UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Deivison André Schast

Desenvolvimento de sistema para controle remoto de um canhão de água para combate a incêndios em lavouras.

Três de Maio
2022

Deivison André Schast

Desenvolvimento de sistema para controle remoto de um canhão de água para combate a incêndios em lavouras.

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Werner Kraus Júnior, Dr.
Supervisor: Diego Renan Follmann Bittencourt, Eng.

Três de Maio
2022

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

http://portalbu.ufsc.br/ficha

Deivison André Schast

Desenvolvimento de sistema para controle remoto de um canhão de água para combate a incêndios em lavouras.

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 02 de agosto de 2022.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Werner Kraus Junior, Dr.
Orientador
UFSC/CTC/DAS

Diego Renan Follmann Bittencourt, Eng.
Supervisor
Engenharia de Produto da São José Industrial

Prof. Eduardo Rauh Muller, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

Este trabalho é dedicado aos meus queridos pais, sem eles nada disso seria possível.

AGRADECIMENTOS

À minha família, por todo o trabalho árduo que tiveram e por terem me dado todo apoio durante a graduação, garantindo que eu pudesse escolher os meios os quais me interessava trabalhar.

Ao supervisor deste trabalho, Diego Renan Follmann Bittencourt, por estar sempre disponível para auxiliar no projeto, e principalmente por ter me aberto as portas da empresa, dando a oportunidade de realizar o projeto dentro da São José Industrial.

Aos meus colegas de trabalho, o qual eu pude conviver diariamente, e sendo disponíveis nas horas que foram precisas para resolução de dúvidas.

A empresa São José Industrial, pela estrutura disponibilizada para realização do projeto, bem como os demais funcionários presentes nos diversos setores dentro da empresa, onde por vezes precisei buscar auxílio.

Ao Sérgio Pereira da Rocha, pois sem a ajuda dele eu não teria conseguido iniciar a graduação em Engenharia de Controle e Automação.

Aos meus colegas de curso de Engenharia de Controle e Automação, que contribuíram para o meu desenvolvimento pessoal e profissional, ao longo dos anos, e que dividiram alegrias e tristezas ao longo da graduação. Um agradecimento especial ao Pedro Henrique Neves, colega de apartamento, e que durante boa parte da graduação compartilhou o dia a dia comigo.

E, por fim, à Universidade Federal de Santa Catarina, pública e de qualidade, que me proporcionou experiências diversas, ricas e engrandecedoras, agregando conhecimento para o meu desenvolvimento pessoal e profissional.

“Não podemos resolver nossos problemas
com o mesmo pensamento que tínhamos quando os criamos.”
(Einstein, Albert)

RESUMO

Com a expansão da agricultura e dos implementos para o desenvolvimento de cultivares, é preciso também poder manter uma segurança na lavoura. Muitas causas levam ao deterioramento e a perda de lavouras, e uma delas é o incêndio, que em alguns estados do Brasil é recorrente devido ao longo período do ano sem chuva, com bastante seca, e áreas de alta incidência de irradiação solar durante o dia. Pensando nisso, a empresa, a qual será citada nesse trabalho, desenvolveu um produto para o combate a incêndios em lavouras denominado Tanque Mata Fogo. Esse tanque conta com um sistema de canhão de água com mistura de uma solução química para fazer o combate a incêndio. Diante deste cenário, o objetivo geral deste trabalho é desenvolver uma ferramenta para o controle deste canhão de água a distância, utilizando comunicação *wireless* para a comunicação entre um *joystick* e o tanque de água, e também comunicação *bluetooth* para o controle do canhão através de um aplicativo *mobile*. Para isso, criou-se uma ferramenta em aplicativo móvel utilizando o *framework* de desenvolvimento de aplicativos híbridos *Flutter*. E para o desenvolvimento do *joystick*, criou-se um controle utilizando componentes eletrônicos capazes de fazer a comunicação e o controle necessário do canhão de água através da plataforma de programação de Arduino, cuja linguagem de programação é o *C++*. Os resultados alcançados durante os testes preliminares foram positivos, cumprindo os principais requisitos do sistema.

Palavras-chave: Tanque Mata Fogo. Controle. Canhão de água.

ABSTRACT

With the expansion of agriculture and implements for the development of cultivars, it is also necessary to be able to maintain security in the field. Many causes lead to the deterioration and loss of crops, and one of them is the fire, which in some states of Brazil is recurrent due to the long period of the year without rain, with a lot of drought, and areas of high incidence of solar irradiation during the day. With that in mind, the company, which will be mentioned in this work, developed a product to combat to fires in crops called *Tanque Mata Fogo*. This tank has a water cannon system with mixing a chemical solution to do combat to fire. Given this scenario, the general objective of this work is to develop a tool for controlling this water cannon at a distance, using communication wireless for communication between a joystick and the water tank, and also Bluetooth communication for controlling the cannon through a mobile application. For this, a mobile application tool was created using the Flutter hybrid application development framework. And for the development of the joystick, it was created a control using electronic components capable of communicating and the necessary control of the water cannon through the programming platform of Arduino, whose programming language is *C++*. The results achieved during preliminary tests were positive, meeting the main system requirements. The tool can still be further developed, with an external control structure more well defined. However, for design tests, it met what was expected.

Keywords: Tanque Mata Fogo. Control. Water cannon.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

app	aplicativo
MF	Mata Fogo

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo será apresentado a empresa São José Industrial, bem como sua divisão de Engenharia de Produto, processos industriais, e a equipe de suporte dentre os processos da empresa. Além disso, será introduzido o problema a ser atacado pelo projeto em questão, juntamente com as motivações que levaram à sua escolha. Então, expõe-se a solução proposta para tal problema, quais os objetivos do trabalho e a metodologia aplicada.

1.1 A EMPRESA SÃO JOSÉ INDUSTRIAL

A São José Industrial é uma empresa desenvolvedora de implementos agrícolas de alta performance, fundada em 1993 e sediada em São José do Inhacorá, uma pequena cidade do noroeste do Rio Grande do Sul (INDUSTRIAL, 2022c). A São José Industrial surgiu para abastecer os pequenos mercados que rodeavam a região, visando produzir ferramentas para área agrícola. A região era constituída na sua maior parte por pequenos e médios agricultores, e os implementos produzidos para essa região eram de porte pequeno também.

A agricultura está em constante evolução, e isso faz com que as empresas voltadas para o agronegócio busquem sempre inovação, trazendo assim produtos de altíssima qualidade bem como produtos que facilitem a vida no campo. Pensando nisso a São José Industrial, que antes tinha uma pequena fábrica com sede dentro da cidade, buscou recursos e fez uma ampliação dos seus processos, levando a fábrica para uma área industrial, com maior porte e maior possibilidade de crescimento.

Hoje, a empresa conta com uma ampla fábrica de produção de itens agrícolas, o seu quadro de funcionários é em torno de 470 pessoas, e sua carteira de implementos é amplamente vasta. Trazendo desde simples arados para o pequeno agricultor, até tanques graneleiros de altíssima qualidade para grandes agricultores e produtores de grãos.

A empresa sempre teve um orgulho muito grande em relação aos seus produtos, uma vez que eram implementos pequenos, pois atendiam a região próxima a cidade, oferecendo assim um bom relacionamento com os clientes, buscando sempre os *feedback* para trazer melhorias aos implementos.

Porém, a agricultura se moderniza dia após dia, inovando cada vez em máquinas autônomas, capaz de fazer todo o serviço de plantio, cultivo e colheita com base em dados presentes nos mais modernos softwares embarcados nos implementos. Através de sensores, as máquinas fazem os mais diversos tipos de leitura, como por exemplo umidade do solo na hora do plantio, quantas sementes caem por metro plantado, e uma infinidade de dados relevantes para que todo o ciclo de cultivo seja realizado com o máximo de aproveitamento e lucratividade.

E a São José Industrial não pensa diferente disso. Hoje, conta com implementos capazes de gerir de forma autônoma alguns processos na lavoura. Máquina para espalhar sólidos do tipo calcário, adubo ou uréia, já vem com tecnologia embarcada de taxa variável no espalhamento desses sólidos, podendo variar o peso de sólido distríbuido por hectare.

1.2 A DIVISÃO DA ENGENHARIA

A divisão da Engenharia de Produto fica localizada dentro da fábrica da São José Industrial, buscando trazer assim uma proximidade entre o chão de fábrica e os engenheiros, para buscar entender os processos e agilizar a comunicação entre o projeto e o produto final que será comercializado.

Dentro da divisão de engenharia, tem-se engenheiros encarregados por cada parte da produção. Como já dito, a gama de produtos é grande, e por tanto precisa haver uma divisão de tarefas e execução de projetos. As principais áreas dentro da engenharia são:

- Carretas graneleiras;
- Escarificadores e grades;
- Calcareadeiras e distribuidores de sólidos;
- Tanques de água e chorume.

As carretas graneleiras são implementos acoplados nos tratores que ficam responsáveis pelo transporte de grãos oriundos das colheitas, podendo ser usadas tanto dentro da lavoura para fazer a descarga de máquinas colheitadeiras, bem como fora para fazer a transposição desses grãos para uma outra área ou local. Essa família de implementos conta com diversos modelos, podendo ser encontradas carretas graneleiras, desde 10.500 litros até 33.000 litros. Ainda, dentro dessa gama de variáveis, as carretas graneleiras podem ser classificadas de acordo com seu poder de descarga, e com sua tecnologia agregada (INDUSTRIAL, 2022b).

Os escarificadores e grades são produtos próximos ao que eram comercializados no início da empresa, pois é um produto com tecnologia agregada mais na forma de produção e dos materiais que são utilizados para tal (INDUSTRIAL, 2022d). Porém, a inovação existe, pois o mercado cada dia mais traz tratores e maquinários mais potentes, e, portanto, conseguem acoplar implementos mais pesados e maiores. Um exemplo disso são os escarificadores, que são equipamentos que trabalham a superfície e a sub-superficie do solo para promover desagregação de camadas compactadas, a fim de facilitar a penetração de raízes das culturas, da água e do ar, para as camadas mais profundas do solo, sem incorporar a matéria orgânica.

As calcareadeiras e os distribuídores de sólidos também são produtos inovadores, pois trazem a taxa variável para a aplicação de produtos nas lavouras (INDUSTRIAL, 2022a). No caso do distribuidor de sólidos, ou distribuidor autopropelido, ele possui um divisor de produtos onde o sólido é alojado e é indispensável para a aplicação, e existem esteiras que transportam esse sólido até o centro de um disco na parte traseira do implemento onde é lançado ao solo, podendo chegar a até 20 metros de largura de lançamento. As calcareadeiras, distribuidores autopropelidos de calcário, são produtos similares ao distribuidor de sólido, porém é desenvolvida separadamente pois o peso e a densidade de produto é muito diferente, e portanto, é uma configuração quase que totalmente diferente na hora do lançamento.

Os tanques são responsáveis pelo transporte de líquidos (INDUSTRIAL, 2022g). Eles são classificados em dois grandes modelos, que são os tanques de água e os tanques de chorume. Os tanques de chorume são utilizados por fazendeiros para fazer a aplicação de chorume na lavoura pois é um adubo natural, podendo ser aplicado em qualquer área. Os tanques de água, como propriamente o nome diz, são para o transporte de água nas propriedades. Eles são divididos em tanques de transporte de água, tanques de combate a incêndio, e o mais recente projeto da empresa, os tanques Mata Fogo (INDUSTRIAL, 2022f). O que difere o tanque normal de combate a incêndio do tanque Mata Fogo, é que no segundo modelo tem um produto auxiliar que se mistura com a água para fazer o combate ao incêndio, neutralizando e quebrando as moléculas responsáveis pela reignição do fogo.

Além desses implementos citados acima, ainda tem-se outros implementos menores, como plataformas de transporte, carretas menores para transporte de cultivares, grades e aradores de subsolo, distribuidores de sólido de pequeno e médio porte, roçadeiras, e outros produtos menores (INDUSTRIAL, 2022e).

1.3 MODELO DE NEGÓCIO

A São José Industrial faz a comercialização de seus produtos através de revendas parceiras, e também diretamente na fábrica. Cada implemento tem uma série de acessórios que pode ser requisitado pelo cliente, fazem parte de um valor a mais, pois utilizam uma tecnologia ou material diferente. Os tanques de combate a incêndio por exemplo, podem vir com canhão de água manual ou eletrônico, dependendo da solicitação do cliente.

Geralmente, novas soluções são requisitadas por clientes que já testaram algum produto, e precisam fazer aplicações diferentes. Essas soluções são encaixadas no plano de desenvolvimento da equipe de Engenharia de Produto, de acordo com sua prioridade. São então testadas e com a validação são vendidas ao cliente final com uma nova solução.

Além disso, equipes de pós-vendas ficam responsáveis pelo treinamento de pro-

dutores que adquirem os produtos, bem como as revendas que estão comercializando esses produtos.

1.4 ENTREGA DE SOLUÇÃO

O projeto em questão foi desenvolvido na equipe de Engenharia de Produto, na divisão de tanques. O Tanque Mata Fogo é um dos carro-chefes da empresa, pois traz uma inovação que até então não existia no mercado.

Devido ao fato de ser um produto novo, e que é pioneiro no mercado, alguns *feedbacks* se tornam necessários para que possa ser feito o aprimoramento do equipamento. Dito isso, as equipes de pós-vendas ficam responsáveis por trazer as solicitações dos clientes referente a adição ou exclusão de opcionais no tanque. Essas solicitações são analisadas por um supervisor da engenharia, que se for plausível realizar, repassa para a equipe de engenharia, e nesse caso para o engenheiro responsável pelo devido equipamento.

1.5 PROBLEMA

Um dos grandes problemas enfrentados durante a operação do tanque Mata Fogo é o operador que fica exposto na parte de cima do tanque, controlando o canhão de água. Buscando então melhorar isso, procurou-se junto a empresas terceirizadas um canhão de água que fosse eletrônico, ou seja, tivesse uma espécie de *joystick* para controlar à distância o canhão de água. Porém, no mercado hoje, apenas é encontrado um canhão de água em que o *joystick* esteja conectado via fio, fazendo com que ainda assim o operador fique próximo da máquina para operar ela, apesar de estar mais a salvo.

O canhão de água que é utilizado pela empresa, depois de muitos testes, é de uma empresa terceirizada. Esse canhão já vem com tecnologia embarcada, e com um controle remoto. Porém, esse controle é com fio, como já citado, e isso é um limitador para a operação. O canhão de água possui motores que fazem o direcionamento da água. E também, possui um atuador linear que é o responsável por abrir ou fechar mais o esguicho do canhão, comumente chamado de bico do canhão, mudando assim do modo jato para o modo neblina, que é um leque de água.

Em lugares de alta vegetação, como é o caso de lugares no centro-oeste brasileiro, em que o gado de corte fica em meio as pastagens, são locais muitas vezes de difícil acesso, com terrenos bem acidentados, extremas imperfeições, com pedaços de madeira espalhados, pedras e etc. Nesses locais, é difícil para um operador se manter em pé, mesmo cercado por uma gaiola, em cima do tanque, e ainda tem o problema de comunicação entre o operador do trator com o operador do canhão de água, visto

que em muitas situações a máquina precisa estar em movimento enquanto é feita a aplicação de água para apagar o incêndio.

