1. **Introdução**

Este documento tem o objetivo de descrever o projeto de programação desenvolvido para o robô de sumô autônomo de 500g chamado “Tails”. O projeto foi desenvolvido pelo aluno Silas dos Santos Vergílio em linguagem C quando cursava o quarto ano de engenharia elétrica na UNESP , mas é resultado do trabalho e esforço de anos de desenvolvimento de projetos de robótica móvel autônoma da equipe FEG-Robótica. Este documento em conjunto com os relatórios das áreas da mecânica e eletrônica são responsáveis por descrever o desenvolvimento robusto de um projeto de robótica competitiva. O robô em questão foi destinado a competição de robótica “Winter Challenge” que é o maior evento de robótica competitiva do Brasil nas categorias de sumô de robôs e combate de robôs. A organização não exige a apresentação do processo de desenvolvimento do robô ou valoriza projetos de qualidade. Para o grupo, competir é apenas uma maneira de pôr o conhecimento adquirido à prova. O programa foi desenvolvido usando o programa PIC CCS Compiler e foi projetado para operar no microcontrolador PIC18F 2431, ele é o cérebro do sistema e suas características deve ser muito bem estudadas antes de escolher o ideal O mesmo conta com algumas propriedades essenciais para o sucesso do desafio que são portas A/D, que convertem um valor de tensão em um número entendível de 0 a 1023(isso será usado na interpretação do sinal dos sensores). Power PWM que será muito importante no momento de controlar a velocidade dos motores. Estas são apenas algumas das caracterísitas que são aproveitadas neste elemento eletrônico. A figura 1 apresenta o desenho renderizado do robô que foi impresso em uma impressora 3D.



Figura 1 - Desenho renderizado de Tails

1. **Fluxograma**

O primeiro passo no desenvolvimento do projeto é definir o fluxo de comandos, funções, tomadas de decisões que espera-se obter com o programa. É importante salientar as principais regras e objetivos da competição.

O robô compete em uma arena circular que tem em sua borda uma faixa branca. Dois robôs se posicionam com a sua frente em sentidos opostos. Ambos devem esperar 5 segundos após dado o início da partida, parados sem movimento e depois, sem o controle externo de qualquer tipo, devem encontrar o robô adversário e empurrá-lo para fora da arena. Tails deve ser capaz de interpretar sensores infra vermelhos para identificar seu adversário, sensores de linha para identificar o fim da arena, controlar motores (inclusive sua velocidade), tudo isso de maneira eficiente, estratégica e estável. A figura 2 apresenta o fluxograma do projeto.

