



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

RELATÓRIO DO PROJETO 1
SISTEMAS DIGITAIS - ROBÔ DETECTOR DE GÁS

Alunos: Felipe Lima de Medeiros

Flávio Henrique Lopes Barbosa

Gildson Bezerra dos Santos

Thatiana Jéssica da Silva Ribeiro

Natal-RN

2021

ROBÔ DETECTOR DE GÁS

Relatório do projeto apresentado à disciplina de Sistemas Digitais, correspondente parcialmente à avaliação da 1ª unidade do semestre 2021.1 do curso de Engenharia de Computação e Automação da Universidade Federal do Rio Grande do Norte, sob orientação do **Prof. Marcelo Augusto Costa Fernandes**.

Professor: Marcelo Augusto Costa Fernandes

Natal-RN

2021

1. INTRODUÇÃO

Utilizando técnicas de sistemas embarcados, foi proposta a implementação de um robô capaz de trafegar no interior de dutos e fazer a coleta de informações, utilizando sensores, para assim determinar a presença de gases no ambiente. O robô proposto, movimenta-se no percurso base, partindo de p_0 até p_5 , conforme mostrado na Figura 1. A medida que ele atinge os pontos intermediários demarcados no esquemático, ele utiliza sensores para coletar informações de: temperatura, luminosidade, presença de gases e de movimento. Esses pontos são denominados pontos de coleta.

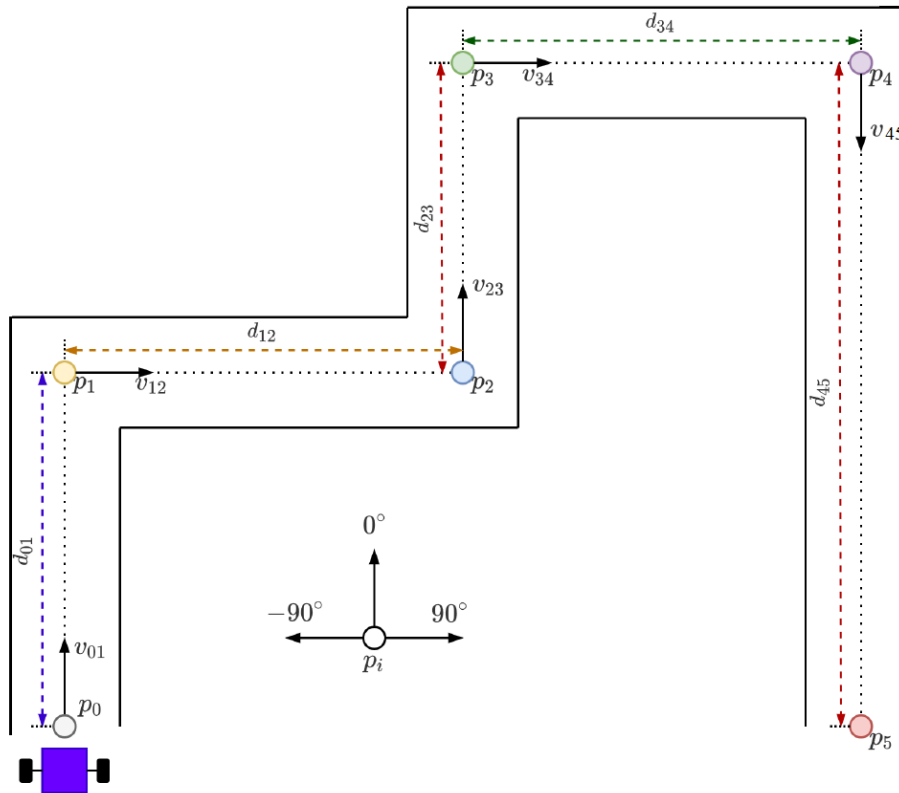


Figura 1 - Percurso base percorrido pelo robô

As distâncias entre pontos de coleta são arbitrárias, e algumas modificações foram feitas na orientação, então, para o robô desenvolvido neste estudo, o percurso não necessariamente deverá seguir esse formato. Na primeira demonstração ele será igual ao da imagem, mas em seções futuras outras simulações de percurso distinto também serão mostradas. O programa de modelagem computacional Tinkercad foi utilizado para a elaboração deste projeto. Os requisitos deste projeto são sintetizados nos tópicos que seguem:

☐ O robô deve ser ligado e desligado pela chave SW_1 , ou pelo módulo de comunicação MC1;

☐ O LED L_1 , no caso deste projeto será o de cor vermelha, deverá permanecer ligado enquanto o robô estiver ligado;

☐ Nos pontos de coleta(p_1, \dots, p_4) o robô deve parar e coletar informações de temperatura, nível de gás, luminosidade e movimentação e enviar via MC₁ para a estação base. No caso da coleta de nível de gás e de movimentação, a coleta deve ser realizada em 3 direções diferentes. No caso desse projeto, as direções foram determinadas como sendo 0°, 90° e 180°. A movimentação desses dois sensores nestas direções é feita pela ação do servo-motor M_s . Enquanto os sensores estão coletando a informação, o LED L_2 , que para este projeto tem cor verde, deverá permanecer ligado;

☐ O robô se movimenta com velocidades diferentes entre os pontos de coleta. Essas velocidades são sintetizadas na Tabela 1.

Tabela 1 - Velocidade linear do robô para trajetos

Trajeto	Velocidade
1 - v_{01}	$0,75 \cdot v_{\max}$
2 - v_{12}	$0,50 \cdot v_{\max}$
3 - v_{23}	$0,25 \cdot v_{\max}$
4 - v_{43}	$0,50 \cdot v_{\max}$
5 - v_{45}	$1 \cdot v_{\max}$

☐ Quando chega aos pontos de coleta, o robô primeiro gira e depois faz a coleta. Esse giro é sempre no mesmo eixo, ou seja, a velocidade angular das rodas da direita e da esquerda são iguais em módulo, mas em direções diferentes. Essa velocidade angular com a qual ele gira deve ser sempre igual a $0,25 \cdot \omega_{\max}$;

☐ Na coleta, o sistema deve enviar apenas quatro níveis de gás, temperatura e luminosidade;

☐ A posição do robô deve ser sempre enviada para a estação base, pelo módulo MC₁;

☐ Ao chegar no ponto final do percurso, o robô deve dar duas voltas em torno do seu próprio eixo com ambos os leds piscando e depois desligar tudo, ou seja, sair do loop principal.

2. MODELO ESQUEMÁTICO E MODELO NO TINKERCAD

O modelo esquemático do robô que foi implementado e seus componentes é mostrado na Figura 2.

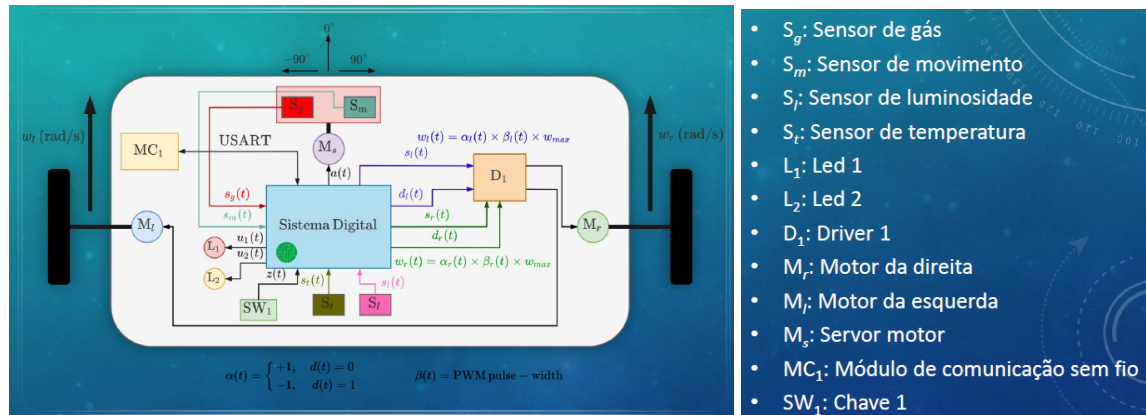


Figura 2 - Modelo esquemático do robô

A partir deste esquema, o robô foi então implementado no Tinkercad, conforme mostrado na Figura 3.

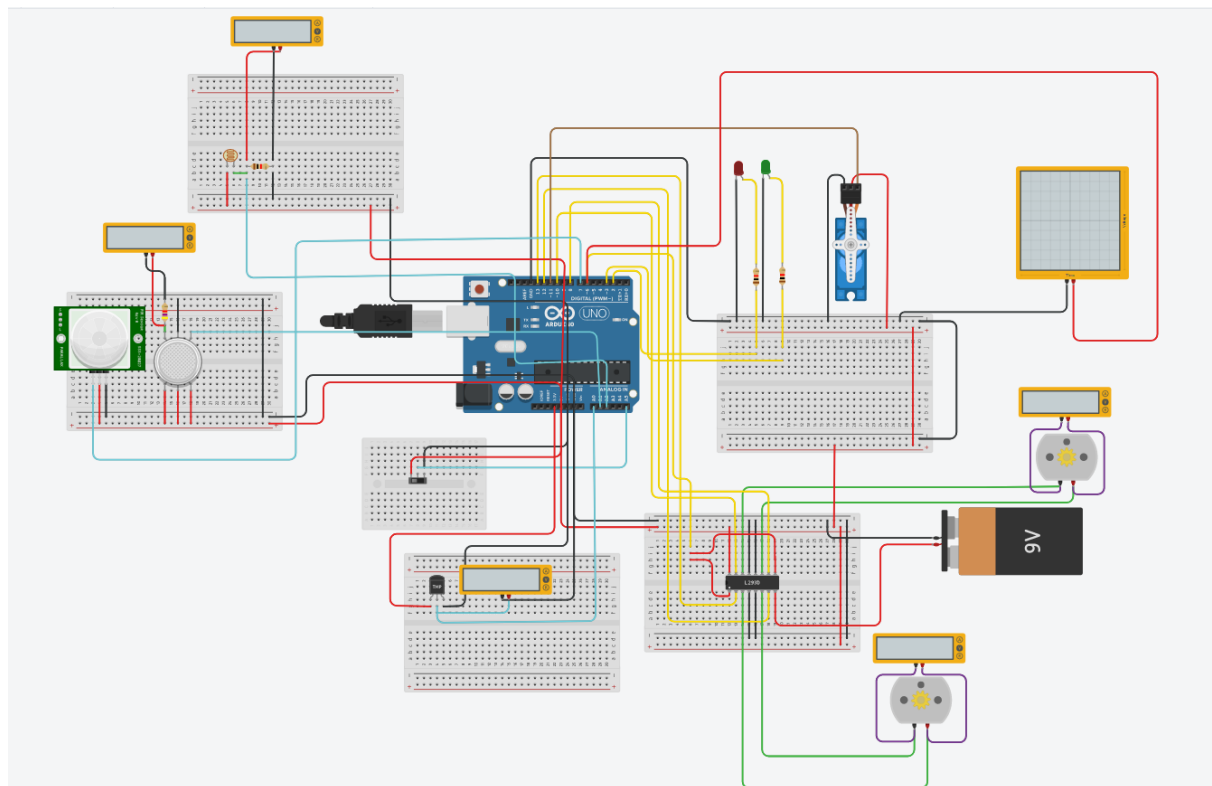




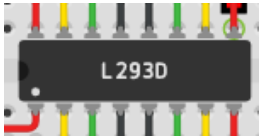

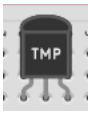

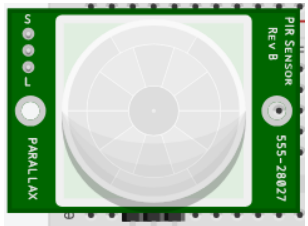
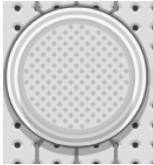
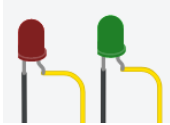
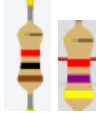

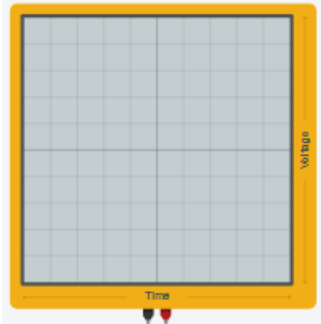


Figura 3 - Modelo implementado no Tinkercad

O principal componente utilizado na construção do robô foi o microcontrolador Atmega238p, que está contido no kit do Arduino Uno R3. Os demais componentes são mostrados na Tabela 2.

Tabela 2 - Componentes utilizados

Componente	Representação
Motores DC	
Servo motor	
Protoboards	
Bateria de 9V	
Driver ponte H modelo L293D	
Chave seletora	
Sensor de temperatura	
Fotoreistor (sensor de luminosidade)	
Sensor de movimento - PIR	

Sensor de gás	
LEDS	
Resistores (1k, 4,72k,	
Multímetros	
Osciloscópio	

O mapeamento das portas tanto do arduino quanto do MCU é sintetizado na Tabela 3.

Tabela 3 - Mapeamento das portas utilizadas

Porta no Atmega328p	Porta no arduino	Sinal	Tipo
PC0	A0	Sensor de temperatura	Entrada
PC1	A1	Sensor de gás	Entrada
PC2	A2	Sensor de luminosidade	Entrada
PC5	A5	Chave seletora	Entrada
PD2	D2	LED ₁ (vermelho)	Saída
PD3	D3	LED ₂ (verde)	Saída
PD6	D6	PWM motor DC	Saída
PD7	D7	Sensor de movimento	Entrada

PB0	D8	Input 3 ponte H (pino 10) - roda esquerda	Saída
PB2	D10	Input 4 ponte H (pino 15) - roda esquerda	Saída
PB3	D11	PWM servo-motor	Saída
PB4	D12	Input 2 ponte H (pino 7) - roda direita	Saída
PB5	D13	Input 1 ponte H (pino 2) - roda direita	Saída

Visualmente, as portas (e mapeamentos entre Atmega328p e arduino) são mostrados na Figura 4.