Como citado, o operador acaba ficando exposto aos perigos do fogo, uma vez que a capacidade de proteção da gaiola é quase nula. Em locais de extrema seca, ou de períodos de estiagem muito grandes, faz com que uma série de fatores ajudem a agravar o poderio do fogo. É o caso do centro-oeste brasileiro, onde pode haver tempos de estiagem próximo a sete ou oito meses do ano, elevando assim a temperatura ambiente, diminuindo a umidade do ar, e tornando plantas e ervas daninhas muito rígidas e secas, agravando ainda mais focos de incêndios.. Além disso, é possível notar que é uma região com fortes ventos, o que pode ajudar no alastro do fogo, inclusive jogando-o contra operador e máquina.

1.6 SOLUÇÃO

Como comentado na seção 1.5, há uma necessidade de melhoria no equipamento, buscando trazer cada vez mais comodidade para o operador da máquina, bem como trazer qualidade para o seu serviço, eliminando-o de potenciais riscos na hora de apagar incêndio.

Para isso, foi discutido uma melhoria no equipamento para atender as necessidades do cliente. Essa melhoria consiste em desenvolver um controle remoto para operar a máquina a distância, elevando assim o nível de segurança e podendo dar mais autonomia para quem for aplicar o produto, visto que ele poderá se locomover e entender melhor onde aplicar o jato de água.

Com isso, também eliminamos a falta de comunicação entre operador do trator e operador do tanque, uma vez que os dois podem estar juntos no trator, ou então o operador do trator se locomover pela lavoura e o operador do canhão apenas ir direcionando o jato de água para o local que há fogo, bastando estar num ponto que atenda a comunicação.

E o principal risco que eliminamos é o risco de possíveis queimaduras no operador. Como podemos ver na Figura 1, quem está operando o canhão de água, fica em uma espécie de gaiola. Isso elimina possíveis riscos relacionados a cair de cima do tanque. Porém, como já dito, em épocas de seca e pouca umidade, o risco aumenta, pois além de termos esses dois fatores, ainda temos os fortes ventos que podem assolar o local. Assim, jogando labaredas de fogo contra o tanque, impedindo que o operador fique em cima do tanque, e faça o manuseio do canhão de água.

1.7 METODOLOGIA DE TRABALHO

Para conseguirmos chegar em uma solução viável, foi analisado o que o mercado apresenta em questão de produtos similares e quais seriam as necessidades de

cada cliente usuário do tanque MF. A avaliação da solução se dá em cima de algumas condutas que a empresa traz para dentro da fábrica, e a principal delas é citada já, que são os *feedbacks* dos clientes.

Como citado, o controle atual vem com fio e isso traz um valor agregado alto no custo final do produto, algo em torno de R$ 8.000,00 a mais, pois além de ter mais matéria prima presente, a empresa fornecedora também precisa tirar seu lucro.

Seguindo então a lógica de redução de custos, foi estudado no mercado os componentes que poderiam ser usados para a fabricação interna do controle remoto *wireless*, e decidido qual seria o melhor caminho a ser tomado. A redução de custos deveria ser de, no mínimo, 50% do que o controle com fio.

A fabricação de componentes não eletrônicos, foi realizada dentro da empresa, e apenas componentes eletrônicos necessários para a realização do processo foram comprados de fora. Os componentes que foram utilizados vão desde microcontroladores, programados em *C++*, até botões e relés para acionamento de equipamentos elétricos.

Além de um controle remoto, também seguindo a lógica de redução de custos, foi pensado na solução para controle do canhão através de um aplicativo para celulares. Esse aplicativo foi desenvolvido em *Flutter*, um kit de desenvolvimento de interface do usuário, de código aberto, criado pelo *Google*, em 2015, baseado na linguagem de programação *Dart*, que possibilita a criação de aplicativos compilados nativamente para os sistemas operacionais iOS, Android, Windows, Mac, Linux e Web (FLUTTER, 2015).

ESTRUTURA DO DOCUMENTO

No capítulo 2 é feita uma descrição do produto, e qual a sua utilidade dentro dos produtos realizados pela empresa. A tecnologia que lhe é empregada, e a forma como é utilizada essa tecnologia.

No capítulo 3 é apresentada uma fundamentação teórica, onde é discutido quais componentes e tecnologias foram empregadas durante o processo de construção do projeto, buscando demonstrar sua usabilidade.

No capítulo 3.2 é exposto o desenvolvimento do projeto, tanto a parte física quanto a parte de programação, buscando demonstrar onde e quando foram utilizados os componentes e quais suas funcionalidades.

No último capítulo é feita a conclusão do relatório, demonstrando os resultados adquiridos, e apontando melhorias que ainda serão feitas no projeto que entrará em produção.

2 TANQUE MATA FOGO

Como apresentado na seção 1.4, o tanque Mata Fogo é um equipamento para transporte de água, com o intuito de fazer o combate a incêndios em lavouras, galpões e também em incêndios em máquinas agrícolas, como no caso tratores e colheitadeiras.

Figura 1 – Tanque de combate a incêndio Mata Fogo.

Fonte: Documento interno da São José Industrial, 2022.

A tecnologia do tanque Mata Fogo tem o poder de combate ao fogo com uma eficiência até dez vezes maior que os tanques agrícolas de combate ao incêndio convencionais.

A tecnologia consiste em um tanque agrícola especialmente projetado para elevar o tempo de combate ao incêndio com o conceito de alta pressão e baixa vazão. O tanque possui dois reservatórios sendo um principal para água e outro para o condimento da exclusiva calda Mata Fogo F500 EA (SAFETY, 2022) que são misturados através de válvulas e misturadores especialmente projetado para este equipamento. Possui uma bomba de diafragrama, com capacidade de até 2,0 l/min, suportando pressões de até 150 PSI (10,3 BAR), com uma elevação de produto de 2,4 metros e tensão de entrada de 12V. Possibilita também que o tanque possa ser utilizado para diversas fi-

nalidades como lavagem de máquinas, galpão, entre outros afazeres que utilize apenas água, mas estando sempre preparado para o combate eficiente a incêndios através da simples abertura da válvula misturadora da calda Mata Fogo F500 EA. Evitando assim desperdícios e garantindo a economia.

A calda Mata Fogo F500 EA é um produto encapsulador de hidrocarbonetos, patenteado, sendo o número um em combate a incêndios tridimensionais nos EUA e em mais de 50 países. Com o kit Mata Fogo F500 EA, além da redução do tempo de combate ao incêndio, bloqueia a reignição do fogo, auxilia na criação de barreiras para melhorar o controle do fogo, além de diminuir a produção de fumaça tóxica, melhorando assim a visibilidade de equipes e operadores para atingir diretamente o ponto principal do incêndio.

O tanque Mata Fogo sai de fábrica com alguns itens básicos e padrões. O Mata Fogo tem opção de comercialização de três modelos: MF 6.500, MF 10.000 e MF 12.000. Nos tanques de 6.500 litros e 10.000 litros, possui uma tampa traseira que pode ser totalmente removida para fazer limpeza interna no tanque. No modelo de 12.000 litro não possui essa tampa, pois quando cheio a pressão aumenta, podendo haver rompimento dos parafusos borboleta.

Todos os modelos contam com uma bomba lobular 4", com vazão de 1.333l/min com acionamento hidráulico. Mangueira de abastecimento, gaiola superior para comando do canhão. Canhão de água manual, com vazão de 200l/min e reservatório de calda de 120 litros.

Nos tanques de 12.000 litros ainda contam com um opcional interessante, que é a mangueira esguicho de 25 metros. Ela fica em um carretel acoplado ao lado do tanque, podendo haver combate a incêndio em nível de solo. Nesse modelo de tanque, tem uma válvula direcional que pode ser aplicada a saída para o canhão de água, para o carretel ou para ambos ao mesmo tempo.

Um dos diferenciais nos modelos de 10.000 litros e 12.000 litros, é que existe o opcional do canhão eletrônico, como citado na seção 1.5, canhão esse da empresa N. MICHELIN (N.MICHELIN, 2022), que fica responsável por fazer toda a parte elétrica e eletrônica presente no canhão, bem como a instalação de motores que fazem a direção do canhão.

A Figura 1 tem um tanque com o canhão eletrônico já instalado. Esse canhão possui o *joystick* com fio como padrão de fábrica. A distância do fio pode ser variada de acordo com o cliente e a sua necessidade, mas por padrão se utiliza um cabo de 10 metros.

Como citado na seção 1.6, há uma necessidade de proteger ainda mais o operador da máquina, e elevar a segurança de toda a equipe envolvida no combate.

2.1 ACESSO

O sistema que possui um canhão eletrônico, é todo desenvolvido para atender a necessidade da utilização do *joystick* com fio.

O que buscamos desenvolver nesse projeto foi utilizar toda a estrutura física do equipamento como motores, cabos, conexões, e fazer o sistema rodar com um *joystick* sem fio. Para isso, utilizamos conexões acoplada as conexões já existentes no sistema com fio, mantendo as características gerais do sistema, porém com acesso remoto.

No projeto em questão, iremos fazer com que dois microcontroladores se comuniquem e troquem informações entre si. Um dos microcontroladores, presente no *joystick* será o responsável por enviar dados ao segundo microcontrolador que ficará acoplado no tanque, esperando instruções para executá-las.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, faz-se uma contextualização acerca dos conceitos e tecnologias utilizados durante o projeto, os quais são necessários para o entendimento completo do trabalho desenvolvido. Dentre os tópicos abordados, comenta-se sobre: os tipos de programação para aplicativos móveis, o *framework Flutter* em mais detalhes, programação em *C++* para desenvolvimento do *joystick*, componentes eletrônicos utilizados para o desenvolvimento, conceitos básicos de eletrônica e dimensionamento de sinais.

3.1 COMPONENTES ELETRÔNICOS

Para a confecção do controle *joystick*, estávamos limitado ao tamanho do produto. Pensando na ergonomia do produto, ele deveria ser de fácil uso, com botões de fácil acesso, tornando a utilidade dele requerida e proveitosa. Porém, temos que pensar na junção dos componentes dentro do controle, e as conexões que facilitariam a sua confecção.

No decorrer dessa seção irei mostrar os componentes que foram utilizados, quais suas propriedades, e qual a sua função em relação ao projeto.

3.1.1 Chaves Seletoras

As chaves seletoras são interruptores simples, usados para abrir ou fechar circuitos elétricos (ALVARO, 2020). Conectadas em um microcontrolador, conseguem enviar sinais de *HIGH* e *LOW* para o sistema, identificando assim se estão em sinal de nível lógico alto ou baixo. Existem diversos modelos de chave seletora para utilização em projeto, e as definidas para esse em questão, foram as chaves alavancas, como na Figura 2. Quando conectadas a um microcontrolador, podemos obter o sinal lógico, e com essa informação fazer alguma ação.

Figura 2 – Chave Seletora tipo Alavanca.

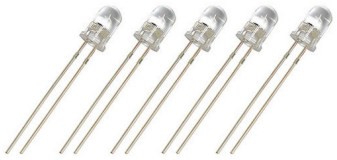
Fonte: Loja Baú da Eletrônica.

A usabilidade delas dentro do projeto se dá para que possamos ter controle sobre alguns componentes dentro do tanque MF. A escolha por esses componentes é porque que são de fácil acesso no mercado, e conseguimos obter os seus sinais para fazer o controle. Quando acionadas, podemos ter controle sobre os componentes presentes no tanque. Botões do estilo *switch*, podem ser encontrados aos montes no mercado, porém só conseguimos utilizá-los quando devemos acionar algo diretamente conectado nele.

3.1.2 LED

A palavra LED origina do inglês "Light Emitting Diode", que significa Diodo Emissor de Luz. Os LED's geram a luz através de um meio sólido maciço, enquanto que nas lâmpadas incandescentes, por exemplo, a luz é gerada através de um filamento que quando aquecido, incandesce (CIRCUITO, 2021). A tensão de operação sobre os seus terminais depende de cada cor de *LED* e também de cada fabricante. Podendo variar de 1,9V nos *LED's* da cor amarela por exemplo, até 3,2V nos *LED's* da cor branca.

Figura 3 – LED Alto Brilho.



Fonte: FilipeFlop, 2020.

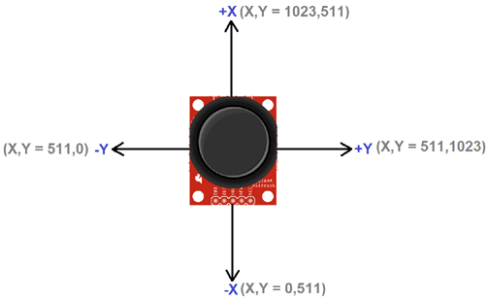
A utilização dos *LED's* é importante para que possamos manter o operador em alerta, indicando para ele se o componente que ele está tentando controlar está de fato em funcionamento. A escolha por *LED's* de alto brilho se dá pois podemos utilizar o controle em vários locais e terrenos diversos, e por isso precisa ser bem alertado o que está acontencendo na mão do operador. Existem no mercado diversos tipos de *LED*, mas são os chamados *LED's* difusos, que são elementos que possuem uma luz interna e são pintados de determinada cor por fora. Esses componentes tendem a possuir uma mesma tensão, e portanto não precisam ter certos cuidados com resistores na conexão de fios. Porém, são mais fracos e são menos duradouros. Por isso a escolha se deu

para *LED's* de alto brilho, que possuem uma vida útil muito maior e uma visualização melhor.

3.1.3 *Joystick* 3 eixos

Este *joystick* é um módulo com dois potenciômetros de 10k Ohms bidirecionais de saída analógica que controlam os eixos X e Y, e um botão central com saída digital para controle do eixo Z (OLIVEIRA, 2020). A alimentação deste componente se dá através do pino de 5V. O restante dos pinos são para o aterramento, para o valor lido do eixo X e o valor lido do eixo Y. Além disso, ainda possui um pino representando o eixo Z, que nesse projeto não foi utilizado.

Figura 4 – *Joystick* 3 eixos.

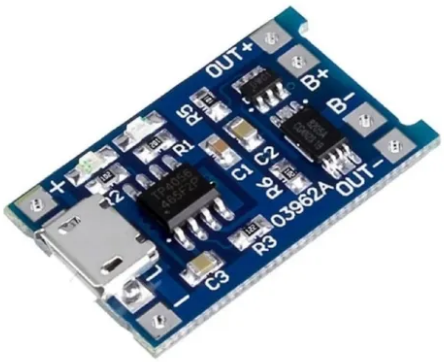


Fonte: Google, 2022.

3.1.4 Módulo Carregador de Bateria de Lítio

Este módulo é ideal para carregar baterias de Lítio. A placa possui um circuito de carga, proteção contra sobrecarga e fácil conexão por cabo micro USB. A bateria não precisa ser removida do circuito e nem trocada. Possui *LED's* para demonstrar quando a bateria está completa ou não. Baseado no chip TP4506, que é um carregador linear completo de corrente constante e tensão constante para célula única de baterias (CALAZANS, 2020).