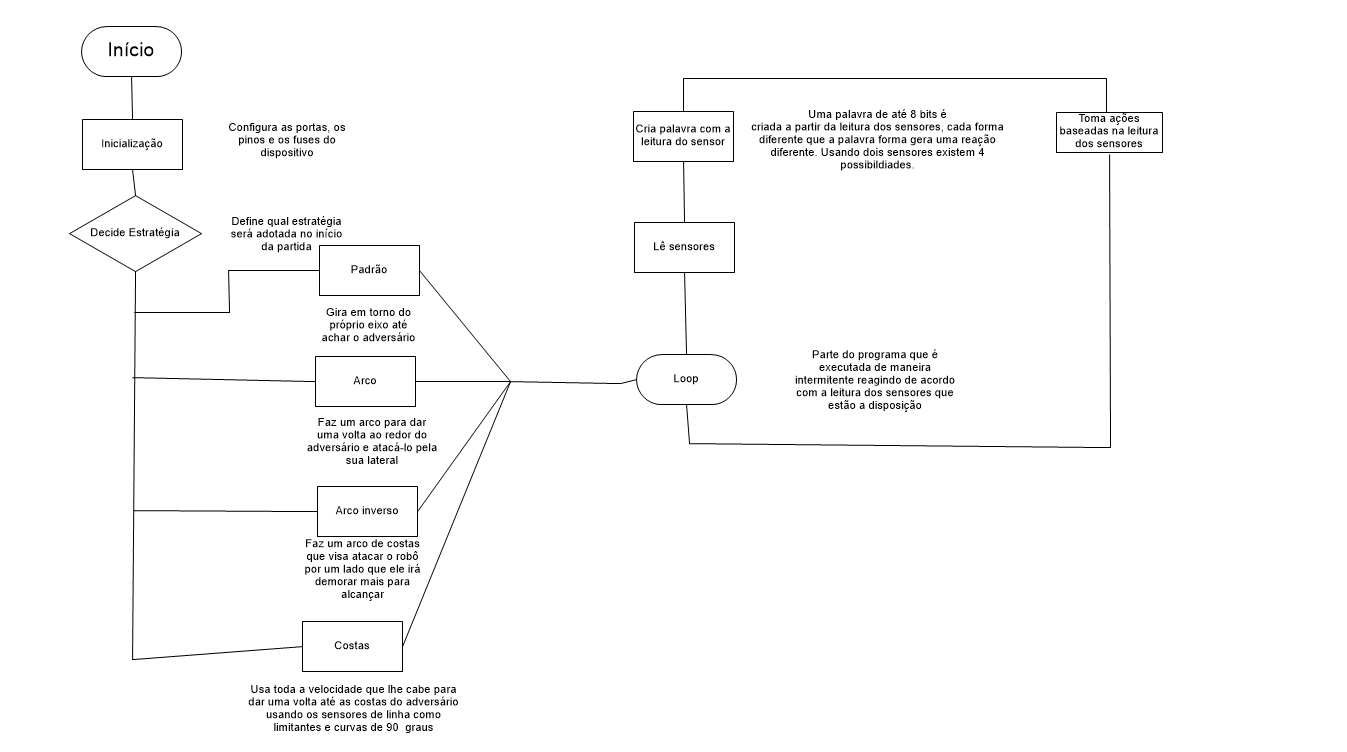


Figura 2 - Fluxograma do programa

O programa é dividido em três principais partes: configuração, que é muito importante para que o dispositivo funcione adequadamente, o detalhamento das configurações é apresentado nos próximos tópicos. A ideia desse código é inovar na competição com a opção de selecionar diferentes estratégias, o que possibilita enfrentar com mais inteligência e diversidade diferentes desafios. A terceira e última parte é o Loop, que nada mais é do que um laço que repete infinitamente uma sequência de comandos, estes comandos devem ser suficientes para que o robô execute sua rotina e reaja ao que acontecer; robôs adversários, linha indicando o fim do dojô, entre outras coisas.

1. **Desenvolvimento**

Como descrito no fluxograma, o primeiro passo do código é aplicar configurações. Isso ocorre em dois momentos, o primeiro logo no começo e o segundo ocorre no começo da função *main()* que é a função principal do código que é executada em primeiro plano. A figura 3 apresenta as primeiras configurações.

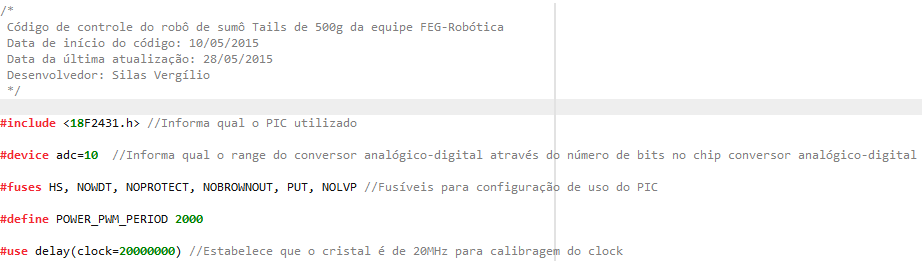


Figura 3 - Primeiras configurações do PIC.

Primeiramente é preciso adicionar a biblioteca do dispositivo que contém todas as funções que são disponibilizadas pelo compilador, lembrando que cada compilador contém uma maneira diferente de usar as funções do dispositivo; as funções que estão na biblioteca indicam ao programador como utilizar os diferentes periféricos(conversores, memória,envio de sinais, recebimento de sinais, entre outros). Em seguida é definido o range do conversor A/D que no caso é de 10 bits, o que quer dizer que ao converter uma tensão de 5V é possível entende-la como um valor de 0 a 1023, isso é usado na interpretação do sinal dos sensores. Um conversor A/D tem a função de converter um valor de tensão (5V, 10V, etc) em um número, afinal quando estamos programando não é possível ler um valor de tensão, mas sim valores na base 2 dependendo do alcance da conversão do dispositivo; por exemplo, um dispositivo que tenha um conversor A/D de 8 bits, irá ler uma tensão e convertê-la para um valor de 0 a 255 (), o cálculo desse valor é feito com uma simples regra de 3, o mesmo irá aparecer em outro momento deste documento. Depois disso vem os *Fuses*, configurações do hardware interno do PIC. HS indica que temos um cristal de alta velocidade (no caso 20Mhz, o que torna o tempo entre a execução de cada comando bem pequena que dá a impressão de que todas as linhas são executas de uma só vez), dessa maneira a leitura dos sensores é quase simultânea. NOWDT indica que o dispositivo não ativa a função de *watchdog timer*, função essa que impede que o programa entre em algum loop muito demorado, o que geralmente indica que o programa travou em algum ponto, todavia no caso do robô é esperado que ele fique em um laço infinito portanto a função *watchdog timer* não pode estar ativada. Depois são declaradas as variáveis globais do código, elas podem ser chamadas a qualquer momento no código. NOPROTECT é uma indicação de que o código não tem proteção contra cópia, nossa equipe promove justamente a disseminação do conhecimento e não esconder ele. NOBROWNOUT faz com que o PIC não reinicie suas atividades caso a sua tensão de alimentação fique abaixo de 4V. PUT significa *power up timer* que é um atraso de 92ms