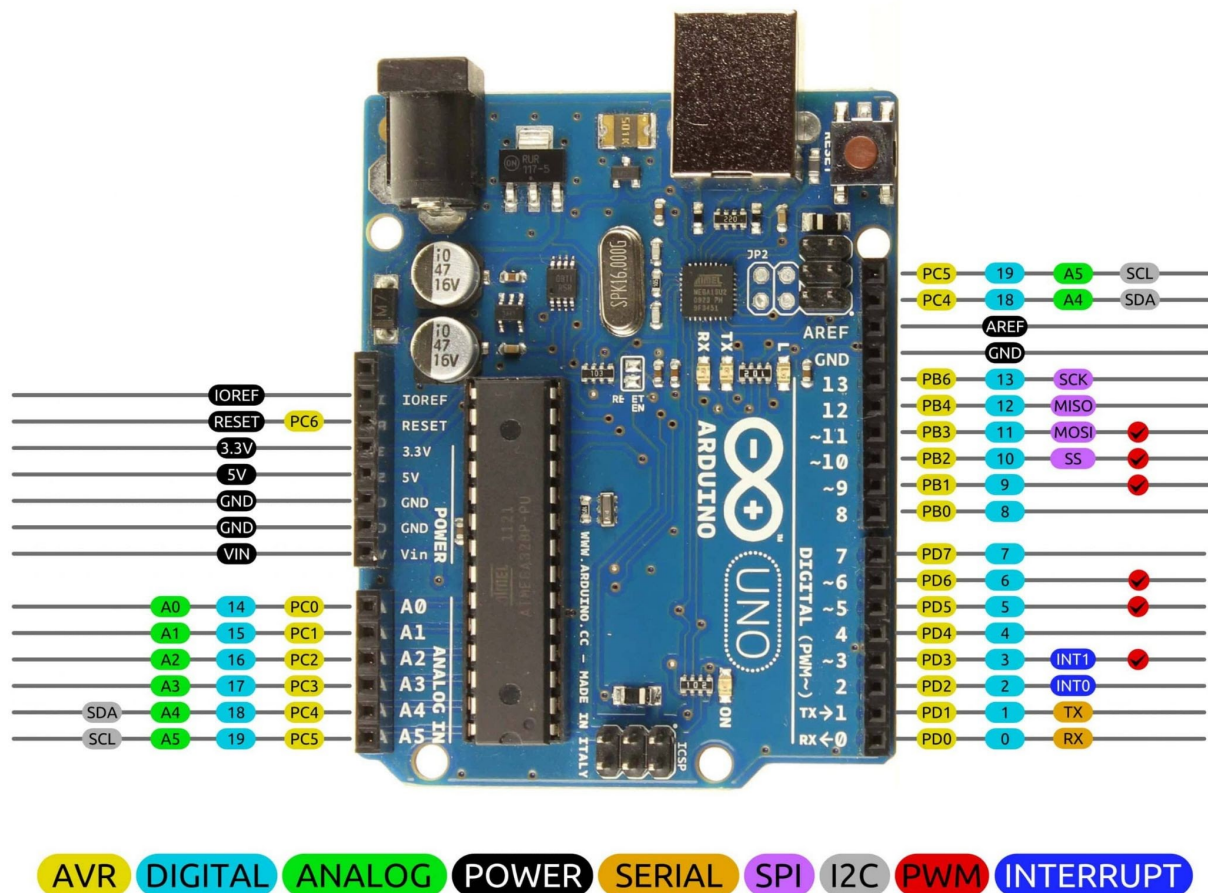


Figura 4 - Pinos arduino Uno R3 e mapeamento no Atmega328p

3. METODOLOGIA

Nessa sessão será descrito os detalhes de implementação do robô detector de gás.

3.1 - O robô

Neste projeto, o robô possui os importantes parâmetros: coordenada x, coordenada y, ângulo de orientação, em metros e graus, respectivamente. Além disso, também foram adicionados dois parâmetros adicionais: sentido de giro e sentido robô. O sentido de giro auxilia na orientação do robô na hora de fazer o giro para mudar de direção de trajetória. É uma variável inteira que possui a seguinte codificação: 0 indica que o robô está parado, 1 indica que o robô gira no sentido anti-horário, 2 indica que o robô gira no sentido horário. Já a variável de sentido robô auxilia no deslocamento do robô. Também é uma variável inteira que possui a seguinte codificação: 0 indica que o robô está parado, 1 indica movimentação para frente e 2 indicaria movimentação para trás. Entretanto, para este projeto não foi utilizada a parte de movimentação para trás. Inicializamos o robô considerando que ele está posicionado na origem e parado. As funções relacionadas à criação do robô são mostradas na Figura 5.

```
struct Robo{
    float x;
    float y;
    float angle;//em graus
    int sentido_giro;// 0 = parado, 1= antihorario, 2= horario
    int sentido_roboto;// 0 = parado, 1= pra frente, 2= pra tras
};

struct Robo criar_roboto(float _x, float _y, float _angle){
    Robo robo;
    robo.x = _x;
    robo.y = _y;
    robo.angle = _angle;
    robo.sentido_giro = 0;
    robo.sentido_roboto = 0;
    return robo;
}

Robo robo = criar_roboto(0,0,0);
```

Figura 5 - Criação do robô

Uma modificação que foi feita em relação às especificações iniciais do projeto foi que, ao invés de utilizar a orientação 0°, 90°, -90° (conforme mostrado na Figura 1), optou-se por trabalhar com ângulos conforme orientação do ciclo trigonométrico, partindo do 0°

igualado ao eixo das abscissas numa representação cartesiana 2D, até 180° para o semicírculo superior, e entre -180° e 0° para o semicírculo inferior.

O robô é movimentado através do movimento das rodas, que vai ser feito pela alimentação dos dois motores de corrente contínua. Para fazer o acionamento dos motores, foi gerado um sinal PWM configurado a partir do arduino, que é alimentado a um dispositivo de ponte H. Esse circuito de ponte H é bastante utilizado para o acionamento de motores, pois ele permite a variação do sentido da corrente em uma determinada carga. Neste projeto, foi utilizado o contador 0, de 8 bits para gerar o sinal PWM usado no acionamento do motor. Esse sinal tem frequência de 250 kHz, valor de contagem é o máximo $2^8 - 1 = 255$ e será alimentado as entradas de Enable (enable 1,2 e enable 3,4) da ponte H do dispositivo L293D. A configuração desse PWM é mostrada na Figura 6.

```
void init_pwm8bits(){
    TCCR0A = 0b10000011;
    TCCR0B = 0b00000011;
}
```

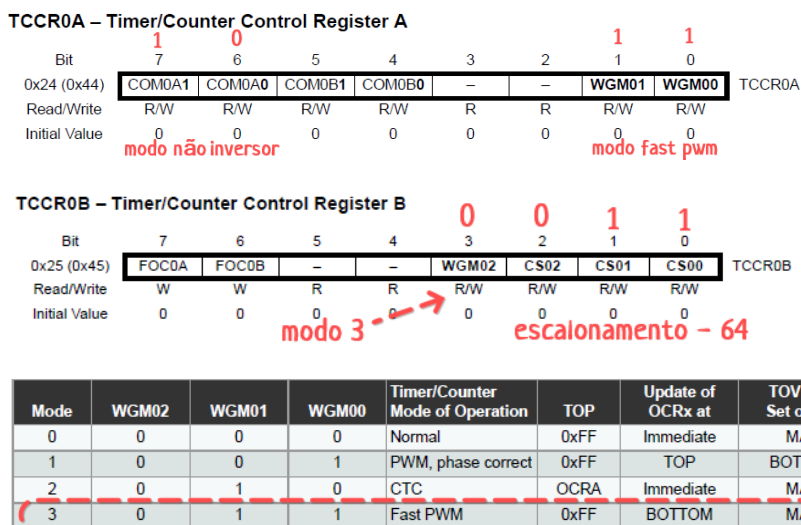


Figura 6 - Configuração do PWM para acionamento do motor

Cada motor DC representa uma roda, como o robô tem duas rodas, foi configurado dois motores, que recebem o output da ponte H para serem acionados. A ponte H é alimentada por uma bateria de 9v. A alternância de acionamento do motor (para ambos os motores), no sentido horário e anti horário, é feita a partir da seleção de nível lógico baixo/alto para as saídas do arduino que vão ser inputs da ponte H, conforme mostrado na função mostrada na Figura 7.

```
void spin_motor1_clockwise(){//combinacao 10
    PORTB |= 0b00100000;//saida PB5(D13) é setado em alta ->input1
    PORTB &= 0b11101111;//saida PB4(D12) é setado em baixa->input2
}
```

```

void spin_motor1_counterclockwise() { //combinacao 01
    PORTB |= 0b00010000; //saida PB4(D12) é setado em alta->input2
    PORTB &= 0b11011111; //saida PB5(D13) é setado em baixa->input1
}

void spin_motor2_clockwise() { //combinacao 10
    PORTB |= 0b00000001; //saida PB0(D8) é setado em alta->input3 d
    PORTB &= 0b11111011; //saida PB2(D10) é setado em baixa->input4
}

void spin_motor2_counterclockwise() { //combinacao 01
    PORTB |= 0b00000100; //saida PB2(D10) é setado em alta->input4
    PORTB &= 0b11111110; //saida PB0(D8) é setado em baixa->input3
}

```

Figura 7 - Configuração dos pinos para acionamento dos motores

Outros parâmetros interessantes referentes ao robô são: raio da roda igual a 0,05 m (50 cm) e a distância da roda ao centro do eixo de rotação do robô é de 0,075 m (7,5 cm).

3.2 - O trajeto

Para definir o trajeto a ser percorrido pelo robô, considerou-se a criação de estruturas para denotar os pontos de coleta. Esses pontos vão ser passados como objetivo na função relativa a movimentação do robô. Os pontos de coletas apresentam os seguintes parâmetros: coordenada x, coordenada y. As funções relacionadas aos pontos de coleta são mostradas na Figura 8.

```

struct Ponto_coleta{
    float x;
    float y;
};

struct Ponto_coleta criar_ponto_coleta(float _x, float _y){
    Ponto_coleta ponto;
    ponto.x = _x;
    ponto.y = _y;
    return ponto;
}

```

Figura 8 - Criação do ponto de coleta

Para criar o trajeto base inicial solicitado no projeto, foram arbitradas as coordenadas dos pontos de coleta conforme mostrado na Figura 9. As distâncias entre os pontos são mostradas visualmente na Figura 10. Foram escolhidas distâncias pequenas devido a limitações impostas pela velocidade linear do robô. Ao utilizar velocidades muito elevadas o robô demonstrou dificuldade em estabilizar no cálculo tanto do ângulo de orientação, quanto do ponto de vista de atualização de suas coordenadas x,y.

```

Ponto_coleta p1 = criar_ponto_coleta(0, 0.5);
Ponto_coleta p2 = criar_ponto_coleta(0.5, 0.5);
Ponto_coleta p3 = criar_ponto_coleta(0.5, 1);
Ponto_coleta p4 = criar_ponto_coleta(1, 1);
Ponto_coleta p5 = criar_ponto_coleta(1, 0);

```

Figura 9 - Criação do ponto de coleta do trajeto base

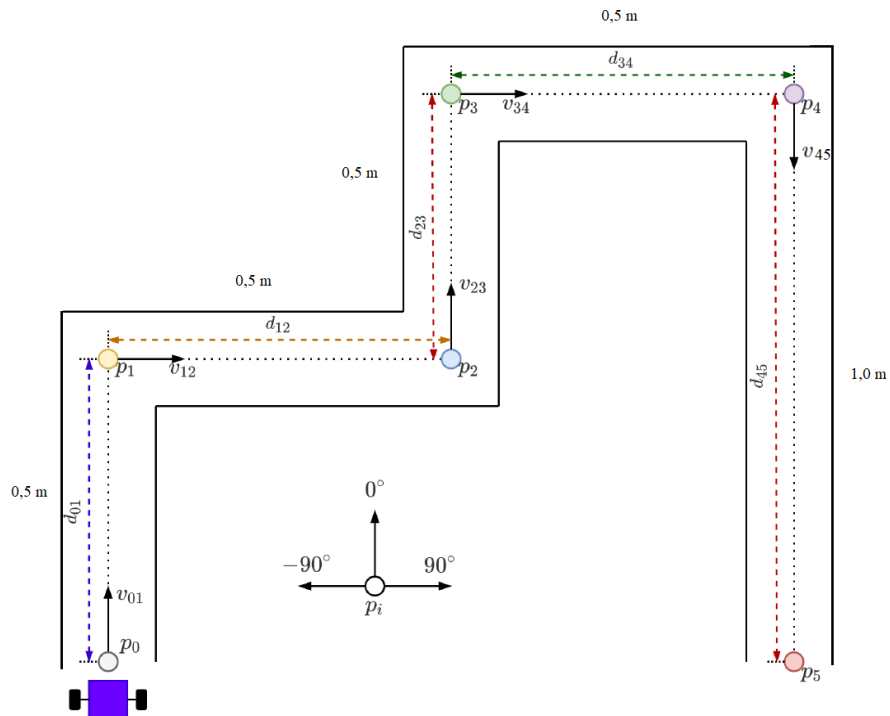


Figura 9 - Trajeto base

3.3 - O funcionamento do robô

O robô pode ser ligado de duas formas: através da chave seletora ou através da digitação das letras “I” ou “L” no teclado (implementado através do módulo de comunicação USART). O robô só funcionará enquanto o resultado da função que checa o nível lógico da porta de acionamento for verdadeiro. Essa função é mostrada na Figura 10.

```

bool is_on(){
    int value = (PINC & 0b00100000);
    if (value == 32){
        return true;
    }else{
        return false;
    }
}

```

Figura 10 - Função de verificação se robô está ligado

Ao ligar o robô, o led vermelho acende (e permanece aceso enquanto estiver ligado) e os motores começam a girar. As funções de acionamento dos motores e de acender o led são mostradas na Figura 11.

```
void set_motor1(bool on, bool forward){
    if(on == true){
        if(forward == true){
            spin_motor1_clockwise();
        }else{
            spin_motor1_counterclockwise();
        }
    }else{
        stop_motor1();
    }
}

void set_motor2(bool on, bool forward){
    if(on == true){
        if(forward == true){
            spin_motor2_clockwise();
        }else{
            spin_motor2_counterclockwise();
        }
    }else{
        stop_motor2();
    }
}

void set_red_led(bool status){
    if (status == true){
        PORTD |= 0b00000100;
    }else{
        PORTD &= 0b11111011;
    }
}
```

Figura 11 - Funções de acionar motor e ligar led vermelho

O robô inicia seu estado inicial com a seguinte orientação: coordenadas x e y na origem (0,0) e ângulo igual a zero, também na origem, conforme indicado na Figura 11.

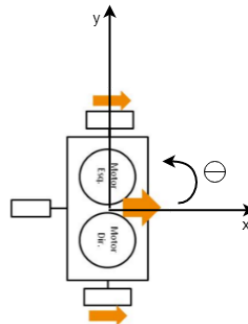


Figura 11 - Orientação inicial do robô

Utiliza-se uma variável global de controle, para distinguir o estado que o robô se encontra. Se o estado do robô é 0, indica que ele está parado. Quando o robô inicia o movimento de rotação em torno de seu eixo, para alinhar-se com o objetivo, esse estado é o 1. Uma vez alinhado, o robô inicia o movimento de translação em direção ao objetivo, esse estado é o estado 3. Uma vez que o robô atingiu o objetivo, ele entra no estado 2, que corresponde ao estado de coleta dos dados dos sensores. Por fim os estados 4 e 5 correspondem, respectivamente, ao ponto de coleta final (criado separadamente do estado 2 porque neste último ponto, é necessário preparar o cenário para o estado de finalização 5, então aqui é necessário zerar a variável que acumula a rotação feita pelo robô, para que ele precisamente dê as duas voltas em torno de si mesmo no estado de finalização) e o estado de

finalização 5 no qual o robô rotaciona em torno de seu eixo duas vezes ($2 \times 360 = 720$ graus) enquanto os leds vermelho e verde se alternam a cada 1 segundo, aproximadamente.

No giro do robô (para alinhar-se com o ponto objetivo desejado), os motores terão a mesma velocidade de rotação, porém sentido de giro distintos. A representação dos motores enquanto há o giro, juntamente com a função de giro são mostradas na Figura 12. Nessa função, inicialmente o sinal PWM que alimenta o motor é configurado, setando o valor de contagem do registrador OCR0A para no caso 25% de duty cycle, que faz parte dos requisitos do projeto.