Figura 5 – Módulo Carregador de Bateria de Lítio.



Fonte: Google, 2022.

A utilização de baterias para o controle é essencial, e precisa-se manter uma segurança quanto ao seu uso. Para tal, é preciso dimensionar a bateria para um tamanho adequado de tensão fornecida, mas também é imprescindível que se possa recarregar essa bateria quando não está usando o controle. O módulo em questão, consegue fornecer informações referente a quanto da bateria está carregada, ou quanto está descarregada. Importante para a continuidade do projeto.

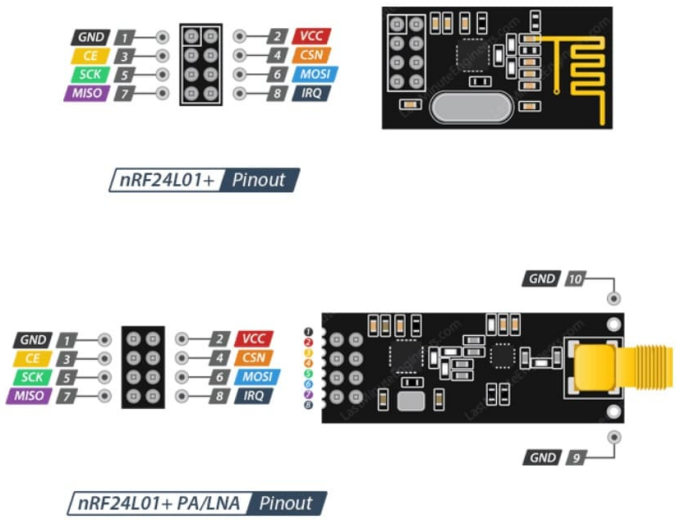
3.1.5 Módulo Wireless NRF24L01

O módulo transceptor *wireless* NRF24L01 trabalha numa frequência de 2.4 GHz e possui uma antena embutida, o que faz desse módulo um dos mais compactos do mercado (ENGINEERS, 2018). Possui um conector de sete pinos, como na Figura 6. Como esses pinos são muito próximos um dos outros, o ideal é que se trabalhe com esse módulo com algum outro tipo de conector, como o jumper macho-fêmea, e ligá-lo diretamente aos pinos do microcontrolador.

As conexões que nos interessam ficam restritas as apresentadas na Tabela 1. No *chip* interno do módulo, ainda existem outras conexões, como por exemplo a interface da antena.

Alguns recursos referentes a esse módulo:

Figura 6 – Módulo Wireless nRF24L01.



Fonte: *How To Mechatronics*, 2019.

- Taxa de transmissão de dados ao ar livre de 1 Mbps ou 2 Mbps;
- Velocidade da interface digital (SPI) de 0 Mbps até 8 Mbps;
- Até 125 canais de rádio frequência para operação;
- Baixa tensão de operação, variando de 1,6V a 3,6V;
- Compatibilidade com diversos modelos de módulo NRF24XX.
- Módulo sem antena auxiliar pode enviar dados em até 50 metros em locais abertos;
- Módulo com antena auxiliar pode enviar dados em até 100 metros em locais abertos.

Uma vantagem de se usar esse módulo é que a tensão de alimentação é baixa, e por tanto pouco consumo de energia para operar, e também o consumo de corrente é muito baixo, apenas 9,0 mA com uma potência de saída de -6 dBm e 12,3 mA no modo RX. Enquanto que os microcontroladores, em cada pino, podem fornecer ou receber uma corrente máxima de 40 mA.

A funcionalidade dos pinos apontados na Tabela 1 são descritas abaixo:

- GND: é o pino Ground. Esse pino é responsável por fazer o aterramento do módulo;

Tabela 1 – Pinagem do módulo *wireless* NRF24L01.

Pino	Nome	Função	Ligação Microcontrolador
1	GND	Terra	GND
2	VCC	Alimentação	3,3V
3	CE	Chip Enable RX/TX	Pino 7
4	CSN	SPI Chip Select	Pino 8
5	SCK	SPI Clock	Pino SCK
6	MOSI	SPI Slave Data Input	Pino MOSI
7	MISO	SPI Slave Data Output	Pino MISO
8	IRQ	Interrupção	Não utilizado

Fonte: *All Datasheet*, 2020.

- VCC: Fornece energia para o módulo. Ele pode variar de 1,9V até 3,6V, podendo ser conectado a saída 3,3V do Arduino;
- CE (Chip Enable): é um pino ativamente alto (HIGH). Quando selecionado, o módulo vai receber ou enviar dados, depdendendo do modo em que estiver;
- CSN (Chip Select Not): é um pino ativamente baixo (LOW). Quando este pino está em *LOW*, o módulo começa a captar dados da porta SPI e processa-os de acordo com as especificações;
- SCK (Serial Clock): entrada de pulsos de clock providas pela conexão Mestre do SPI;
- MOSI (Master Out Slave In): entrada SPI para o módulo;
- MISO (Master In Slave Out): saída SPI para o módulo;
- IRQ: é um pino interruptor que avisa o Master quando novos dados estão disponíveis para serem processados.

3.1.6 Módulo Bluetooth BLE1010

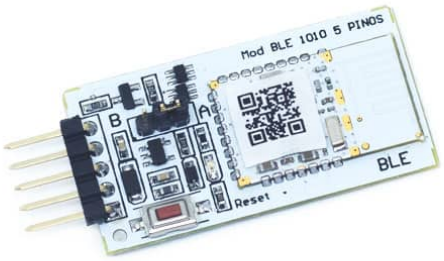
O módulo BLE-1010 4.0 é homologado pela ANATEL, e é um módulo para soluções em Bluetooth de baixo consumo, *"Bluetooth Low Energy"* (ELETRÔNICA, 2018).

Esse módulo faz dele um bom componente eletrônico devido a sua potência em questão de distância de alcance. Quando em local aberto, pode chegar a até 100m, tornando sua facilidade de uso muito grande. Possui 5 pinos para conexão sendo um para alimentação, um para aterramento, e dois pinos responsáveis pela comunicação. O quinto pino, denominado *ST*, serve para verificar o *status* da conexão.

A comunicação entre o módulo e o microcontrolador se dá através das portas UART *TX* e *RX*. Essa comunicação é unidirecional, isto é, os dados são enviados pelo transmissor e recebidos pelo receptor.

UART é o acrônimo de *Universal Asynchrounous Receiver/Transmiter* ou *Receptor/Transmissor Universal Assíncrono*. E como dito já, sua finalidade é possibilitar a transmissão e a recepção de dados originalmente disponíveis de forma paralela.

Figura 7 – Módulo Bluetooth BLE1010.



Fonte: FilipeFlop, 2022.

3.1.7 Driver Ponte H Dupla - L298n

O Driver Ponte H L298n é um módulo para cargas indutivas, relés, solenóides, motores *CC* e motores de passo. Este módulo é estruturado em cima do *chip* L298n. Utilizando este driver, é possível controlar independentemente a velocidade de rotação de dois motores DC ou de um motor de passo.

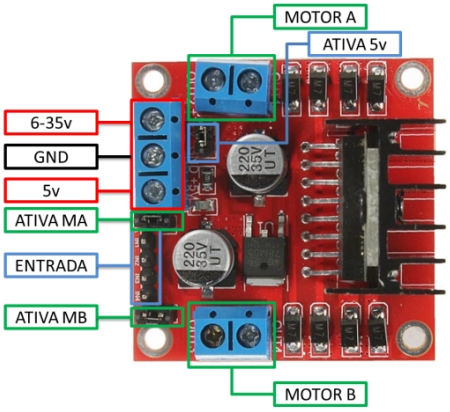
O módulo permite o controle de sentido de rotação de motores *CC* bem como a sua velocidade. Para isso, basta utilizarmos os pinos PWM do Arduino (NERY, 2020).

A tensão de operação deste módulo varia de 4V a 35V. Como o microcontrolador só consegue fornecer até 5V, o módulo possui um pino de entrada para tensão acima de 5V, podendo fazer o abastecimento de motores mais potentes. Corrente de operação máxima de 2A por canal ou 4A no máximo. Ainda, a corrente lógica varia de 0 mA a 36 mA, estando assim dentro da faixa de operação dos microcontroladores.

Como apresentado na Figura 8, podemos verificar a pinagem do Driver Ponte H:

- Motor A e Motor B: se referem aos conectores para ligação de dois motores DC ou um motor de passo;

Figura 8 – Driver Ponte H - L298n.



Fonte: Loja Baú da Eletrônica, 2022.

- Ativa MA e Ativa MB: pinos responsáveis pelo controle PWM dos motores A e B. Se estiver com jumper, não haverá controle de velocidade, pois os pinos estão ligados aos de 5V.
- Entrada: Este barramento é composto por IN1, IN2, IN3, e IN4. Sendo estes pinos responsáveis pela rotação do Motor A (IN1 e IN2) e Motor B (IN3 e IN4).

O pino denominado "Ativa 5V"é um regulador de tensão integrado. Quando o driver está operando entre 6-35V, este regulador disponibiliza uma saída regulada de +5V no pino (5V) para um uso externo, podendo alimentar por exemplo outro componente eletrônico. Então, é necessário manter o jumper que está nesse pino para que não haja danos ao microcontrolador, quando estiver trabalhando com tensões maiores que 5V.

A entrada denominada "6-35V"é onde será conectada a fonte de alimentação externa quando o driver estiver controlando um motor que opere nessa faixa de tensão.

3.1.8 Módulo Relé 2 Canais - 5V

Os relés são componentes eletromecânico, isso é, eles tem uma parte elétrica e outra mecânica. Eles são muito utilizados para acionar cargas maiores do que a tensão dos microcontroladores (ENGINEERS, 2019). Um exemplo é o acionamento de bombas 12V, lâmpadas 12V e afins.

O relé em si possui dois pinos que estão conectados a uma bobina e três pinos de saída, chamados de *comum*, *normalmente aberto* e *normalmente fechado*. Quando a bobina não está energizada, o *comum* fecha o circuito com o *normalmente fechado*.

Figura 9 – Módulo Relé de 2 Canais - 5V.

Fonte: ArduRobotica, 2021.

Quando a bobina é energizada, o *comum* fecha o circuito com o *normalmente aberto*. Desse jeito, pode-se ligar e desligar circuitos conectados a esses pinos.

O módulo relé da Figura 9 possui três pinos que irão conectados ao microcontrolador: Vcc (5V), IN (entrada de controle do relé) e GND. E três pinos de saída como já citados.

No caso do módulo de dois canais, ele pode controlar até duas saídas, e dessa forma fazer o controle sobre o fluxo de tensão de dois equipamentos.

3.1.9 Resistor

Os resistores são componentes eletrônicos onde a sua principal função é limitar o fluxo de cargas elétricas por meio da conversão da energia elétrica em energia térmica. São feitos a partir de materiais dielétricos, de grande resistência (101, 2019).

A unidade de medida de um resistor é o Ω (Ohm). E seu dimensionamento deve seguir a Lei de Ohm: $R = \frac{U}{i}$. Onde "U"é a tensão ou potencial elétrico (V) e "i"é a corrente elétrica.

Os resistores são classificados em vários tipos, e para identificá-los eles possuem faixas de cores diferentes. Na Figura 10 temos uma tabela onde podemos calcular a resistência do resistor através das cores apresentadas em sua construção. Resistores padrão possuem quatro faixas de identificação. Então, devemos levar em conta a 1ª faixa que é um valor, a 2ª faixa que também é um valor, a 4ª faixa que é fator multiplicador e a 5ª faixa que é a tolerância do resistor.

Existem resistores das mais variadas dimensões de carga. Para o seu uso em circuitos é necessário saber dimensionar a sua usabilidade. Não precisamos, por exemplo, usar um resistor de 10k Ohm para acionar uma lâmpada se a nossa tensão for

Figura 10 – Tabela de códigos para identificar resistores.

COR	1ª FAIXA	2ª FAIXA	3ª FAIXA	4ª FAIXA	5ª FAIXA
	VALOR	VALOR	VALOR	MULTIPLICADOR	TOLERÂNCIA
PRETO		0	0	X 1 Ω	
MARROM	1	1	1	X 10 Ω	Mais ou menos 1 %
VERMELHO	2	2	2	X 100 Ω	Mais ou menos 2 %
LARANJA	3	3	3	X 1.000 Ω	
AMARELO	4	4	4	X 10.000 Ω	
VERDE	5	5	5	X 100.000 Ω	Mais ou menos 0,5%
AZUL	6	6	6	X 1.000.000 Ω	Mais ou menos 0,25%
VIOLETA	7	7	7	X 10.000.000 Ω	Mais ou menos 0,1%
CINZA	8	8	8		Mais ou menos 0,05%
BRANCO	9	9	9		
DOURADO				X 0.1 Ω	Mais ou menos 5 %
PRATA				X 0.01 Ω	Mais ou menos 10 %
SEM COR					Mais ou menos 20 %

Fonte: Aprendendo Elétrica, 2020

baixa, pois assim estaremos "limitando"demais a tensão e a corrente, e não acendendo a lâmpada.

Na Figura 11 podemos ver um conjunto de resistores de 220Ω. A primeira faixa é vermelha com um valor 2, a segunda também é vermelha com um valor 2 e a terceira faixa é marrom indicando o fator de multiplicação de 10Ω, com uma tolerância de $\pm$5% identificado pela faixa dourada.

Figura 11 – Resistor de 220 Ω.

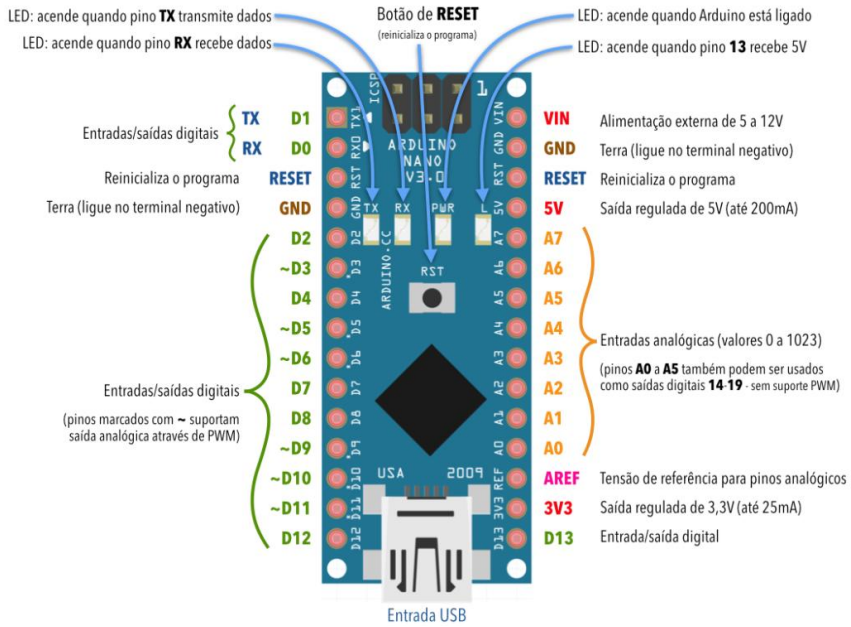
Fonte: FilipeFlop, 2022.

