 151; //Valor onde se deve informar R1 e R2 lidos
5     float ValMax = 0, RMax = 0; //Declara variáveis auxiliares e inicia em Zero
6     for(float R = 0; R < 1000000; R++){ //Incrementa R de Zero até 1 Bilhão
7         float Resultado = ( (5*R1)/(R+R1) ) - ( (5*R2)/(R+R2) ); //Calcula S1-S2
8         if(Resultado>ValMax){ //Verifica se S1-S2 é o maior valor até então
9             RMax = R; //Se for, salva o valor atual de R como o ideal até então
10            ValMax = Resultado; //salva o novo S1-S2 como o maior até então
11        } //Finaliza a operação de salvar o valor ideal
12        // Se S1-S2 for menor que algum valor encontrado anteriormente, não faz nada
13    } //Finaliza os 1 bilhão de teste
14    // Ao final, se tem o melhor valor de R para R1 e R2 informados
15    printf("%.2f", RMax); //Mostra na tela o valor ideal de R que era almejado
16    printf("\n%.2f", ValMax); //Mostra na tela o valor de S1-S2 com o R ideal
17    return 0; //Finaliza a função principal do programa
18 } //Encerra e fecha o programa

```

Fonte: Equipe ERA2-D2.

Uma proteção 3D foi produzida na cor preta, que absorve mais luz, para o LDR. Isso diminui a interferência de luminosidade externa. Em conjunto com a proteção, os LEDs (*Light Emitting Diode*, em português, Diodo Emissor de Luz) de alto brilho foram colocados ao lado, o que também ajuda a diminuir interferências, realizando as leituras com mais precisão. Construiu-se uma PCB com os pedaços restantes da original, e todos os componentes foram soldados na placa.

LEDs de alto brilho operam com 3.6v entre 10 e 40 miliamperes (mA), precisando de resistores para não queimar. Utilizando novamente a Lei de Ohm, calculou-se que a resistência deveria ser de 39Ω , assim gerando uma corrente de 35mA, forte o bastante para gerar alto brilho, mas segura para não queimar o componente.

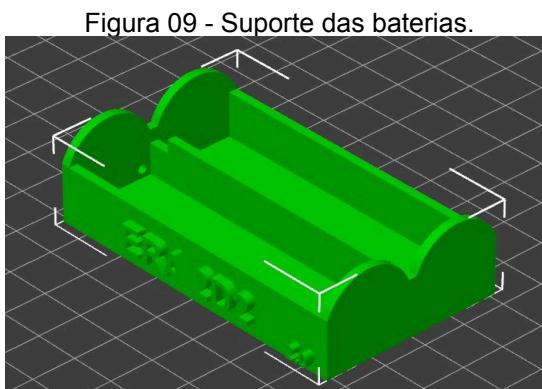
Figura 08 - Sensor de cor.



Fonte: Equipe ERA2-D2.

O sistema de alimentação contém quatro baterias de lítio, duas em série localizadas à esquerda, e duas em série à direita do robô. Baterias em série somam a tensão, totalizando 8.4 volts (4.2V + 4.2V). Vale ressaltar que os GNDs (terra)

estão em comum, ou seja, todos conectados. O suporte para as baterias foi modelado em 3D e molas para a fixação foram presas no suporte.



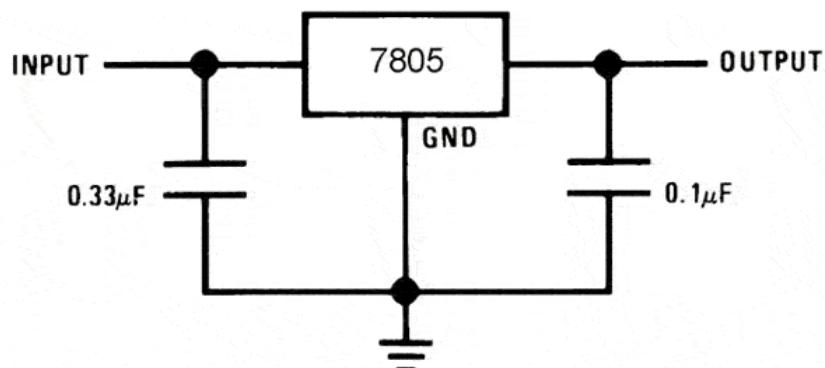
Fonte: Equipe ERA2-D2.

O uso de dois *packs* de 8.4V se deu devido ao alto consumo dos servomotores, que variam de 100mA a 750mA. O *pack* esquerdo alimenta todos os componentes, enquanto o direito é usado apenas para os servos.

Com exceção do Arduino Mega 2560, os outros componentes são alimentados com 5V, para que não possuam variações de tensão, principalmente os motores que necessitam da mesma tensão para que funcionem corretamente. Para isso, a tensão total passa pelo regulador de tensão LM7805cv. Esse componente serve para diminuir e manter a tensão em uma faixa específica, nesse caso 5V, com pequenas variações entre 4.98 e 5.02 volts ($\pm 0.4\%$), mas mantendo-se estável na maior parte do tempo em 5V.

O LM7805cv funciona com uma tensão de no mínimo 2V a mais do que a sua tensão de saída, ou seja, as baterias podem chegar a sua carga mínima de 7.4V, a tensão de saída continua a mesma, sofrendo apenas pequenas variações. Com o auxílio de 2 capacitores eletrolíticos, um na entrada e um na saída do regulador, é obtida uma regulagem altamente precisa da tensão. A tensão das baterias da direita passam pelo regulador de tensão HW-286, que foi configurado para reduzi-la a 6V, alimentando os três (3) servomotores.

Figura 10 - Esquema do LM7805cv.



Fonte: Muhanned Al-Rawi, 2021.

Na montagem final, diversos componentes foram integrados para garantir um desempenho otimizado. O controle das operações do robô ficou a cargo de um microcontrolador Arduino Mega 2560, enquanto dois motores CC N20, com caixas de redução 100/1 e pneus de silicone de 22mm, garantiram a movimentação. Uma esfera metálica de 18mm foi utilizada para apoio e equilíbrio. Dois sensores ultrassônicos HC-SR04 foram responsáveis pela detecção de obstáculos, e a ponte H L298N controlou os motores. Diodos SK3G 0285 evitaram danos elétricos, enquanto dissipadores de calor e capacitores eletrolíticos asseguraram a estabilidade térmica e elétrica do sistema.

Ferramentas auxiliares, como multímetro, osciloscópio e computadores, foram essenciais durante o processo de montagem e programação. Uma máquina CNC Delta L9060 foi utilizada para fabricar os PCBs, e peças específicas foram impressas na CREALITY K1. Após a soldagem e fixação dos componentes, verificou-se a ausência de curtos nas trilhas de cobre, resultando em um robô funcional e confiável, pronto para enfrentar os desafios do torneio.

O desenvolvimento da programação é uma parte crítica do processo, pois garante que o robô seja capaz de realizar todas as tarefas de forma autônoma. Para assegurar a eficiência do código, foram realizados testes em ambientes controlados, com o objetivo de avaliar e aprimorar as capacidades do robô.

O código é estruturado em várias seções, sendo a primeira de inclusão das bibliotecas usadas, a segunda de criação de constantes e variáveis, a terceira de funções a serem realizadas ao longo do programa, a quarta de declaração de pinos (*void setup*), e a última o programa a ser executado, o *loop*.

Bibliotecas:

Figura 11 - Inclusão das bibliotecas.

```
// Bibliotecas
#include <EEPROM.h>
#include <HCSR04.h>
#include <Servo.h>
```

Fonte: Equipe ERA2-D2.

EEPROM:

EQUIPE ERA2-D2

A memória EEPROM (*Electrically Erasable Programmable Read-Only Memory*, em português, Memória Somente de Leitura Programável e Apagável Eletricamente) é utilizada para a calibração do sensor de cor, guardando de modo não volátil os valores de leitura feitos em objetos vermelhos e pretos. Sua programação usa uma biblioteca chamada “EEPROM.h”. Ela é chamada na função “SensorCor()” para passar os dados de calibração. Em específico, essa memória no Arduino Mega 2560 armazena apenas 8 bits (um byte, ou 256 valores de 0 a 255), porém o sensor é calibrado com 10 bits (valores entre 0 e 1023), por isso é necessária a divisão por 4 ($1024/4 = 256$) no momento de calibração, e no momento de utilização é multiplicado por 4.

Figura 12 - Leitura do valor salvo nos *slots* 5 e 6 da EEPROM.

```
// Mínimos e Máximos de leitura da cor vermelha.
int CORVERMELHAMin = 4 * EEPROM.read(5);
int CORVERMELHAMax = 4 * EEPROM.read(6);
```

Fonte: Equipe ERA2-D2.

Servo:

Os servomotores são utilizados para a movimentação da garra. Sua programação usa uma biblioteca chamada “Servo.h”. O código consiste em nomear os servomotores, informar as portas PWM que estão conectados, e escrever o ângulo desejado de 0 a 180.

Figura 13 - Declaração dos Servomotores.

```
// Garra.
Servo servol; // 60 a 140 graus -graus, mais pra tras.
Servo servo2; // 50 a 140 graus -graus, mais alto.
Servo servo3; // 180 a 90 graus -graus, garra fechada.

int i = 0;
```

Fonte: Equipe ERA2-D2.

Figura 14 - Movimentação dos Servomotores.

```
for (i = 70; i <= 130; i++) { // Servo2 vai lentamente de 70 a 130.
    servo2.write(i);
    delay(12);
}
```

Fonte: Equipe ERA2-D2.

HCRS04:

Existem diversos modos de realizar medições com os sensores ultrassônicos, sendo com bibliotecas ou com fórmulas e leituras de portas digitais. Foram feitos testes e foi escolhida a biblioteca “HCRS04.h”, por ser a mais precisa para medir distâncias.

Figura 15 - Declaração dos sensores ultrassônicos.

```
// Ultrassônico.  
#define trig1 11  
#define trig2 10  
#define echol 8  
#define echo2 9  
  
UltraSonicDistanceSensor usl(trig1, echol); // (Trig, Echo)  
UltraSonicDistanceSensor us2(trig2, echo2);
```

Fonte: Equipe ERA2-D2.

O sensor ultrassônico principal mede a distância e, com uso do bool, retorna se a árvore grande é falsa ou verdadeira.

Figura 16 - Medição dos sensores ultrassônicos.

```
// Ultrassônico.  
bool teste;  
bool Ultrassonico() {  
    //distancial = usl.measureDistanceCm();  
    distancia2 = us2.measureDistanceCm();  
  
    if (distancia2 < 10.0) {  
        arvoreG = true;  
    } else {  
        arvoreG = false;  
    }  
    return arvoreG;  
}
```

Fonte: Equipe ERA2-D2.

Constantes e Variáveis:

As constantes e variáveis são utilizadas para a execução da maior parte do código, guardando valores de calibração de sensores, dados necessários para cálculos, constantes importantes para o funcionamento do robô, armazenar valores em geral. O código possui 4 tipos de variáveis, as do tipo *bool*, *byte*, *int*, *float* e *long*.

- O tipo *bool* (booleano) é binário (1 bit), armazena apenas 0 e 1.
- O tipo *byte* armazena 8 bits, 256 valores, de 0 a 255.
- O tipo *int* armazena 16 bits, de -32.768 a +32.767, ou com *unsigned*, que significa “não sinalizado”, fazendo com que vá de 0 a 65.535.

- O tipo *long* armazena 32 bits, 4.294.967.296 valores, de -2.147.483.648 a -2.147.483.647, ou com *unsigned*, 0 a 4.294.967.295.
- O tipo *float* armazena valores decimais de -3.4028235E+38 ou +3.4028235E+38 (E+38 significa 10^{38}), com precisão de 6-7 casas após a vírgula.

Tabela 06 -Tamanhos e tipos de variáveis.

	<i>bool</i>	<i>byte</i>	<i>int</i>	<i>long</i>	<i>float</i>
Bits	1	8	16	32	32
Tamanho	2	256	65.536	4.294.967.296	3.4028235E+38
Valores	0 e 1	0 a 255	-32.768 a +32.767	-2.147.483.648 a +2.147.483.647	-3.4028235E+38 ou +3.4028235E+38
Valores <i>Unsigned</i>			0 a 65.535	0 a 4.294.967.295	

Fonte: Equipe ERA2-D2.

Figura 17 - Constantes e Variáveis.

```

bool CORPRETA;
bool leituraCorPreta;
byte contagemDerrubados;
unsigned long distancial;
unsigned long distancia2;
int i = 0;
bool arvoreP = false;
bool arvoreG = false;

```

Fonte: Equipe ERA2-D2.

Figura 18 - Constantes e Variáveis.