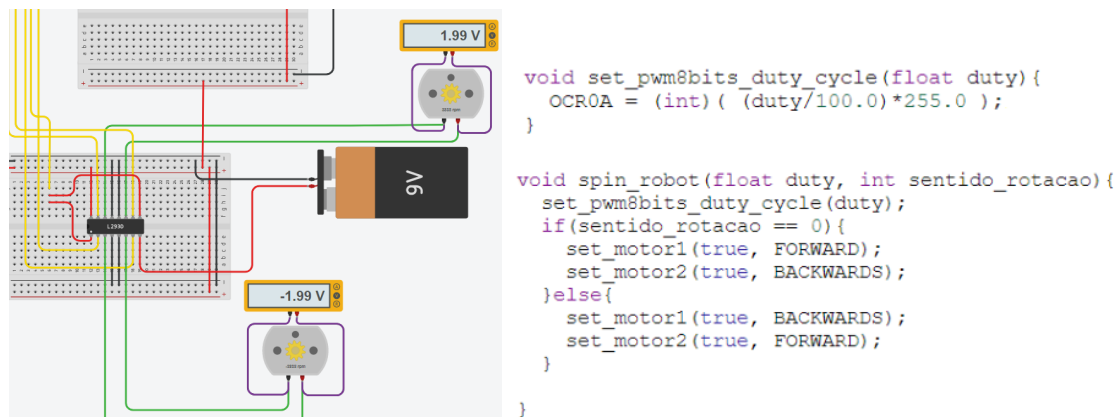


Figura 12 - Giro do robô

Na movimentação para frente, ambos os motores vão ser configurados com o mesmo sentido de rotação (sentido horário, por exemplo). A Figura 13 ilustra o caso de movimentação para frente, com sua respectiva função de movimento. Percebe-se que a velocidade com a qual o robô se deslocará é função do duty cycle definido no sinal PWM (nesta figura foi mostrado para um duty cycle de 75 %).

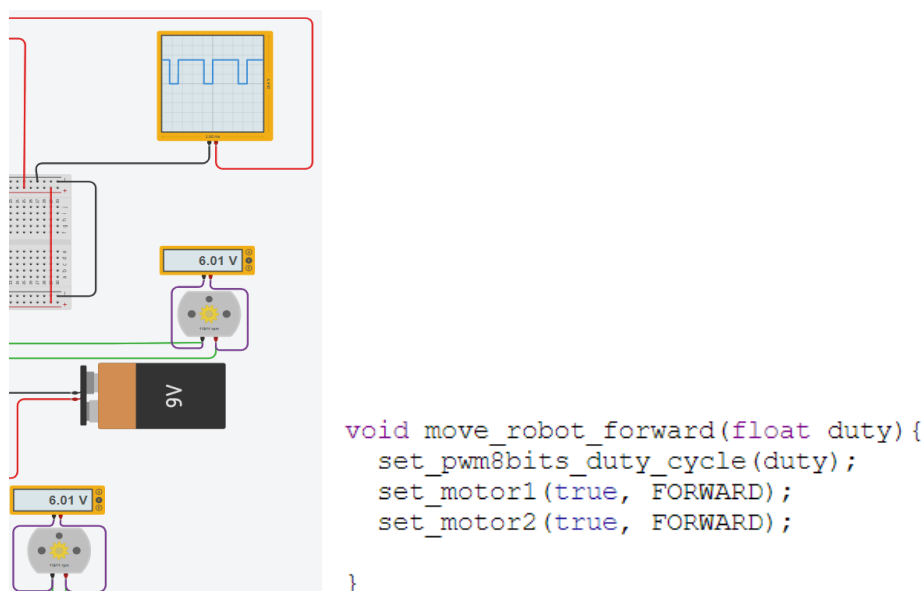


Figura 13 - Movimento do robô para frente (75% de duty cycle)

Quando está próximo de atingir o ponto de coleta, foi considerada uma redução na velocidade do robô (reduziu-se o duty cycle para 10 %, simulando uma espécie de freio) para que ele consiga atingir o ponto de coleta de maneira mais estabilizada. Ao chegar no ponto de coleta, o robô para, o led verde acende e inicia-se o processo de leitura dos sensores: temperatura, nível de luz, nível de gás e detecção de movimento. Esses dois últimos têm suas leituras feitas em 3 direções: 0°, 90° e 180°. A Figura 14 ilustra o processo de coleta de dados dos sensores.

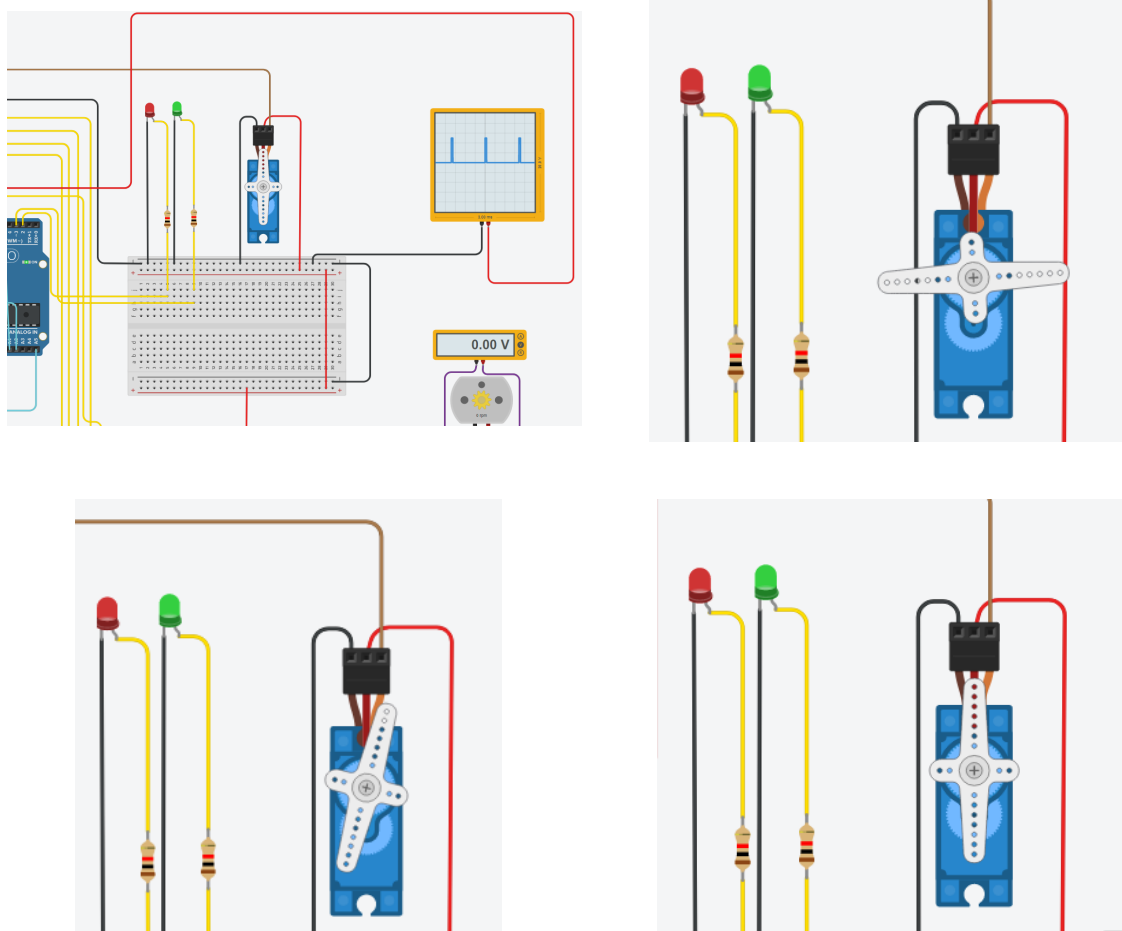


Figura 14 - Coleta de dados dos sensores

A movimentação dos sensores de nível de gás e detecção de movimento é feita utilizando-se um servo motor, que é acionado por um sinal PWM. Inicialmente deveria ser configurado utilizando o contador 1 conforme o modo 14, para que fosse possível configurar a frequência do pulso e a frequência relacionado ao duty cycle responsável pela alteração de posição informado no datasheet (ou seja, se faz necessário a presença de dois registradores para armazenamento de valor de contagem, que nesse modo seriam OCR1A para fazer a

variação de frequência do pulso e ICR1 para fazer a variação de frequência do duty cycle), conforme mostrado na Figura 15.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Figura 15 - Modo que deveria ser utilizado para movimento do servomotor

Entretanto, foi constatado que este modo apresenta um certo “bug”, que faz com que o tempo de simulação fique travado. Sendo assim, optou-se por utilizar o contador 2 (de 8 bits) e alterar manualmente o parâmetro de contagem armazenado em para obtenção visual da posição 90° e 180 °. As funções que fazem a configuração do PWM e alteração de posição são mostradas na Figura 16. Foi utilizado o modo não inversor, fast pwm e prescale de 1024.

```

void init_pwm8bits_servo(){
    TCCR2A = 0b10000011;
    TCCR2B = 0b00000111;
}

void set_comp_reg_timer2(int value)
{
    OCR2A = value;
}

void servo_angle_0()
{
    set_comp_reg_timer2(0);
}

void servo_angle_90()
{
    set_comp_reg_timer2(24);
}

void servo_angle_180()
{
    set_comp_reg_timer2(50);
}

```

Figura 16 - Funções de PWM servo motor e sua movimentação

Os valores lidos pelos sensores de temperatura e luminosidade são alterados arrastando-se uma slidebar. Já os detalhes da medição de nível de gás e de detecção de movimento são mostrados na Figura 17. Conforme a nuvem de gás é aproximada do sensor, ocorre a variação de tensão sob ele. Já no de movimento, conforme o movimento é detectado, um sinal digital é enviado para o microcontrolador. Nível alto significa que há movimento detectado e visualmente aparece uma região avermelhada.

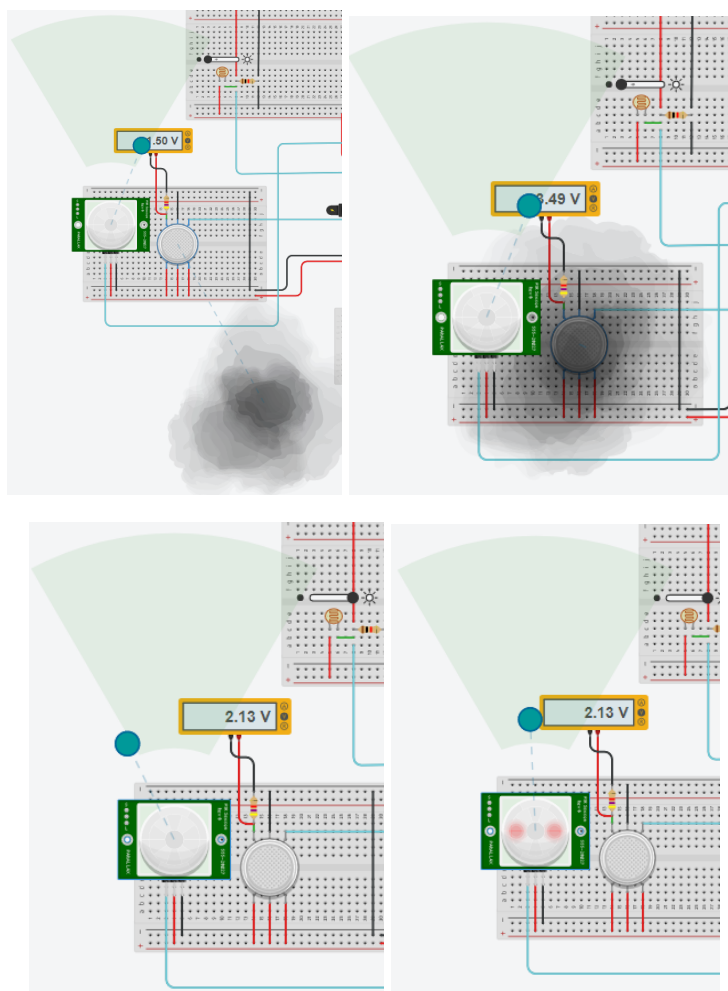


Figura 17 - Detalhes do funcionamento dos sensores de gás e movimento

A configuração dos registradores do conversor analógico-digital é mostrada na Figura

18.

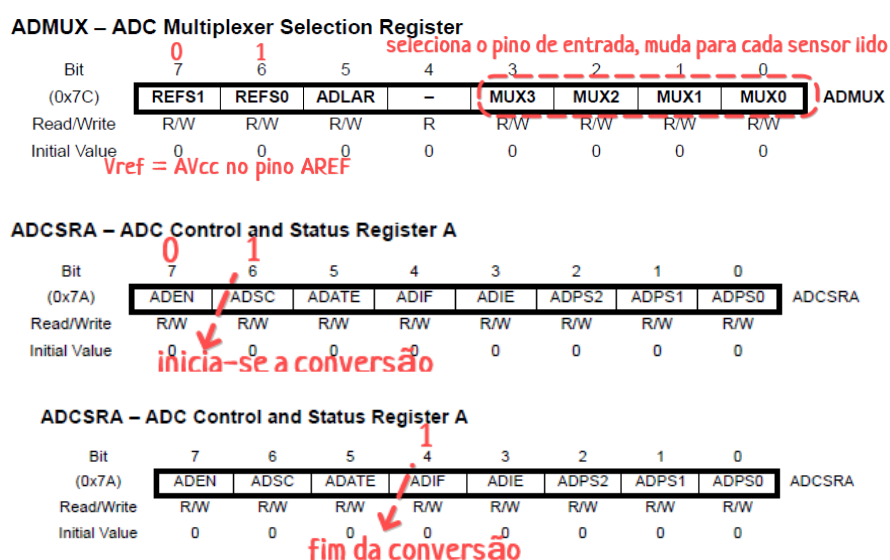


Figura 18 - Registradores utilizados durante conversão analógico-digital

A função de leitura dos sensores é mostrada na Figura 19. Para poder mostrar o resultado em níveis, conforme requisitos do projeto, há uma série de funções auxiliares utilizadas para possibilitar a conversão de variáveis inteiras, float e double para que sejam escritas como strings. Essas funções auxiliares somente serão mostradas no código, na sessão de Anexo.

```
void read_sensors(){
    set_green_led(true);

    //sensor de temperatura
    ADMUX = 0b01000000;
    ADCSRA |= 0b01000000;
    while( !(ADCSRA & 0b00010000));
    temperaturaADC = ADC;
    float step = 5.0/1023.0;
    float tempV = step * temperaturaADC;
    tempC = ((tempV-0.5)/10) * (1000);
    fill_temp_buffer();

    //sensor de gas
    /***** Leitura Gas INCIO *****/
    ADMUX = 0b01000001;

    servo_angle_0();
    _delay_ms(1000);
    ADCSRA |= 0b01000000;
    while( !(ADCSRA & 0b00010000));
    gasADC = ADC;
    n_gas_0 = (gasADC/1023)*100;

    //sensor de movimento - em 0 graus
    if((PIND & 0b10000000) == 128){
        println("Movimento detectado em 0 graus");
    }

    servo_angle_90();
    _delay_ms(1000);
    ADCSRA |= 0b01000000;
    while( !(ADCSRA & 0b00010000));
    gasADC = ADC;
    n_gas_90 = (gasADC/1023)*100;

    //sensor de movimento - em 90 graus
    if((PIND & 0b10000000) == 128){
        println("Movimento detectado em 90 graus");
    }

    servo_angle_180();
    _delay_ms(1000);
    ADCSRA |= 0b01000000;
    while( !(ADCSRA & 0b00010000));
    gasADC = ADC;
    n_gas_180 = (gasADC/1023)*100;
    fill_gases();

    //sensor de movimento - em 180 graus
    if((PIND & 0b10000000) == 128){
        println("Movimento detectado em 180");
    }

    servo_angle_0();
    _delay_ms(1000);
    /***** Leitura Gas FIM *****/

    //sensor de luminosidade
    ADMUX = 0b01000010;
    ADCSRA |= 0b01000000;
    while( !(ADCSRA & 0b00010000));
    luzADC = ADC;
    n_luz = (luzADC/1023)*100;

    fill_luz();

    set_green_led(false);
    print_medicoes();
}

}
```

Figura 19 - Função de leitura dos sensores

Uma vez que os dados são coletados, eles são enviados para a estação base utilizando o módulo de comunicação USART. Esses valores são escritos no monitor serial, conforme mostrado na Figura 20.