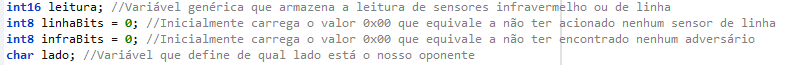


Figura 4 - Declaração das variáveis globais

A variável *leitura* é usada sempre que a leitura de algum sensor ocorre, ela é totalmente genérica porque ao invés de ter uma variável para cada leitura, a cada vez que o valor é lido ele forma uma parte da variável *linhaBits* ou *infraBits* e depois é reiniciado, assim usando apenas 3 varíaveis somos capazes de interpretar 5 sensores, no caso deste projeto, ou até mais se necessário. As variáveis *infraBits* e *linhaBits* funcionam de maneira idêntica. Cada bit da variável indica o estado de um sensor, se o mesmo encontrou ou não o que se procura (um adversário ou a linha branca), o programa futuramente lê a palavra de uma vez só possibilitanto interpretar muitas leituras de uma só vez. Por fim a variável lado pode ter dois valores, ‘e’ ou ‘d’ indicando qual o lado que o robô espera que o adversário esteja, cada estratégia programa precisa ter uma versão para cada lado para que a mesma seja eficiente.

Uma técnica para facilitar a visualização e utilização do código é o vasto uso de funções, para que cada bloco de instruções que puder ser entendido de maneira clara como uma ação é transformado em uma função que é chamada na função *main()* isso deixa o código principal mais limpo e sua mudança mais prática, além de facilitar o entendimento do fluxo das instruções. A figura 5 mostra as primeiras funções escritas.

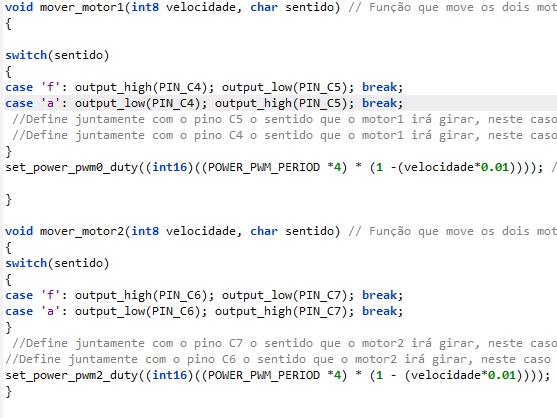


Figura 5 - Funções que controlam os motores dos robôs

Estas são funções muito importantes e que no curso da equipe inovam no controle de motores, pois até agora a equipe criava “manualmente” uma onde PWM para controlar a velocidade, neste programa os pinos Power PWM foram usados, eles tem alguns comandos já definidos na biblioteca do dispositivo. Antes disso é preciso ver de maneira superficial como o controle dos motores (que usa um circuito de ponteH) pode ser acionado. Cada motor tem 3 pinos de controle, sendo que dois deles definem o sentido do motor, eles devem estar em níveis lógicos diferentes, um em *high*(aproximadamente 5v) e outro em *low*(aproximadamente 0v), inverter isso faz com que o motor esteja pronto para girar no outro sentido. O outro pino é o pino de PWM. Basicamente este pino liga e desliga o motor, afim de controlar a velocidade do mesmo é enviada uma onde que varia apenas entre dois valores: 5v e 0v,basicamente *on* e *off*, a porcentagem do tempo, em relação ao período total da onda em que a onda deixa o motor ligado indica o quão rápido ele será em relação a sua velocidade máxima. A função *set\_power\_pwm0\_duty* controla justamente essa porcentagem , também chamada de *ciclo de trabalho*, multiplicando o período do *power pwm* do pic por 4 (informação indicada pelo datasheet dele) e multiplicando isso por uma porcentagem do mesmo. Os parâmetros da função são *porcentagem* e *sentido*, ambos de maneira bastante intuitiva controlam o funcionamento de cada motor. Quase tão importante quanto mover os motores é parar os motores, para isso basta mandar nível lógico baixo nos dois pinos de sentido como mostra a figura 6. Importante notar que além desses comandos foi enviado também o comando de velocidade do motor para 0, isso foi feito como uma medida para garantir a parada do motor que não estava eficiente nos testes.