```

float Distancia(double distancia) {
    distancia = distancia * 1.0238 + 0.486;
    float tempo = distancia / tempoReto;
    return tempo;
}

```

Fonte: Equipe ERA2-D2.

O “#define” é uma diretiva de pré-processador, ou seja, ela serve para definir substituições de texto feitas antes de enviar para o processador, não ocupando

memórias indesejadas, por exemplo, todos locais escritos “mEa” (figura 19) serão substituídos por “29”, antes do envio para o microprocessador.

Figura 19 - Utilização do `#define`.

```
#define mEa 29
#define mEb 27
#define mDa 25
#define mDb 23
#define mEv 2
#define mDv 3
```

Fonte: Equipe ERA2-D2.

Funções:

As funções são comandos pré definidos para efetuar uma determinada ação, podendo conter poucos ou diversos passos, operações matemáticas, entre outros. Ao todo o código possui quatorze (14) funções que serão listadas na respectiva ordem do código.

Figura 20 - Função para parar o robô.

```
void Para(int para) {
    digitalWrite(mEa, LOW);
    digitalWrite(mEb, LOW);
    analogWrite(mEv, 0);
    digitalWrite(mDa, LOW);
    digitalWrite(mDb, LOW);
    analogWrite(mDv, 0);
    delay(para);
}
```

Fonte: Equipe ERA2-D2.

Figura 21 - Função para o robô mover-se para frente.

```
void Frente(int tempo) {
    digitalWrite(mEa, HIGH);
    digitalWrite(mEb, LOW);
    analogWrite(mEv, velE);
    digitalWrite(mDa, HIGH);
    digitalWrite(mDb, LOW);
    analogWrite(mDv, velD);
    delay(tempo);
    Para(100);
}
```

Fonte: Equipe ERA2-D2.

Figura 22 - Função para o robô mover-se para trás.

```
void Tras(int tempo) {  
  
    digitalWrite(mEa, LOW);  
    digitalWrite(mEb, HIGH);  
    analogWrite(mEv, velE);  
    digitalWrite(mDa, LOW);  
    digitalWrite(mDb, HIGH);  
    analogWrite(mDv, velD);  
    delay(tempo);  
    Para(100);  
}
```

Fonte: Equipe ERA2-D2.

Figura 23 - Função para o robô girar no próprio eixo para a direita.

```
void DireitaEixo(int angulo) {  
  
    int t;  
    t = angulo * tempoDir;  
    digitalWrite(mEa, HIGH);  
    digitalWrite(mEb, LOW);  
    analogWrite(mEv, 255);  
    digitalWrite(mDa, LOW);  
    digitalWrite(mDb, HIGH);  
    analogWrite(mDv, 255);  
    delay(t);  
    Para(100);  
}
```

Fonte: Equipe ERA2-D2.

Figura 24 - Função para o robô girar no próprio eixo para a esquerda.

```
void EsquerdaEixo(int angulo) {  
  
    int t;  
    t = angulo * tempoEsq;  
    digitalWrite(mEa, LOW);  
    digitalWrite(mEb, HIGH);  
    analogWrite(mEv, 255);  
    digitalWrite(mDa, HIGH);  
    digitalWrite(mDb, LOW);  
    analogWrite(mDv, 255);  
    delay(t);  
    Para(100);  
}
```

Fonte: Equipe ERA2-D2.

Figura 25 - Função para leitura do sensor de cor.

```
void SensorCor() {
    if (analogRead(sc) > CORVERMELHAMax) {
        CORPRETA = 1;
    } else {
        CORPRETA = 0;
    }
}
```

Fonte: Equipe ERA2-D2.

Figura 26 - Função para auxiliar na comparação da cor lida pelo sensor.

```
void SensorCorAux() {
    if (analogRead(sc) > CORVERMELHAMax) {
        leituraCorPreta = 1;
    } else {
        leituraCorPreta = 0;
    }
}
```

Fonte: Equipe ERA2-D2.

Figura 27 - Função para colocar o braço robótico na posição inicial.

```
void Inicial() {
    int v1 = servol.read();
    int v2 = servo2.read();

    // Ajusta o servol para 145 graus.
    if (v1 < 145) {
        for (int i = v1; i <= 145; i++) {
            servol.write(i);
            delay(12);
        }
    } else if (v1 > 145) {
        for (int i = v1; i >= 145; i--) {
            servol.write(i);
            delay(12);
        }
    }

    // Ajusta o servo2 para 70 graus.
    if (v2 < 70) {
        for (int i = v2; i <= 70; i++) {
            servo2.write(i);
            delay(12);
        }
    } else if (v2 > 70) {
        for (int i = v2; i >= 70; i--) {
            servo2.write(i);
            delay(12);
        }
    }

    servo3.write(180);
}
```

Fonte: Equipe ERA2-D2

Figura 28 - Função para pegar a árvore grande.

```
void ArvoreGrande() {
    for (i = 145; i >= 35; i--) {
        servol.write(i);
        delay(12);
    }

    for (i = 70; i <= 130; i++) {
        servo2.write(i);
        delay(12);
    }
}
```

Fonte: Equipe ERA2-D2.

Figura 29 - Função para pegar a árvore pequena.

```
void ArvorePequena() {
    for (i = 145; i >= 30; i--) {
        servol.write(i);
        delay(12);
    }

    for (i = 70; i <= 130; i++) {
        servo2.write(i);
        delay(12);
    }
}
```

Fonte: Equipe ERA2-D2.

Figura 30 - Função para leitura do sensor e análise da árvore identificada.

```
bool Ultrassonico() {
    //distancial = us1.measureDistanceCm();
    distancia2 = us2.measureDistanceCm();

    if (distancia2 < 10.0) {
        arvoreG = true;
    } else {
        arvoreG = false;
    }
    return arvoreG;
}
```

Fonte: Equipe ERA2-D2.

Figura 31 - Função para subir a rampa.

```
void Rampa(int tempo) {

    for (i = 145; i >= 48; i--) {
        servol.write(i);
        delay(12);
    }

    for (i = 70; i >= 65; i--) {
        servo2.write(i);
        delay(12);
    }

    servo3.write(85);

    Turbo(tempo);

    tempo = 2000;

    for (i = 48; i >= 20; i--) {
        servol.write(i);
        delay(12);

        Turbo(tempo);
        break;
    }
}
```

Fonte: Equipe ERA2-D2.

Figura 32 - Função para aumentar a velocidade das rodas.

```
void Turbo(int tempo){  
  
    digitalWrite(4, LOW);  
    digitalWrite(mEa, HIGH);  
    digitalWrite(mEb, LOW);  
    analogWrite(mEv, velE);  
    digitalWrite(mDa, HIGH);  
    digitalWrite(mDb, LOW);  
    analogWrite(mDv, velD);  
    delay(tempo);  
    digitalWrite(4, HIGH);
```

Figura 33 - Função para o braço robótico ajustar-se para leitura de cor principal.

```
void Cor() {  
    for (i = 45; i <= 60; i++) {  
        servol.write(i);  
        delay(12);  
    }  
    servo2.write(65);  
}
```

Fonte: Equipe ERA2-D2.

Setup:

O *setup* é a parte do código na qual são declarados os pinos de saída e entrada de dados e a função “*Inicial()*”, tudo que precisa ser feito apenas uma única vez.

Figura 34 - *Setup* do Código.

Fonte: Equipe ERA2-D2

Subseção principal:

É a parte do código que executa todas as ações do robô, chamando as funções desejadas em momentos específicos. Aqui são dadas as ordens de execução, as entradas e saídas de dados recebidos e enviados.

Figura 35 - Recorte da parte inicial do *loop*.