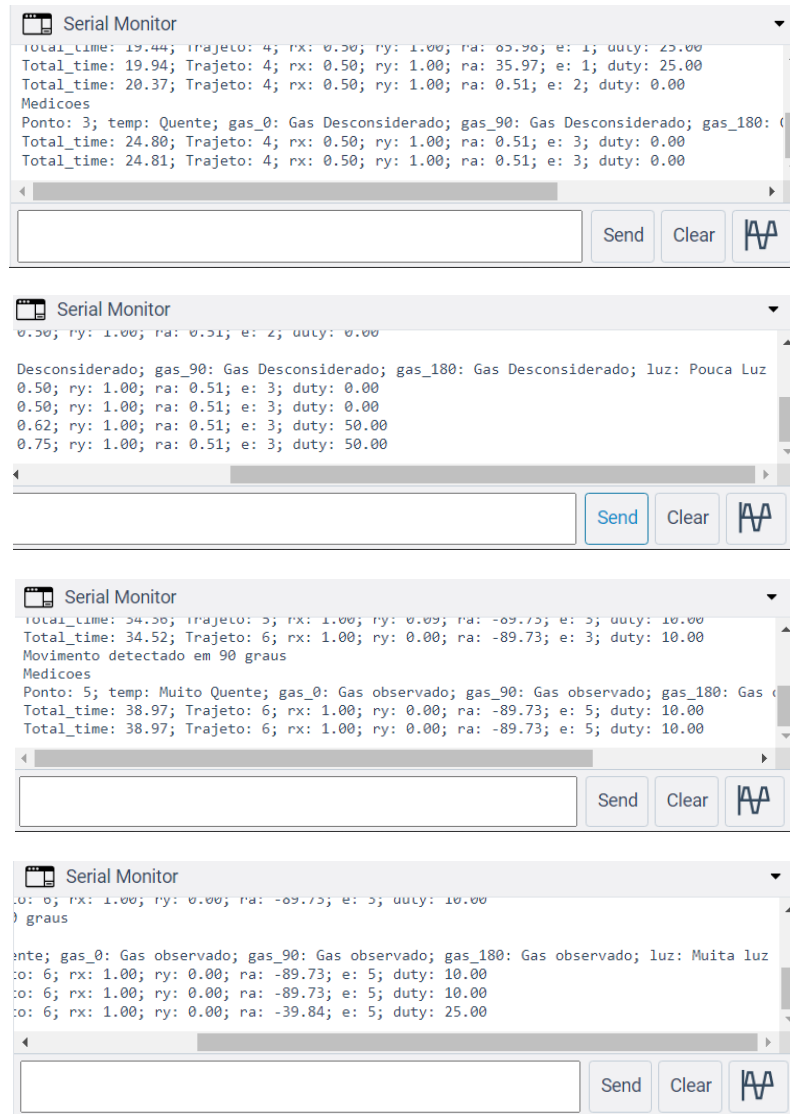


Figura 20 - Exemplos de leitura dos sensores

Uma vez que a leitura é concluída o LED verde é apagado e inicia-se novamente as etapas de movimentação do robô. A lógica referente a movimentação do robô é feita no loop while, e o cálculo da atualização de coordenadas e ângulo é tratado no interior da interrupção. A interrupção se faz necessária porque é desejado saber a posição atual do robô enquanto ele se movimenta, para que esta seja enviada sempre para a estação base. Foi configurada a interrupção com intervalos de 0,001 s (1ms), utilizando o contador 1. O valor de contagem para obter esse intervalo é calculado conforme mostrado nas equações a seguir.

$$f_c = \frac{f_{clk,atmega}}{N * (OCR1A + 1)} \longrightarrow \frac{1}{t_c} = \frac{f_{clk,atmega}}{N * (OCR1A + 1)}$$

$$OCR1A = \frac{16000000 * 0,001}{1024} - 1 \longrightarrow OCR1A \approx 15$$

A configuração desses registradores é feita conforme mostrado na Figura 21.

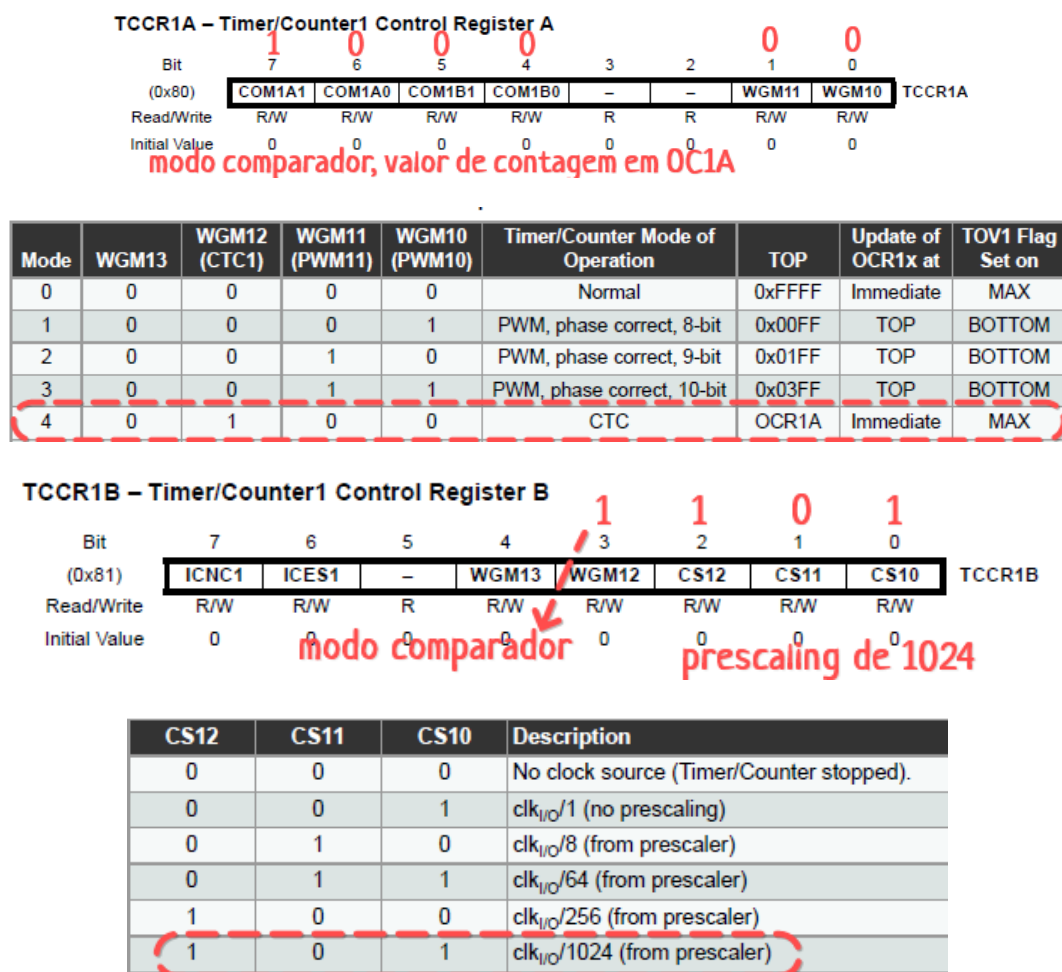
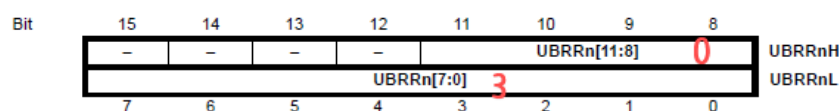


Figura 21 - Configuração dos registradores da interrupção

Por fim, para a configuração do módulo de comunicação USART, foram utilizadas as seguintes configurações de registradores, mostradas na Figura 22. O registrador UBRRnL/H é usado para ajustar a taxa de transferência de dados (foi escolhido 250kBps). Os registradores de controle utilizados são os UCSR0B e UCSR0C. No UCSR0B é ativada a interrupção, e habilitada a transmissão e recepção. No UCSR0C é configurado o tamanho do dado enviado/transmitido (8bits, para este caso).

UBRRnL and UBRRnH – USART Baud Rate Registers



Baud Rate (bps)	$f_{osc} = 16.0000\text{MHz}$			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIE _n	TXCIE _n	UDRIE _n	RXE _n	TXE _n	UCSZ _{n2}	RXB _n	TXB _n	UCSR _n B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ativa a interrupção ativa recepção ativa transmissão

UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSEL _{n1}	UMSEL _{n0}	UPM _{n1}	UPM _{n0}	USBS _n	UCSZ _{n1}	UCSZ _{n0}	UCPOL _n	UCSR _n C
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

seleciona tamanho de dado enviado/transmitido

UCSZ _{n2}	UCSZ _{n1}	UCSZ _{n0}	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit

Figura 22 - Configuração dos registradores da USART

Por fim, as funções de transmitir e enviar dados são mostradas na Figura 23.

```

void print(const char* message)
{
    int i=0;
    while(message[i] != '\0'){
        while (!(UCSR0A & (0b00100000)));
        UDR0 = message[i];
        i++;
    }
}

unsigned char recebeDado() {
    while (!(UCSR0A & (0b10000000)));
    return UDR0;
}

```

Figura 23 - Funções para recebimento/transmissão de dados utilizando USART

O registrador UDR_n é o buffer no qual são armazenados os dados transmitidos/recebidos. A transmissão/recepção de dados é controlada pelo registrador UCSR0A. Quando está ocorrendo transmissão de dados, é travada no loop while até que a flag UDRE_n seja ativada, indicando que o buffer está vazio e pronto para receber dado para transmitir. Já para receber dados, ocorre um travamento no loop while até que a flag RXC_n é

ativada, indicando que há dado não lido no buffer. Esses últimos registradores são mostrados na Figura 24.

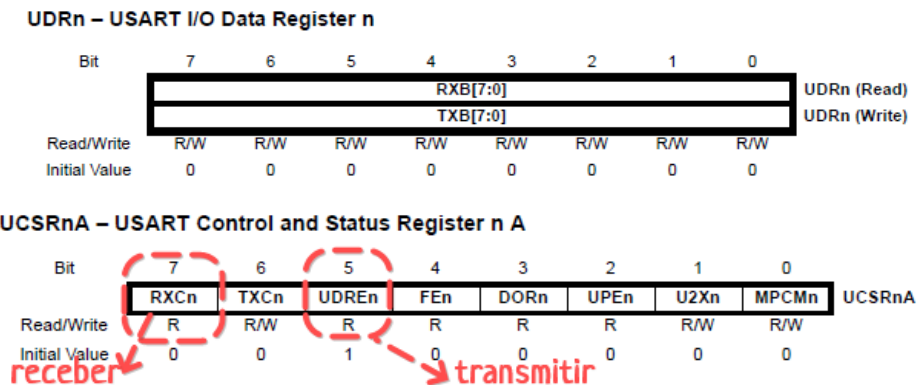


Figura 24 - Configuração dos registradores da USART

Depois de percorrer todo o trajeto, o robô faz duas rotações em torno de seu eixo piscando os leds vermelho e verde, desliga os motores, os leds e sai do loop principal. No meio da execução o robô pode ser desligado, ou pela própria chave seletora, ou pela digitação da letra “d” ou “D” (que teve sua recepção configurada utilizando o USART).

4. FUNCIONAMENTO TRAJETO BASE

Inicialmente, o robô é ligado ou pela chave seletora ou pelo comando “I” ou “L” digitado no serial monitor e recebido via módulo de comunicação USART. Ao ligar, o robô vai iniciar fazendo o giro para alinhar-se com o seu objetivo, que está a 90° da sua posição inicial, conforme já explicado nas seções anteriores. A velocidade de giro é calculada com base em 25 % da velocidade linear máxima, pois $v_{\text{linear}} = w \cdot r$, pela seguinte equação:

$$w = \frac{r(w_r - w_l)}{L}$$

No caso do giro, a velocidade das rodas são iguais em modulo, mas com sentidos contrários. Logo no numerador vai aparecer um fator multiplicativo 2, e o denominador L é a distância entre as rodas. A Figura 25 ilustra a curva do sinal PWM gerado para este movimento, com 25 % de duty cycle. Com esse duty cycle a velocidade de giro é 1,74 rad/s.

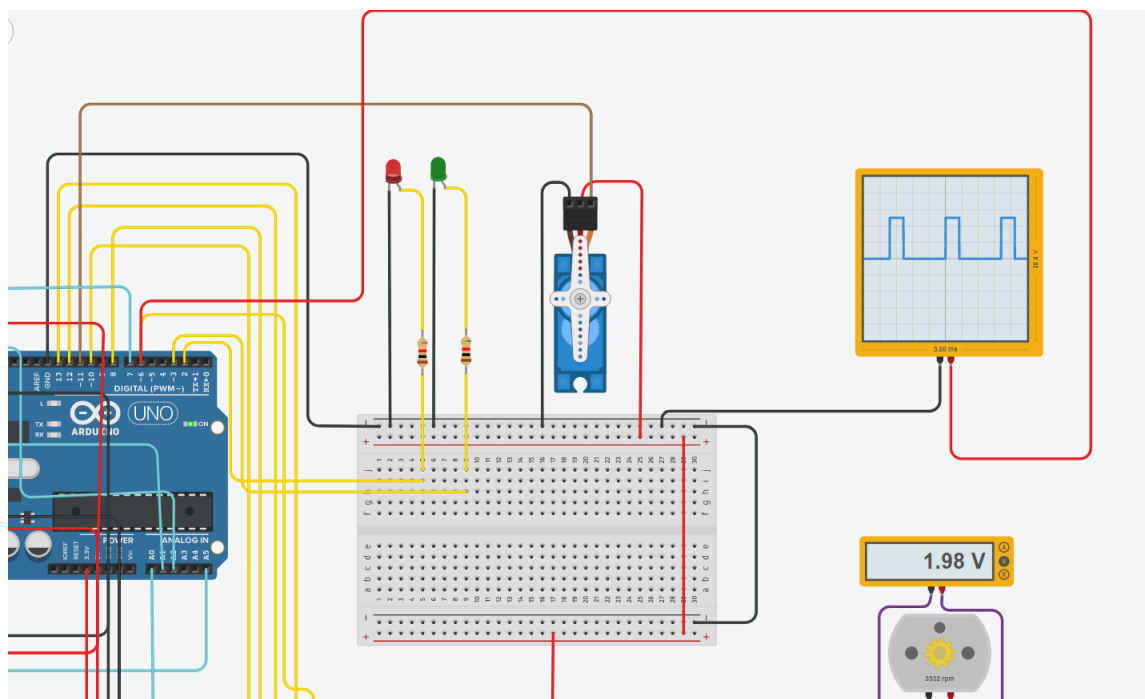


Figura 25 - Giro do robô, com 25 % de duty cycle

Em seguida, inicia-se o trajeto 1, que vai do ponto P0 até o ponto P1, com 0,5 metros. O sinal de PWM do motor é acionado para 75 % de duty cycle e a velocidade linear correspondente a esse sinal é de 0,39 m/s. A Figura 26 demonstra o movimento do robô no trajeto 1.