3.1.10 Microcontrolador - Arduino Nano

Os microcontroladores, como o próprio nome diz, são pequenos computadores num único circuito integrado o qual contém um núcleo de processador, memória e periféricos programáveis de entrada e saída. A memória de programação pode ser RAM, NOR flash ou PROM. A qual, muitas vezes, é incluída no chip. Esses dispositivos são normalmente utilizados para controlar as coisas a partir de informações que são passadas para eles. No caso do nosso *joystick*, o microcontrolador fica responsável por acionar motores, enviar informações e fazer leitura de variáveis como dos botões.

O microcontrolador escolhido para ser o transmissor de dados foi o Arduino Nano (ARDUINO, 2014), mostrado na Figura 12. O Arduino Nano é uma excelente escolha para projetos pequenos/médios pois dentre várias características, temos o tamanho reduzido, e sua tensão de entrada, que deve ser entre 7V e 12V, o que uma bateria de 9V consegue atender perfeitamente. Além disso, a escolha do microcontrolador ter sido o Arduino Nano, também se deve ao fato de que precisávamos reduzir custos, e ele é amplamente visto no mercado com um valor acessível.

Figura 12 – Arduino Nano V3.0.



Fonte: Fórum Arduino, 2021.

O Arduino Nano possui 14 pinos digitais, sendo que 6 destes também servem para saídas PWM, 8 portas de entradas analógicas, corrente DC por pino de 40 mA, velocidade de clock de 16 MHz e um processador Atmel Atmega 328, presente na grande maioria dos microcontroladores.

Os pinos analógicos recebem valores de 0 a 1023, enquanto os pinos digitais recebem valores de 0 a 255.

3.1.11 Microcontrolador - Arduino Mega 2560

O microcontrolador Arduino Mega 2560, Figura 13, foi desenvolvido para projetos mais complexos, com mais componentes conectados, podendo atender a vários sensores e atuadores ao mesmo tempo. Tem um processador Atmel Atmega 2560, com clock de 16 MHz, porém com mais memória que os microcontroladores menores (ARDUINO, 2022b).

O clock do processador é o que determina a velocidade com que o processador irá trabalhar e é importante salientar , pois tanto no caso do Arduino Nano apresentado na seção 3.1.10 quanto no Arduino Mega da Figura 13, eles suportam realizar 16 milhões de instruções por segundo. Lembrando que 1Hz quer dizer um ciclo por segundo, e por tanto:

$$16MHz = 16000000Hz = 0,0000000625s = 62,5ns. \tag{1}$$

Cada operação leva 62,5ns para 1s, então temos:

$$\frac{1s}{62,5ns} = 16M. \tag{2}$$

Possui uma memória *flash* de 256 KBytes. Essa memória é responsável por armazenar dados sobre os programas. Ela é uma memória não volátil e de estado sólido, o que significa que não há partes móveis que possam ser danificadas.

Além disso, também possui 8 KBytes de memória estática SRAM. Essa memória é constituída por um circuito com transistores e esse modelo não precisa de atualização constante para manter os dados ativos. Isso faz com que possa ficar muito tempo sem precisar atualizar os dados que estão alocados nessa placa microcontroladora.

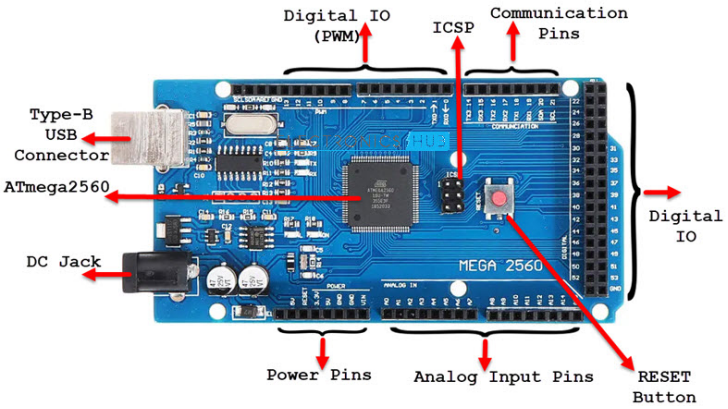
Além do mais, possui 4 interfaces seriais, o que significa que podem se comunicar com até quatro portas de comunicação. Essas portas utilizam pinos diferentes que possibilitam a comunicação *full-duplex*, em que a informação pode trafegar nos dois sentidos ao mesmo tempo.

O Arduino Mega 2560 possui 54 portas digitais, onde 15 destas podem ser utilizadas como PWM. Possui 16 entradas analógicas, o que traz uma facilidade para fazer projetos complexos, pois possui uma ampla variadade de portas que servem para os mais variados componentes eletrônicos.

Na Figura 14 podemos ver com mais clareza quais são os pinos e suas funções no Arduino Mega.

O Arduino Mega consegue trabalhar com uma fonte de alimentação externa de 6V a 20V. No entanto, se a fonte for menor que 7V, o pino 5V pode fornecer menos que cinco volts, e a placa se mostrar instável. E se for maior que 12V, o regulador de tensão embarcado na placa pode super aquecer e danificá-la. A alimentação externa

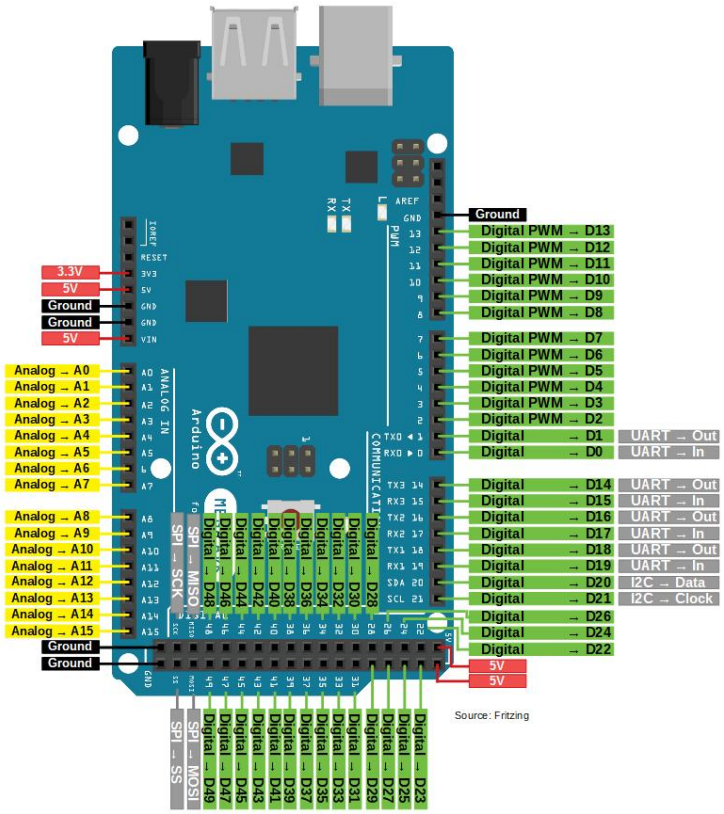
Figura 13 – Arduino Mega 2560.



Fonte: Eletronic Hubs, 2022.

se dá através do cabo estilo *jack*, encontrado em demasia no mercado, ou através do cabo USB conectado diretamente no computador.

Figura 14 – Esquemático do Arduino Mega 2560.



Fonte: Embarcados, 2020.

O Arduino Mega 2560 escolhido para fazer parte do projeto, é um componente, dentro do que buscava-se, um tanto quanto grande e espaçoso. Por isso, a escolha se

dá quanto a sua utilização por parte da central de comunicação presente no tanque de água. Além do mais, para realizarmos todos os testes e fazer todos os componentes funcionarem de forma conjunta, precisaríamos de portas digitais suficientes. Dessa forma, a utilização de um segundo Arduino Nano, não seria suficiente, deixando assim algum componente ficar de fora do projeto, o que o tornaria inviável.

3.2 PLATAFORMA DE DESENVOLVIMENTO - IDE ARDUINO

Um ambiente de desenvolvimento integrado (IDE - Integrated Development Environment) é um software para criar aplicações utilizando ferramentas de desenvolvimento em uma determinada linguagem de programação. A IDE Arduino é uma aplicação de plataforma cruzada, escrita em funções de *C* e *C++*. É usada para escrever e fazer upload de programas em placas compatíveis com o Arduino. Mas com bibliotecas de terceiros, também pode ser utilizada para fazer upload de códigos de placas de outros desenvolvedores (ARDUINO, 2022a).

A IDE do Arduino permite escrever códigos referenciando os componentes que irão ser utilizados. Através da IDE também é possível identificar a placa que vai ser utilizada para ser feito o upload do código na placa correta. Existe a possibilidade ainda de instalar novos pacotes com placas que não estão listadas na própria IDE, aumentando assim a gama de microcontroladores que podem ser programados pela interface.

A linguagem segue alguns padrões e, se o programa não for simples, começa-se identificando as bibliotecas que serão utilizadas e farão parte do projeto. As bibliotecas tem funções desenvolvidas especificamente para executar tarefas determinadas como por exemplo, configurar o GPS de um determinado componente eletrônico. A utilização dessas bibliotecas facilita o desenvolvimento do programa, tornando assim o código mais simples e organizado. Um exemplo de inclusão de bibliotecas pode ser visto na Figura 15, onde na primeira linha temos a inclusão da biblioteca SPI, esta biblioteca permite que você se comunique com dispositivos SPI, através do microcontrolador. A segunda biblioteca incluída é responsável por *printar* o que vem na porta SPI e mostrar para o usuário. Por fim, a inclusão da biblioteca *RF24* que é o driver de rádio para os módulos NRF24L01, uma biblioteca de comunicação.

Figura 15 – Exemplo de código IDE Arduino - Inclusão de bibliotecas.

```
#include <SPI.h>
#include "printf.h"
#include "RF24.h"
```

Fonte: IDE Arduino, 2022.

A seguir, são declarados os pinos que serão utilizados para conexão entre componente e microcontrolador e por quais componentes serão utilizados. E aqui, nessa parte do código, é preciso identificar as variáveis, se são do tipo *int* (inteiro), *bool* (verdadeiro ou falso), *char* (caractere), *array* (arranjo ou lista), *float* (flutuante), *string* (sequência de caracteres), entre outras mais que são importantes, que estão

na documentação da biblioteca Arduino (REFERENCE, 2022). Podemos ver na Figura 16 a declaração do tipo de variável e os pinos que irão representar o componente na placa microcontroladora. No código demonstrado, o projeto referente a ele terá quatro botões e um *LED* que acende quando algum desses botões é acionado. Os botões representam pinos digitais, e retornam um estado de *HIGH* (alto) ou *LOW* (baixo), ou então, como referenciado no código, irão retornar 0 ou 1. Então, declaramos os pinos que irão representar os botões como *int*, e o pino que irá representar o *LED* também como *int* pois representa um estado de *HIGH* ou *LOW* também. É declara três variáveis juntamente com os pinos que guardam os estados que os botões se encontram, para depois fazer comparativos dentro do código.

Figura 16 – Exemplo de código Arduino - Pinagem.

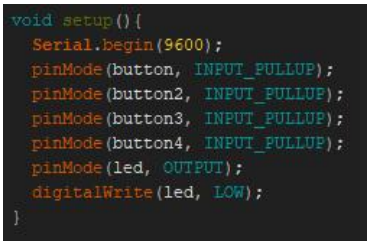
```
const int button = 4;
const int button2 = 3;
const int button3 = 5;
const int button4 = 6;
const int led = 11;

int preButtonState;
int buttonState, buttonState2;
int buttonState3, buttonState4;
```

Fonte: IDE Arduino, 2022.

Na Figura 17 temos uma estrutura que é de suma importância e precisa estar em todos os programas que forem desenvolvidos para microcontroladoras. O *setup*, como o nome mesmo diz, é a parte de "configurar"os componentes que irão fazer parte do projeto. Alguns componentes são do tipo *INPUT*, ou seja, é lido o estado atual do componente. No caso dos botões, eles são declarados como entradas pois, quando pressionados, eles fecham o circuito interno e retornam um valor inteiro para o usuário, configurando assim um cenário de liga e desliga. Os *LED's* são componentes de saída, por isso a declaração de *OUTPUT*, pois quando precisam ser acionados, é enviado um sinal para o *LED*, ou seja, é um pino de saída. Ainda, no caso de *LED's*, por exemplo, para acionamento dos mesmos é necessário enviar um sinal, via porta digital, para que o estado do *LED* seja *HIGH* ou *LOW*. No estado de *HIGH* o *LED* deve acender, e em *LOW*, apagar. Por isso, é totalmente configurável o estado inicial do *LED* dentro do *setup*.

Na segunda linha do código, podemos ver que é declarado um *begin* na porta *Serial*. Essa parte do código faz a liberação para que o usuário possa "assistir"o que está sendo comunicado entre componente e microcontrolador, através do *Monitor Serial*, presente na IDE do Arduino. Esse acompanhamento só poderá ser feito quando conectado o microcontrolador no computador, pois é através do cabo USB que é lida essas variáveis.

Figura 17 – Exemplo de código Arduino - *setup*.



Fonte: IDE Arduino, 2022.

No código demonstrado na Figura 18 temos a função *loop*. Essa função faz precisamente o que o seu nome sugere, e repete-se consecutivamente enquanto a placa estiver ligada, permitindo que o programa mude e responda a essas mundanças.

Dentro da IDE do Arduino podemos fazer várias funções, como o usuário quiser e achar mais determinante, mas todos esses códigos precisam ser referenciados dentro da variável loop se eles quiserem ser executados. Então, pode-se criar uma função fora do *loop* desde que, ela seja chamada dentro do *loop* de qualquer forma.

No código da Figura 18 pode-se notar a presença da variável *buttonState*, essa variável recebe o valor de leitura do *button* que foi declarado no inicio do código. Essa leitura é feita através da leitura do pino digital, com a função *digitalRead*. Logo após os botões serem lidos, parte-se para a parte do código em que é tomadas as decisões, através da função *if*. Nesse caso em específico, podemos ver uma decisão bem simples, onde a função de decisão verifica se o estado do botão é *HIGH*, se for verdadeira essa informação, ele escreve para o usuário o estado do botão através da função *Serial.println()* e ao mesmo tempo coloca o pino digital do *LED* em *HIGH*, indicando para ligar. Se essa condição não for verdadeira, ou seja, se o botão não estiver em *HIGH*, o programa decide por desligar o *LED*, colocando-o em estado *LOW*.

E dessa forma, ele repete para os quatro botões a mesma decisão. Por fim, é colocado um *delay* de 1s para que o programa volte a executar o laço *loop* novamente.