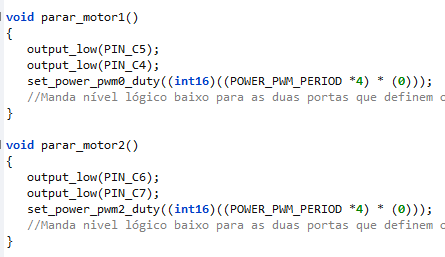


Figura 6 - Funções que param os motores

Após garantir o controle dos motores vem a parte que torna o robô inteligente que é o tratamento dos sinais dos sensores. Todo tratamento de sinal de sensores é dividido em três funções: função leitura, função que chama a leitura, função que trata o valor lido. Antes de mostrar o código de cada uma é necessário entender o método que o sinal é tratado. Em primeiro lugar a leitura é feita através de um canal ADC, ele converte um valor de tensão em um valor proporcional de 0 a 1023, esse valor é bastante importante afinal ele é quem indica um valor que podemos trabalhar, já que 4,0 volts, 3,3 volts não são valores que conseguimos trabalhar na programação. No caso dos sensores usados pelo robô (sensores de linha e sensores de distância), seus valores intermediários não importam, tudo que importa é se a linha branca foi achada ou não; se o adversário foi achado ou não. Desta maneira quando a função de leitura é chamada o valor lido é comparado com algum parâmetro que define a transição entre algo que deseja ser percebido ou não, como a presença de um adversário ou a linha branca do dojô. No caso deste projeto um destes valores usados foi proveniente da análise de uma curva característia enquanto que o outro de testes. A figura 7 apresenta o gráfico de resposta em tensão de acordo com a distância de um objeto do sensor infravermelho usado.

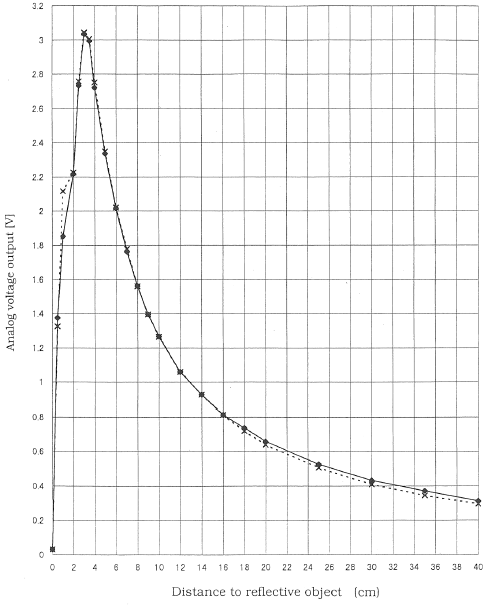


Figura 7 - Gráfico da resposta em tensão de acordo com a distância de um objeto ao sensor

Por uma análise do gráfico percebe-se que o menor valor de tensão em que um objeto pode ser visto é aproximadamente 0,3V. A tradução deste valor de tensão no valor que ele será representado para o PIC é simples:

A partir da resolução da regra de três acima obteve-se o valor de 65. Todavia o que é feito com esse valor é ainda mais importante. O artifício utilizado para tratar todas as leituras de uma só vez é fazer com que cada leitura forme uma palavra e o programa tenha que leia a palavra de uma vez e já saiba o que está havendo com todo um grupo de sensores. A palavra a que se refere não é uma palavra como geralmente se está acostumado, e sim um conjunto de zeros e uns em que cada bit representa o estado de um sensor. 1 querendo dizer que algo foi visto (adversário ou linha branca) e 0 indicando que nada foi encontrado. A figura 8 explica um pouco esta ideia.