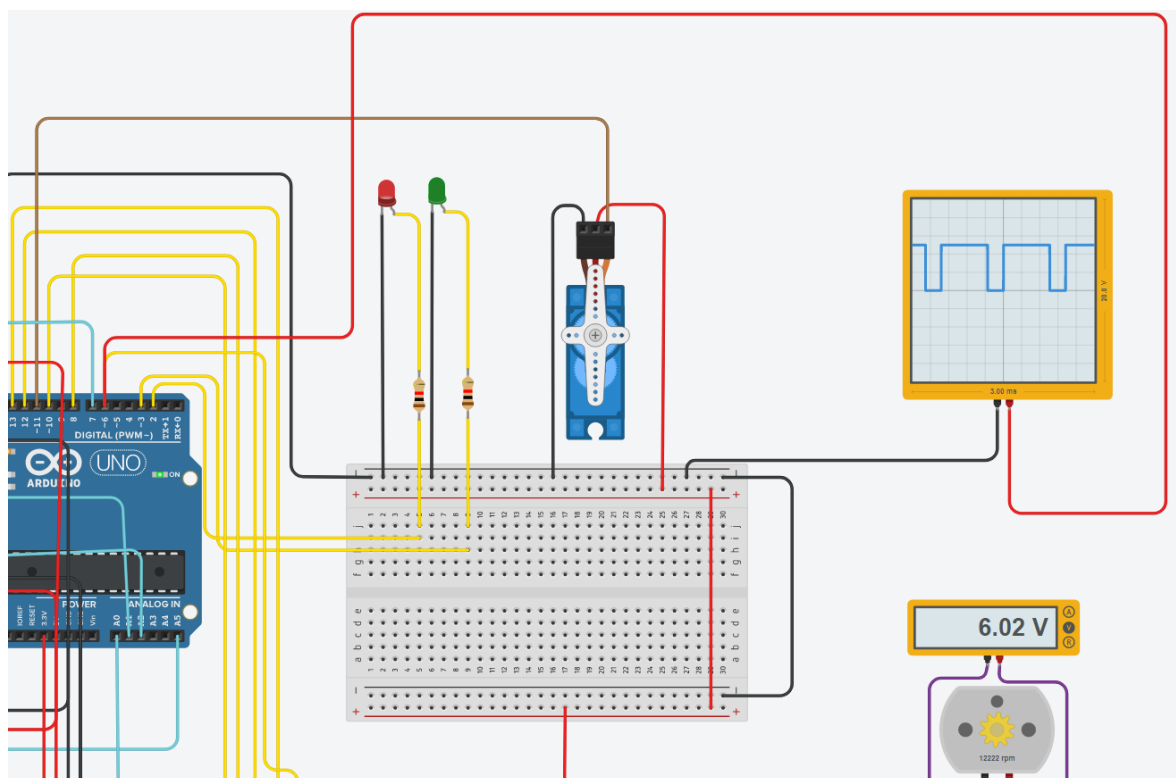


Figura 26 - Trajeto 1, com 75 % de duty cycle

Quando está próximo de atingir o ponto de coleta P1, o robô freia, por meio de uma redução no duty cycle para 10 %. Quando atinge este ponto, o robô para e rotaciona novamente, com o mesmo duty cycle de 25 % como mostrado anteriormente na Figura 25. Durante a coleta dos dados dos sensores, o led verde fica aceso e as medições são feitas nas direções já descritas anteriormente, conforme mostra a Figura 27.

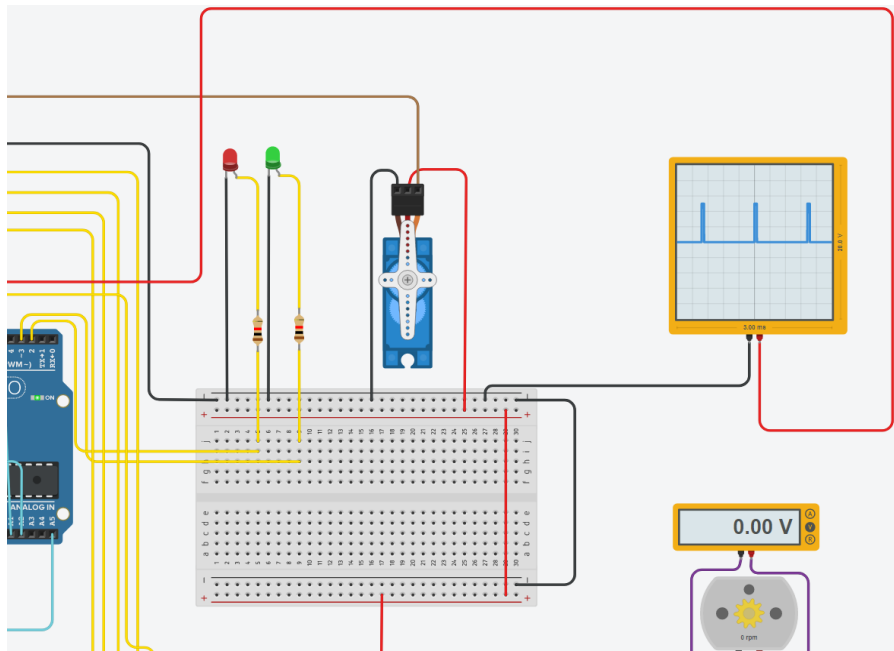


Figura 27 - Robô durante a coleta de dados dos sensores

As medições realizadas nesse ponto são mostradas na Figura 28.

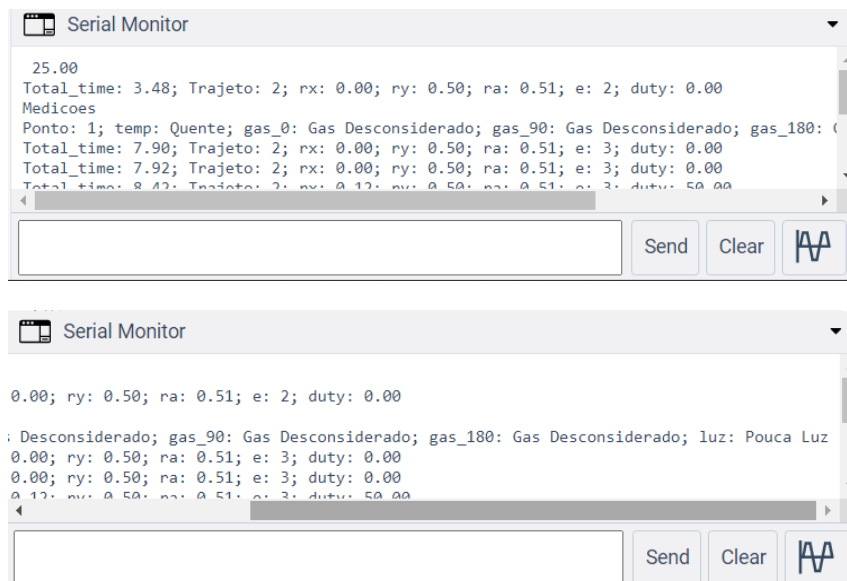


Figura 28 - Robô durante a coleta de dados dos sensores

Em seguida, inicia-se o trajeto 2, que vai do ponto P1 até o ponto P2, com 0,5 metros. O sinal de PWM do motor é acionado para 50 % de duty cycle e a velocidade linear

correspondente a esse sinal é de 0,26 m/s. A Figura 29 demonstra o movimento do robô no trajeto 2.

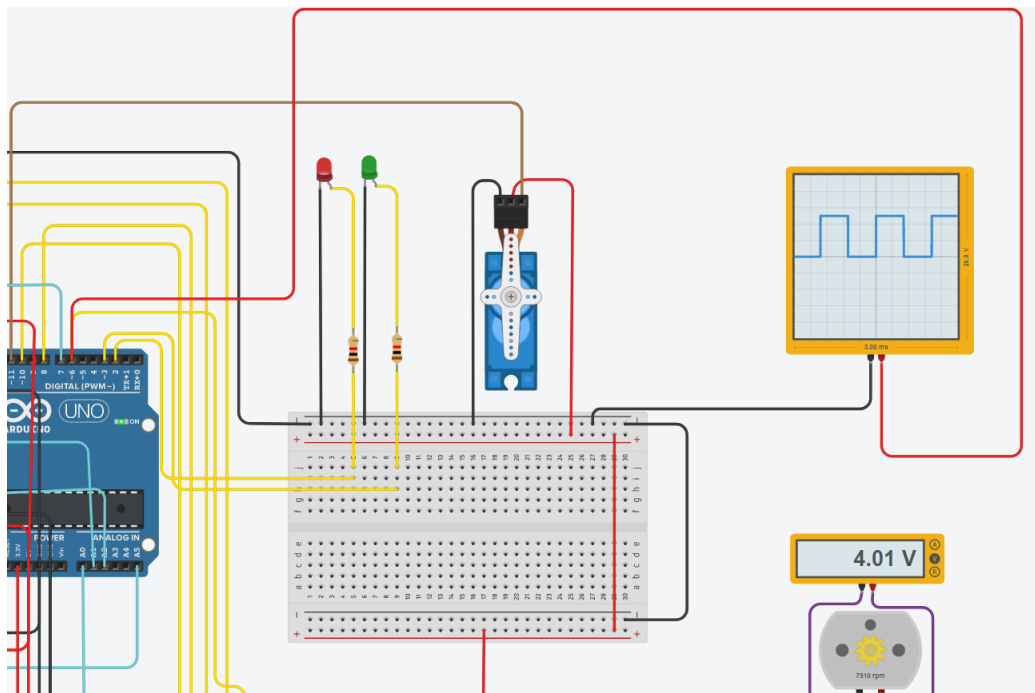


Figura 29 - Trajeto 2, com 50 % de duty cycle

Quando atinge o ponto P2, o robô para e rotaciona novamente, com o mesmo duty cycle de 25 % como mostrado anteriormente na Figura 25. Durante a coleta dos dados dos sensores, o led verde fica aceso e as medições são feitas nas direções já descritas anteriormente, conforme mostra a Figura 27. As medições feitas no ponto P2 são mostradas na Figura 30.

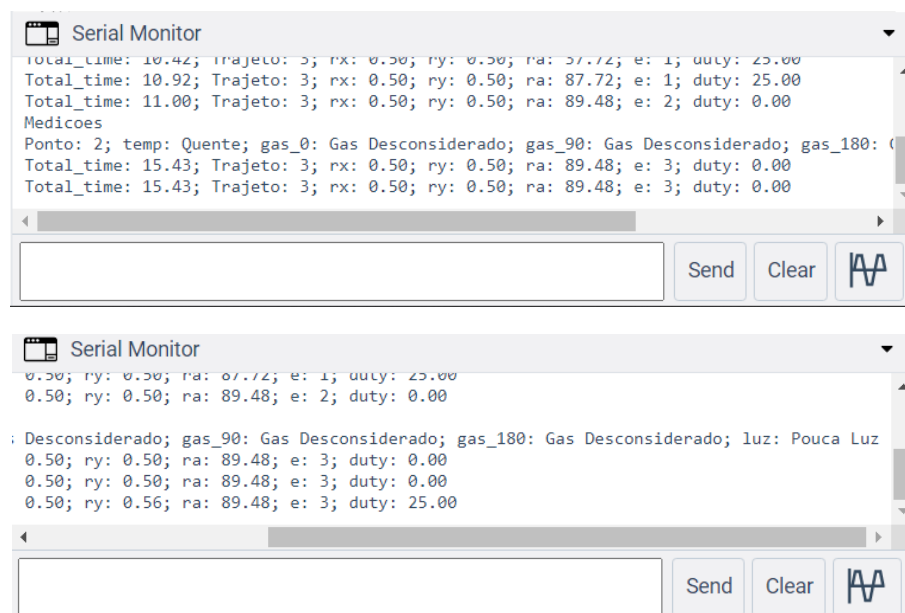


Figura 30 - Robô durante a coleta de dados dos sensores

Em seguida, inicia-se o trajeto 3, que vai do ponto P2 até o ponto P3, com 0,5 metros. O sinal de PWM do motor é acionado para 25 % de duty cycle e a velocidade linear correspondente a esse sinal é de 0,13 m/s. A Figura 31 demonstra o movimento do robô no trajeto 3.

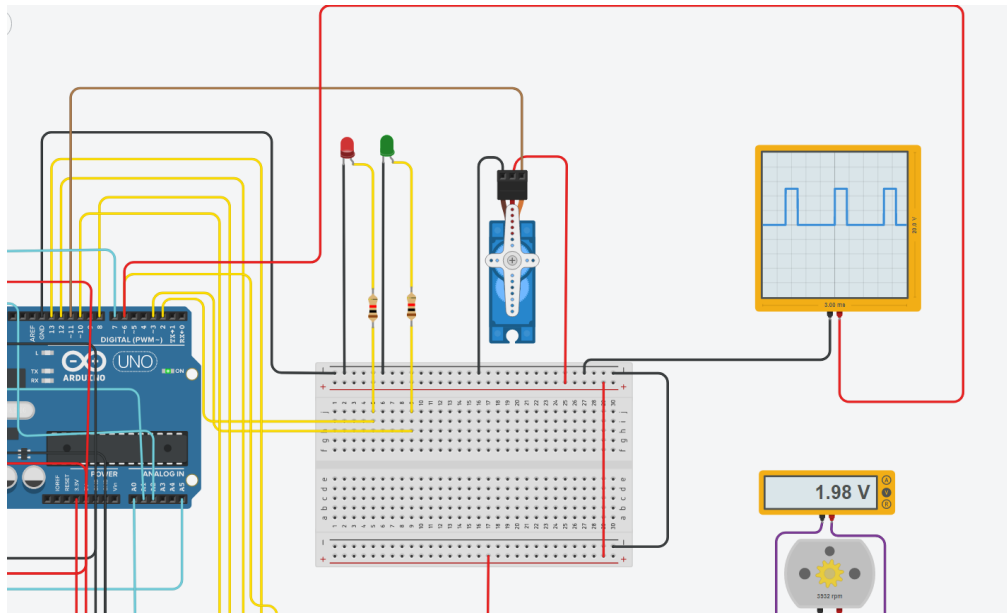


Figura 31 - Trajeto 3, com 25 % de duty cycle

Quando atinge o ponto P3, o robô para e rotaciona novamente, com o mesmo duty cycle de 25 % como mostrado anteriormente na Figura 25. Durante a coleta dos dados dos sensores, o led verde fica aceso e as medições são feitas nas direções já descritas anteriormente, conforme mostra a Figura 27. As medições feitas no ponto P3 são mostradas na Figura 32.