A IDE do Arduino é simples e intuitiva de ser utilizada. Os passos para a configuração e realização de procedimentos com componentes eletrônicos se dá de forma bem limpa, com o usuário podendo interagir diretamente com esses componentes, e verificar o que está sendo comunicado entre ambos. Arquivos complexos exigem do processador do microcontrolador um pouco mais, porém, ainda assim, é realizado de forma simples, mantendo a integridade dos componentes e trazendo informações precisas.

Figura 18 – Exemplo de código Arduino - *loop*.

```
void loop(){
  buttonState = digitalRead(button);
  buttonState2 = digitalRead(button2);
  buttonState3 = digitalRead(button3);
  buttonState4 = digitalRead(button4);

  if(buttonState == HIGH){
    Serial.print("Estado Botao 1 Ligado: "); Serial.println(buttonState);
    digitalWrite(led, HIGH);
  } else {
    Serial.print("Estado Botao 1 Desligado: "); Serial.println(buttonState);
    digitalWrite(led, LOW);
  }
  if(buttonState2 == HIGH){
    Serial.print("Estado Botao 2 Ligado: "); Serial.println(buttonState2);
    digitalWrite(led, HIGH);
  } else {
    Serial.print("Estado Botao 2 Desligado: "); Serial.println(buttonState2);
    digitalWrite(led, LOW);
  }
  if(buttonState3 == HIGH){
    Serial.print("Estado Botao 3 Ligado: "); Serial.println(buttonState3);
    digitalWrite(led, HIGH);
  } else {
    Serial.print("Estado Botao 3 Desligado: "); Serial.println(buttonState3);
    digitalWrite(led, LOW);
  }
  if(buttonState4 == HIGH){
    Serial.print("Estado Botao 4 Ligado: "); Serial.println(buttonState4);
    digitalWrite(led, HIGH);
  } else {
    Serial.print("Estado Botao 4 Desligado: "); Serial.println(buttonState4);
    digitalWrite(led, LOW);
  }
  delay(1000);
}
```

Fonte: IDE Arduino, 2022.

3.3 *FLUTTER*

Flutter é um kit de desenvolvimento de interface de usuário, desenvolvida pelo Google, baseado em lingaugem de programação *Dart*, que possibilita a criação de aplicativos para sistemas Android, Web, iOS, Windows e etc (FLUTTER, 2015).

O *Flutter* é construído através de *widgets*, e esses *widgets* são construídos usando uma estrutura moderna que se inspira em *React* (REACT, 2022). A ideia é que cada desenvolvedor construa a sua prórpia interface de usuário a partir destes *widgets*.

O *Flutter*, como mencionado acima, utiliza da linguagem de programação *Dart*, uma linguagem também criada pelo Google, sendo essa linguagem compatível com a orientação a objetos e programação funcional (o que diminui muito a curva de aprendizagem). O *Dart*, assim como a linguagem de programação para microcontroladores, é fácil e simples de usar (GOOGLE, 2022).

A Figura 19 é um exemplo de código *Dart*, na plataforma de desenvolvimento Android Studio (DEVELOPERS, 2022). Quando inicia-se um projeto novo na plataforma, é gerado automaticamente um primeiro arquivo para exemplificação, e se for rodado, gerará um aplicativo bem simples com um botão de incrementação de uma

Figura 19 – Exemplo de código *Flutter - main*.

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),  // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );  // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }
```

Fonte: Android Studio, 2022.

variável inteira. Nesse exemplo, podemos ver a importação das bibliotecas no início do código. Logo após, é iniciado o desenvolvimento do aplicativo, e aqui fica a critério do desenvolvedor utilizar o nome do aplicativo que desejar. As classes que são criadas, são necessárias para o desenvolvimento e execução do projeto.

Dentro das classes também é feita toda a parte de inteligência do aplicativo, com funções que serão chamadas caso algum evento venha a acontecer em determinada parte do código. No exemplo da Figura 19 temos uma função para incrementar valores sempre que o botão for selecionado. Então, ela é invocada e adiciona um na contagem.

Na parte do código em que começa o *Widget build*, é a parte de criação da parte estática do aplicativo, e nesse em específico, quando for rodado o código, será possível ver na tela uma aplicação com uma barra de ferramentas azul com o título *Flutter Demo Home Page*.

Logo em seguida é criada a classe *StatefulWidget*, cujo *Widget* é responsável por criar a aplicação em si, criando a *Home Page* do aplicativo.

Figura 20 – Exemplo de código *Flutter - build*.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),  // AppBar
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const Text(
            'You have pushed the button this many times:',
          ),  // Text
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headline4,
          ),  // Text
        ],  // <Widget>[]
      ),  // Column
    ),  // Center
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: const Icon(Icons.add),
    ),  // FloatingActionButton
  );  // Scaffold
}
}
```

Fonte: Android Studio, 2022.

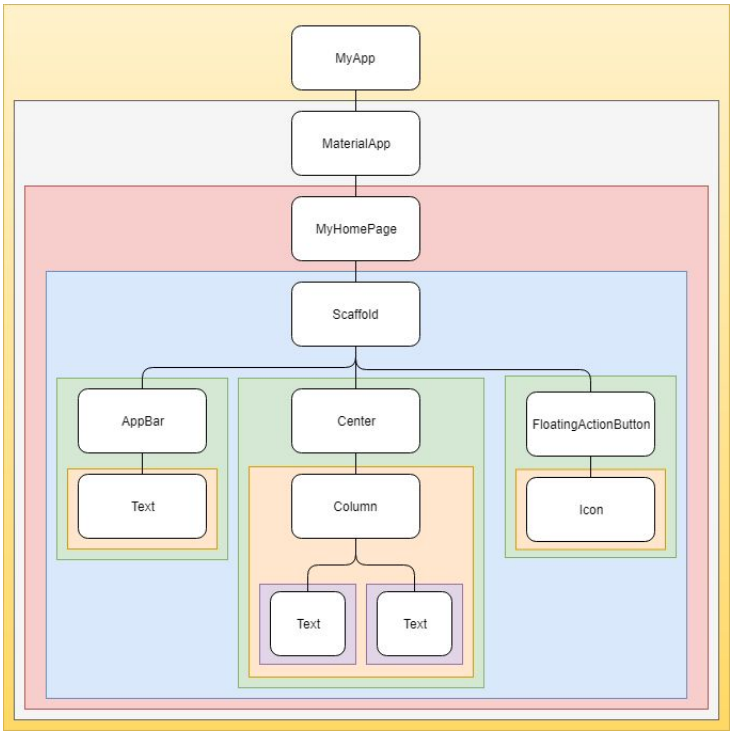
Na Figura 20 podemos ver a parte de criação da interface do usuário, e tudo que irá aparecer na tela para o usuário ver e interagir. Na linha de comando que temos o *body*, é onde começa as instruções para o aplicativo se centralizar na tela e se adaptar as diferentes telas que existirão. Dentro do "corpo"centralizado, comumente chamado como *widget* pai, temos um *child*, que é o *widget* filho. Nesse caso, cria-se um filho *Column* com alinhamentos a serem definidos de acordo com o desenvolvedor. Ainda, dentro de *child*, criam-se instâncias e variáveis denominadas *children* que, como no caso anterior, são os "filhos dos filhos", e assim por diante se for preciso desenvolver.

Algumas funções precisam ser invocadas para serem contabilizadas. É o caso da função *_incrementCounter*, que é demonstrada na Figura 20. Essa função é chamada quando o botão for pressionado, através da propriedade *onPressed* dentro da *child floatingActionButton*.

A Figura 21 demonstra como é a estrutura do código *Flutter* que foi citado

anteriormente.

Figura 21 – Estrutura de código do *Flutter*.



Fonte: Medium, 2019.

4 DESENVOLVIMENTO

Neste capítulo será apresentado os requisitos do sistema, quais são os passos que precisamos seguir para que o projeto funcione de acordo com o que esperamos. Também neste capítulo será apresentado os procedimentos utilizados para o desenvolvimento do projeto, bem como a utilização dos componentes citados na seção 3.1. Será ainda apresentada as soluções de desenvolvimento de software, os métodos utilizados, e toda a parte de aplicação de métodos com a interação entre eletrônicos.

Para a realização do projeto, é necessário saber que foi desenvolvido o rádio transmissor, conhecido por controle remoto, e acoplado ao tanque existe uma central que ficará responsável por receber os dados do controle e fazer as ações necessárias. Será apresentado os dois modelos de sistema, e todos os componentes juntamente com a sua programação e seu modelo de desenvolvimento.

Vale ressaltar que o autor deste trabalho contou com o auxílio dos colaboradores do setor de Engenharia de Produto da São José Industrial para tomar decisões de projeto e assimilar o funcionamento tanto dos sistemas físicos do tanque de água, quanto do sistema eletrônico presente no canhão de água. No entanto, o desenvolvimento em si do sistema foi completamente produzido pelo autor.

4.1 REQUISITOS DO SISTEMA

Os requisitos do sistema podem ser divididos em duas partes: os requisitos para funcionamento do controle remoto e os requisitos para funcionamento do aplicativo móvel. Os primeiros têm como finalidade definir o que o projeto deve ser capaz de realizar e executar em relação a comunicação *wireless*. Já os requisitos de funcionamento do aplicativo móvel, são mais flexíveis, uma vez que o *app* fica responsável por ser um auxiliar na gestão de controle.

Tendo em vista isso, apresenta-se os requisitos globais abaixo:

- Ser capaz de controlar a direção do canhão, girando na horizontal num eixo de 330°, na vertical 115°, e abertura do esguicho do canhão para obter a aspersão da água em modo jato ou névoa;
- Acionamento da bomba de água responsável pela mistura de calda e água, presentes no tanque;
- Acionamento do giroflex, presente no tanque, para aviso de proximidade de máqinas;
- Ser intuitivo para o cliente, ao ponto de apenas ligar e funcionar, sem muitos obstáculos;

- Fácil comunicação, minimizando os efeitos de perda de sinal ou perda de comunicação;
- Aumentar a área de controle do canhão, maximizando o seu uso para locais abertos.

4.2 CONTROLE REMOTO - PARTE FÍSICA DO RÁDIO TRANSMISSOR

Buscando trazer comodidade para o cliente final do produto tanque Mata Fogo (MF), demonstrado na seção 2, foi então desenvolvido um controle remoto via *wireless* para a utilização e o controle do tanque durante o combate ao incêndio. Para isso, todo a programação de linguagem para o controle foi desenvolvido na plataforma IDE Arduino, apresentado na seção 3.2, em conjunto com os componentes eletrônicos apresentados na seção 3.1.

Para a realização do projeto, foi optado utilizar para o controle remoto componentes que utilizassem a tecnolgia *wireless* para fazer a comunicação. Para que houvesse uma redução de custos, visto que hoje o mercado de componentes eletrônicos tem uma variedade grande de fabricantes do mesmo componente, e também para que se aumentasse a qualidade perante outros modelos presentes no mercado. A empresa já havia orçado de outros fornecedores um modelo de controle remoto, mas os valores cobrados por essas empresas faziam com que o produto final, o tanque Mata Fogo, ganhasse um valor agregado ainda maior, impossibilitando a competitividade de valores no mercado. Pensando nisso, começou-se o desenvolvimento do projeto.

O tanque Mata Fogo possui alguns componentes essenciais para o seu funcionamento, e dentre todos eles, há a bomba de diafragma que fica responsável por fazer a mistura do produto F-500 EA junto a água saindo do tanque. Então, alguns itens precisavam ser respeitados para que o projeto funcionasse da melhor forma possível, mantendo as características iniciais de um canhão de água manual.

Os requisitos para a construção do controle remoto eram os seguintes:

- Chave alavanca para acionamento da bomba;
- Chave alavanca para ligar giroflex que fica presente em cima do canhão de água;
- Chave alavanca para fazer o controle do atuador linear presente no canhão de água, responsável por mudar a posição do esguicho do canhão, podendo optar entre as opções jato de água e névoa;
- *LED's* para indicação de acionamento dos botões;
- Botão de direcionamento do canhão.

Seguindo esse pré-requisitos, foi feito o esboço do projeto e como ele deveria seguir para que pudesse ser feito o desenvolvimento do mesmo.

Buscando uma forma ergonômica, o controle precisava ser acessível, de fácil manuseio, e fácil transporte, evitando assim eventuais problemas com toques inesperados ou errados.

A Figura 22 mostra o controle remoto que foi desenvolvido e impresso, via impressora 3D, dentro da empresa. Podemos ver algumas imperfeições no projeto impresso, na parte cinza, e isso se deve ao filamento da impressora 3D ser de uma qualidade inferior ao da parte amarela. A intenção era desenhar e buscar um fornecedor externo para a parte plástica do controle, porém isso não foi conseguido fazer em tempo hábil devido a escassez de fornecedor.

Figura 22 – Protótipo do Controle Remoto impresso em impressora 3D.

Fonte: Arquivo pessoal, 2022.

Na parte física do controle remoto, vista na Figura 22, temos a chave 1 responsável por fazer a abertura e fechamento do bocal do canhão. A chave 2 fica responsável pelo acionamento da bomba, a chave 3 liga e desliga o giroflex, os *LED's* mostram qual dos botões está acionado e o *joystick* fica responsável por fazer a direção do canhão.

Na parte interna do controle, ainda na parte física, foi utilizado os seguintes componentes:

- Arduino Nano, igual ao apresentado na seção 3.1.10;
- Módulo NRF24L01;
- Resistores para fazer o acionamento dos *LED's*;

- Resistores para garantir que as chaves não oscilem entre os estados *HIGH* e *LOW*;
- Fios e conexões;
- Bateria de 9V;
- Entrada para carregador de bateria via cabo *jack*;
- Módulo carregador de bateria;
- Botão para ligar e desligar o controle remoto.

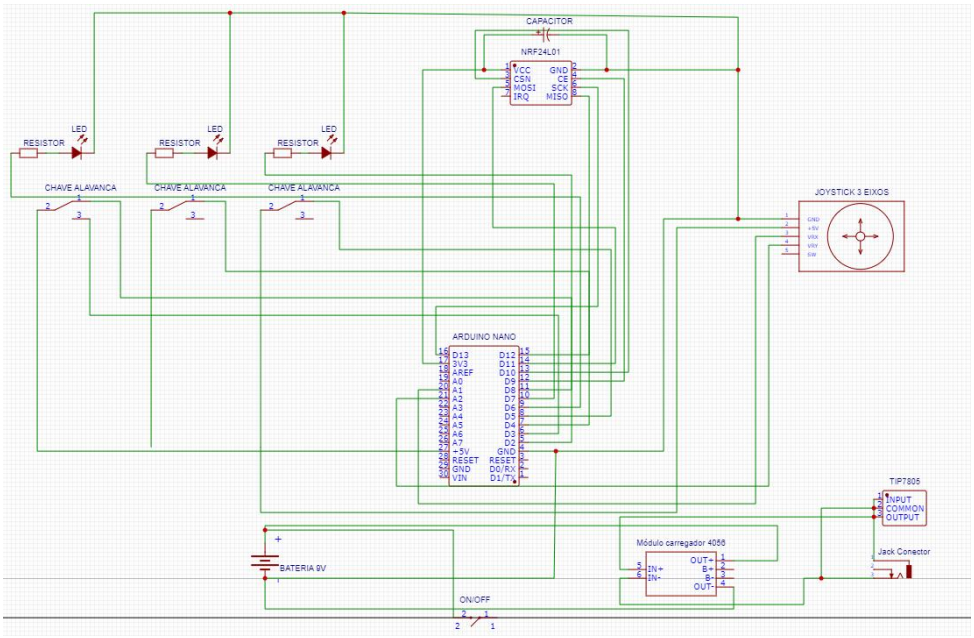
O esquemático apresentado na Figura 23 representa as conexões entre os componentes que foram utilizados para fazer a parte física do controle. Todos os componentes presentes na parte do controle são alimentandos pelo pino de 5V proveniente do Arduino Nano.