```
void loop() {  
    EsquerdaEixo(90);  
    Frente(Distancia(56));  
    EsquerdaEixo(90);  
    Frente(Distancia(12));  
}
```

Fonte: Equipe ERA2-D2

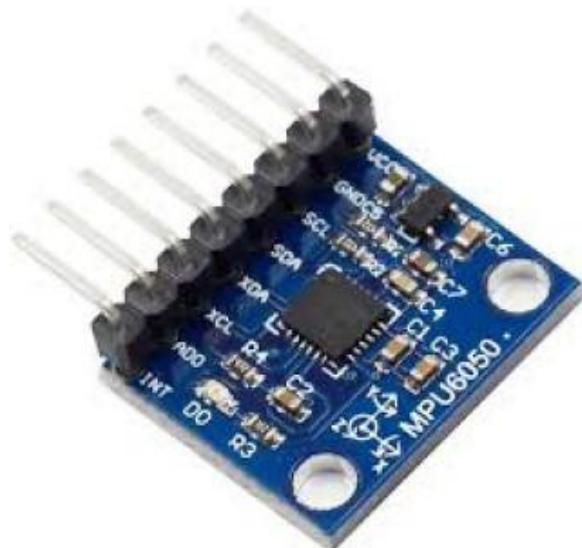
Após a participação no Torneio Regional, realizado em 21 de setembro, realizamos uma série de melhorias no robô com base nas observações feitas durante os testes e a competição. Identificamos que, por depender do tempo como variável para se locomover, e não de um sistema de sensoriamento adequado, o robô apresentava desvios em sua trajetória. Esses problemas eram especialmente evidentes em giros no próprio eixo e em deslocamentos de longas distâncias, comprometendo sua precisão.

Inicialmente, buscamos solucionar o problema utilizando um módulo acelerômetro e giroscópio, o MPU6050. No entanto, após diversas tentativas de implementação, decidimos descartar o componente devido à sua complexidade e às interferências em seu funcionamento, que impediam a obtenção de resultados consistentes. Essa experiência nos motivou a explorar alternativas mais eficazes para garantir a estabilidade e a precisão do robô em futuras competições.

MELHORIAS DE HARDWARE:

Encoder:

Figura 36 - Sensor acelerômetro e giroscópio MPU6050.



Fonte: Equipe ERA2-D2

Uma das soluções encontradas foi a utilização de um *encoder*, um dispositivo que converte movimento ou posição em um sinal elétrico. Ele pode ser usado para medir velocidade, direção ou posição de um motor, por exemplo. Existem dois tipos principais:

- *Encoder incremental*: Gera pulsos para medir deslocamentos relativos. É útil para medir velocidade ou distância percorrida.
- *Encoder absoluto*: Fornece um valor único para cada posição, permitindo saber a posição exata mesmo após as interrupções.

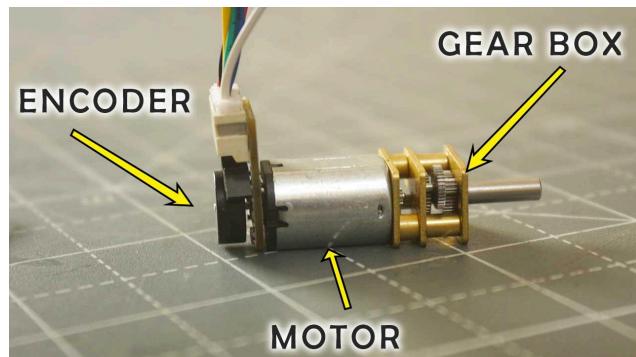
Utilizou-se para o projeto o incremental, que funciona gerando pulsos elétricos enquanto o eixo do motor gira, permitindo medir deslocamentos relativos, como distância percorrida ou velocidade. Ele possui dois canais principais (A e B) que fornecem sinais em quadratura, permitindo identificar a direção do movimento e a contagem de pulsos. Não utiliza-se o canal B para o projeto, já que é possível saber qual o movimento o robô faz.

Esses pulsos são contabilizados pelo Arduino, a resolução é determinada pelo número de pulsos gerados por volta. O motor N20 com *encoder* foi comprado. Devido a falta de motores, o mais próximo de 200 RPM, (motor utilizado) foi o de 150 RPM. De acordo com o fabricante do motor, o *encoder* gera 7 pulsos por revolução e, como a redução do motor é de 100:1 (cem rotações do motor para uma do eixo da roda) obtemos 700 pulsos por volta da roda.

Assim, implementou-se novos motores, instalados no local dos antigos. Foi necessário remodelar o suporte 3D e organizar os fios, já que é inviável refazer a

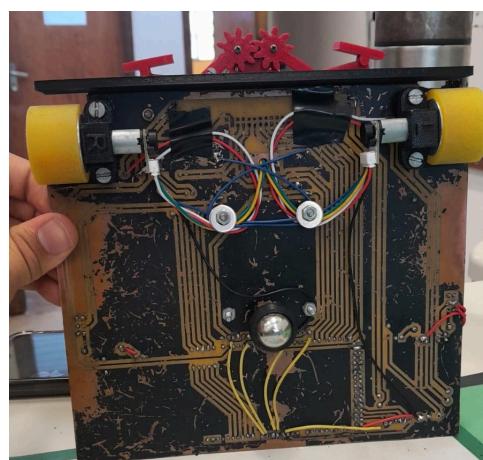
PCB em um curto prazo. Os fios foram ajustados da melhor maneira possível para não causar problemas e permanecerem organizados.

Figura 37 - Motor N20 com encoder.



LABORATÓRIO DIY ELÉTRICO. [Controlando N20 Micro Gear Motor com Encoder usando arduino].
Disponível em: <https://electricdiylab.com> . Acesso em: 27 nov. 2024.

Figura 37 - Motores instalados na PCB.

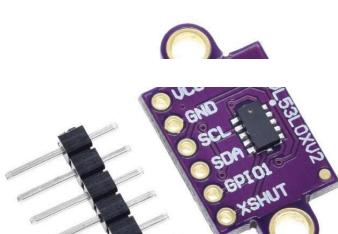


Fonte: Equipe ERA2-D2

LASER:

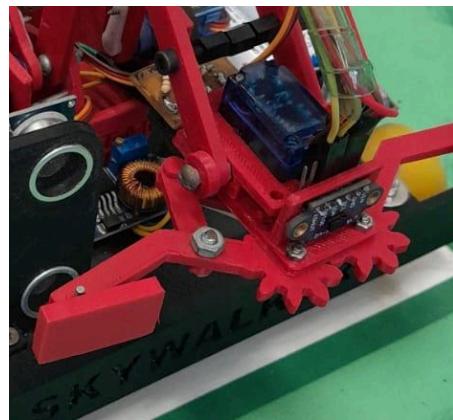
Para melhorar o tempo de leitura da distância, anteriormente feita pelos ultrassônicos, o que demandava manobras extras do robô, já que são deslocados do centro para não atrapalhar a movimentação do braço robótico, obteve-se a ideia de implementar um pequeno sensor a LASER (*Light Amplification by Stimulated Emission of Radiation*, em português "amplificação da luz por emissão estimulada de radiação"), VI53I0XX na parte frontal da garra.

Figura 38 - Sensor de distância a laser VI53I0XX.



CURTO CIRCUITO. [Sensor de Distância Laser - VL53L0X] Disponível em:
<https://www.curtocircuito.com.br>. Acesso em: 27 nov. 2024.

Figura 39 - Sensor de distância a laser instalado na garra.



Fonte: Equipe ERA2-D2

O sensor possui vantagens em comparação ao ultrassônico, pois é pequeno, realiza medidas em milímetros e, principalmente, ser mais preciso por ser um disparo linear, já que o outro é uma onda sonora, dispersando-se em várias direções, gerando interferências.

Melhorias Gerais do Projeto:

Um agrupador de fios feito a partir de uma garrafa PET reutilizada, foi implementado para manter a organização dos fios do braço robótico e melhorar sua mobilidade.

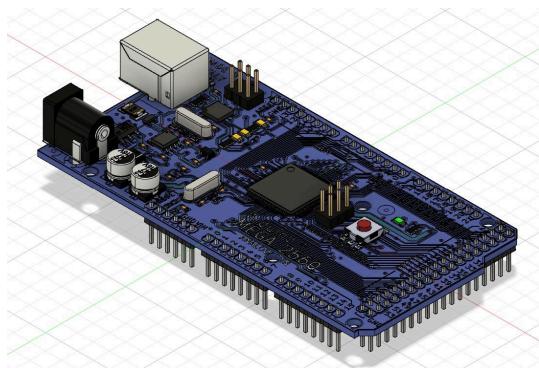
Figura 40 - Organizador de fios feito de garrafa PET.



Fonte: Equipe ERA2-D2

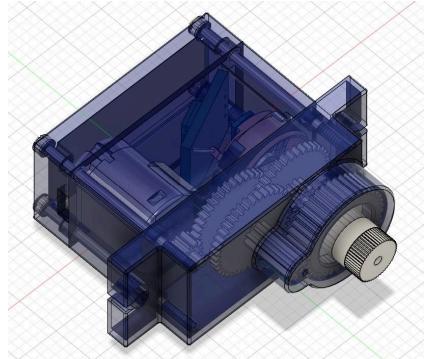
Decidiu-se também que seria feito um modelo 3D do protótipo, a fim de agregar e dar peso ao trabalho. O modelo foi desenvolvido na plataforma Fusion 360, apresentando todas as peças encaixadas em seus devidos lugares (com exceção aos fios). Uma animação de movimento também foi desenvolvida, link para o vídeo no *Google Drive* no apêndice(B).

Figura 41 - Arduino MEGA 2560 modelado em 3D no *software* Fusion 360.



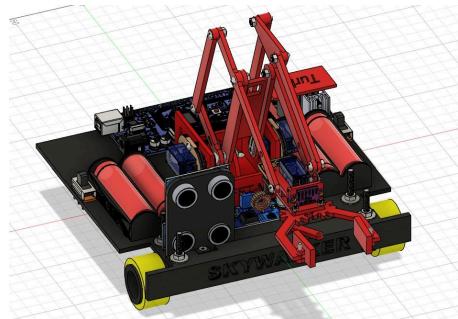
Fonte: Equipe ERA2-D2

Figura 42 - Servomotor S9g modelado em 3D no *software* Fusion 360.



Fonte: Equipe ERA2-D2

Figura 43 - Robô completo modelado em 3D no software Fusion 360.



Fonte: Equipe ERA2-D2

MELHORIAS DE SOFTWARE:

Devido às novas implementações de sensores, foi necessário fazer novas linhas de códigos para programá-los corretamente. Foram adicionadas novas bibliotecas, funções, e uma nova lógica de controle PID (Controlador Proporcional Integral Derivativo) para o *encoder*, além de uma otimização do controle do braço robótico e do *loop*.

O código, anteriormente com 809 linhas, foi reduzido para 620 linhas, representando uma otimização de 23,336% em seu tamanho, sem comprometer as funcionalidades. Além de manter o mesmo desempenho, a nova versão incorpora melhorias significativas, incluindo comentários detalhados que facilitam o entendimento. As estruturas, como o *loop* e funções, foram reformuladas para serem mais eficientes, modulares e legíveis, proporcionando maior clareza e manutenção mais ágil.

Bibliotecas:

Figura 44 - Inclusão das bibliotecas.

```
// Bibliotecas.  
#include <VL53L0X.h>      // Sensor a Laser.  
#include <EEPROM.h>        // Memória EEPROM.  
#include <HCSR04.h>        // Sensor Ultrassônico.  
#include <Servo.h>          // Servomotores.  
#include <Wire.h>           // Comunicação I2C.  
TURMA: Equipe ERA2-D2
```

Novas bibliotecas foram implementadas para controlar o sensor a laser, sendo elas a “VL53L0X.h” e “Wire.h”.

VL53L0X:

É a biblioteca auxiliar que faz a leitura da medição do sensor, em milímetros, de dois modos diferentes, leitura contínua (mais rápido possível) ou de tempos em tempos dados em milissegundos.

WIRE:

O protocolo de comunicação I2C (*Inter-Integrated Circuit*, em português, Circuito Inter-Integrado) é síncrona, e ocorre com a comunicação chefe-funcionário, sendo o Arduino o “chefe” e os outros dispositivos (no caso, o sensor a laser) os “funcionários”, que respondem aos comandos do chefe.

Uma vantagem desse tipo de comunicação é que são utilizados apenas 2 fios, o SDA (*Serial Data*) para enviar e receber dados, e o SCL (*Serial Clock*) para sincronizar os dados. Outra vantagem é que cada funcionário possui um endereço de memória, assim sendo possível conectar até 127 dispositivos a um mestre.

A biblioteca “Wire.h” abstrai a complexidade do protocolo I2C, tornando seu uso simples.

Constantes e Variáveis:

Novas constantes e variáveis foram adicionadas, além de ajustar algumas das antigas para o tipo “double”. Os “#define” que eram utilizados em operações matemáticas foram substituídos por não serem recomendados para esse tipo de tarefa. Continuaram sendo usados para definições de pinos.

- O “double” armazena o dobro de casas decimais em comparação ao *float*.

- O *double* armazena de -3.4028235E+308 ou +3.4028235E+308 (E+308 significa 10^{308}), com precisão de 15-16 casas após a vírgula.

Tabela 07 - Tamanhos e tipos de variáveis *double*.

	<i>Double</i>
Bits	64
Tamanho	3,4028235E+308
Valores	-3,4028235E+308 ou 3,4028235E+308

Fonte: Equipe ERA2-D2

Figura 45 - *Double* no código.

```
const double perimetro = 10.68141502; // Perímetro da roda.
const double circunferencia = 54.35393641; // Circunferência do robô (diâmetro = distância entre eixo das rodas).

// Parâmetros PID.
const double Kp = 5.9; // Constante Proporcional.
const double Ki = 0.6; // Constante Integrativa.
const double Kd = 0.7; // Constante Derivativa.
const double Tm = 0.03; // Tempo médio entre leituras.

double erro, errol = 0, erro2 = 0;
double PWME, PWMD, PWME1 = veLE, PWMD1 = veLD;

double distanciaE = 0, distanciaD = 0, distanciaT = 0;
```

Fonte: Equipe ERA2-D2

Interrupções:

Como o próprio nome diz, as interrupções interrompem o código no momento em que algum pulso é mandado em um pino digital especial, fazem uma função específica e voltam ao andamento do programa. Isso é importante para obter maior precisão das leituras e, é necessário pois ler continuamente um pino demanda, em si, um programa próprio para isso.

Pode-se dizer que as interrupções fazem um programa “paralelo” (não é paralelo pois não acontece ao mesmo tempo do código principal). Existem 4 tipos de sinais diferentes, sendo eles:

- **LOW**: Acionar a interrupção quando o estado do pino for *LOW*, (0).
- **RISING**: Acionar a interrupção quando o estado do pino for de *LOW* (0) para *HIGH* (1), apenas.
- **FALLING**: Acionar a interrupção quando o estado do pino for de *HIGH* (1) para *LOW* (0) apenas.
- **CHANGE**: Acionar a interrupção sempre que o estado do pino mudar (0 para 1 e 1 para 0).

Como citado anteriormente, o *encoder* mandará 700 pulsos por volta e, para maior precisão, utilizou-se o *CHANGE* nos pinos 18 e 19, gerando assim o dobro de precisão, 1400 pulsos por volta.

É preciso indicar ao programa qual será o pino e o número da interrupção. Esses parâmetros foram observados na tabela a seguir.

Tabela 08 - Interrupções associadas a pinos em diferentes versões de Arduino

Placa	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega 2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

AUTOMAÇÃO IFRS RG. [Arduino e Interrupções]. Disponível em: <https://automacaoifrsrg.wordp.com>. Acesso em: 28 nov. 2024.

Figura 46 - Sintaxe, INT 4 e 5, função chamada, borda de mudança (CHANGE).

```
attachInterrupt(4, ContadorEsq, CHANGE);
attachInterrupt(5, ContadorDir, CHANGE);
```

Fonte: Equipe ERA2-D2

Figura 47 - Funções chamadas pelas interrupções, somam um pulso.

```
void ContadorEsq() { pulsosE++; }
void ContadorDir() { pulsosD++; }
```

Fonte: Equipe ERA2-D2

Lógica PID:

Utilizou-se a lógica PID para controle de velocidade, o que consequentemente gerou um protótipo que possui um sistema de navegação preciso para andar em linha reta.

Figura 48 - Função responsável por calcular o PID e ajustar os motores.