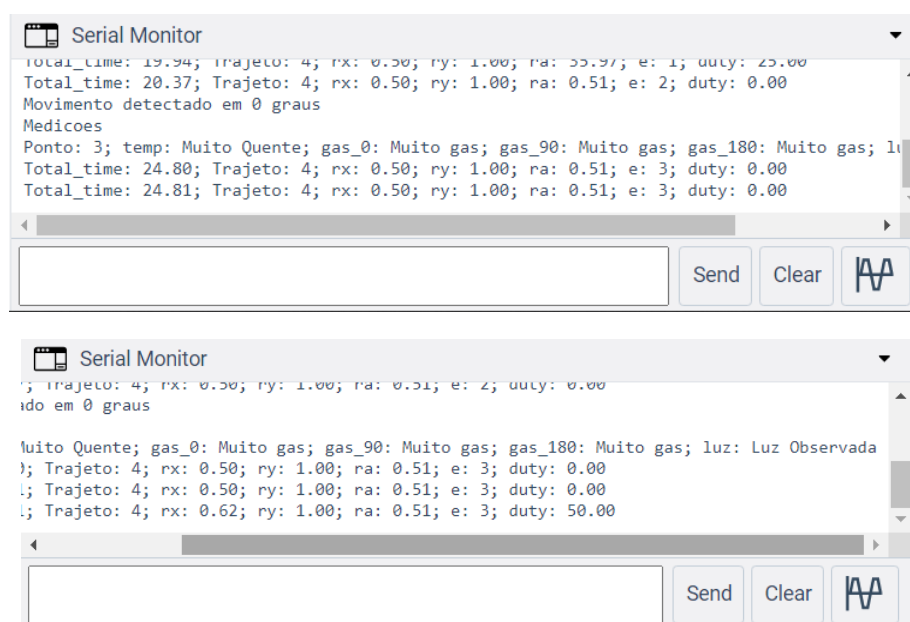


Figura 32 - Robô durante a coleta de dados dos sensores

Em seguida, inicia-se o trajeto 4, que vai do ponto P3 até o ponto P4, com 0,5 metros. O sinal de PWM do motor é acionado para 50 % de duty cycle e a velocidade linear correspondente a esse sinal é de 0,26 m/s. A Figura 33 demonstra o movimento do robô no trajeto 4.

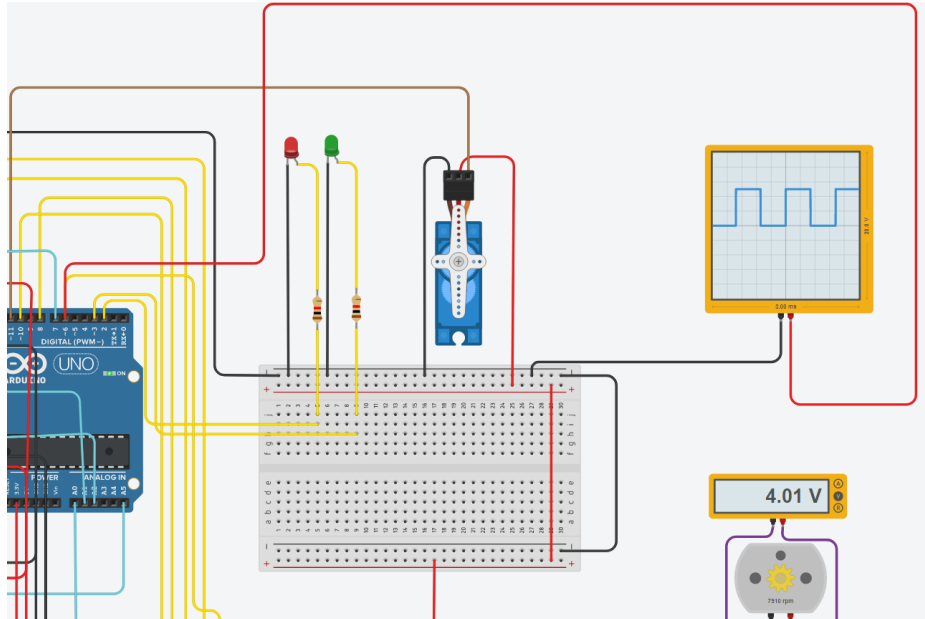


Figura 33 - Trajeto 4, com 50 % de duty cycle

Quando atinge o ponto P4, o robô para e rotaciona novamente, com o mesmo duty cycle de 25 % como mostrado anteriormente na Figura 25. Durante a coleta dos dados dos sensores, o led verde fica aceso e as medições são feitas nas direções já descritas anteriormente, conforme mostra a Figura 27. As medições feitas no ponto P4 são mostradas na Figura 34.

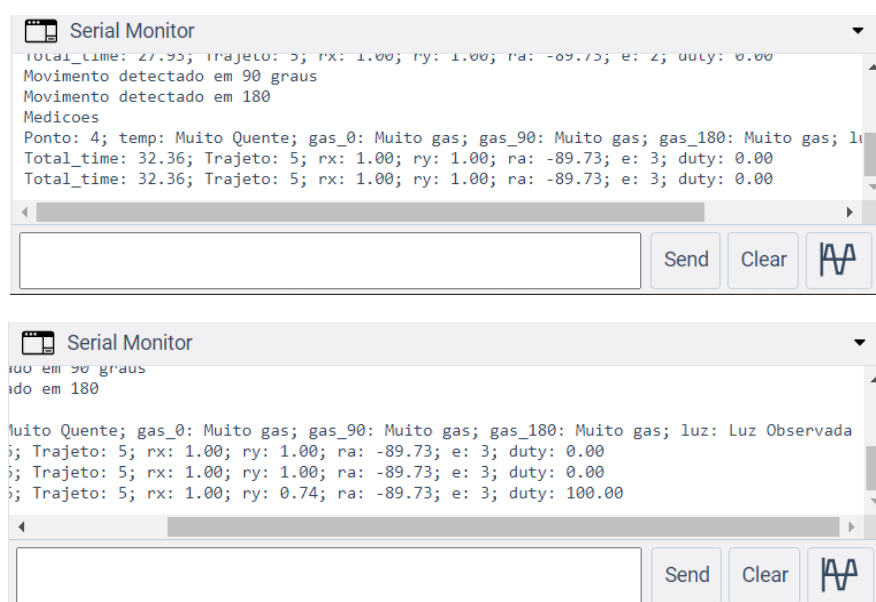


Figura 34 - Robô durante a coleta de dados dos sensores

Em seguida, inicia-se o trajeto 5, que vai do ponto P4 até o ponto P5, com 1 metro. O sinal de PWM do motor é acionado para 100 % de duty cycle e a velocidade linear correspondente a esse sinal é de 0,52 m/s. A Figura 35 demonstra o movimento do robô no trajeto 5.

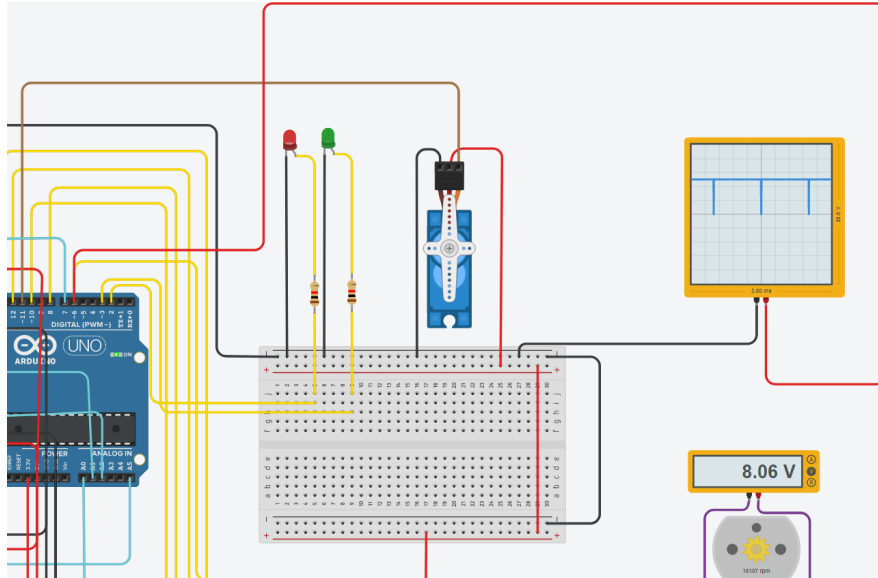


Figura 35 - Trajeto 4, com 100 % de duty cycle

Percebemos que a tensão medida no motor é de 8.06 V. Ele é alimentado por uma bateria de 9 V, o que indica que há algumas perdas até que chegue nele. Quando atinge o ponto P5, o robô para e rotaciona novamente, com o mesmo duty cycle de 25 % como mostrado anteriormente na Figura 25. Durante a coleta dos dados dos sensores, o led verde fica aceso e as medições são feitas nas direções já descritas anteriormente, conforme mostra a Figura 27. As medições feitas no ponto P5 são mostradas na Figura 36.

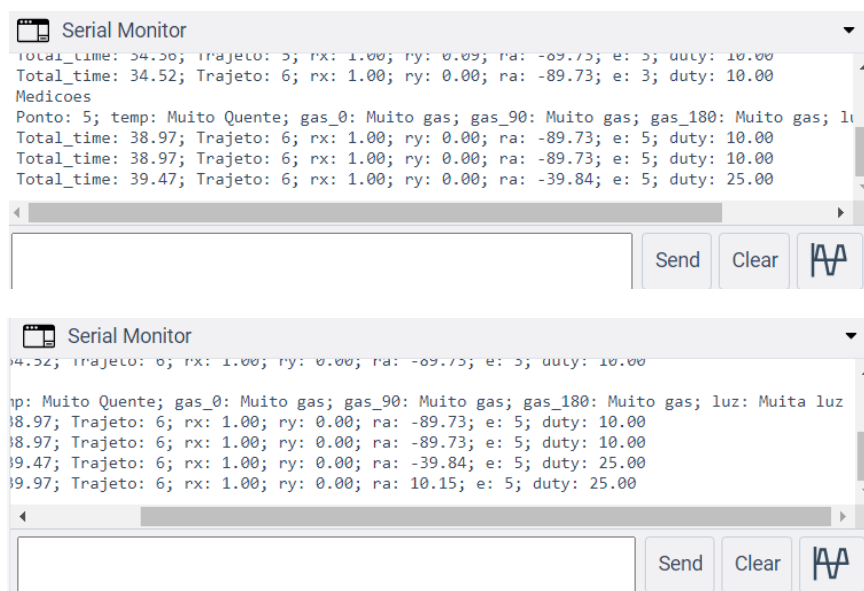


Figura 36 - Robô durante a coleta de dados dos sensores

O ponto P5 é o ponto final do trajeto, ao atingir esse ponto, após coleta de dados dos sensores ser realizada, o robô inicia um movimento de giro em torno de seu próprio eixo, também com a velocidade de 25 % fornecida pelo sinal PWM. Ao finalizar duas rotações completas (retornando então ao ângulo que ele chega ao ponto P5, nesse caso -90°), ele então desliga automaticamente. Durante o giro duplo, os leds vermelho e verde se alternam piscando, conforme mostrado na Figura 37.

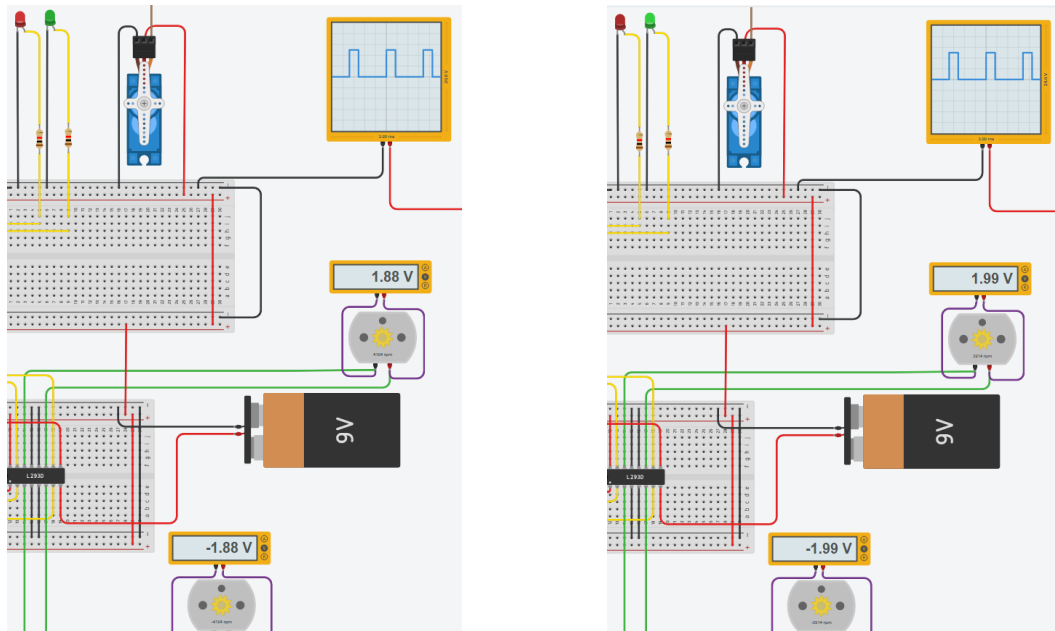


Figura 37 - Robô durante o giro duplo, com leds alternando

O percurso percorrido pelo robô é mostrado no gráfico da Figura 38.

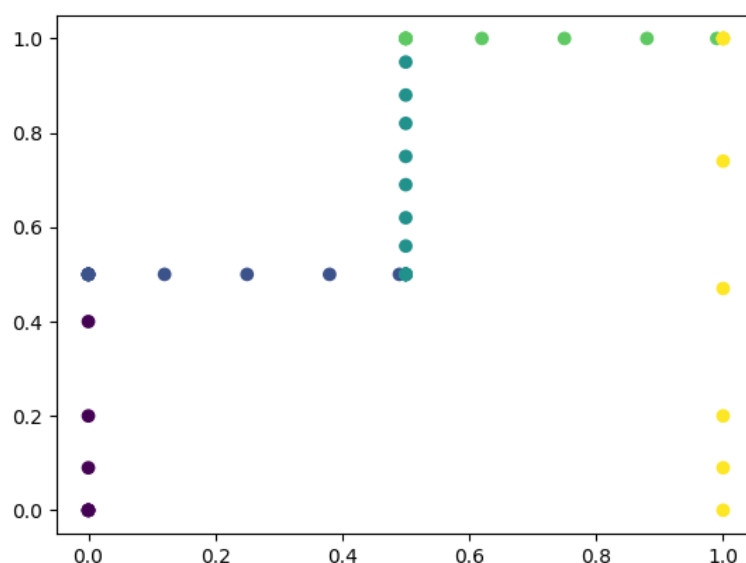


Figura 38 - Percurso trajeto base

5. TRAJETO ESTRELA

Para ilustrar como ao adicionar a orientação de ângulo do robô possibilita trajetórias mais flexíveis, foi proposto um segundo trajeto, em formato de estrela, também com 5 segmentos e pontos de coleta similares ao visto no trajeto base. As velocidades também foram consideradas as mesmas. O gráfico deste trajeto é mostrado na Figura 39.