Para utilizar o módulo carregador da bateria, precisamos fazer a instalação do conector *jack*. Porém, temos o problema que a tensão entrada do cabo de alimentação para carregamento é de 12V enquanto que o módulo carregador só suporta 5V nas suas entradas. Então para corrigir esse problema, foi utilizado um transistor TIP7805 para fazer a regulagem da tensão, e na sua saída ter 5V alimentando o carregador.

Quanto a alimentação do Arduino Nano, como já citado na seção 3.1.10, ele suporta uma entrada de 9V, por tanto, pode ser conectada a bateria diretamente na placa, no pino denominado V_{in}, alimentando assim todo o circuito.

As chaves seletoras presentes no controle, por serem de material mais simples e portanto mais baratas, acabaram apresentando oscilações no retorno de estados que se encontravam. Para a realização do projeto, foram testados os componentes separadamente, e foram monitorados através do *Monitor Serial* da IDE Arduino. Durante os testes, foi percebido que quando desligada, a chave permanecia com valores em nível *HIGH*, que é o estado que se espera quando feita a programação responsável pela leitura do estado da chave. Porém, quando a chave era acionada, e por tanto deveria ir para estado *LOW*, acionando assim o seu resistor interno, ele fazia a chave oscilar muito entre estado *LOW* e *HIGH*, o que é muito ruim pensando num projeto que precisa ligar e desligar uma bomba, por exemplo. Portanto, para corrigir esse problema, foi utilizado um resistor na chave seletora, entre o pino de leitura digital do Arduino e a chave seletora. Dessa forma o problema foi corrigido e foi verificado que o acionamento das chaves não oscilava mais. Chegou-se a constatação que isso acontecia devido a simplicidade do material interno da chave. A escolha do valor do resistor neste caso foi feita através de testes, e, por tanto, não apresentado nenhum cálculo nesta seção. Os componentes escolhidos foram resistores de 1KΩ.

Figura 23 – Esquemático do transmissor.



Fonte: Arquivo pessoal, 2022.

Para fazer a conexão com os *LED's* responsáveis por demonstrar o acionamento das chaves, foi utilizado um resistor de 360 Ω. Isso se deve ao fato de que os *LED's* não suportam uma tensão alta, como já demonstrado na seção 3.1.2, e precisa-se que a tensão proveniente da bateria seja reduzida para não queimar os *LED's*. Esses resistores, puderam ser calculados através da Lei de Ohm, que implica que:

$$R = \frac{V}{i}. \tag{3}$$

O valor do resistor utilizado varia de acordo com a tensão que lhe é aplicado e com a corrente que é aplicado sobre seus terminais. No caso do controle, há uma bateria de 9V e a corrente aplicada nos terminais é proveniente da saída digital do Arduino, e portanto, a corrente designada para o acionamento é de 40 mA, porém o *LED* da cor vermelha trabalha na faixa de 1,8V a 2,0V e com corrente de 20mA. Então para fazer o cálculo dos resistores, se utilizou da Lei de Ohm citada na Equação 3, onde pudemos chegar no valor de:

$$R = \frac{V_{bateria} - V_{led}}{i} = \frac{9V - 2,0V}{20mA} \Rightarrow R = \frac{7V}{0,02A} = 350\Omega. \tag{4}$$

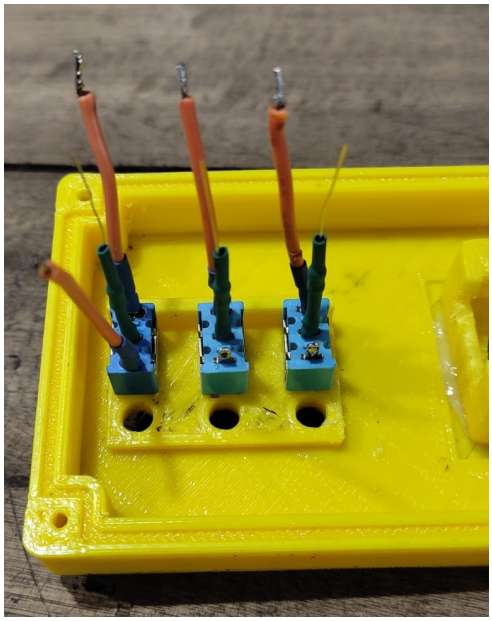
O resistor escolhido de 360Ω tem uma tolerância de $\pm$5%, e pode atender perfeitamente o projeto.

Na Figura 23 podemos notar a presença de um capacitor logo acima do módulo *wireless*. Esse capacitor é responsável por manter o módulo menos suscetível a oscilações, fixando assim sua tensão e mantendo o bom funcionamento. O uso desse

capacitor se deu após os testes terem mostrado que quanto a não presença do capacitor, o módulo *wireless* iniciava com a alimentação e oscilava muito, não fazendo conexão com os módulos escravos. Quando adicionado, o módulo para de oscilar pois não começa com sua carga em zero. A forma de resolver esse problema foi indicado através do site do fabricante do módulo.

Por fim, essas conexões foram feitas utilizando soldadores de estanho, e realizadas pelo próprio autor desse trabalho. Na Figura 24, podemos ver o esquema de ligação das chaves alavanca, onde na parte central (cabo verde) há a soldagem dos resistores, responsáveis por não deixar os níveis lógicos das chaves oscilarem.

Figura 24 – Soldagem das chaves alavanca.



Fonte: Arquivo pessoal, 2022.

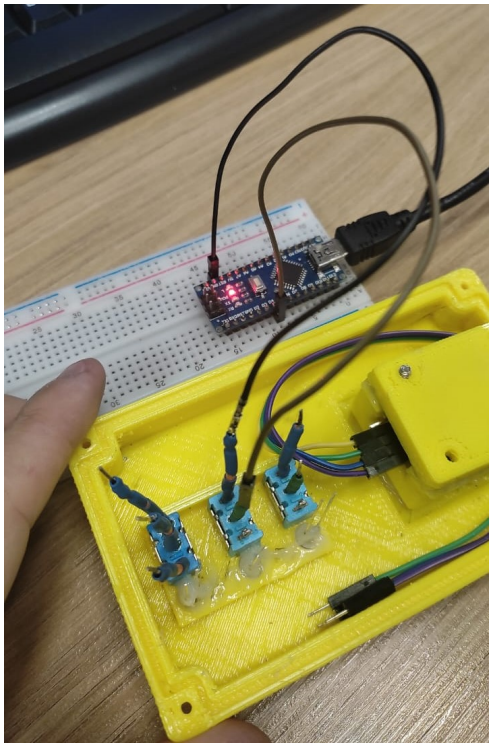
Na Figura 25 temos a testagem das chaves, pois como já citado, os componentes foram testados individualmente a fim de manter a integridade final do projeto.

Todos os demais componentes e fios foram soldados seguindo a composição do esquema da Figura 23, mantendo a familiaridade entre o que foi planejado e escrito em software com o que foi executado na parte prática.

4.3 CONTROLE REMOTO - PARTE FÍSICA DO RECEPTOR

Para receber os dados oriundos do rádio transmissor, é necessário junto ao tanque ter uma central que ficará captando os pacotes de dados e executando as ações necessárias. Para isso, a composição da central fica um pouco diferente da composição do controle remoto, visto que no transmissor temos que apenas enviar

Figura 25 – Teste da chave alavanca.



Fonte: Arquivo pessoal, 2022.

informações, enquanto que no receptor precisamos executar as informações. Para isso, foi utilizado alguns componentes eletrônicos que foram demonstrados na seção 3.1.

Para a recepção de dados, temos na central também um módulo NRF24L01. Esse módulo, presente tanto no transmissor quanto no receptor, não precisam necessariamente ser do mesmo fabricante ou o mesmo modelo, desde que possua o mesmo chip processador.

Na parte do receptor, temos que executar alguns passos para fazer com que o sistema obedeça o que passamos a ele. Alguns requisitos devem ser respeitados na hora de desenvolver o sistema:

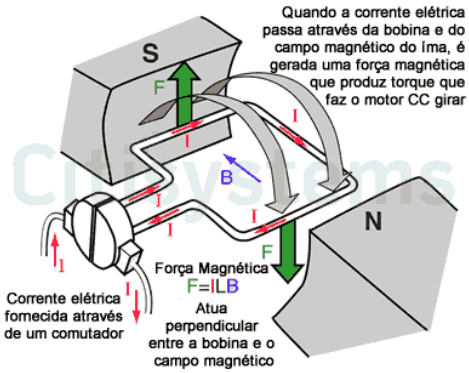
- O canhão de água precisa se mexer para a direita e para a esquerda;
- O canhão de água precisa se mexer para cima e para baixo;
- O canhão de água precisa abrir e fechar o esguicho, fazendo jato d'água ou névoa;
- A bomba para mistura do F-500 EA precisa ser acionada e desacionada;
- O giroflex precisa ligar e desligar.

Para que isso aconteça, precisamos ter alguns componentes que irão executar as tarefas.

Como apresentado na seção 3.1.7, o Driver Ponte H é um módulo utilizado para controlar motores DC. Com a conexão dos fios de carga do motor conectados diretamente nas saídas do driver, podemos fazer com que o motor gire no sentido horário ou anti-horário. Essa mudança de direção se dá devido ao fato de que o driver simplesmente muda o sentido da tensão enviada. Quando a corrente elétrica atravessa a bobina e o campo magnético do imã presente no motor, é gerada uma força mecânica que produz o torque e o giro do eixo do motor. Quando invertida a tensão de entrada, o campo magnético muda de direção e portanto o eixo do motor gira ao contrário. Esse módulo será responsável por fazer a conexão dos motores que giram o canhão para a direita e para a esquerda, bem como para cima e para baixo e também do atuador linear que faz o esguicho se abrir mais ou fechar.

Na Figura 26 podemos ver o princípo de funcionamento do motor de corrente contínua, o mesmo que faz parte do sistema eletrônico do canhão de água.

Figura 26 – Princípio de funcionamento do motor *CC*.



Fonte: Citisystems, 2022.

Então, os motores serão acionados usando esse driver, pois assim invertendo a tensão através desses, poderemos aplicar a corrente em dois sentidos.

Para fazer a implementação do acionamento da bomba e do giroflex, a opção pela utilização do módulo relé se dá por causa da possibilidade de acionamento das saídas através de sinais lógicos oriundos do microcontrolador. Quando mandamos um sinal de *LIGA*, o circuito do módulo relé se fecha e deixa então passar tensão, aplicando essa tensão na bomba ou giroflex, conforme for a programação das entradas e saídas, definidas pelo usuário.

Porém, foi identificada uma dificuldade para execução do projeto nessa parte, pois esperava-se utilizar o mesmo microcontrolador no transmissor e no receptor. Mas isso não foi possível, pois nos faltava portas digitais para fazer a conexão de todos os

componentes, no receptor. As portas digitais que precisávamos eram as seguintes:

- 4 portas digitais para o Módulo Ponte H responsável por acionar o motor na vertical e na horizontal;
- 2 portas digitais para o Módulo Ponte H responsável pelo atuador linear no esguicho do canhão;
- 5 portas digitais para o Módulo NRF24L01, responsável pela comunicação;
- 2 portas digitais para o Módulo Relé, para o acionamento da bomba e do giroflex;
- 2 portas digitais para o Módulo BLE, responsável pela comunicação Bluetooth.

Totalizando 15 portas digitais necessárias, enquanto que o Arduino Nano nos possibilita apenas 14 portas digitais.

Pensando nisso então, decidimos utilizar no projeto, a fim de testes, um Arduino Mega 2560 no receptor, possibilitando assim ter mais portas digitais e por consequência, conseguir atingir todos os objetivos listados acima.

Por fim, na parte física do receptor teremos o Arduino Mega 2560 como o microcontrolador responsável por fazer toda a parte computacional, o módulo *wireless* NRF24L01, dois módulos Ponte H, um módulo relé de 2 canais, e o módulo Bluetooth. A tensão de entrada do sistema é feita através dos cabos conectados diretamente na bateria do trator, já que este estará manobrando o tanque e portanto terá proximidade para energizar o sistema.

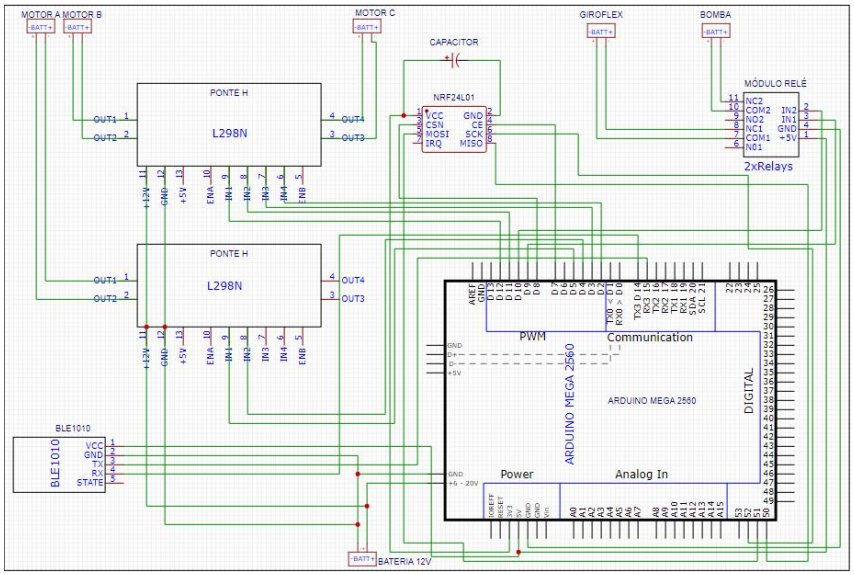
Os motores presentes no canhão de água, são motores CC de 12V. E para isso precisamos que, além do microcontrolador receber a tensão de 12V, os motores também recebam essa tensão. Por isso, foi ligado os cabos oriundos da bateria diretamente nos dois módulos Ponte H, fazendo assim o abastecimento de energia nos motores.

Para os demais módulos, foram utilizadas as saídas de 5V e 3,3V do Arduino Mega, que possuem uma estabilização perfeita para a energização dos componentes.

Na Figura 27, podemos verificar o esquemático relacionado ao desenvolvimento do receptor, e assim entender as conexões que foram feitas.

