ESCOLA SUPERIOR NÁUTICA INFANTE D. HENRIQUE
DEPARTAMENTO DE ENGENHARIA MARÍTIMA

Projeto USV

Relatório de Programação do Comando

Prof. Pedro Teodoro

Trabalho elaborado por:

- Christian Rodrigues 15202
- Vasco Ferreira 15200
- Rodrigo Maria 15217
- Gabriel Oliveira 15197
- Tiago Macedo 15224
- Tiago Pereira 15223

ENIDH - RIA – 2024/2025

ÍNDICE

Introdução

O código do comando define as instruções interpretadas pelo sistema de controlo do barco em modo manual. Permite ao operador enviar comandos diretos para a navegação, manobras e ajustes operacionais do USV, substituindo temporariamente a lógica autónoma. É essencial para garantir resposta precisa em situações onde o controlo humano é necessário.

Objetivos

O principal objetivo é tornar o código do comando mais dinâmico e adaptável a diferentes situações em que o controlo manual é essencial. Para isto foi necessário compreender a estrutura do código existente, analisar a sua lógica de funcionamento e aplicar os conhecimentos adquiridos para realizar os ajustes necessários após a instalação de um novo motor, garantindo compatibilidade e desempenho adequado do sistema.

Desenvolvimento

Durante a primeira semana até à terceira semana, foi necessário dedicar tempo a compreender detalhadamente o funcionamento do código. Surgiram algumas dificuldades em perceber a estrutura da codificação e os valores que precisavam de ser enviados e de ser recebidos entre o recetor e o emissor, o que originou alguns erros ao longo do processo. Apesar destas dificuldades, esta fase foi importante para perceber melhor como a comunicação funciona e para facilitar alterações e melhorias no futuro do projeto.

Durante o desenvolvimento do código de controlo remoto do USV, tivemos problemas porque o Arduino recebia valores aleatórios ou até caracteres estranhos quando enviávamos números grandes (por exemplo, 40100 ou 40200), enquanto valores mais baixos como 30100 funcionavam bem. Depois de analisar, percebemos que o tipo de variável podia estar a limitar o valor máximo suportado, por isso mudámos para *uint16_t*, o que resolveu o problema e garantiu que todos os comandos fossem transmitidos corretamente, mesmo com prefixos maiores.

```
int mensagem;       // Valor enviado via radio até 32767
// Alteração
uint16_t mensagem;  // Valor enviado via radio até 65534
```

Da a quarta semana até à quinta semana, procedeu-se à implementação de um terceiro motor no sistema. Inicialmente, o código desenvolvido apresentava uma abordagem bastante métrica e específica, o que resultou numa menor flexibilidade.

Após análise por parte do professor, foi sugerido que, em vez de uma estrutura de código demasiado rígida, se optasse por uma solução mais dinâmica. O objetivo dessa recomendação foi aumentar a flexibilidade do sistema, facilitando adaptações e melhorias futuras.

Melhorias ao código

```
if (switchState == LOW) {
  potValueY = analogRead(potPinA);
  if ((potValueY > 515) && (potValueY < 525)) {

    potValueY = 512;
  }

  potValueX = analogRead(potPinB);
  //serial.print("potValueX")

  if ((potValueX > 512) && (potValueX < 520)) {

    potValueX = 512;
  }

  vel_MC = map(potValueY, 0, 1023, 1800, 1200);
  vel_Turn = map(potValueX, 0, 1023, 200, -200);
```

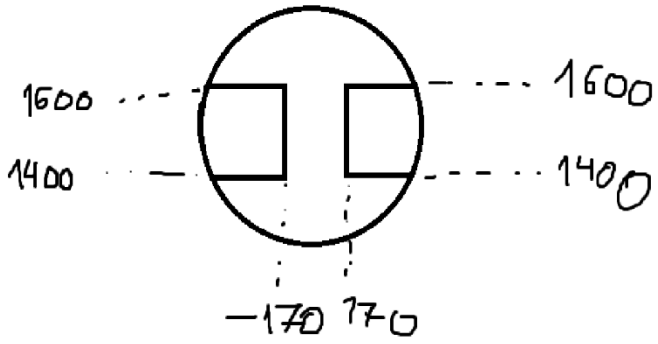
```
//Função para obter os valores do joystick de x
int obterValor(int joyStick, int eixo){
  int valor = 0;
  int x, y = 0;

  valor = analogRead(joyStick);

  // Fazer verificação se está no meio
  if ((valor > 515) && (valor < 525)) {

    valor = 512;
  }

  // Conversão - Uso o 1 para identificar que quer obter o valor de Y e 2 para o X
  if(eixo == 1){
    valor = map(valor, 0, 1023, 200, -200); // Y
  }
  else if(eixo == 2){
    valor = map(valor, 0, 1023, 1800, 1200); // X
  }
  else {
    valor = 0; // Não existe
  }

  return valor;
}
```

Na segunda versão do código foram feitas várias melhorias importantes, tendo em conta os comentários e orientações do professor Carlos Gonçalves. Uma das principais melhorias foi a organização do código em funções específicas, o que permitiu otimizar a leitura, facilitar alterações futuras e tornar o funcionamento mais claro. Para além disso, otimizou-se a leitura dos valores dos joysticks e o mapeamento foi ajustado para garantir um controlo mais preciso dos motores. Estas mudanças ajudaram a tornar o código mais eficiente, modular e robusto para responder melhor às necessidades do projeto.