```
void CalculaPID() {  
  
    if ((millis() - primeiroMillis) >= (Tm * 1000)) {  
  
        noInterrupts();  
        distanciaE = (double)(perimetro * pulsosE * debugEnc) / 1400;  
        distanciaD = (double)(perimetro * pulsosD) / 1400;  
        distanciaT = distanciaT + (double)(perimetro * pulsosE) / 1400;  
        interrupts();  
  
        pulsosD = pulsosE = 0;  
        primeiroMillis = millis();  
    }  
  
    erro = distanciaD - distanciaE;  
  
    //-----SATURAR PWM PID-----//  
    if (PWME1 > 255) PWME1 = 255;  
    if (PWME1 < 0) PWME1 = 0;  
  
    if (PWMD1 > 255) PWMD1 = 255;  
    if (PWMD1 < 0) PWMD1 = 0;  
  
    // Retirou-se "-Kp" de "(-Kp + Ki*Tm (-2*Kd/Tm))". O erro do motor direito é oposto ao erro do esquerdo.  
    PWME = PWME1 + (Kp + Kd / Tm) * erro + (Ki * Tm - 2 * Kd / Tm) * errrol + (Kd / Tm) * erro2;  
    PWMD = PWMD1 + (Kp + Kd / Tm) * (-1 * erro) + (Ki * Tm - 2 * Kd / Tm) * (-1 * errrol) + (Kd / Tm) * (-1 * erro2);  
    PWME1 = PWME;  
    PWMD1 = PWMD;  
    erro2 = errrol;  
    errrol = erro;  
  
    //-----SATURAR PWM PID-----//  
    if (PWME > 255) PWME = 255;  
    if (PWME < 0) PWME = 0;  
  
    if (PWMD > 255) PWMD = 255;  
    if (PWMD < 0) PWMD = 0;  
  
    //-----COMANDO MOTORES-----//  
    analogWrite(mEv, PWME);  
    analogWrite(mDv, PWMD);  
}
```

Fonte: Equipe ERA2-D2

A função é separada em 6 partes, sendo elas: Ler a distância percorrida pelos motores; calcular o erro; saturar o PWM de entrada; calcular o PID; saturar o PWM de saída; escrever o valor de PWM na saída dos motores. A figura 3.47 (página 38) será utilizada como apoio para as explicações.

Termos: Termos “E” se referem ao motor esquerdo, e termos “D” se referem ao motor direito.

Parte 1: De 30 em 30 milissegundos (Tm , em segundos vezes 1000 para conversão em ms) o código confere a distância percorrida nesse intervalo de tempo e a distância total percorrida pelo robô é atualizada.

Parte 2: O erro é calculado tomando como base referencial o motor esquerdo, por isso é dado como a distância direita menos a esquerda. Se o erro for positivo, o motor esquerdo deve acelerar e o direito desacelerar.

Parte 3: O PWM de entrada (o que inicia no primeiro estado e, consequentemente, os que vão se atualizando ao longo da função e do tempo) é saturado para que não seja menor que 0, e maior que 255, que são valores aceitos pela sintaxe “*analogWrite*”.

Parte 4: O PID é um tipo de controlador amplamente utilizado em sistemas de controle automático, industriais e de automação. Ele serve para regular variáveis de processo, como temperatura, pressão, velocidade e, nesse caso, ajustar a distância percorrida pelos motores para que se mantenham igualmente alinhados e não desviam o robô da rota.

O controlador PID ajusta a saída de um sistema de acordo com três componentes principais:

1. Constante Proporcional (P):

- Representa a resposta diretamente proporcional ao erro do sistema, que é a diferença entre o valor desejado (*setpoint*) e o valor medido (variável de processo).
- Fórmula básica: $P = Kp \cdot e(t)$ onde:
 - Kp : Constante de ganho proporcional.
 - $e(t)$: Erro em um dado instante t .
- Um ganho proporcional alto reduz o erro, mas pode introduzir oscilação e instabilidade.

2. Constante Integral (I):

- Atua acumulando o erro ao longo do tempo, corrigindo desvios persistentes (erro acumulado) para garantir que o sistema alcance exatamente o valor desejado.
- Fórmula básica: $I = Ki \int e(t)dt$, onde:
 - Ki : ganho integral.
- Esse termo é útil para eliminar erros persistentes ao longo do tempo, entretanto um valor alto pode causar instabilidade.

3. Constante Derivativo (D):

- Responde à taxa de variação do erro, antecipando erros futuros e proporcionando uma resposta mais rápida e estável.
- Fórmula básica: $D = Kd \cdot \frac{de(t)}{dt}$, onde:
 - Kd : ganho derivativo.
- Ajuda a reduzir oscilações quando o erro é pequeno, porém, se mal ajustado, pode amplificar ruídos.

Fórmula de Controle PID:

A saída do controlador é a soma dos três componentes:

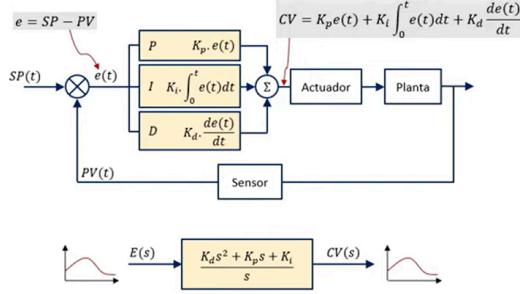
$$u(t) = Kpe(t) + Ki \int e(t)dt + Kd \cdot \frac{de(t)}{dt} (t)$$

Para implementar isso em um microcontrolador, foi necessário utilizar o PID discreto, que resumidamente transforma um sinal “analógico” em digital. Como base, utilizou-se o vídeo da plataforma YouTube, do canal Jesus Correa - PLC, como mostrado na figura 3.51.

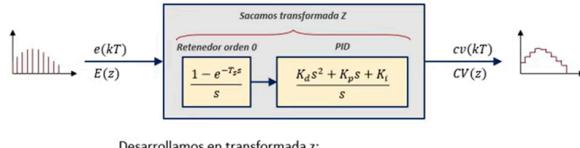
Figura 49 - Discretización de uma equação PID.

Ecuación de diferencias PID

Sabemos por la definición de control PID, que el controlador multiplica, integra y deriva el error, por lo tanto lo que debemos hacer es establecer la suma siguiente:



Si queremos discretizar dicho controlador, debemos entonces poner un retenedor a la entrada, en este caso será por ejemplo un retenedor de orden cero:



Desarrollamos en transformada z:

$$PID_d = \frac{CV(z)}{E(z)} = z \left[\frac{1 - e^{-T_s s}}{s} \left(\frac{K_d s^2 + K_p s + K_i}{s} \right) \right]$$

$$\begin{aligned} &= z \left[\frac{1 - e^{-T_s s}}{s} \left(K_p + \frac{K_i}{s} + K_d s \right) \right] \\ &= (1 - z^{-1}) z \left[\frac{K_p}{s} + \frac{K_i}{s^2} + K_d \right] \\ &= (1 - z^{-1}) \left[\frac{K_p}{(1 - z^{-1})} + \frac{K_i T_s z^{-1}}{(1 - z^{-1})^2} + \frac{K_d}{T_s} \right] \\ \frac{CV(z)}{E(z)} &= \frac{K_p (1 - z^{-1}) + K_i T_s z^{-1} + \frac{K_d}{T_s} (1 - z^{-1})^2}{(1 - z^{-1})} \\ \frac{CV(z)}{E(z)} &= K_p + K_i T_s \frac{1}{z - 1} + \frac{K_d}{T_s} \frac{z - 1}{z} \end{aligned}$$

Ecuación de un PID discreto: Tenga en cuenta que las constantes K_p , K_d y K_i son obtenidas de un sistema continuo.

$$CV(z) - CV(z)z^{-1} = K_p E(z) - K_p E(z)z^{-1} + K_i T_s E(z)z^{-1} + \frac{K_d}{T_s} E(z) + \frac{K_d}{T_s} E(z)z^{-2} - 2 \frac{K_d}{T_s} E(z)z^{-1}$$

$$CV(z) - CV(z)z^{-1} = (K_p + \frac{K_d}{T_s}) E(z) + (-K_p + K_i T_s - 2 \frac{K_d}{T_s}) E(z)z^{-1} + \frac{K_d}{T_s} E(z)z^{-2}$$

$$cv(n) = cv(n-1) + \left(K_p + \frac{K_d}{T_s} \right) e(n) + \left(-K_p + K_i T_s - 2 \frac{K_d}{T_s} \right) e(n-1) + \frac{K_d}{T_s} e(n-2)$$

Ecuación de diferencias de un PID discreto



Fonte: Canal “Jesus Correa - PLC”.

Equação:

$$cv(n) = cv(n-1) + \left(K_p + \frac{K_d}{T_m} \right) e(n) + \left(-K_p + K_i \cdot T_m - 2 \cdot \frac{K_d}{T_m} \right) e(n-1) + \frac{K_d}{T_m} e(n-2)$$

Explicação algébrica:

(n) ≡ Número de passagens pela equação.

cv(n) ≡ PWM atual.

e(n) ≡ Primeiro erro.

cv(n - 1) ≡ PWM anterior.

e(n - 1) ≡ Segundo erro.

e(n - 2) ≡ Terceiro erro.

Utilizou-se o método Ziegler-Nichols e empírico/tentativa e erro, para ter um bom caminho para ajustar as constantes, sendo ele feito pelos seguintes passos:

1. Zerar Ki e Kd .
2. Testar valores de Kp até que o sistema se torne instável (o robô corrigir bruscamente diante de pequenos erros), feito isso o Kp será um pouco menor que o valor testado. O valor encontrado foi 6.
3. Definir Kd como metade de Kp e fazer testes para ver a suavidade do sistema (o robô faz curvas suaves). Kd definido como 3 suavizou muito o sistema (o protótipo faz grandes curvas para corrigir), testes foram feitos e o melhor valor encontrado foi 1.
4. Com o valor de Kd ajustado, definir Ki como metade de Kd , e testar. (o sistema deve diminuir a constância de erros (O robô, visivelmente, não deve oscilar.).

Durante esse passo, percebeu-se que, quando a parte integral da equação era reativada para os testes, o " - Kp " anula o ganho proporcional, fazendo com que o retorno da equação fosse 0. O tempo de amostragem (Tm), que inicialmente era de 10 milissegundos foi substituído por $30ms$, o que fez com que o sistema reagisse melhor a prevenção dos erros futuros, já que com um tempo maior, prevê-se que o erro será maior em um maior período de tempo, evitando que o Kd estoure a equação. Por fim, chegou-se a seguinte equação:

$$cv(n) = cv(n - 1) + \left(Kp + \frac{Kd}{Tm} \right) e(n) + \left(Ki \cdot Tm - 2 \cdot \frac{Kd}{Tm} \right) e(n - 1) + \frac{Kd}{Tm} e(n - 2)$$

5. Fazer o ajuste fino das constantes.

As constantes, após o ajuste fino, ficaram definidas como:

$$\begin{aligned} Kp &= 5,8. \\ Ki &= 0,6. \\ Kd &= 0,7. \end{aligned}$$

Parte 5: O PWM de saída (o novo valor que o motor receberá) é saturado para que não seja menor que 0, e maior que 255.

Parte 6: O motor recebe o novo valor gerado pelo cálculo PID.

A utilização lógica dessa função será mais desenvolvida no bloco “Funções”.

Funções:

Existem 8 novas funções, “ContadorEsq()”, “ContadorDir()”, “CalculaPID()”, “Garra()”, “LerArvore()”, “Laser()”, “TesteMotores()” e “AbaixaRampa()”, além de 2 funções, “Direcao()” e “Gira()”, que substituíram 4 outras, “Frente()”, “Tras()”, “DireitaEixo()” e “EsquerdaEixo()”, totalizando assim 20 funções no código. A seguir, figuras retratando as funções (exceção as já mostradas anteriormente e posteriormente).

Figura 50 - Função para ajustar a altura de leitura das árvores.

```
void LerArvore() { Garra(120, 125); }
```

Fonte: Equipe ERA2-D2

Figura 51 - Função para utilização do sensor laser.

```
VL53L0X laser;  
  
// Função do Laser.  
bool Laser() {  
  
    distanciaLaser = laser.readRangeContinuousMillimeters() - 20; // 20 é o erro do sensor.  
    if (distanciaLaser < 100) arvoreG = true;  
    else arvoreG = false;  
  
    return arvoreG;  
}
```

Fonte: Equipe ERA2-D2

Figura 52 - Função para teste de balanceamento dos motores.

```
void TesteMotores() {  
  
    digitalWrite(mEa, HIGH);  
    digitalWrite(mEb, LOW);  
    digitalWrite(mDa, HIGH);  
    digitalWrite(mDb, LOW);  
    analogWrite(mEv, velE);  
    analogWrite(mDv, velD);  
    delay(120000); // 2 minutos.  
}
```

Fonte: Equipe ERA2-D2

Figura 53 - Função para abaixar a rampa.

```
void AbaixaRampa() {  
    Garra(70, 180); // Garra aberta  
    Garra(50, 180); // Garra aberta  
}
```

Fonte: Equipe ERA2-D2

Explicação das funções:

“Direcao()” e “Gira()”:

Figura 54 - Função de controle dos motores em linha reta.

```
void Direcao(float distancia, bool frente = true) {  
  
    bool a = frente ? true : false;  
    bool b = frente ? false : true;  
  
    attachInterrupt(4, ContadorEsq, CHANGE);  
    attachInterrupt(5, ContadorDir, CHANGE);  
  
    digitalWrite(mEa, a);  
    digitalWrite(mEb, b);  
    digitalWrite(mDa, a);  
    digitalWrite(mDb, b);  
    analogWrite(mEv, veLE);  
    analogWrite(mDv, veld);  
    primeiroMillis = millis();  
  
    while (distanciaT < distancia) { PIDEncoder(); }  
  
    digitalWrite(mEa, b);  
    digitalWrite(mEb, a);  
    digitalWrite(mDa, b);  
    digitalWrite(mDb, a);  
    analogWrite(mEv, veLE);  
    analogWrite(mDv, veld);  
    delay(30);  
  
    Para(50);  
    detachInterrupt(4);  
    detachInterrupt(5);  
    PWME = velE, PWMD = velD, PWMEl = velE, PWMDl = velD;  
    erro = errol = erro2 = pulsosE = pulsosD = distanciaE = distanciaD = distanciaT = 0;  
}  
Fonte: Equipe ERA2-D2
```

Figura 55 - Função de controle dos motores em curvas no próprio eixo.