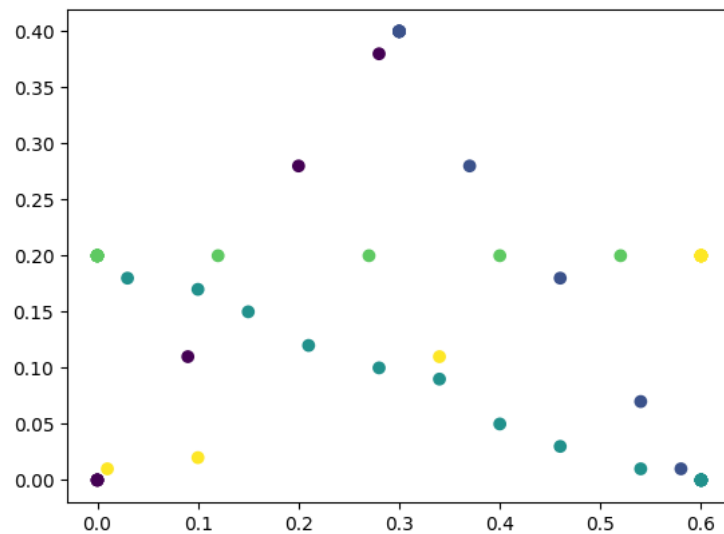


Figura 39 - Percurso trajeto estrela

6. LINK PARA PROJETO

O projeto está disponível no link:

https://www.tinkercad.com/things/9yEr8d620MV-copy-of-projeto1unidadealltrajetos/editel?sharecode=RoYrv_w517mAc-DHiYLJm0UpRYXEPwx4xgbiAFRiU6U

ANEXO

```
#include <avr/io.h>
```

```
#include <util/delay.h>
```

```
#include <math.h>
```

```
#include <avr/interrupt.h>
```

```
#define FORWARD true
```

```
#define BACKWARDS false
```

```
#define PI 3.14159265358979323846
```

```
//RAIO_RODA - m
```

```
#define RAIO_RODA 0.05
```

```
//D_RODA_CENTROIDE - m
```

```
#define D_RODA_CENTROIDE 0.075
```

```
float tensao_bateria = 4; //Volts
```

```
float perdas = 0.94; //V
```

```
float max_tensao_motor = tensao_bateria - perdas;
```

```
float temperaturaADC = 25;
```

```
bool read_temp_sensor = false;
```

```
float tempC = 25;
```

```
float gasADC = 0;
```

```
float n_gas_0 = 0;
```

```
float n_gas_90 = 0;
```

```
float n_gas_180 = 0;
```

```
float n_gas_level = 0;
```

```
float luzADC = 0;
```

```
float n_luz = 0;
```

```
float tempo_atual = 0;
```

```
float tol_ang_1 = 2;
```

```
float tol_ang_2 = 0.01;
```

```
float tol_dist_1 = 0.02;
```

```
float tol_dist_2 = 0.005;
```

```
bool motor_ligado = false;
```

```
float interruption_interval = 0.001;//segundos
```

```
double total_time = 0;//segundos
```

```
float duty_atual = 0;
```

```
float rotacao_acumulada = 0;
```

```
float vel_giro = 0;
```

```
float delta_angle = 0;
```

```
float delta_angle_graus = 0;
```

```
float vel_linear_robo = 0;
```

```
float delta_vel = 0;
```

```
float angle_robo_rad = 0;
```

```
bool toggle_leds = false;
```

```
bool timer_interruption_flag = false;
```

```
uint32_t count_print = 0;
```

```
uint32_t count_total = 0;
```

```
uint16_t count_led = 0;
```

```
char buffer_int[40];
```

```
char buffer_float[40];
```

```
char temp_string[20];
```

```
char gas0_string[20];
```

```
char gas90_string[20];
```

```
char gas180_string[20];
```

```
char luz_string[30];
```

```
unsigned char switch_usart;
```

```
/*
```

```
1 -> p0 para p1
```



```

2 -> p1 para p2
3 -> p2 para p3
4 -> p3 para p4
5 -> p4 para p5
*/

int trajeto = 1;
/*
    0 -> parado
    1 -> alinhando com ponto
    2 -> coletando dados
    3 -> movendo para ponto
    4 -> coleta ultimo ponto
    5 -> comemoracao
*/
int estado_robo =0;

int count_timer1 = 0;

struct Ponto_coleta{
    float x;
    float y;
};

struct Robo{
    float x;
    float y;
    float angle;//em graus
    int sentido_giro;// 0 = parado, 1= antihorario, 2= horario
    int sentido_robo;// 0 = parado, 1= pra frente, 2= pra tras
};

struct Ponto_coleta criar_ponto_coleta(float _x, float _y){
    Ponto_coleta ponto;

```

```

    ponto.x = _x;
    ponto.y = _y;
    return ponto;
}

```

```

struct Robo criar_robo(float _x, float _y, float _angle){
    Robo robo;
    robo.x = _x;
    robo.y = _y;
    robo.angle = _angle;
    robo.sentido_giro = 0;
    robo.sentido_robo = 0;
    return robo;
}

```

```

float calculate_angle(struct Robo robo, struct Ponto_coleta ponto){//computar o angulo para
alinhar o robo com o ponto
    float delta_y = ponto.y - robo.y;
    float delta_x = ponto.x - robo.x;
    float angle = atan2(delta_y, delta_x); //angulo em radiano
    float angle_graus = (angle*180)/PI;
    return angle_graus;
}

```

```

float calculate_distance(struct Robo robo,struct Ponto_coleta ponto){
    float delta_y = ponto.y - robo.y;
    float delta_x = ponto.x - robo.x;
    return sqrt(delta_y*delta_y + delta_x*delta_x);//distancia euclideana entre robo e o
ponto
}
/*

```

```

Ponto_coleta p1 = criar_ponto_coleta(0, 100);
Ponto_coleta p2 = criar_ponto_coleta(50, 100);
Ponto_coleta p3 = criar_ponto_coleta(50, 250);

```

```
Ponto_coleta p4 = criar_ponto_coleta(150, 250);
Ponto_coleta p5 = criar_ponto_coleta(150, 0);
*/
```

```
Ponto_coleta p1 = criar_ponto_coleta(0, 0.5);
Ponto_coleta p2 = criar_ponto_coleta(0.5, 0.5);
Ponto_coleta p3 = criar_ponto_coleta(0.5, 1);
Ponto_coleta p4 = criar_ponto_coleta(1, 1);
Ponto_coleta p5 = criar_ponto_coleta(1, 0);
```

```
Robo robo = criar_robo(0,0,0);
```

```
Ponto_coleta pegar_ponto_trajeto(int trajeto){
    if(trajeto == 1){
        return p1;
    }
    if(trajeto == 2){
        return p2;
    }
    if(trajeto == 3){
        return p3;
    }
    if(trajeto == 4){
        return p4;
    }
    if(trajeto == 5){
        return p5;
    }
}
```

```
float pegar_duty_trajeto(int trajeto){
    if(trajeto == 1){
        return 75;
    }
}
```

```
if(trajeto == 2){  
    return 50;  
}  
if(trajeto == 3){  
    return 25;  
}  
if(trajeto == 4){  
    return 50;  
}  
if(trajeto == 5){  
    return 100;  
}else{  
    return 0;  
}  
}
```

```
void set_red_led(bool status){  
    if (status == true){  
        PORTD |= 0b00000100; //liga led vermelho q ta na porta A2  
    }else{  
        PORTD &= 0b11110111; //apaga led vermelho  
    }  
}
```

```
void set_green_led(bool status){  
    if (status == true){  
        PORTD |= 0b00001000; //liga led verde q ta na porta A3  
    }else{  
        PORTD &= 0b11110111; //apaga led verde  
    }  
}
```

```

bool is_on(){
    int value = (PINC & 0b00100000); //valor de entrada q vem do switch p/ PC5 ta no reg
    PINC 2^5=32
    if (value == 32){
        return true;
    }else{
        return false;
    }
}

```

```

void spin_motor1_clockwise(){ //combinacao 10
    PORTB |= 0b00100000; //saida PB5(D13) é setado em alta -> input1 do l293D
    PORTB &= 0b11101111; //saida PB4(D12) é setado em baixa -> input2 do l293D
}

```

```

void spin_motor1_counterclockwise(){ //combinacao 01
    PORTB |= 0b00010000; //saida PB4(D12) é setado em alta -> input2 do l293D
    PORTB &= 0b11011111; //saida PB5(D13) é setado em baixa -> input1 do l293D
}

```

```

void stop_motor1(){ //combinacao 00
    PORTB &= 0b11001111; //saida PB4(D12) e PB5(D13) em baixa
}

```

```

void set_motor1(bool on, bool forward){
    if (on == true) { //so haverá movimento se o robo tiver on
        if (forward == true){
            spin_motor1_clockwise(); //gira motor1 no sentido horario
        }else{
            spin_motor1_counterclockwise(); //gira motor1 no sentido antihorario
        }
    }else{
        stop_motor1(); //nao rotaciona motor se o robo estiver off
    }
}

```

```
}
```

```
void spin_motor2_clockwise(){//combinacao 10
    PORTB |= 0b00000001;//saida PB0(D8) é setado em alta->input3 do l293D
    PORTB &= 0b11111011;//saida PB2(D10) é setado em baixa->input4 do l293D
}
```

```
void spin_motor2_counterclockwise(){//combinacao 01
    PORTB |= 0b00000100;//saida PB2(D10) é setado em alta->input4 do l293D
    PORTB &= 0b11111110;//saida PB0(D8) é setado em baixa->input3 do l293D
}
```

```
void stop_motor2(){//combinacao 00
    PORTB &= 0b11111010;
}
```

```
void set_motor2(bool on, bool forward){
    if(on == true){//so haverá movimento de motor se o robo tiver on
        if(forward == true){
            spin_motor2_clockwise();//gira motor2 no sentido horario
        }else{
            spin_motor2_counterclockwise();    //gira motor2 no sentido antihorario
        }
    }else{
        stop_motor2();//nao rotaciona motor se o robo estiver off
    }
}
```

```
void stop_motors(){
    stop_motor1();
    stop_motor2();
}
```

```
void spin_robot(){//para fazer a rotacao do robo -> motores giram em sentidos opostos
    set_motor1(true, FORWARD);
    set_motor2(true, BACKWARDS);
}
```

```
void spin_robot_ms(float time_ms){//fazendo rotacao baseado em tempo, em seguida para
robo
    spin_robot();
    _delay_ms(time_ms);
    stop_motor1();
    stop_motor2();
}
```

```
void shutdown_robot(){
    stop_motor1();
    stop_motor2();
    set_red_led(false);
    set_green_led(false);
}
```

```
void init_pwm8bits(){//usa contador 0 - que é 8bits
    TCCR0A = 0b10000011;//modo fast pwm (ultimos 2 bits 1) / modo nao inversor (primeiros
bits 10)
    TCCR0B = 0b00000011;    //escalonando por 64, fico com freq de 250kHz
}
```

```
void init_pwm8bits_servo(){//usa contador 0 - que é 8bits
    TCCR2A = 0b10000011;//modo fast pwm (ultimos 2 bits 1) / modo nao inversor (primeiros
bits 10)
    TCCR2B = 0b00000011;    //escalonando por 64, fico com freq de 250kHz
}
```

```
void set_comp_reg_timer2(int value)
{
```

```

        OCR2A = value;
    }

void servo_angle_0()
{
    set_comp_reg_timer2(0);
}

void servo_angle_90()
{
    set_comp_reg_timer2(24);
}

void servo_angle_180()
{
    set_comp_reg_timer2(50);
}

void set_pwm8bits_duty_cycle(float duty){
    OCR0A = (int)( (duty/100.0)*255.0 );//valor de duty cycle de 0 - 100 * valor da contagem
    (2^8 = 256, entao tem 255 valores contados)
}

float get_rpm(float duty){//a rpm vai variar para cada percurso, pq a tensao enviada pro
motor vai mudar (usando o pwm)
    float rpm = 100 * (duty/100);//rpm alterado para ficar mais visivel todo o processo
    return rpm;
}

float convert_rpm_rad_s(float rpm){
    float vel_rad_s = (rpm*2*PI)/60;//1 rot = 2PI rad e 1 min = 60s
    return vel_rad_s;
}

```


float get_velocidade_linear(float duty){//Essa é a velocidade que o robo vai andar, considera igual pra as duas rodas. Vai depender da vel angular, q por sua vez depende da alimentacao do motor a ser ajustada pelo pwm

```
float rpm = get_rpm(duty);
```

```
float vel_rad_s = convert_rpm_rad_s(rpm);
```

```
float vel_linear = vel_rad_s * RAOIO_RODA; //v_linear = v_angular * raio
```

```
return vel_linear;
```

```
}
```

float get_spin_vel(float duty){//Essa vai ser a velocidade de giro do robo. Necessario anexar o desenho pra entender a formula no relatorio

```
float wheel_vel_linear = get_velocidade_linear(duty);
```

```
float spin_vel = (2*wheel_vel_linear)/ (2*D_RODA_CENTROIDE);
```

```
return spin_vel;
```

```
}
```

void move_robot_forward(float duty){//Passo o duty, pq movimento vai depender disso para variar a velocidade nos trechos. distancia - m

```
set_pwm8bits_duty_cycle(duty);
```

```
set_motor1(true, FORWARD);//ligando o motor
```

```
set_motor2(true, FORWARD);
```

```
}
```

/*por enquanto, ta uma funcao acessorio, no percurso o robo so vai pra frente e gira*/

void move_robot_backwards(float duty, float distancia){//distancia - m

```
float vel_robo = get_velocidade_linear(duty);
```

```
float tempo_movimento = distancia/vel_robo;
```

```
set_motor1(true, BACKWARDS);//ligando o motor
```

```
set_motor2(true, BACKWARDS);
```

```
_delay_ms(tempo_movimento*1000);
```

```
set_motor1(false, BACKWARDS);//parando o motor
```

```
set_motor2(false, BACKWARDS);
```

```
}
```

/*primeiro pego vel de giro do robo, que vai depender da rotacao, que vai depender do duty cycle do pwm para os diferentes percursos*/

void spin_robot2(float duty, float angle){//angulo em grau

float vel_giro = get_spin_vel(duty);

float angle_rad = fabs(angle)*((2*PI)/360);//supondo q passa o angulo em graus, faz a conversao

float tempo_giro = angle_rad/vel_giro; //vel_giro ta em rad/s, entao sai em segundo aqui

if(angle >= 0){

set_motor1(true, FORWARD);

set_motor2(true, BACKWARDS);

}else{

set_motor1(true, BACKWARDS);

set_motor2(true, FORWARD);

}

_delay_ms(tempo_giro*1000);//posso usar pq nao to fazendo nada enquanto gira

}

void spin_robot(float duty, int sentido_rotacao){

set_pwm8bits_duty_cycle(duty);

if(sentido_rotacao == 0){

set_motor1(true, FORWARD);

set_motor2(true, BACKWARDS);

}else{

set_motor1(true, BACKWARDS);

set_motor2(true, FORWARD);

}

}

void print_medicoes()

{

```

println("Medicoes");
    print("Ponto: ");
print_int(trajeto-1);
print("; temp: ");
    print2(temp_string);
    print("; gas_0: ");
    print2(gas0_string);
    print("; gas_90: ");
    print2(gas90_string);
    print("; gas_180: ");
    print2(gas180_string);
    print("; luz: ");
    print2(luz_string);
    println("");

}

void fill_temp_buffer(){
    String stringTemperatura = "";

    if(tempC >= -40 && tempC < 0){
        stringTemperatura += "Muito Frio";
    }else if(tempC >= 0 && tempC < 20){
        stringTemperatura += "Frio";
    }else if (tempC >= 20 && tempC < 80){
        stringTemperatura += "Quente";
    }else{
        stringTemperatura += "Muito Quente";
    }
    strcpy(temp_string, stringTemperatura.c_str());
}

const char * get_gas_label(){
    if(n_gas_level >= 0 && n_gas_level < 35){

```

```

        return "Gas Desconsiderado";
    }else if(n_gas_level >= 35 && n_gas_level < 50){
        return "Gas observado";
    }else if (n_gas_level >= 50 && n_gas_level < 75){
        return "Muito gas";
    }else{
        return "Gas critico";
    }
}

```

```

void fill_gases(){
    String stringGas0 = "";
    String stringGas90 = "";
    String stringGas180 = "";

    n_gas_level = n_gas_0;
    stringGas0 += get_gas_label();

    n_gas_level = n_gas_90;
    stringGas90 += get_gas_label();

    n_gas_level = n_gas_180;
    stringGas180 += get_gas_label();

    strcpy(gas0_string, stringGas0.c_str());
    strcpy(gas90_string, stringGas90.c_str());
    strcpy(gas180_string, stringGas180.c_str());
}