Buscando manter o mesmo esquema que apresentado na Figura 27, foi implementado a central junto ao tanque. Nos lugares dos motores A, B e C, foram conectados os motores reais responsáveis pela dinâmica de movimentação do canhão de água. Nas saídas ditas GIROFLEX e BOMBA, os nomes já são auto explicativos, e foram designadas para o acionamento dos respectivos componentes do tanque. Para fazer o acionamento dos motores, o módulo Ponte H conta com quatro entradas de sinal denominadas IN1, IN2, IN3 e IN4. Cada duas dessas entradas fica responsável por acionar um motor. No caso da entrada IN1 e IN2, elas controlam a saída 1 do

Figura 27 – Esquemático do receptor.



Fonte: Arquivo Pessoal, 2022.

módulo. E a entrada IN3 e IN4, acionam a saída 2 do módulo. Dessa forma, conectamos as portas digitais do Arduino Mega diretamente nesses pinos do módulo, a fim de comandar os motores.

No caso do módulo relé, como possuímos dois canais de comunicação, temos duas entradas: IN1 e IN2. Seguindo o modelo do módulo Ponte H, o módulo relé tem uma entrada para controlar uma saída, e outra entrada para controlar outra saída. Mas o esquema de funcionamento se dá de forma diferente, já que por originalidade de fábrica, o módulo relé vem com estado digital *HIGH*. Então, se queremos fechar o circuito e assim acionar a bomba ou giroflex, precisamos enviar um sinal de estado *LOW*. Quando enviado esse sinal, o sistema se fecha e começa a passar corrente por entre os terminais.

4.3.1 Controle Remoto - Programação do Transmissor

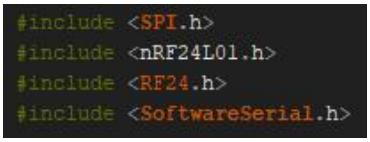
Para fazer a programação do controle remoto, foi utilizada a IDE Arduino, já apresentada na seção 3.2.

4.3.1.1 Inclusão de bibliotecas

Para começarmos o desenvolvimento, precisamos importar as bibliotecas que serão necessárias para a execução dos componentes. Como já citado, o módulo *wireless* tem uma biblioteca própria para o seu desenvolvimento para facilitar o uso do componente. Na Figura 28, podemos ver a inclusão da biblioteca *SPI*, que fica responsável pela leitura das portas *Serial*. A inclusão da biblioteca *nRF24L01*, cuja função é

habilitar o módulo *wireless* e a biblioteca *RF24*, que transforma o módulo em um rádio transmissor de alta frequência.

Figura 28 – Inclusão de bibliotecas.



Fonte: Arquivo Pessoal, 2022.

4.3.1.2 Declaração de pinos e estrutura.

Após esse evento, devemos declarar quais portas iremos utilizar para quais componentes. Na Figura 29 podemos ver que a declaração dos pinos é bem intuitiva, uma vez que você coloca qual o tipo de variável irá ser usada, nesse caso uma variável do tipo inteira, o nome da variável que você quer utilizar e o pino no qual será conectado o componente no microcontrolador.

Nesta parte, é importante salientar que os pinos declarados devem ser expressamente conectados aos mesmos pinos na placa controladora. Na primeira linha do código, podemos ver uma declaração com nome *RF24*, e essa declaração nos dá uma variável chamada radio, onde os pinos 7 e 8 são responsáveis pelos pinos *CE* e *CSN* do módulo *wireless*. Na segunda linha do código é declarado um endereço, e esse endereço ele precisa ser lembrado, pois a comunicação se dá através desse endereço. Uma vez programado o transmissor com qualquer tipo de endereço, desde que respeitado o máximo de 8 *bit's* por causa da variável *const byte*, o receptor precisa ter o mesmo endereço para que o transmissor consiga localizá-lo e mandar os dados necessários.

Como já citado na seção 3.1.3, o *joystick* de 3 eixos é um componente que utiliza de saídas analógicas para a sua leitura. Dessa forma, declaramos que os pinos A0 e A1 ficarão responsáveis por lerem os pinos dos eixos X e Y, respectivamente.

Finalizando a parte de declaração de pinos, temos a prescrição dos pinos relacionados aos botões aos *LED's*. Quando programados, procura-se manter uma escrita intuitiva do nome das variáveis, para que não se perca tempo tentando decifrar o que cada uma faz. No caso da Figura 29, podemos ver que as variáveis foram declaradas de forma que qualquer usuário possa entender do que está tratando cada pino.

Durante a construção do código do transmissor, foi pensado o que precisaria enviar ao receptor, e o quanto os dados seriam relevantes. Na linguagem de programação da IDE Arduino, contamos com uma classe denominada *struct*. A classe *struct* permite que o programador crie uma variável que estruture um conjunto de dados

Figura 29 – Declaração de pinos.

```
RF24 radio(7,8); // CE, CSN
const byte address[6] = "00001";


// pinos
int x_key = A0;
int y_key = A1;
int BotaoNevoa = 2;
int BotaoJato = 3;
int BotaoBomba = 4;
int BotaoGiroled = 5;
int LedBotaoNevoaJato = 6;
int LedBotaoBomba = 9;
int LedBotaoGiroled = 10;

struct posicoes {
  int x_pos;
  int y_pos;
  int EstadoBotaoNevoa;
  int EstadoBotaoJato;
  int EstadoBotaoBomba;
  int EstadoBotaoGiroled;

};

posicoes data;
```

Fonte: Arquivo Pessoal, 2022.

selecionados. No caso do projeto em questão, foi criada uma estrutura com os dados que seriam relevantes enviar informações ao receptor, como os estados dos botões e a posição do *joystick*. Essa classe reúne todos esses valores dentro de um pacote chamado *posicoes* e envia-o para o rádio. Dessa forma, garantimos que nenhum dado se perca no caminho, e que enviemos todos de uma única vez. Na última linha do código apresentado, podemos ver que a variável *posicoes* é do tipo *data*, ou seja, um dado.

4.3.1.3 Configuração da função *setup*

Após feita as declarações das variáveis, devemos fazer a configuração do nosso rádio transmissor e das chaves que iremos utilizar. Na Figura 30 temos as declarações que iremos utilizar para fazer o controle remoto. Primeiro configuramos a taxa de transferência por segundo da porta *serial*. Essa porta serve para ver através do computador tudo que está sendo transmitido, nesse caso, do receptor. Logo após, precisamos dar um "start"no rádio, através da função *radio.begin*.

Com o rádio já em funcionamento, precisamos declarar algumas funções que fazem parte do pacote da biblioteca *nRF24L01*:

- **radio.openWritingPipe(address)**: Essa função seta o rádio como um mestre, ou seja, como o responsável por "escrever"os dados no endereço;

- **radio.setPALevel**: Essa função é responsável por setar o nível de potência do amplificador (PA). O nível de *PA* define o consumo de energia do chip, e portanto, a potência de transmissão. Setamos em mínimo pois o número de dados que precisamos enviar é baixo, e a distância que precisamos percorrer para a tranmissão não precisa ser a maior possível;
- **radio.setDataRate**: Define a taxa de transmissão de dados. Quanto mais baixo o valor do parâmetro, nesse caso 250Kbps, mais rápida sera a taxa de transmissão;
- **radio.stopListening**: Essa função é definida para que o rádio pare de ouvir qualquer informação que possa vir do receptor.

Figura 30 – Configurações de *setup*.

```
void setup(){
  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.setDataRate(RF24_250KBPS);
  radio.stopListening();
  pinMode(x_key, INPUT);
  pinMode(y_key, INPUT);
  pinMode(BotaoNevoa, INPUT_PULLUP);
  pinMode(BotaoJato, INPUT_PULLUP);
  pinMode(BotaoBomba, INPUT_PULLUP);
  pinMode(BotaoGiroled, INPUT_PULLUP);

  pinMode(LedBotaoNevoaJato, OUTPUT);
  pinMode(LedBotaoBomba, OUTPUT);
  pinMode(LedBotaoGiroled, OUTPUT);
}
```

Fonte: Arquivo Pessoal, 2022.

Pinos digitais podem ser usados como entrada (*INPUT*), entrada com *pull-up* (*INPUT_PULLUP*) ou saída (*OUTPUT*) Mudar o modo de um pino com a função *pinMode()* muda o comportamento elétrico do pino. Na construção do transmissor, os botões são conectados por dois fios: um ligado a um pino digital e o outro ao terra. Quando acionado, pode haver uma oscilação entre o deslocamento do estado *HIGH* para *LOW*. E para evitar isso, juntamente com os resistores implementados, utilizamos a entrada *INPUT_PULLP*, para garantir que não tenha nenhum tipo de oscilação. O modo *INPUT_PULLUP* faz com que seja acionado o resistor interno da chave também. Dessa forma, temos que todas as chaves sejam do tipo entrada.

Por fim, definimos que os pinos dos *LED's* sejam do tipo saída, já que iremos aplicar uma corrente neles para que sejam acesos quando a sua respectiva chave seja selecionada.

4.3.1.4 Configuração e programação da função *loop*

A parte mais importante no desenvolvimento do transmissor se dá dentro da função *loop*. É nessa parte que serão executadas todas as ações que são necessárias para o controle do tanque Mata Fogo.

Em um primeiro momento, como demonstrado na Figura 31, temos que fazer a leitura dos pinos analógicos através da função *analogRead()*. Como visto na seção 4.3.1.2, temos duas variáveis para armazenar a posição do eixo X e do eixo Y. Por isso, fazemos a leitura da porta analógica referente ao pino do eixo X e o pino do eixo Y, e essas posições lidas são armazenadas na variável *data*, definida fora da função *loop*. Os valores variam de 0 a 1023. Lembrando que a variável *data* guarda valores na função *posicoes*, que por sua vez fica alojada na classe *struct*. Então, ao mesmo tempo em que vai se fazendo a leitura da posição X e Y, já vai armazenando esses valores para serem enviados pelo rádio transmissor.

Assim como as entradas analógicas precisam ser lidas, os estados dos botões, chaves alavanca, também precisam ser lidos e jogados dentro da classe *struct*. Os valores lidos pela função *digitalRead()* são necessariamente valores 0 ou *LOW*, 1 ou *HIGH*. Se a leitura for 1, significa que o botão está "desligado". Porém, se for do tipo 0, o botão está indicando que está "ligado".

Figura 31 – Configurações de *loop*.

```
void loop() {
  data.x_pos = analogRead(x_key);
  data.y_pos = analogRead(y_key);


  data.EstadoBotaoNevoa = digitalRead(BotaoNevoa);
  data.EstadoBotaoJato = digitalRead(BotaoJato);
  data.EstadoBotaoBomba = digitalRead(BotaoBomba);
  data.EstadoBotaoGiroled = digitalRead(BotaoGiroled);


  if(data.EstadoBotaoNevoa == LOW) {
    digitalWrite(LedBotaoNevoaJato, LOW);
  } else {
    digitalWrite(LedBotaoNevoaJato, HIGH);
  }
  if(data.EstadoBotaoJato == LOW ){
    digitalWrite(LedBotaoNevoaJato, LOW);
  }else {
    digitalWrite(LedBotaoNevoaJato, HIGH);
  }

  if(data.EstadoBotaoBomba == LOW){
    digitalWrite(LedBotaoBomba, LOW);
  } else {
    digitalWrite(LedBotaoBomba, HIGH);
  }
  if(data.EstadoBotaoGiroled == LOW){
    digitalWrite(LedBotaoGiroled, LOW);
  } else {
    digitalWrite(LedBotaoGiroled, HIGH);
  }

  radio.write(&data, sizeof(posicoes));
  delay(100);
}
```

Fonte: Arquivo Pessoal, 2022.

Esses são os valores que precisam ser enviados via rádio para a central que fica alojada no tanque. Daí por diante, toda a programação é feita para validar os componentes que ficarão no controle.

Nessa parte do código, as funções de decisão e comparação ficam responsáveis por analisar o estado das chaves e aplicar estados nos *LED's*. Então, se o

EstadoBotaoNevoa for um estado de *LOW*, deve acender o *LED* que sinaliza que a chave está ativa. Ou seja, a chave está fazendo a função de manobrar o atuador linear presente no canhão de água. Se não, se for diferente de *HIGH*, permanece o *LED* em *HIGH*, indicando que não precisa acender. E essa comparação segue para todas as chaves, verificando sempre se as mesmas estão ativas ou não.

Ao fim do processo de comparação e decisão, é chamada a função *radio.write* que faz um compilado dos dados gravados na variável *data*, e joga-os na variável *posicoes*. Essa variável está dentro da classe *struct* e portanto, faz parte do pacote de dados que precisam ser enviados para central do tanque. Os dados são enviados em forma de pacote, com a estrutura total das variáveis, mesmo que algumas não tenham valor, ou permaneçam "zeradas", elas continuam sendo enviadas. Quando há alteração, imediatamente é enviado o dado novo.

4.3.2 Controle Remoto - Programação do Receptor

Nessa parte do relatório, irei demonstrar a programação por parte do receptor, bem como as suas configurações e ações que irão ser tomadas.

4.3.2.1 Inclusão de bibliotecas

Começamos no receptor também declarando as bibliotecas, e aqui temos a adição da biblioteca *SoftwareSerial*, e isso acontece pois na central presente no tanque teremos também um ponto de recepção de sinal Bluetooth, a ser explicado nas próximas seções. A biblioteca em questão permite a comunicação *serial* nos pinos RX e TX do microcontrolador. Esta biblioteca tem a limitação de não poder transmitir e receber dados ao mesmo tempo, porém vamos utilizar somente um sentido de informação, então não precisamos nos preocupar com isso no momento. Se os motores presentes no canhão de água precisasse controlar também a sua velocidade de rotação, deveríamos incluir a biblioteca *Stepper*, que é uma biblioteca responsável por fazer o controle de velocidade de rotação de motores CC. Mas não é o caso nesse trabalho.

Figura 32 – Inclusão de bibliotecas.

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <SoftwareSerial.h>
```

Fonte: Arquivo Pessoal, 2022.

4.3.2.2 Declaração de pinos e estrutura.

Na Figura 33 podemos ver um aumento de variáveis em relação ao lado do transmissor.

Figura 33 – Declaração de pinos e estrutura do receptor.