```
void Gira(float angulo, bool esquerda = true) {  
  
    bool a = esquerda ? false : true;  
    bool b = esquerda ? true : false;  
    double distancia = (double)((angulo / 360) * circunferencia);  
  
    attachInterrupt(4, ContadorEsq, CHANGE);  
    attachInterrupt(5, ContadorDir, CHANGE);  
  
    digitalWrite(mEa, a);  
    digitalWrite(mEb, b);  
    digitalWrite(mDa, b);  
    digitalWrite(mDb, a);  
    analogWrite(mEv, velE);  
    analogWrite(mDv, velD);  
    primeiroMillis = millis();  
  
    while (distanciaT < distancia) { PIDEncoder(); }  
  
    digitalWrite(mEa, b);  
    digitalWrite(mEb, a);  
    digitalWrite(mDa, a);  
    digitalWrite(mDb, b);  
    analogWrite(mEv, velE);  
    analogWrite(mDv, velD);  
    delay(30);  
  
    Para(50);  
    detachInterrupt(4);  
    detachInterrupt(5);  
    PWME = velE, PWMD = velD, PWME1 = velE, PWMD1 = velD;  
    erro = errol = erro2 = pulsosE = pulsosD = distanciaE = distanciaD = distanciaT = 0;  
}
```

Fonte: Equipe ERA2-D2.

As funções “Direcao()” e “Gira()” recebem parâmetros da distância a ser percorrida e direção, a lógica de *true* e *false* auxilia em enviar a lógica 0 e 1 para os pinos dos motores, fazendo com que girem para frente ou para trás. O mesmo ocorre com a função gira, porém a lógica é invertida.

As interrupções são ligadas para que o programa faça as leituras adequadas dos *encoders* acoplados aos motores.

Ambas chamam a função “CalculaPID()”, explicada anteriormente, para fazer o ajuste dos motores. O programa fica dentro do *while* até que a distância andada seja igual a distância desejada. O robô freia bruscamente para parar exatamente no local desejado, para que isso ocorra, inverte-se o sentido de rotação dos motores

durante $30ms$. Após isso a função “Para()” é chamada por $50ms$ para que não ocorra a troca imediata de direção, evitando possíveis erros.

As interrupções são desligadas para que não ocorra leituras desnecessárias na passagem de uma função para outra, melhorando a velocidade e prevenindo possíveis problemas (não utiliza-se a função “interrupts()” e “noInterrupts()” para que não desligue diretamente o *clock*, pois isso faz com que o *delay* do arduino desligue simultaneamente).

Logo depois, o PWM volta a ser o original e os erros e distâncias são zerados para evitar possíveis acúmulos de erros que interfiram nos próximos movimentos.

“Garra()”:

O sistema possui um funcionamento simples, sendo dividido em partes:

- Trava de segurança (saturação) que impõe os limites máximos da garra.
- Análise do tipo de movimento, se irá aumentar ou diminuir os ângulos.
- Movimentação constante dos servomotores ao local desejado, 1 grau a cada $12ms$.
- Atualização do último estado do servomotor, para que a próxima vez que a função seja chamada haja o parâmetro certo para as comparações.

Figura 56 - Função de controle da garra.

```
void Garra(byte x, byte y, bool fechada = false) {
    if (x <= 30) x = 30;
    if (x >= 145) x = 145;

    // Servol.
    if (ultimoEstadoX > x) {
        for (byte i = ultimoEstadoX; i >= x; i--) {
            servol.write(i);
            delay(12);
        }
    } else if (ultimoEstadoX < x) {
        for (byte i = ultimoEstadoX; i <= x; i++) {
            servol.write(i);
            delay(12);
        }
    }

    // Servo2.
    if (ultimoEstadoY > y) {
        for (byte i = ultimoEstadoY; i >= y; i--) {
            servo2.write(i);
            delay(12);
        }
    } else if (ultimoEstadoY < y) {
        for (byte i = ultimoEstadoY; i <= y; i++) {
            servo2.write(i);
            delay(12);
        }
    }

    // Servo3.
    if (fechada == true) {
        if(ultimoEstadoGarra == true) ultimoEstadoGarra = 90;
        for (byte i = ultimoEstadoGarra; i >= 90; i--) {
            servo3.write(i);
            delay(10);
        }
    }
    else servo3.write(180);
    ultimoEstadoGarra = fechada;
    ultimoEstadoX = x;
    ultimoEstadoY = y;
}
```

□

Fonte: Equipe ERA2-D2

Setup:

No *setup* foram incrementadas as novas inicializações necessárias para o funcionamento do código.

Figura 57 - Parte do *Setup* referente ao encoder.

```
// Encoder Esquerdo.  
pinMode(pinAE, INPUT_PULLUP); // Canal A.  
//pinMode(pinBE, INPUT_PULLUP); // Canal B.  
  
// Encoder Direito.  
pinMode(pinAD, INPUT_PULLUP); // Canal A.  
//pinMode(pinBD, INPUT_PULLUP); // Canal B.  
  
// Interrupções.  
attachInterrupt(4, ContadorEsq, CHANGE);  
attachInterrupt(5, ContadorDir, CHANGE);
```

Fonte: Equipe ERA2-D2

Figura 58 - Inicialização do Laser no *Setup*.

```
// Laser.  
Wire.begin();  
laser.setTimeout(500);  
while (!laser.init());  
  
laser.startContinuous();  
...
```

Fonte: Equipe ERA2-D2

Loop:

Após as atualizações no *software*, o *loop* ficou mais enxuto e de melhor compreensão, facilitando a lógica e a sintaxe para a movimentação do robô.

Figura 59 - Parte inicial do novo *loop*.

```
void loop() {  
  
    //-----PRIMEIRA MISSÃO-----  
    //TesteDirecao(); // Utilizar apenas para calibrar a velocidade dos motores.  
  
    Gira(90 * inercia); // Esquerda  
    Direcao(62); // Frente  
    Gira(90); // Esquerda  
  
    LerArvore(); // Garra  
  
    if (Laser()) {  
        //-----Árvore grande-----  
  
        ArvoreGrande(); // Pega a árvore grande  
        servo3.write(90); // Garra fechada  
        Garra(120, 125, true); // Garra fechada  
        Garra(110, 90, true); // Garra fechada  
  
        Direcao(6, false); // Trás  
        Gira(90, false); // Direita  
        Direcao(5); // Frente  
        Gira(90, false); // Direita  
        Direcao(10); // Frente  
        Gira(90); // Esquerda  
        Direcao(9); // Frente  
  
        Garra(120, 90); // Garra aberta  
  
        //-----Pegar próxima árvore-----//  
        Direcao(10, false); // Trás  
        Gira(90); // Esquerda
```

Fonte: Equipe ERA2-D2

6. CRONOGRAMA DE TRABALHO

Tabela 09 - Cronograma de fevereiro até maio de 2024.

AÇÕES PREVISTAS [EQUIPE ERA2-D2 2024]	RESPONSÁVEL	PRAZO TOTAL (DIAS)	RESULTADOS ESPERADOS	MESES			
				FEV	MAR	ABR	MAI
Reunião de finalização da Temporada 2023	Equipe	1	Despedida de antigos membros e patrocinadores.				
Processo Seletivo de Novos Membros	Equipe(Técnico e mentor)	5	Seleção de novos membros comprometidos com o projeto.				
Reunião de Integração com Novos Membros	Equipe	1	Apresentação da equipe, do projeto, das regras e das funções de cada membro.				
Publicação de Vídeo de Agradecimento	Equipe	1	Agradecimento aos patrocinadores e colaboradores				
Abertura Oficial da temporada 2024	Equipe	1	Temporada 2024 iniciada				
Elaboração do orçamento da temporada	Equipe (Gestores)	10	Orçamento detalhado e realista para todo o projeto.				
Documentação para patrocínio	Equipe (Gestores)	20	Documentação pronta para a solicitação de patrocínio				
Definição do Design do Robô	Equipe (Robô)	20	Design funcional, considerando as regras da competição.				
Escolha do projeto para mérito científico	Equipe (Mérito científico)	20	Escolha do projeto com melhor custo-benefício				
Construção do robô	Equipe (Robô)	15	Robô construído e pronto para ser testado				
Início da documentação do TBR	Equipe (Gestores)	1	Documentação da temporada iniciada				
Início das programações do projeto científico	Equipe (Mérito científico)	1	Programações pronta para serem testadas				
Início das programações do robô	Equipe (Robô)	1	Programações pronta para serem testadas				

Fonte: Equipe ERA2-D2.

Tabela 10 - Cronograma de junho até setembro de 2024.

AÇÕES PREVISTAS [EQUIPE ERA2-D2 2024]	RESPONSÁVEL	PRAZO TOTAL (DIAS)	RESULTADOS ESPERADOS	MESES			
				JUN	JUL	AGO	SET
Término da construção do robô	Equipe (Robô)	1	Robô finalizadas para testes				
Início do planejamento de abordagens para o desafio prático	Equipe (Robô)	30	Planejamento iniciado				
Início vendas da rifa	Equipe	1	Venda de rifas iniciados				
Término das estratégias de abordagens no desafio prático	Equipe (Robô)	1	Estratégias finalizadas				
Início dos testes do robô no tapete	Equipe (Robô)	1	Iniciados os testes do robô no tapete				
Início da construção do projeto científico	Equipe (Mérito científico)	1	Iniciada a construção do projeto científico				
Sorteio da rifa	Equipe	1	Rifa sorteada				
Término dos testes do robô	Equipe (Robô)	1	Teste concluídos				
Término da construção do projeto científico	Equipe (Mérito científico)	1	Projeto científico construído				
Teste do projeto científico	Equipe (Mérito científico)	20	Projeto científico testado e aprovado				
Término da documentação	Equipe (Gestores)	1	Documentação concluída				
Finalização e conferência de todo o progresso da equipe	Equipe	5	Progresso finalizado				
Ensaio para apresentação	Equipe	10	Ensaio completo				
Entrega de documentos com 7 dias de antecedência	Equipe (Gestores)	1	Documentos entregues				
Participação da etapa regional do TBR 2024	Equipe	1	Participação da equipe				

Fonte: Equipe ERA2-D2.

Tabela 11 - Cronograma setembro até dezembro de 2024.

AÇÕES PREVISTAS [EQUIPE ERA2-D2 2024]	RESPONSÁVEL	PRAZO TOTAL (DIAS)	RESULTADOS ESPERADOS	MESES			
				SET	OUT	NOV	DEZ
Classificação para etapa nacional do TBR 2024	Equipe	1	Classificação da equipe para nacional				
Melhorias no mérito científico para nacional	Equipe (Mérito científico)	60	Mérito científico aprimorado				
Melhorias no robô e na programação para nacional	Equipe (Robô)	60	Robô e programação aprimorado				
Melhorias na documentação para nacional	Equipe (Gestores)	60	Documentação Aprimorada				
Ensaio para apresentação no nacional	Equipe	15	Ensaio para nacional completo				
Término das melhorias no mérito científico para nacional	Equipe (Mérito científico)	1	Mérito científico pronto para nacional				
Término das melhorias no robô e na programação para nacional	Equipe (Robô)	1	Robô e programação prontos para nacional				
Término das melhorias na documentação para nacional	Equipe (Gestores)	1	Documentação para nacional concluída				
Finalização e conferência de todo o progresso da equipe para nacional	Equipe	5	Progresso para nacional finalizado				
Entrega de documentos para o nacional com 7 dias de antecedência	Equipe (Gestores)	1	Documentos para nacional entregues				
Participação da etapa nacional do TBR 2024	Equipe	1	Participação da equipe na etapa nacional 2024				

Fonte: Equipe ERA2-D2.

7. RESULTADOS COLHIDOS

O desempenho do robô ao longo do projeto foi amplamente satisfatório, demonstrando a competência da equipe desde o planejamento até a execução final. O desenvolvimento da programação foi concluído dentro do cronograma, o que foi crucial para manter um fluxo de trabalho consistente e garantir a integração eficiente entre as partes mecânicas e eletrônicas do robô. O resultado foi um dispositivo que operou com alta precisão e confiabilidade, cumprindo todos os objetivos propostos.

Desde o início, o planejamento envolveu uma escolha cuidadosa dos componentes, como o microcontrolador Arduino Mega 2560, escolhido pela sua compatibilidade com uma ampla gama de sensores e atuadores. A equipe optou por sensores ultrassônicos para reconhecimento de obstáculos e sensores de cor para posicionamento e categorização de objetos. Esses sensores, combinados com os servomotores SG90, permitiram manobras precisas e a realização de tarefas complexas, como o manuseio da garra robótica.

Apesar do planejamento meticuloso, o projeto enfrentou desafios durante sua execução. A montagem do robô exigiu ajustes significativos para garantir que a estrutura fosse robusta o suficiente para suportar todos os componentes. Um dos maiores desafios foi a precisão da garra robótica, que inicialmente não atingia o nível desejado. Isso foi superado com ajustes nos servomotores e melhorias no controle. Além disso, a programação do robô também passou por refinamentos. Embora os algoritmos iniciais tivessem funcionado, os primeiros testes revelaram a necessidade de ajustes para melhorar a precisão e os tempos de execução das tarefas.

O desempenho do robô ao longo do projeto foi satisfatório, demonstrando a competência da equipe desde a fase de planejamento até a execução final. O desenvolvimento da programação foi concluído dentro do cronograma estabelecido, o que foi fundamental para garantir um fluxo de trabalho consistente e uma integração eficiente entre os componentes mecânicos e eletrônicos.

A equipe participou do Torneio Regional no dia 21 de setembro, e após uma análise estratégica, decidiu focar exclusivamente na execução da missão 2. Durante o desafio prático, o robô realizou a missão com excelência, neutralizando corretamente os poluentes, o que garantiu uma pontuação de 290 pontos. Esse desempenho assegurou o primeiro lugar na competição prática. A apresentação dos documentos técnicos e a exposição do projeto de tecnologia e engenharia permitiram à equipe conquistar o primeiro lugar geral na competição, destacando a qualidade e a eficiência do trabalho realizado.

Além disso, a equipe também participou da 33º Mostra Específica de Trabalhos e Aplicações (META), um evento bastante respeitado, que reúne projetos de destaque em diversas áreas da ciência e tecnologia. Durante o evento, a equipe apresentou sua plataforma de monitoramento aquático e o robô desenvolvido para o torneio. Essa participação no META contribuiu para que o trabalho da equipe fosse reconhecido.

Após o torneio, a equipe fez ajustes adicionais no robô. Substituíram-se os motores e as baterias, e o sensor foi regulado para melhorar seu desempenho. A programação também foi aprimorada para permitir que o robô executasse todas as missões do torneio. Com esses ajustes, o robô passou a realizar todas as tarefas com sucesso em 114 segundos, demonstrando a eficiência dos aprimoramentos implementados.

8. ANEXO

Citações:

A Lei de Ohm foi postulada em 1827 pelo físico alemão Georg Simon Ohm (1787-1854).

A Lei de Kirchhoff das Tensões (LKT) foi desenvolvida em 1845 pelo físico alemão Gustav Robert Kirchhoff (1824-1887).

Links:

[Proteus](#)

[Arduino - Home](#)

[Alldatasheet.com](#)

[Thingiverse](#)

[RoboCore](#)

[Nosso melhor robô de todos os tempos! #ManualMaker Aula 10, Vídeo 2](#)

[Micro Servo Motor Tower Pro 9g Sg90 para Arduino | Tecnotronics](#)

[Divisores de Tensão - Introdução para MS e Microcontroladores](#)

[Control PID para Motor DC con encoder - Arduino](#)

[Autodesk Fusion | Software CAD, CAM, CAE 3D e PCB em nuvem](#)

9. APÊNDICE

APÊNDICE A - Materiais e instrumentos.

Materiais:

1. Um (1) microcontrolador Arduino Mega 2560.
2. Um (1) braço robótico modelado em 3D.
3. Dois (2) suportes para baterias modelados em 3D.
4. Quatro (4) baterias de lítio modelo 18650 4.2v.
5. Quatro (4) molas para contato de bateria.
6. Dois (2) reguladores de tensão, modelos LM7805cv e HW-286.
7. Dois (2) capacitores eletrolíticos, 0.1 μ F e 0.33 μ F.

8. Dois (2) sensores ultrassônicos HC-SR04.
9. Uma (1) ponte H modelo L298N.
10. Três (3) servomotores SG90 9 gramas.
11. Um (1) LDR.
12. Dois (2) diodos modelo SK3G 0285.
13. Dois (2) motores CC N20 com caixa de redução 100/1.
14. Dois (2) pneus de silicone 22 milímetros de diâmetro.
15. Uma (1) esfera metálica 18 mm de diâmetro.
16. Dois (2) LED's brancos de alto brilho.
17. Um (1) dissipador de calor.
18. Quatro (4) parafusos, 4 milímetros de diâmetro.
19. Vinte e dois (22) parafusos 3 milímetros de diâmetro.
20. Dois (2) parafusos franceses pequenos.
21. Três (3) parafusos pequenos.
22. Quatro (4) tiras de velcro.
23. Duas (2) chaves deslizantes.
24. Um (1) módulo relé HL-52s v1.0.
25. Uma (1) placa de fenolite cobreada 20cm X 30cm.
26. Conectores macho e fêmea.
27. *Jumpers* metálicos.
28. Fios de cobre vermelhos e amarelos.

Instrumentos auxiliares:

1. Multimetro.
2. Osciloscópio.
3. Computadores.
4. Delta CNC L9060.
5. Impressora 3D CREALITY K1.

APÊNDICE B - DOCUMENTO TÉCNICO DO ROBÔ:

<https://docs.google.com/document/d/1qfaXImFF9rA0MJOla2-Mk9JZPolsYvTirgmmbiL3Px0/edit?usp=sharing>

APÊNDICE C - ANIMAÇÃO 3D DO PROTÓTIPO ROBÓTICO:

https://drive.google.com/drive/folders/1Vy_ZCleIRn55KMFLFU6ARa01si_wWNVY?usp=sharing

APÊNDICE D - Códigos.

TBR Regional Belo Horizonte 2024:

```
// Bibliotecas.  
#include <EEPROM.h>  
#include <HCSR04.h>  
#include <Servo.h>  
  
// Motores.  
#define mEa 29  
#define mEb 27  
#define mDa 25  
#define mDb 23  
#define mEv 2  
#define mDv 3  
  
// Constantes.  
#define tempoDir 8.1111 //original 8.1111 //02-08: 7.73 // 07-08: 7.72 // 09-08:  
8.36  
#define tempoEsq 8.3625 //original 8.3625 //02-08: 8.005 // 07-08: 8.36 // 09-08:  
7.95  
#define tempoReto 0.01890359  
  
#define velE 255  
#define velD 235  
  
#define porcentagemMotor 1  
#define porcentagemEsquerda 1  
#define porcentagemDireita 1.1  
  
// Sensor de cor.  
#define sc A0  
bool CORPRETA;  
bool leituraCorPreta;  
byte contagemDerrubados = 0;  
  
// Mínimos e Máximos de leitura da cor vermelha.  
int CORVERMELHAMin = 4 EEPROM.read(5);  
int CORVERMELHAMax = 4 EEPROM.read(6);  
  
// Calibração de cores.  
#define PRETO 1  
#define BRANCO 0
```

```

// Garra.
Servo servo1; // 60 a 140 graus - graus, mais pra tras
Servo servo2; // 50 a 140 graus - graus, mais alto
Servo servo3; // 180 a 90 graus - graus, garra fechada

int i = 0;

// Ultrassônico.
#define trig1 11
#define trig2 10
#define echo1 8
#define echo2 9

UltraSonicDistanceSensor us1(trig1, echo1); // (Trig, Echo)
UltraSonicDistanceSensor us2(trig2, echo2);

unsigned long distancia1;
unsigned long distancia2;

bool arvoreP = false;
bool arvoreG = false;

// Funções dos Motores.
void Para(int para) {

    digitalWrite(mEa, LOW);
    digitalWrite(mEb, LOW);
    analogWrite(mEv, 0);
    digitalWrite(mDa, LOW);
    digitalWrite(mDb, LOW);
    analogWrite(mDv, 0);
    delay(para);
}

void Frente(int tempo) {

    digitalWrite(mEa, HIGH);
    digitalWrite(mEb, LOW);
    analogWrite(mEv, velE);
    digitalWrite(mDa, HIGH);
    digitalWrite(mDb, LOW);
    analogWrite(mDv, velD);
}

```

```
delay(tempo porcentagemMotor);
Para(100);
}
```

```
void Tras(int tempo) {

digitalWrite(mEa, LOW);
digitalWrite(mEb, HIGH);
analogWrite(mEv, velE);
digitalWrite(mDa, LOW);
digitalWrite(mDb, HIGH);
analogWrite(mDv, velD);
delay(tempo porcentagemMotor);
Para(100);
}
```

```
void DireitaEixo(int angulo) {
```

```
int t;
t = angulo tempoDir;
digitalWrite(mEa, HIGH);
digitalWrite(mEb, LOW);
analogWrite(mEv, 255);
digitalWrite(mDa, LOW);
digitalWrite(mDb, HIGH);
analogWrite(mDv, 255);
delay(t porcentagemDireita);
Para(100);
```

```
}
```

```
void EsquerdaEixo(int angulo) {
```

```
int t;
t = angulo tempoEsq;
digitalWrite(mEa, LOW);
digitalWrite(mEb, HIGH);
analogWrite(mEv, 255);
digitalWrite(mDa, HIGH);
digitalWrite(mDb, LOW);
analogWrite(mDv, 255);
delay(t porcentagemEsquerda);
Para(100);
```

```

}

// Funções Sensor de Cor.
void SensorCor() {
    if (analogRead(sc) > CORVERMELHAMax) {
        CORPRETA = 1;
    } else {
        CORPRETA = 0;
    }
}

void SensorCorAux() {
    if (analogRead(sc) > CORVERMELHAMax) {
        leituraCorPreta = 1;
    } else {
        leituraCorPreta = 0;
    }
}

// Garra.
void Inicial() {
    int v1 = servo1.read();
    int v2 = servo2.read();

    // Ajusta o servo1 para 145°.
    if (v1 < 145) {
        for (int i = v1; i <= 145; i++) {
            servo1.write(i);
            delay(12);
        }
    } else if (v1 > 145) {
        for (int i = v1; i >= 145; i--) {
            servo1.write(i);
            delay(12);
        }
    }
}

// Ajusta o servo2 para 70°.
if (v2 < 70) {
    for (int i = v2; i <= 70; i++) {
        servo2.write(i);
        delay(12);
    }
}

```

```

        }
    } else if (v2 > 70) {
        for (int i = v2; i >= 70; i--) {
            servo2.write(i);
            delay(12);
        }
    }

    servo3.write(180);
}

void ArvoreGrande() {
    for (i = 145; i >= 35; i--) { // Servo1: 35 para a árvore pequena e 55 para a árvore grande.
        servo1.write(i);
        delay(12);
    }

    for (i = 70; i <= 130; i++) {
        servo2.write(i);
        delay(12);
    }
}

void ArvorePequena() {
    for (i = 145; i >= 30; i--) { // Servo1: 35 para a árvore pequena e 55 para a árvore grande.
        servo1.write(i);
        delay(12);
    }

    for (i = 70; i <= 130; i++) {
        servo2.write(i);
        delay(12);
        // Não fique torto por favor, te amamos por isso ❤.
    }
}

// Ultrassônico.
bool Ultrassonico() {
    //distancia1 = us1.measureDistanceCm();
    distancia2 = us2.measureDistanceCm();
}

```

```

if (distancia2 < 10.0) {
    arvoreG = true;
} else {
    arvoreG = false;
}
return arvoreG;
}

// Missão da Cor.
void Turbo(int tempo){

    digitalWrite(4, LOW);
    digitalWrite(mEa, HIGH);
    digitalWrite(mEb, LOW);
    analogWrite(mEv, velE);
    digitalWrite(mDa, HIGH);
    digitalWrite(mDb, LOW);
    analogWrite(mDv, velD);
    delay(tempo);
    digitalWrite(4, HIGH);
}

void Rampa(int tempo) {

    for (i = 145; i >= 48; i--) { // Servo1: 35 para a árvore pequena e 55 para a árvore grande.
        servo1.write(i);
        delay(12);
    }

    for (i = 70; i >= 65; i--) { // Servo1: 35 para a árvore pequena e 55 para a árvore grande.
        servo2.write(i);
        delay(12);
    }

    servo3.write(85);

    Turbo(tempo);

    tempo = 2000;
}

```

```

    for (i = 48; i >= 30; i--) { // Servo1: 35 para a árvore pequena e 55 para a árvore
        grande.
        servo1.write(i);
        delay(12);

        Turbo(tempo);
        break;

    }
}

void Cor() {

    for (i = 30; i <= 60; i++) {
        servo1.write(i);
        delay(12);
    }
    servo2.write(65);
}

// Tema Star Wars.
void StarWars() {
    ;
}

// Tempo.
float Distancia(double distancia) {
    distancia = distancia * 1.0238 + 0.486;
    float tempo = distancia / tempoReto;
    return tempo;
}

void setup() {

    // Motores.
    pinMode(mEa, OUTPUT);
    pinMode(mEb, OUTPUT);
    pinMode(mDa, OUTPUT);
    pinMode(mDb, OUTPUT);
    pinMode(mEv, OUTPUT);
    pinMode(mDv, OUTPUT);
    pinMode(4, OUTPUT);
}

```

```

// Sensor de Cor.
pinMode(sc, INPUT);