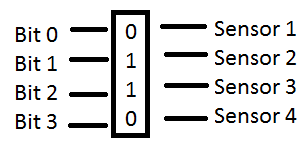


Figura 8 - Descrição gráfica de lógica de armazenamento de situação de sensores

Portanto cada vez que um sensor vai ser lido é preciso definir em qual porta conversor A/D do PIC ele está conectado e qual bit da palavra que representa o grupo dos sensores ele irá representar. A figura 9 apresenta o código de uma leitura genérica que é usada para os dois tipos de sensores.

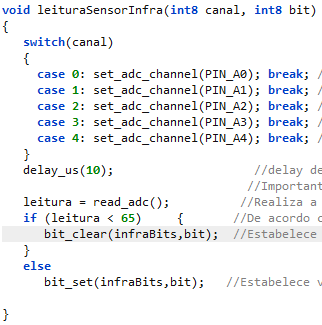


Figura 9 - Código da leitura de um sensor qualquer

Primeiramente de acordo com o parâmetro da função um dos canais A/D é preparado para receber um valor (o PIC faz apenas uma leitura de cada vez, por isso antes de cada leitura é preciso “abrir” um dos canais). Em seguida o valor lido é comparado com o parâmetro 65, caso seja menor ele zera o bit que representa o sensor, caso contrário ele define como 1, indicando que algo foi encontrado. Para o tratamento do que foi lido pelos sensores é importante saber que essa palavra que foi formada como um número binário pode ser escrita em diversas outras bases, como a base 10, base 8, base 6,seja lá qual base for, no fundo será a mesma coisa. Para efeitos de simples compreensão cada palavra foi vista como um número decimal de 0 a 16 (para o caso de usar 4 bits). Para cada número pode haver uma reação diferente do robô. No robô que foi a competição a reação da maioria das palavras foi semelhante, mas num futuro próximo é possível definir reações mais inteligentes e variadas que compreendam um maior número de situações em que o próprio robô pode se encontrar. A figura 10 mostra a função que trata a palavra de um grupo de sensores.

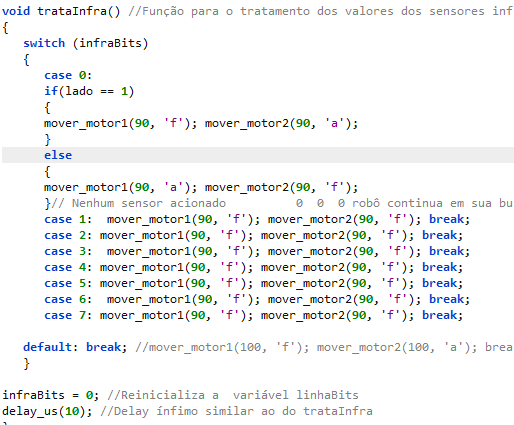


Figura 10 - Função que interpreta e toma ação de acordo com a leitura instantânea dos sensores.

O caso ‘0’ representa a situação em que nenhum sensor foi acionado portanto o robô deve realizar um padrão de busca. De acordo com a posição inicial dos robôs este padrão pode mudar, isto é dito pela variável “lado” que será explica em seguida. Os outros casos (de 1 a 7) representam quando o robô adversário foi visto, então Tails se move para frente com quase toda a sua potência (em testes o uso da potência total causava instabilidade no robô) afim de derrubar o outro robô.

Um aspecto muito simples da competição que teve inclusive sua pesquisa negligenciada devido ao caráter simples de seu desafio, gerou um grande problema: a regra dos cinco segundos. De acordo com as regras da competição, após ligar o robô o mesmo deve esperar cinco segundos antes de começar qualquer ação. Com a função *delay\_ms(tempo\_em\_milisegundos)* que tem a função de executar a última ação e esperar sem fazer nada a mais por um tempo informado pelo programador, isso deveria ser uma tarefa simples. Todavia ao estudar mais profundamente o hardware do microcontrolador que estamos trabalhando nota-se que as estruturas usadas para gerar a onda PWM (comparadores, contadores, entre outros) também são usadas para gerar esse tempo, portanto ao mandar o microcontrolador esperar 3 segundos, ele ás vezes esperava 1 segundo, ás vezes 20 segundos e ás vezes nós desistimos de contar; tudo para o mesmo comando. Ficou claro que usar o comando *delay* não seria proveitoso. Após muitos estudos percebeu-se que o ideal seria usar algo que mesmo com a interação de outros comandos levasse o mesmo tempo para ser executado; tendo então em mente a teoria de microcontroladores usou-se o tempo que o sistema leva para executar uma instrução, este tempo tem pouca ou nenhuma variação e ao repetir a mesma instrução é possível usar isso como uma base de tempo, como mostra a figura 11.