```

```

void fill_luz(){
    String stringLuz = "";

    if(n_luz >= 0 && n_luz < 25){
        stringLuz += "Pouca Luz";
    }
}

```

```

    }
    else if(n_luz >= 25 && n_luz < 50){
        stringLuz += "Luz Observada";
    }
    else if (n_luz >= 50 && n_luz < 75){
        stringLuz += "Muita luz";
    }
    else{
        stringLuz += "Luz critica";
    }
    strcpy(luz_string, stringLuz.c_str());

}

void read_sensors(){
    set_green_led(true);

    //sensor de temperatura
    ADMUX = 0b01000000; //AVCC como Ref. ADC0 como entrada ( primeiros bits 01)
    ADCSRA |= 0b01000000; //inicia a conversao ADSC vai pra 1
    while( !(ADCSRA & 0b00010000)); //trava aqui ate terminar leitura ADIF =1
    temperaturaADC = ADC;
    float step = 5.0/1023.0;
    float tempV = step * temperaturaADC;
    tempC = ((tempV-0.5)/10) * (1000);
    fill_temp_buffer();

    //sensor de gas
    /***** Leitura Gas INCIO *****/
    ADMUX = 0b01000001; //AVCC como Ref. ADC0 como entrada ( primeiros bits 01)

    servo_angle_0();
    _delay_ms(1000);

```

```

ADCSRA |= 0b01000000;//inicia a conversao ADSC vai pra 1
while( !(ADCSRA & 0b00010000));//trava aqui ate terminar leitura ADIF =1
gasADC = ADC;
n_gas_0 = (gasADC/1023)*100;

//sensor de movimento - em 0 graus
if((PIND & 0b10000000) == 128){
    println("Movimento detectado em 0 graus");
}

servo_angle_90();
_delay_ms(1000);
ADCSRA |= 0b01000000;//inicia a conversao ADSC vai pra 1
while( !(ADCSRA & 0b00010000));//trava aqui ate terminar leitura ADIF =1
gasADC = ADC;
n_gas_90 = (gasADC/1023)*100;

//sensor de movimento - em 90 graus
if((PIND & 0b10000000) == 128){
    println("Movimento detectado em 90 graus");
}

servo_angle_180();
_delay_ms(1000);
ADCSRA |= 0b01000000;//inicia a conversao ADSC vai pra 1
while( !(ADCSRA & 0b00010000));//trava aqui ate terminar leitura ADIF =1
gasADC = ADC;
n_gas_180 = (gasADC/1023)*100;

fill_gases();

//sensor de movimento - em 180 graus
if((PIND & 0b10000000) == 128){
    println("Movimento detectado em 180");
}

```

```
}
```

```
servo_angle_0();
```

```
_delay_ms(1000);
```

```
/****** Leitura Gas FIM *****/
```

```
//sensor de luminosidade
```

```
ADMUX = 0b01000010; //AVCC como Ref. ADC0 como entrada ( primeiros bits 01)
```

```
ADCSRA |= 0b01000000; //inicia a conversao ADSC vai pra 1
```

```
while( !(ADCSRA & 0b00010000)); //trava aqui ate terminar leitura ADIF =1
```

```
luzADC = ADC;
```

```
n_luz = (luzADC/1023)*100;
```

```
fill_luz();
```

```
set_green_led(false);
```

```
print_medicoes();
```

```
}
```

```
void piscar_leds(){
```

```
if(toggle_leds == false){
```

```
    set_red_led(true);
```

```
    set_green_led(false);
```

```
    toggle_leds = true;
```

```
}else{
```

```
    set_green_led(true);
```

```
    set_red_led(false);
```

```
    toggle_leds = false;
```

```
}
```

```
}
```

```
void print(const char* message)
```

```
{
```

```

        int i =0;
while(message[i] != '\0'){
    while (!( UCSR0A & (0b00100000))); /* Wait for empty transmit buffer */
    UDR0 = message[i];
    i++;
}
}

void print2(char* message)
{
    int i =0;
while(message[i] != '\0'){
    while (!( UCSR0A & (0b00100000))); /* Wait for empty transmit buffer */
    UDR0 = message[i];
    i++;
}
}

void print_int(int valor)
{

    int i =0;
    String stringOne = String(valor);
    strcpy(buffer_int, stringOne.c_str());

    while(buffer_int[i] != '\0'){
        while (!( UCSR0A & (0b00100000))); /* 1<<UDRE0 Wait for empty transmit
buffer */
        UDR0 = buffer_int[i];
        i++;
    }

}

```



```

void print_float(float valor)
{
    int i =0;
    int valor_int_2decimals = round((valor* 100));
    float valor_float_2decimals = valor_int_2decimals;
    valor_float_2decimals = valor_float_2decimals/100;

    String stringOne = String(valor_float_2decimals);
    if (stringOne.length() == 0){
        valor_int_2decimals = valor_int_2decimals/100;
        stringOne = String(valor_int_2decimals);
    }
    strcpy(buffer_float, stringOne.c_str());

    while(buffer_float[i] != '\0'){
        while (!(UCSR0A & (0b00100000))); /* Wait for empty transmit buffer */
        UDR0 = buffer_float[i];
        i++;
    }

}

void println(const char* message)
{
    int i =0;
    while(message[i] != '\0'){
        while (!(UCSR0A & (0b00100000))); /* Wait for empty transmit buffer */
        UDR0 = message[i];
        i++;
    }
    while (!(UCSR0A & (0b00100000))); /* Wait for empty transmit buffer */
    UDR0 = '\n';
}

```

```
void print_status(){
```

```
    print("Total_time: ");
    print_float(total_time);
    print("; Trajeto: ");
    print_int(trajeto);
    print("; rx: ");
    print_float(robo.x);
    print("; ry: ");
    print_float(robo.y);
    print("; ra: ");
    print_float(robo.angle);
    print("; e: ");
    print_int(estado_robo);
    print("; duty: ");
    print_float(duty_atual);
    println("");
}
```

```
unsigned char recebeDado(){
```

```
    //bit rxc sinaliza que tem bytes nao lidos no buffer
    while(!(UCSR0A & (0b10000000)));
    return UDR0;
}
```

```
void setup()
```

```
{
    /*PC5 PC4 PC3 PC2 PC1 PC0
    DDRC = 0 -> entrada / DDRC = 1 -> saida
    A0(PC0): conversor AD, sensor de temperatura (entrada).
    A1(PC1): conversor AD, sensor de gas (entrada).
    A2(PC2): conversor AD, sensor luminosidade (entrada).
    A3(PC3):
```

A4(PC4):

A5(PC5): liga/desliga robo (entrada).

*/

DDRC &= 0b10000000;

/*PD7 PD6 PD5 PD4 PD3 PD2 PD1 PD0

DDRD = 0 -> entrada / DDRD = 1 -> saida

D0(PD0):

D1(PD1):

D2(PD2): saida para led vermelho

D3(PD3): saida para led verde

D4(PD4):

D5(PD5):

D6(PD6): saida para pwm

D7(PD7): entrada pra sensor de movimento - digital

*/

DDRD |= 0b01001110;

/*PB7 PB6 PB5 PB4 PB3 PB1 PB0

13(PB5): inputs para motor 1 (saida).

12(PB4): inputs para motor 1 (saida).

11(PB3): PWM do servo motor

10(PB2): inputs para motor 2 (saida).

9(PB1) :

8(PB0) : inputs para motor 2 (saida).

*/

DDRB |= 0b00111101;

bool robot_on = false;//inicializa robo desligado

/*PWM motor DC*/

init_pwm8bits();

set_pwm8bits_duty_cycle(100);

```
//PWM servor motor
init_pwm8bits_servo();
servo_angle_0();
```

```
ADCSRA |= 0b01000000; //Inicializar o ADC
```

```
//contador 1 de 16 bits para interrupcao
TCCR1A = 0b10000000; //modo normal de operacao(4 primeiros bits da esq) TOV1 flag
quando atinge maximo
//o bit 3 habilita o CTC
//  $(1/0.1) = 16000000 / (1024 * 2 * (1+X))$ 
//  $(x+1) = ((16 * 10^6) * (0.1)) / (1024 * 2)$ 
TCCR1B = 0b00001101; // (pre scaling de 1024 3 ultimos bits direita) (bit 4 bit da direita
ativa o CTC, o valor de comparacao ta guardado em OCR1A)
OCR1A = 15; //valor da contagem 0 -> 1560, ai ele dispara interrupcao e zera o contador
//ts é o tempo que eu quero que cada interrupção seja chamada e ts= 1s.
```

```
//Habilito o overflow no bit 1
TIMSK1 = 0b000000010;
```

```
sei();
```

```
//para teste.
// Serial.begin(9600);
unsigned int ubrr = 3; //250k de taxa de transmissao
```

```
/* Set Baudrate */
UBRR0H = 0; // Shift the 16bit value ubrr 8 times to the right and transfer the upper 8 bits
to UBBR0H register.
```

UBRR0L = ubrr; // Copy the 16 bit value ubrr to the 8 bit UBBR0L register, Upper 8 bits are truncated while lower 8 bits are copied

UCSR0C = 0b00000110; /* Set frame format: 8data, 1stop bit */

UCSR0B = 0b10011000; /* Enable transmitter and receiver and interruption
(1<<RXCIE0) | (1<<TXEN0) | (1<<RXEN0)*/

print_status();

trajeto = 1;

estado_robo = 1;

}

void loop()

{

// bool ligado = is_on() || switch_usart == 108 || switch_usart == 76;

if(is_on()){

 //if(ligado == true){

 if(estado_robo == 0){

 shutdown_robot();

 duty_atual = 0;

 }

if(estado_robo == 1){//alinhar o robo com o objetivo

 set_red_led(true);

 struct Ponto_coleta p_atual = pegar_ponto_trajeto(trajeto);

 float spin_angle = calculate_angle(robo, p_atual);

```

float angle_dif = spin_angle - robo.angle;

if(angle_dif >= 0){
    if (angle_dif > tol_ang_1 ){
        duty_atual = 25;
    }
    else{
        duty_atual = 5;
    }
    spin_robot(duty_atual, 1);
    motor_ligado = true;
    robo.sentido_giro = 1;
}else{
    angle_dif = fabs(angle_dif);
    if (angle_dif > tol_ang_1 ){
        duty_atual = 25;
    }
    else{
        duty_atual = 5;
    }
    spin_robot(duty_atual, 2);
    motor_ligado = true;
    robo.sentido_giro = 2;
}

if(fabs(angle_dif) < tol_ang_2){
    stop_motors();
    duty_atual =0;
    motor_ligado = false;
    robo.sentido_giro = 0;
    estado_robo = 2;//robo ja esta alinhado, agora vai pro mover
    set_red_led(false);
    print_status();
}

```

```
}
```

```
else if(estado_robo == 2){
```

```
    if(trajeto == 1){
```

```
        set_red_led(true);
```

```
        estado_robo++;
```

```
    }
```

```
else if(trajeto >= 2 || trajeto <= 5){
```

```
    set_red_led(true);
```

```
    read_sensors();
```

```
    estado_robo++;
```

```
    set_red_led(false);
```

```
}
```

```
else{
```

```
    estado_robo++;
```

```
}
```

```
print_status();
```

```
}
```

```
else if(estado_robo == 3){//robo entra em movimento
```

```
    set_red_led(true);
```

```
    struct Ponto_coleta p_atual = pegar_ponto_trajeto(trajeto);
```

```
    float distancia_ate_objetivo = calculate_distance(robo, p_atual);
```

```
    if (distancia_ate_objetivo > tol_dist_1){
```

```
        duty_atual = pegar_duty_trajeto(trajeto);
```

```
        move_robot_forward(duty_atual);
```

```
        robo.sentido_robo = 1;
```

```
        motor_ligado = true;
```

```
    }
```

```
    else if(distancia_ate_objetivo > tol_dist_2){
```

```
        duty_atual = 10;
```

```
        move_robot_forward(duty_atual);
```

```

    robo.sentido_robo = 1;
    motor_ligado = true;
}
else{
    trajeto++;
    print_status();
    stop_motors();
    motor_ligado = false;
    robo.sentido_robo = 0;
    if(trajeto >= 6){
        estado_robo = 4;
        set_red_led(false);
    }else{
        estado_robo = 1;
        set_red_led(false);
    }
}
}

else if(estado_robo == 4){
    set_red_led(true);
    read_sensors();
    estado_robo = 5;
    rotacao_acumulada = 0;
    set_red_led(false);
    print_status();
}

else if(estado_robo == 5){
    duty_atual = 25;
    spin_robot(duty_atual, 1);
    motor_ligado = true;
    robo.sentido_giro = 1;
    if(count_led >= 1000){

```



```
piscar_leds();
count_led = 0;
}
if(rotacao_acumulada > 720){
    stop_motors();
    set_red_led(false);
    set_green_led(false);
    motor_ligado = false;
    estado_robo = 0;
    duty_atual = 0;
    robo.sentido_giro = 0;
    print_status();
    exit(0);
}
}
```

```
// Imprimi o status
if(count_print >= 500){
    count_print = 0;
    print_status();
}
```

```
if(is_on() == false){
    if(switch_usart != 108 && switch_usart != 76){
        stop_motors();
        set_red_led(false);
        set_green_led(false);
        motor_ligado = false;
        estado_robo = 0;
        duty_atual = 0;
        print_status();
        exit(0);
    }
}
```

```
}  
}
```

```
ISR(TIMER1_COMPA_vect){  
    if (is_on()){  
        timer_interruption_flag = true;  
        total_time = total_time + interruption_interval;  
        count_print++;  
        count_total++;  
        count_led++;  
  
        if(motor_ligado == true){  
            //atualizar a orientacao do robo quando robo ta girando  
            if(robo.sentido_giro == 1){  
                vel_giro = get_spin_vel(duty_atual);  
                delta_angle = vel_giro *interruption_interval;  
                delta_angle_graus = (delta_angle*180)/PI;  
                robo.angle += delta_angle_graus;  
                //Serial.print(delta_angle_graus);  
                rotacao_acumulada += delta_angle_graus;  
            }  
            if(robo.sentido_giro == 2){  
                vel_giro = get_spin_vel(duty_atual);  
                delta_angle = -vel_giro *interruption_interval;  
                delta_angle_graus = (delta_angle*180)/PI;  
                robo.angle += delta_angle_graus;  
                rotacao_acumulada += delta_angle_graus;  
            }  
            if(robo.angle > 180){  
                robo.angle -= 360;  
            }  
            if(robo.angle < -180){  
                robo.angle += 360;  
            }  
        }  
    }  
}
```

```

    }

    //atualizando posicao x e y quando robo ta se movimentando
    if(robo.sentido_robo == 1){
        vel_linear_robo = get_velocidade_linear(duty_atual);
        angle_robo_rad = (robo.angle* PI)/180;
        delta_vel = vel_linear_robo * interruption_interval;
        robo.x += delta_vel * cos(angle_robo_rad);
        robo.y += delta_vel * sin(angle_robo_rad);
    }
}
}
}

ISR(USART_RX_vect){
    //receber os comandos pra desligar ou ligar
    switch_usart = recebeDado();
    if(switch_usart == 108 || switch_usart == 76){
        PINC |= 0b00100000;
    }else if (switch_usart == 100 || switch_usart == 68){
        PINC &= 0b11011111;
    }
    else{
        if(is_on()){
            PINC |= 0b00100000;
        }else{
            PINC &= 0b11011111;
        }
    }
}
}

```