// Servomotores.
servo1.attach(5);
servo2.attach(6);
servo3.attach(7);

// Ultrassônicos.
pinMode(trig1, OUTPUT);
pinMode(trig2, OUTPUT);
pinMode(echo1, INPUT);
pinMode(echo2, INPUT);

// Inicializações.
digitalWrite(4, HIGH);
Inicial();
}

void loop() {

    int H = 1;
    if (H == 1) {

        // Primeira parte.
        EsquerdaEixo(100);
        Frente(Distancia(56));
        EsquerdaEixo(90);
        Frente(Distancia(12));

        if (Ultrassonico()) {

            Tras(Distancia(12));
            DireitaEixo(90);
            Frente(Distancia(6));
            EsquerdaEixo(90);
            ArvoreGrande(); // ARVORE GRANDE.
            Frente(Distancia(6.5));

            for (i = 180; i >= 90; i--) {
                servo3.write(i);
                delay(8);
            }
        }
    }
}

```

```

for (i = 35; i <= 110; i++) {
    servo1.write(i);
    delay(12);
}

for (i = 130; i >= 90; i--) {
    servo2.write(i);
    delay(12);
}

Tras(Distancia(7));
DireitaEixo(90);
Frente(Distancia(5));
DireitaEixo(90);
Frente(Distancia(6));
EsquerdaEixo(90);
Frente(Distancia(15));

for (i = 110; i <= 120; i++) {
    servo1.write(i);
    delay(12);
}

servo3.write(180);

// Pegar proxima árvore.
Tras(Distancia(7));
EsquerdaEixo(90);
Frente(Distancia(17));
DireitaEixo(20);

for (i = 90; i <= 130; i++) {
    servo2.write(i);
    delay(12);
}

for (i = 110; i >= 25; i--) {
    servo1.write(i);
    delay(12);
}

for (i = 180; i >= 80; i--) {

```

```

servo3.write(i);
delay(12);
}

for (i = 25; i <= 110; i++) {
    servo1.write(i);
    delay(12);
}

Tras(Distancia(4));
DireitaEixo(60);
Frente(Distancia(16));
servo3.write(180);

Tras(Distancia(8));
EsquerdaEixo(90);
Frente(Distancia(16));
EsquerdaEixo(90);

} else {

//Árvore pequena.
Tras(Distancia(12)); //12
DireitaEixo(90);
Frente(Distancia(6));
EsquerdaEixo(90);
ArvorePequena();
Frente(Distancia(6));

for (i = 180; i >= 80; i--) {
    servo3.write(i);
    delay(12);
}

for (i = 45; i <= 100; i++) {
    servo1.write(i);
    delay(12);
}

Tras(Distancia(1));
DireitaEixo(90);
Frente(Distancia(20));

```

```

for (i = 120; i <= 115; i++) {
    servo1.write(i);
    delay(12);
}

for (i = 130; i <= 150; i++) {
    servo2.write(i);
    delay(12);
}

servo3.write(180);

// Pegar próxima árvore.
Tras(Distancia(10));
EsquerdaEixo(60);

for (i = 150; i >= 100; i--) {
    servo2.write(i);
    delay(12);
}

for (i = 80; i >= 45; i--) {
    servo1.write(i);
    delay(12);
}

Frente(Distancia(9));

for (i = 180; i >= 80; i--) {
    servo3.write(i);
    delay(8);
}

for (i = 45; i <= 110; i++) {
    servo1.write(i);
    delay(12);
}

for (i = 90; i >= 55; i--) {
    servo2.write(i);
    delay(12);
}

```

```

Tras(Distancia(9));
DireitaEixo(60);
Frente(Distancia(3));
DireitaEixo(90);
Frente(Distancia(8));
EsquerdaEixo(90);
Frente(Distancia(14));

for (i = 60; i <= 110; i++) {
    servo2.write(i);
    delay(12);
}

servo3.write(180);
Tras(Distancia(7));
DireitaEixo(90);
Tras(Distancia(19));
DireitaEixo(90);

}
Frente(Distancia(78));
Para(3000);
}

```

// SEGUNDA MISSÃO

```

Frente(Distancia(38));
DireitaEixo(90);
Rampa(4000);
Frente(Distancia(60));

Cor();
Para(500);
SensorCor();

for (i = 60; i <= 80; i++) {
    servo1.write(i);
    delay(10);
}


```

```

for (i = 65; i >= 55; i--) {
    servo2.write(i);
}

```

```

    delay(10);
}

for (i = 180; i >= 90; i--) {
    servo3.write(i);
    delay(5);
}

Tras(Distancia(33));
DireitaEixo(90);
Frente(Distancia(29));

for (int y = 0; y < 4; y++) {

    SensorCorAux();

    if (leituraCorPreta == CORPRETA) {

        if(contagemDerrubados == 4){
            break;
        }

        Tras(Distancia(9));
        EsquerdaEixo(40);

        for (i = 80; i >= 50; i--) {
            servo1.write(i);
            delay(12);
        }

        DireitaEixo(35);
        contagemDerrubados++;

        for (i = 50; i <= 80; i++) {
            servo1.write(i);
            delay(10);
        }

        if (y == 3) {
            break;
        }
    }
}

```

} else {

```

if (y == 3) {
    break;
}

Tras(Distancia(10));
}

EsquerdaEixo(92);
Frente(Distancia(26));
DireitaEixo(90);
Frente(Distancia(10));
}

Tras(Distancia(33));
DireitaEixo(90);
Frente(Distancia(40));

// Outro lado (otherside).
Tras(Distancia(33));
DireitaEixo(90);
Frente(Distancia(33));

for (int y = 0; y < 4; y++) {

    SensorCorAux();

    if (leituraCorPreta == CORPRETA) {

        if(contagemDerrubados == 4){
            break;
        }

        Tras(Distancia(9));
        DireitaEixo(35);

        for (i = 80; i >= 50; i--) {
            servo1.write(i);
            delay(12);
        }

        EsquerdaEixo(40);
    }
}

```

```

contagemDerrubados++;

for (i = 50; i <= 80; i++) {
    servo1.write(i);
    delay(10);
}

if (y == 3) {
    break;
}

} else {

    if (y == 3) {
        break;
    }

    Tras(Distancia(10));
}

EsquerdaEixo(92);
Frente(Distancia(27));
DireitaEixo(90);
Frente(Distancia(10));
}

delay(10000);
}

```

TBR Nacional 2024:

```

// Bibliotecas.
#include <StarWars.h> // Música StarWars.
#include <VL53L0X.h> // Sensor Laser.
#include <EEPROM.h> // Memória EEPROM.
#include <HCSR04.h> // Ultrassônico.
#include <Servo.h> // Servomotores.
#include <Wire.h> // Protocolo I2C.

// Pinos dos Motores.
#define mEa 29 // Pino Digital.
#define mEb 27 // Pino Digital.

```

```

#define mDa 25 // Pino Digital.
#define mDb 23 // Pino Digital.
#define mEv 2 // Pino Digital PWM.
#define mDv 3 // Pino Digital PWM.

// Pinos dos Encoders.
#define pinAD 18 // Canal A.
#define pinAE 19 // Canal A.
//#define pinBD // Canal B.
//#define pinBD // Canal B.

// Constantes dos Motores.
#define velE 255 // 100% de velocidade no Motor Esquerdo.
#define velD 217 // 85% de velocidade no Motor Direito.
const float debugEnc = 0.96; // Erro do encoder. Aumenta vai pra esquerda.

const double perimetro = 10.68141502; // Perímetro da roda.
const double circunferencia = 54.35393641; // Circunferência do robô (diâmetro = distância entre eixo das rodas).

// Parâmetros PID.
const double Kp = 5.8; // 6 Constante Proporcional.
const double Ki = 0.6; // 0.6 Constante Integrativa.
const double Kd = 0.7; // 0.75 Constante Derivativa.
const double Tm = 0.03; // 0.03 // Tempo de amostragem média entre leituras.

double erro, erro1 = 0, erro2 = 0;
double PWME, PWMD, PWME1 = velE, PWMD1 = velD;

double distanciaE = 0, distanciaD = 0, distanciaT = 0;
unsigned long primeiroMillis;

volatile long pulsosE = 0, pulsosD = 0;

//-----ENCODER-----
void ContadorEsq() { pulsosE++; }
void ContadorDir() { pulsosD++; }
void PIDEncoder() {

if ((millis() - primeiroMillis) >= (Tm * 1000)) {
noInterrupts();
distanciaE = (double)(perimetro * pulsosE * debugEnc) / 1400;
distanciaD = (double)(perimetro * pulsosD) / 1400;
distanciaT = distanciaT + (double)(perimetro * pulsosE) / 1400;
interrupts();
}
}

```

```

pulosD = pulsosE = 0;
primeiroMillis = millis();
}

erro = distanciaD - distanciaE;

//-----SATURAR PWM DE ENTRADA-----//
if (PWME1 > 255) PWME1 = 255;
if (PWME1 < 0) PWME1 = 0;

if (PWMD1 > 255) PWMD1 = 255;
if (PWMD1 < 0) PWMD1 = 0;

//-----CALCULO PID-----//
PWME = PWME1 + (double)((Kp + Kd / Tm) * erro + (Ki * Tm - 2 * Kd / Tm) * erro1 + (Kd / Tm) * erro2); // Retirou-se -Kp de (-Kp + Ki*Tm -2*Kd/Tm).
PWMD = PWMD1 + (double)((Kp + Kd / Tm) * (-1*erro) + (Ki * Tm - 2 * Kd / Tm) * (-1 * erro1) + (Kd / Tm) * (-1 * erro2)); // O erro direito é sempre o oposto do esquerdo.
PWME1 = PWME;
PWMD1 = PWMD;
erro2 = erro1;
erro1 = erro;

//-----SATURAR PWM DE SAÍDA-----//
if (PWME > 255) PWME = 255;
if (PWME < 0) PWME = 0;

if (PWMD > 255) PWMD = 255;
if (PWMD < 0) PWMD = 0;

//-----COMANDO MOTORES-----//
analogWrite(mEv, PWME);
analogWrite(mDv, PWMD);
}

//-----GARRA-----//
Servo servo1; // 30 a 145 graus - graus, mais pra tras.
Servo servo2; // 50 a 140 graus - graus, mais alto.
Servo servo3; // 180 a 85 graus - graus, garra fechada.

byte ultimoEstadoX = 0, ultimoEstadoY = 0, ultimoEstadoGarra;

// Funções da Garra.
void Garra(byte x, byte y, bool fechada = false) {

```

```

if (x <= 30) x = 30;
if (x >= 145) x = 145;

// Servo1.
if (ultimoEstadoX > x) {
    for (byte i = ultimoEstadoX; i >= x; i--) {
        servo1.write(i);
        delay(12);
    }
} else if (ultimoEstadoX < x) {
    for (byte i = ultimoEstadoX; i <= x; i++) {
        servo1.write(i);
        delay(12);
    }
}

// Servo2.
if (ultimoEstadoY > y) {
    for (byte i = ultimoEstadoY; i >= y; i--) {
        servo2.write(i);
        delay(12);
    }
} else if (ultimoEstadoY < y) {
    for (byte i = ultimoEstadoY; i <= y; i++) {
        servo2.write(i);
        delay(12);
    }
}

// Servo3.
if (fechada == true) {
    if(ultimoEstadoGarra = true) ultimoEstadoGarra = 90;
    for (byte i = ultimoEstadoGarra; i >= 90; i--) {
        servo3.write(i);
        delay(10);
    }
}
else servo3.write(180);
ultimoEstadoGarra = fechada;
ultimoEstadoX = x;
ultimoEstadoY = y;
}

void Inicial() {
    ultimoEstadoX = servo1.read();
    ultimoEstadoY = servo2.read();
}

```

```

    Garra(145, 70); // Posição inicial: Servo1 em 145º e Servo2 em 70º, garra aberta.
}

void LerArvore() { Garra(90, 110); } // Altura ideal para fazer a leitura das alturas das árvores.
void ArvoreGrande() { Garra(50, 130); } // Altura ideal para pegar a árvore grande.
void ArvorePequena() { Garra(40, 130); } // Altura ideal para pegar a árvore pequena.

//-----SENSOR DE
COR-----//  

#define sc A0 // Pino do Sensor de Cor.

bool CORPRETA;
bool leituraCorPreta;
byte contagemDerrubados = 0;

// Mínimos e Máximos de leitura da cor vermelha.
unsigned int CORVERMELHAMin;
unsigned int CORVERMELHAMax;

// Funções Sensor de Cor.
void SensorCor() {
    if (analogRead(sc) > CORVERMELHAMax) CORPRETA = 1;
    else CORPRETA = 0;
}

void SensorCorAux() {
    if (analogRead(sc) > CORVERMELHAMax) leituraCorPreta = 1;
    else leituraCorPreta = 0;
}

//-----LASER-----//  

unsigned long distanciaUs1; // Armazena a leitura de distância do Ultrassônico 1.
unsigned long distanciaUs2; // Armazena a leitura de distância do Ultrassônico 2.
unsigned long distanciaLaser; // Armazena a leitura de distância do Laser.

bool arvoreG; // Árvore Grande, verdadeira ou falsa.