Durante a sexta semana, debatemos em equipa qual seria a melhor forma de implementar o código. A principal dificuldade foi conciliar as várias opiniões e diferentes maneiras de resolver o problema.

Como não chegámos a um consenso inicial, decidimos experimentar várias abordagens diferentes cada um realizando o seu próprio código e no final escolheríamos aquele que se destacou pela simplicidade, clareza e bom funcionamento.

Durante o desenvolvimento, optou-se por implementar limitações nos eixos do joystick como forma de distinguir as intenções de comando do operador. Esta abordagem permitiu identificar, com maior precisão, os momentos em que os motores deveriam realizar rotações ou manter um movimento linear, melhorando assim a interpretação dos sinais manuais e a resposta do sistema de controlo.



Entre as semanas sete e nove, focámo-nos na implementação dos códigos definidos nas etapas anteriores. Uma das maiores dificuldades foi pensar numa forma de estruturar o código de maneira mais dinâmica, evitando rigidez e permitindo que fosse mais fácil de ajustar ou melhorar no futuro. Foi necessário algum tempo de reflexão e testes para chegar a uma solução mais flexível, sem comprometer o bom funcionamento do comando.

Como se observa na implementação, optou-se por modularizar o código, evitando concentrar toda a lógica dentro da função loop(). Parte significativa do processamento foi deslocada para funções auxiliares, as quais são invocadas apenas quando necessário. Esta abordagem contribuiu para uma estrutura mais limpa, eficiente e de fácil manutenção, além de facilitar a leitura e o reaproveitamento do código.

```
59  //Função para obter os valores do joystick de x
60  int obterValor(int joyStick, int eixo){···
84  }
85
86  int mandarValor(int motor, int x , int y){···
143 }
144
145 //Função para obter os valores do joystick de x
146
147 void loop() {
148
149   switchState = digitalRead(switchPin);
150
151   if (switchState == LOW) {
152     digitalWrite(ledPinA, HIGH);
153     digitalWrite(ledPinB, LOW);
154
155     //Enviar os dados para recetor sovre o LED
156     mensagem = 40100;
157     radio.write(&mensagem, sizeof(mensagem));  // Sending data over NRF 24L01
158     Serial.print("Transmitted msg (Modo Manual): ");
159     Serial.println(mensagem);
160
161     // Obter os Valores do JoyStick
162     Valor_JEY = obterValor(JEY, 1);
163     Valor_JEX = obterValor(JEX, 2);
164     Valor_JDY = obterValor(JDY, 1);
165     Valor_JDX = obterValor(JDX, 2);
166
167     // Mandar os valores para          || Caso queiras ler os valores coloca os delays e tira o comentário dos print's na função
168     m1 = mandarValor(1, Valor_JDX, Valor_JDY);
169     //delay(1000);
170     m2 = mandarValor(2, Valor_JDX, Valor_JDY);
171     //delay(1000);
```

Na semana dez, foi feita a seleção do código a utilizar no projeto final. A principal dificuldade esteve relacionada com o uso do mesmo joystick para controlar dois motores, o que causou alguns bugs em certas versões do código. Por esse motivo, foram realizados vários testes para analisar as diferentes abordagens. Uma das versões mostrou-se mais estável e funcionou corretamente, sendo essa a escolhida para integrar no projeto.

Entre as semanas dez e catorze, continuámos a realizar testes práticos para identificar erros, corrigir bugs e ajustar o comportamento do sistema. Durante este período, o código foi aprimorado, tornando-se ainda mais dinâmico e flexível, o que facilitou ajustes posteriores e melhorou a sua eficiência. Esse ciclo contínuo de testes e melhorias garantiu que a versão final fosse mais dinâmica e adequada aos objetivos do projeto.

Conclusão

Com este trabalho foi possível desenvolver um código funcional para o comando pretendido, atingindo os objetivos propostos. O código criado demonstra ser dinâmico, permitindo adaptações e melhorias futuras conforme as necessidades. Este resultado evidencia a nossa capacidade de planear, implementar e testar soluções de forma eficiente, consolidando os conhecimentos adquiridos e abrindo caminho para possíveis evoluções deste projeto.

Referências

https://github.com/15200-USV/USV.git