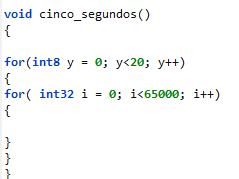


Figura 11 - Delay artificial de cinco segundos para o início da partida

Enfim alcançamos a função *main()* do programa que vai configurar mais alguns aspectos do programa e chamar as funções que serão efetivamente usadas.

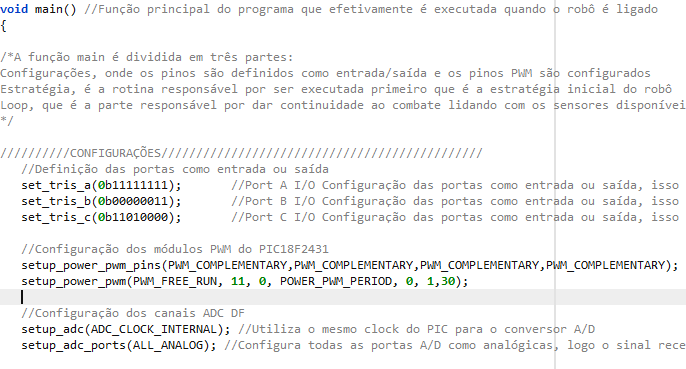


Figura 12 - Configurações dentro da função main()

O comando *set\_tris* define como cada porta do PIC irá se comportar,se será usada como um dispositivo de entrada ou de saída, onde 1 indica dispositivo de entrada (as portas A são onde os sensores são posicionados) e 0 se é um dispositivo de entrada (as portas B indicam sinais de saída como PWM ou controle de direção dos motores).

Mais dois comandos para definir o funcionamento são introduzidos, o primeiro deles diz o modo de operação dos módulos PWM(lembrando que cada módulo PWM é representado por duas portas PWM), no caso do nosso PIC esses módulos operam de maneira complementar, portanto define-se como uma das portas do módulo irá operar e a outra irá operar de maneira contrária automaticamente. O outro comando de configuração é mais específico e define algumas coisas mais específicas, em que a maioria delas segue simplesmente um padrão básico para um funcionamento adequado do controle PWM, é importante salientar a importância do número 30, o último parâmetro da configuração é chamado de *dead time*, ele é o tempo de espera que existe antes de alterar um sentido, ou uma velocidade; a princípio este valor era 0, mas havia muita instabilidade no controle dos motores, além de que se os motores exigissem correntes mais altas isso poderia gerar picos muito altos frequentemente e estragar a eletrônica do robô. Por fim os canais de conversão A/D são configurados todos como analógicos, quer dizer que vão receber um sinal que pode ser de 0V até 5V aproximadamente e transformar em um número de 0 a 1023.

Na figura 10 houve a aparição da variável *lado* e não ficou explicado como ela surgia, a figura 13 apresenta o comando que inicializa a variável



Figura 13 - Inicialização da variável lado

O comando *input()* verifica se determinado pino do PIC está recebendo um nível lógico alto (3,3V até 5V aproximadamente) ou um níve lógico baixo (0V até 2,2V aproximadamente). Essa escolha de lado é feita introduzindo um curto circuito no pino C1 caso deseja-se nível lógico 1 ou deixando ele como está para um nível lógico 0.

Após todas essas definições o programa que irá rodar em um loop infinito fica bastante simples e compacto como mostra a figura 14.

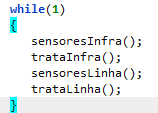


Figura 14 - Laço infinito que rege o programa

Os sensores são lidos, seus sinais tratados, isso a uma velocidade tão rápida de processamento que dá a impressão de que os quatro comandos acontecem simultaneamente e o tempo todo, o que para fins teóricos não é verdade, mas na prática funciona bem.

1. **Conclusão**

O robô Tails completou com êxito a tarefa que lhe foi proposta. Esperar cinco segundos, procurar o adversário com uma estratégia prévia e depois entrar em uma rotina de busca. A maioria das portas do microcontrolador foi usada deixando apenas algumas livres para possíveis expansões, o que demonstra que sua escolha foi muito boa. O programa obteve sucesso nos testes e a resolução de seus problemas, graças a boa organização do código, foi muito produtiva e prática. A caracterísitca modular, uma função principal reduzida e o alta densidade de comentários nas linhas de código, além dos resultados obtidos e conhecimentos adquiridos no projeto, comprovam o sucesso do projeto que foi aqui relatado.