VL53L0X laser; // Declara o Laser 1.

// Função do Laser.
bool Laser() {

    for(byte i = 0; i < 3; i++){
        distanciaLaser = distanciaLaser + (laser.readRangeContinuousMillimeters() - 15); // 15 é o
        erro do sensor.
}

```

```

}

distanciaLaser = distanciaLaser / 4;

if (distanciaLaser >= 100) arvoreG = false;
else arvoreG = true;

return arvoreG;
}

//-----ULTRASSÔNICO-----
// Pinos dos Ultrassônicos.
#define trig1 11 // Pino Digital.
#define trig2 10 // Pino Digital.
#define echo1 8 // Pino Digital.
#define echo2 9 // Pino Digital.

UltraSonicDistanceSensor us1(trig1, echo1); // Declara o Ultrassônico 1.
UltraSonicDistanceSensor us2(trig2, echo2); // Declara o Ultrassônico 2.

// Função do Ultrassônico.
bool Ultrassonico() {

//distanciaUs1 = us1.measureDistanceCm();
distanciaUs2 = us2.measureDistanceCm(); // (Medição do Ultrassônico 2 em centímetros).

if (distanciaUs2 < 10) arvoreG = true;
else arvoreG = false;

return arvoreG;
}

//-----FUNÇÕES DOS MOTORES-----
void Para(unsigned long para) {

digitalWrite(mEa, LOW);
digitalWrite(mEb, LOW);
digitalWrite(mDa, LOW);
digitalWrite(mDb, LOW);
analogWrite(mEv, 0);
analogWrite(mDv, 0);
delay(para);
}

```

```

void Direcao(float distancia, bool frente = true) {

    bool a = frente ? true : false;
    bool b = frente ? false : true;

    attachInterrupt(4, ContadorEsq, CHANGE);
    attachInterrupt(5, ContadorDir, CHANGE);

    digitalWrite(mEa, a);
    digitalWrite(mEb, b);
    digitalWrite(mDa, a);
    digitalWrite(mDb, b);
    analogWrite(mEv, velE);
    analogWrite(mDv, velD);
    primeiroMillis = millis();

    while (distanciaT < distancia) { PIDEncoder(); }

    digitalWrite(mEa, b);
    digitalWrite(mEb, a);
    digitalWrite(mDa, b);
    digitalWrite(mDb, a);
    analogWrite(mEv, velE);
    analogWrite(mDv, velD);
    delay(30);

    Para(50);
    detachInterrupt(4);
    detachInterrupt(5);
    PWME = velE, PWMD = velD, PWME1 = velE, PWMD1 = velD;
    erro = erro1 = erro2 = pulsosE = pulsosD = distanciaE = distanciaD = distanciaT = 0;
}

void Gira(float angulo, bool esquerda = true) {

    bool a = esquerda ? false : true;
    bool b = esquerda ? true : false;
    double distancia = (double)((angulo / 360) * circunferencia);

    attachInterrupt(4, ContadorEsq, CHANGE);
    attachInterrupt(5, ContadorDir, CHANGE);

    digitalWrite(mEa, a);
    digitalWrite(mEb, b);
    digitalWrite(mDa, b);
    digitalWrite(mDb, a);
    analogWrite(mEv, velE);
}

```

```

analogWrite(mDv, velD);
primeiroMillis = millis();

while (distanciaT < distancia) { PIDEncoder(); }

digitalWrite(mEa, b);
digitalWrite(mEb, a);
digitalWrite(mDa, a);
digitalWrite(mDb, b);
analogWrite(mEv, velE);
analogWrite(mDv, velD);
delay(30);

Para(50);
detachInterrupt(4);
detachInterrupt(5);
PWME = velE, PWMD = velD, PWME1 = velE, PWMD1 = velD;
erro = erro1 = erro2 = pulsosE = pulsosD = distanciaE = distanciaD = distanciaT = 0;
}

//-----FUNÇÕES
//-----COMPLEMENTARES-----//
void Turbo(float tempo) {

    digitalWrite(49, LOW);

    digitalWrite(mEa, HIGH);
    digitalWrite(mEb, LOW);
    digitalWrite(mDa, HIGH);
    digitalWrite(mDb, LOW);
    analogWrite(mEv, velE);
    analogWrite(mDv, velD);
    delay(tempo);

    digitalWrite(49, HIGH);
}

void Rampa(unsigned int tempo) {
    if(tempo > 4000) tempo = 4000;
    Garra(105, 65, true); // Garra fechada
    Garra(70, 65, true); // Garra fechada

    Direcao(200);
    Turbo(tempo);
    tempo = 2000;
    for (byte i = 48; i >= 20; i--) {
        servo1.write(i);
    }
}

```

```

delay(12);

Turbo(tempo);
break;
}
}

void AbaixaRampa() {
    Garra(70, 180); // Garra aberta
    Garra(50, 180); // Garra aberta
}

void Cor() {
    Garra(62, 65, true); // Garra fechada
    servo2.write(65); // Força o Servo
}

//-----SETUP-----
-----

void setup() {

    // Motores.
    pinMode(mEa, OUTPUT);
    pinMode(mEb, OUTPUT);
    pinMode(mDa, OUTPUT);
    pinMode(mDb, OUTPUT);
    pinMode(mEv, OUTPUT);
    pinMode(mDv, OUTPUT);

    // Turbo.
    pinMode(49, OUTPUT);
    digitalWrite(49, HIGH); // Desliga o Turbo (relé). Lógica invertida devido a eletrônica.

    // Servomotores.
    servo1.attach(5);
    servo2.attach(6);
    servo3.attach(7);
    Inicial(); // Desloca a garra para a posição inicial.

    // Sensor de Cor.
    pinMode(sc, INPUT);

    // Laser.
    Wire.begin();
    laser.setTimeout(500);
    while (!laser.init());
}

```

```

laser.startContinuous(); // Inicia o modo contínuo consecutivo (Lê o mais rápido possível).
// Para usar o modo de tempo contínuo, forneça um período entre as medições em
millissegundos "laser.startContinuous(100)".

// Ultrassônicos.
pinMode(trig1, OUTPUT);
pinMode(trig2, OUTPUT);
pinMode(echo1, INPUT);
pinMode(echo2, INPUT);

// Encoder Esquerdo.
pinMode(pinAE, INPUT_PULLUP); // Canal A.
//pinMode(pinBE, INPUT_PULLUP); // Canal B.

// Encoder Direito.
pinMode(pinAD, INPUT_PULLUP); // Canal A.
//pinMode(pinBD, INPUT_PULLUP); // Canal B.

// Interrupções.
attachInterrupt(4, ContadorEsq, CHANGE);
attachInterrupt(5, ContadorDir, CHANGE);

// StarWars Music Theme.
// ImperialMarch march(45, 108);

// Inicializações.
CORVERMELHAMin = 4 * EEPROM.read(5);
CORVERMELHAMax = 4 * EEPROM.read(6);
}

//-----LOOP-----
-----
void loop() {

//-----PRIMEIRA
MISSÃO-----//
//TesteMotores(); // Utilizar APENAS para calibrar a velocidade dos motores.
Rampa(1100);
// Direcao(62); // Frente
delay(10000000);

Gira(90); // Esquerda
Direcao(62); // Frente
Gira(90); // Esquerda
}

```

```
LerArvore(); // Garra  
  
if (Laser()) {  
    //-----Árvore  
    grande-----//
```

```
        ArvoreGrande(); // Pega a árvore grande  
        Direcao(3); // Frente  
        servo3.write(90); // Garra fechada  
        Garra(120, 125, true); // Garra fechada  
        Garra(110, 90, true); // Garra fechada
```

```
        Direcao(6, false); // Trás  
        Gira(90, false); // Direita  
        Direcao(5); // Frente  
        Gira(90, false); // Direita  
        Direcao(10); // Frente  
        Gira(90); // Esquerda  
        Direcao(9); // Frente
```

```
        Garra(120, 90); // Garra aberta
```

```
    //-----Pegar próxima
```

```
    árvore-----//  
    Direcao(10, false); // Trás  
    Gira(90); // Esquerda  
    Direcao(17); // Frente  
    Gira(30, false); // Direita
```

```
    Garra(120, 135); // Garra aberta  
    Garra(32, 135, true); // Garra fechada  
    Garra(115, 135, true); // Garra fechada
```

```
    Direcao(4, false); // Trás  
    Gira(60, false); // Direita  
    Direcao(12); // Frente
```

```
    Garra(115, 135); // Garra aberta
```

```
    Direcao(7, false); // Trás  
}
```

```
    //-----Árvore  
    pequena-----//  
    else {
```

```

ArvorePequena();

Direcao(3); // Frente

Garra(30, 130, true); // Garra fechada
Garra(100, 130, true); // Garra fechada
servo1.write(100); // Força o Servo

Direcao(5, false); // Trás
Gira(90, false); // Direita
Direcao(15); // Frente

Garra(120, 130); // Garra aberta

//-----Pegar próxima
árvore-----//  

Direcao(12, false); // Trás
Gira(60); // Esquerda

Garra(45, 100); // Garra aberta

Direcao(10); // Frente

Garra(45, 100, true); // Garra fechada
Garra(110, 55, true); // Garra fechada

Direcao(9, false); // Trás
Gira(60, false); // Direita
Direcao(3); // Frente
Gira(90, false); // Direita
Direcao(12); // Frente
Gira(90); // Esquerda
Direcao(12); // Frente

Garra(110, 110, true); // Garra fechada
Garra(105, 110, true); // Garra fechada
servo3.write(180); // Garra aberta

Direcao(8, false); // Trás
Gira(90, false); // Direita
Direcao(15, false); // Trás
Gira(90); // Esquerda
}

Direcao(30, false); // Trás

```

```

//----- SEGUNDA
MISSÃO-----//

Gira(92, false); // Direita
Direcao(16); // Frente
Rampa(1000); // Turbo 650
Gira(3); // Esquerda
Direcao(35); // Frente

Cor();
SensorCor();

Garra(95, 50, true); // Garra fechada

Direcao(32, false); // Trás
Gira(90, false); // Direita
Direcao(32); // Frente

for (byte y = 0; y < 4; y++) {

    SensorCorAux(); // Lê a cor

    if (leituraCorPreta == CORPRETA) {

        if (contagemDerrubados == 4) break;

        Direcao(10, false); // Trás
        Gira(40, false); // Direita

        Garra(40, 50, true); // Garra fechada

        Gira(40); // Esquerda
        contagemDerrubados++; // Acrescenta 1 a mais na contagem dos poluentes derrubados.

        Garra(95, 50, true); // Garra fechada

        if (y == 3) break;
    }

    else {

        if (y == 3) break;
        Direcao(10, false); // Trás
    }

    if (contagemDerrubados >= 4) break;
}

```

```

Gira(90); // Esquerda
Direcao(25.5); // Frente
Gira(90, false); // Direita
Direcao(11.2); // Frente
}

// Caso não estejam todos derrubados do lado inicial.
if (contagemDerrubados < 4) {

    if (leituraCorPreta == CORPRETA) Direcao(18, false); // Trás
    else Direcao(25, false); // Trás

    Gira(90, false); // Direita
    Direcao(40); // Frente

    //-----OUTRO
    LADO-----//


    Direcao(27, false); // Trás
    Gira(90, false); // Direita
    Direcao(33); // Frente

    for (int y = 0; y < 4; y++) {

        if (contagemDerrubados == 4) break;

        SensorCorAux();

        if (leituraCorPreta == CORPRETA) {

            if (contagemDerrubados == 4) break;

            Direcao(9, false); // Trás
            Gira(40); // Esquerda

            Garra(40, 55, true); // Garra fechada

            Gira(40, false); // Direita
            contagemDerrubados++;

            Garra(95, 50, true); // Garra fechada

            if (y == 3) break;
        }
    }
}

```

```

else {

    if (y == 3) break;
    Direcao(10, false); // Trás
}

if (contagemDerrubados >= 4) break;

Gira(90);      // Esquerda
Direcao(26.8); // Frente
Gira(90, false); // Direita
Direcao(12);   // Frente
}

}

else {
//march.playMelody();
delay(600000); // 10 minutos.
}
//march.playMelody();
delay(600000); // 10 minutos.
}

void TesteMotores() {
digitalWrite(mEa, HIGH);
digitalWrite(mEb, LOW);
digitalWrite(mDa, HIGH);
digitalWrite(mDb, LOW);
analogWrite(mEv, velE);
analogWrite(mDv, velD);
delay(60000); // 1 minuto.
}

```