```
SoftwareSerial BLE10(18,19); // RX = 2, TX = 3
char appData;

String inData = "";
RF24 radio(7,8); // CE, CSN

const byte address[6] = "00001";

/* Aqui começa a declaração das Ponte H */
const int MotorRight = 11;
const int MotorLeft = 12;

const int MotorUp = 2;
const int MotorDown = 3;

const int AtuadorJato = 4;
const int AtuadorNevoa = 5;

/* Aqui termina a declaração das Ponte H */

/* Começa aqui a declaração das porta relé */

int PortaReleUm = 9; // IN1
int PortaReleDois = 10; //IN2

/* Termina aqui a declaração das porta relé */



int x_posicao;

struct posicoes {
  int x_pos;
  int y_pos;
  int EstadoBotaoNevoa;
  int EstadoBotaoJato;
  int EstadoBotaoBomba;
  int EstadoBotaoGiroled;
};

posicoes data;
```

Fonte: Arquivo Pessoal, 2022.

Primeiramente começamos declarando quais pinos serão responsáveis por fazer a comunicação *serial*. Na seção 3.1.11 temos um esquemático do microcontrolador Arduino Mega, e podemos verificar quais são as portas habilitadas para fazerem essa comunicação. Foram escolhidas as portas 18 e 19 para fazer o RX e o TX, respectivamente.

Criamos uma variável *appData* que ficará responsável por guardar os valores que irão ser lidos na porta *serial*, vindas do aplicativo *mobile* a ser apresentado. Também, é criada uma variável do tipo *String*, com nome de *inData* para fazer as comparações e decisões depois, a fim de controlar o sistema com as informações vindas do aplicativo (app) *mobile*.

A declaração dos pinos responsáveis pelo módulo *wireless* não precisam necessariamente serem as mesmas do transmissor. Porém, devem estar ligadas às suas respectivas portas. Por exemplo, o pino de *clock* no transmissor é conectado ao pino 13, enquanto no receptor, por usar um microcontrolador diferente, é conectado ao pino 52. Mas o endereço entre ambos devem ser iguais, e isso é importante que seja, pois

senão o transmissor enviará dados para ninguém, e o receptor não receberá dados. É também necessário que a declaração de quantos *bit's* foram usados permaneça a mesma.

Citado na seção 4.3, a parte física do receptor possui dois módulos Ponte H que são responsáveis por fazer o controle sobre os motores. Nesta parte, devemos declará-los a fim de habilitar quais as entradas que serão responsáveis por fazer o controle de cada motor. No caso da variável *MotorRight*, definimos o pino 11 como responsável, e essa conexão será feita entre o pino 11 do microcontrolador e o pino IN1 do módulo Ponte H. É importante ver que aqui nesse ponto, a variável é do tipo *const int*, cuja função é ser um qualificador de variáveis que modifica o comportamento da mesma, fazendo com que essa variável seja de "apenas-leitura". Significa que ela pode ser usada como qualquer outra variável do tipo *int* mas seu valor não pode ser mudado. É apenas o que recebe do transmissor.

E esta regra vale para todas as variáveis que são de atuação, como a definição do motor girando para a esquerda, para cima e para baixo, além do atuador linear, que ficará responsável por fazer o esguicho se movimentar e abrir ou fechar mais a "boca"do canhão, transformando a aspersão em jato d'água ou névoa d'água.

Por fim, a classe *struct* precisa ser a mesma que utilizada no transmissor, pois o pacote de dados que serão enviados devem ser os mesmos dos dois lados.

4.3.2.3 Configuração da função *setup*

Nesta parte do projeto podemos ver que as configurações dos componentes é um pouco diferente do transmissor, com mais componentes sendo declarados. Começamos habilitando a porta *serial* que ficará lendo o que vem do aplicativo através do módulo Bluetooth. O módulo BLE1010 fica conectado aos pinos da porta *serial* apenas esperando informações que possam ser enviadas pelo app *mobile*. Declaramos a habilitação do módulo *wireless*, como fizemos no transmissor.

A parte de configuração do módulo *wireless* diferem um pouco do lado do transmissor. Nesta etapa, é preciso declarar que o rádio receptor será sempre um "escravo", e que sempre irá receber valores e não enviar nada. Para isso, se define a função *openReadingPipe* com valor 0 para que seja apenas leitura de dados vindos do endereço *address*. Se o escravo precisasse enviar dados, deveria ser o valor 1 alocado na função. Ele seria o escravo ainda, porém, poderia enviar informações para o mestre. Ainda, habilita-se a função *startListening* para que o rádio inicie a leitura de dados.

Na parte de declaração de pinos de atuadores, todos precisam ser do tipo *OUTPUT*, pois são pinos que irão enviar dados para os respectivos componentes. Por exemplo o pino responsável pelo *MotorLeft* precisa enviar um sinal para o módulo Ponte H quando ele irá atuar no módulo.

Precisamos também fazer com que todos os atuadores iniciem em nível lógico

Figura 34 – Configurações do *setup*.

```
void setup() {
  Serial.begin(9600);
  BLE10.begin(9600);
  radio.begin();
  pinMode(MotorLeft, OUTPUT);
  pinMode(MotorRight, OUTPUT);
  pinMode(MotorDown, OUTPUT);
  pinMode(MotorUp, OUTPUT);
  pinMode(AtuadorJato, OUTPUT);
  pinMode(AtuadorNevoa, OUTPUT);
  pinMode(PortaReleUm, OUTPUT);
  pinMode(PortaReleDois, OUTPUT);

  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.setDataRate(RF24_250KBPS);
  radio.startListening();

  digitalWrite(MotorLeft, LOW);
  digitalWrite(MotorRight, LOW);
  digitalWrite(MotorUp, LOW);
  digitalWrite(MotorDown, LOW);
  digitalWrite(AtuadorJato, LOW);
  digitalWrite(AtuadorNevoa, LOW);
}
```

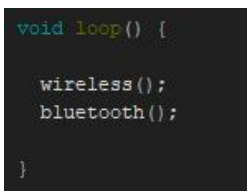
Fonte: Arquivo Pessoal, 2022.

LOW. Pois se não dermos nenhum sinal, eles podem entender que já se iniciam "energizados"e comecem a se movimentar sem nenhum dado ter sido enviado. Por isso, todas as portas digitais dos motores e atuadores começam em nível baixo.

4.3.2.4 Função *wireless*

Para melhor visualização do que está acontecendo dentro do código, foi desenvolvido duas funções separadas para cada tecnologia que foi utilizada. Ainda assim, é necessário criar a função *loop* e chamar essas funções auxiliares dentro do mesmo, como apresentado na Figura 35.

Figura 35 – Função *loop*.



```
void loop() {

  wireless();
  bluetooth();

}
```

Fonte: Arquivo Pessoal, 2022.

A Figura 36 mostra a função responsável por ler e interpretar os dados vindos do controle remoto *wireless*. A função começa analisando se existem bytes (caracteres) disponíveis para leitura na porta *serial*. Se esses dados existem, o rádio faz uma leitura, através da função *radio.read()*, do pacote que está sendo recebido, e nesse caso é o pacote de dados da estrutura *posicoes* oriundos do transmissor.

Figura 36 – Função *wireless*.

```
void wireless() {

    if (radio.available()) {

    radio.read(&data, sizeof(posicoes));
    delay(500);

    if(data.x_pos < 350) {
      digitalWrite(MotorLeft, HIGH);
      digitalWrite(MotorRight, LOW);
      digitalWrite(MotorUp, LOW);
      digitalWrite(MotorDown, LOW);
    }
    if(data.x_pos > 650){
      digitalWrite(MotorLeft, LOW);
      digitalWrite(MotorRight, HIGH);
      digitalWrite(MotorUp, LOW);
      digitalWrite(MotorDown, LOW);
    }
    if(data.y_pos < 350){
      digitalWrite(MotorLeft, LOW);
      digitalWrite(MotorRight, LOW);
      digitalWrite(MotorUp, HIGH);
      digitalWrite(MotorDown, LOW);
    }
    if(data.y_pos > 650){
      digitalWrite(MotorLeft, LOW);
      digitalWrite(MotorRight, LOW);
      digitalWrite(MotorUp, LOW);
      digitalWrite(MotorDown, HIGH);
    }
    if(data.x_pos > 350 && data.x_pos < 650 && data.y_pos > 350 && data.y_pos < 650){
      digitalWrite(MotorLeft, LOW);
      digitalWrite(MotorRight, LOW);
      digitalWrite(MotorUp, LOW);
      digitalWrite(MotorDown, LOW);
    }
```

Fonte: Arquivo Pessoal, 2022.

Após a leitura ser feita, as tomadas de decisões começam a ser executadas. Essas variáveis vindas em forma de pacote são "desmembradas"e analisadas separadamente.

No caso da variável *x_pos*, ela é a variável que no transmissor faz a leitura do *joystick*, e portanto, ela mostra um valor entre 0 e 1023. Se o valor retornado for menor que 350, significa que o usuário está deslocando o eixo X para a esquerda, e assim é necessário que o motor do canhão elétrico gire para o sentido anti-horário, girando assim sua base para a esquerda.

Para que possamos acionar o motor e fazê-lo girar, precisamos aplicar uma tensão no motor. Nesse caso, enviamos um sinal para que o pino responsável por *MotorLeft* seja acionado, indo para o estado *HIGH* e mantendo os demais em *LOW*. Quando esse pino vai para o estado *HIGH*, o pino IN1 presente no módulo Ponte H interpreta essa informação e aplica uma tensão nos terminais do motor. Para que o motor gire no sentido anti-horário, o sentido da tensão aplicada tem que ser invertida,

ou seja, aplica-se uma tensão "negativa"no terminal positivo do motor, e uma tensão "positiva"no terminal negativo do motor. Se a variável *x_pos* for maior que 650, significa que o usuário está virando o botão do *joystick* para a direita, e portanto, o pino responsável por virar o motor para a direita é acionado, e nesse caso é o pino *MotorRight* no microcontrolador e IN2 no módulo Ponte H. Isso implica em uma tensão "positiva"no terminal positivo do motor, e uma tensão "negativa"no terminal negativo do motor, girando o mesmo no sentido horário.

O mesmo pensamento aplica-se para o motor que fica responsável por elevar ou abaixar o canhão de água, porém usando a variável *y_pos*. Essa variável retorna também valores entre 0 e 1023. O que muda aqui, é os pinos que são acionados. No caso do motor girando para cima, o pino que interpreta a variável *MotorUp* no módulo Ponte H é o pino IN3, e no motor para baixo, o pino responsável é o IN4. Para fazer a rotação dos motores, aplica-se o mesmo pensamento e a mesma fórmula que para girar no sentido horizontal.

Se as informações que vierem do transmissor nas variáveis *x_pos* e *y_pos* forem entre 350 e 650 nas duas, quer dizer que o botão *joystick* está parado, e portanto centralizado. Então, não precisamos que haja qualquer tipo de ação, e por isso setamos os motores novamente em estado *LOW*, para que não façam nada enquanto não vier novas informações.

Na Figura 37, temos a parte de acionamento da bomba e do giroflex. Ainda seguindo o raciocínio das posições do *joystick*, temos a análise de estados das chaves do controle transmissor. Aqui, as decisões são tomadas dependendo do valor que chega para o receptor. Se a chave alavanca responsável pelo atuador linear for acionada, ela irá gerar um valor de estado 0, e no receptor se esse valor chegar, a variável *AtuadorNevoa* é acionada. Essa variável fica responsável por aplicar uma tensão no motor do atuador linear presente no canhão, que faz o fechamento do bico, e assim fazendo uma névoa na aspersão da água. Então, se o valor da variável *EstadoBotaoNevoa*, oriunda do transmissor, for 0, aciona-se o atuador linear através da variável *AtuadorNevoa*, aplicando uma tensão negativa nos terminais do atuador linear através do módulo Ponte H, fazendo o atuador se retrair, fechando o bicho do canhão.

Se o valor da variável *EstadoBotaoJato* for 0, aplica-se uma tensão positiva nos terminais do atuador linear, através do módulo Ponte H, fazendo assim o atuador linear se expandir, abrindo o bico do canhão.

Por fim, analisa-se o estado dos botões responsáveis pelo acionamento da bomba e do giroflex. Esses estados são mais simples, pois juntamente com o módulo relé, a única função é fechar o circuito e deixar passar a tensão para abastecimento dos componentes. Então, se o *EstadoBotaoBomba* for 0, é acionada a entrada IN1 do módulo relé, que internamente faz o circuito se fechar, deixando passar tensão entre os terminais. Se o *EstadoBotaoGiroflex* for 0, aciona-se o pino IN2 do módulo relé, e

Figura 37 – Função *wireless*.

```
    if(data.EstadoBotaoNevoa == 0) {
      digitalWrite(AtuadorNevoa, HIGH);
    } else {
      digitalWrite(AtuadorNevoa, LOW);
    }
    if(data.EstadoBotaoJato == 0) {
      digitalWrite(AtuadorJato, HIGH);
    }
    if(data.EstadoBotaoBomba == 0){
      digitalWrite(PortaReleUm, LOW);
    } else {
      digitalWrite(PortaReleUm, HIGH);
    }
    if(data.EstadoBotaoGiroled == 0) {
      digitalWrite(PortaReleDois, LOW);
    } else {
      digitalWrite(PortaReleDois, HIGH);
    }
  }
}
```

Fonte: Arquivo Pessoal, 2022.

assim fecha o circuito passando tensão entre os terminais.

O módulo relé funciona exatamente como uma chave interruptora. O relé vem padrão de fábrica com sinal lógico *HIGH* e quando precisa ser fechado o circuito, precisamos passar a informação de padrão lógico *LOW*, assim fechando circuito e acionando a bomba ou giroflex.

4.3.2.5 Função *bluetooth*

Nesta parte, irei demonstrar a configuração do *Bluetooth* por parte da central acoplada no tanque. Posteriormente, será apresentado o aplicativo móvel que foi desenvolvido, e como se comunica com o módulo *Bluetooth* conectado na central.

A Figura 38 mostra a função *bluetooth* desenvolvida para receber e interpretar sinais que são vindos do aplicativo móvel.

O primeiro passo que precisamos executar é habilitar a porta *serial* e "escutar"o que está chegando através dela.

A condição *while* é necessária para ficarmos "escutando"o que vem da porta *serial*. Enquanto houver dados, lemos eles e adicionamos numa variável denominada *appData*. Transformamos essa variável para um tipo *String*, pois todos os dados que vierem do aplicativo, serão em forma de *String*. Por isso, é essencial que todo dado que chegue na porta, seja transformado neste tipo.

Podemos notar que segue a mesma lógica que apresenta a seção 4.3.2.4. Onde, após receber um tipo de dados, temos um evento associado a ele. Para fazer a direção do canhão, chegarão dados numéricos através da porta *serial*, sendo os números 0, 1,

Figura 38 – Função *Bluetooth*.

```
void bluetooth() {

  BLE10.listen(); // Escuta o que vem da porta BLE10

  while(BLE10.available() > 0) {
    appData = BLE10.read();
    inData = String(appData);
  }

  if(inData == "2"){
      digitalWrite(MotorLeft, HIGH);
      digitalWrite(MotorRight, LOW);
      digitalWrite(MotorUp, LOW);
      digitalWrite(MotorDown, LOW);
  }
  if(inData == "0"){
      digitalWrite(MotorLeft, LOW);
      digitalWrite(MotorRight, HIGH);
      digitalWrite(MotorUp, LOW);
      digitalWrite(MotorDown, LOW);
  }
  if(inData == "3"){
      digitalWrite(MotorLeft, LOW);
      digitalWrite(MotorRight, LOW);
      digitalWrite(MotorUp, HIGH);
      digitalWrite(MotorDown, LOW);
  }
  if(inData == "1"){
      digitalWrite(MotorLeft, LOW);
      digitalWrite(MotorRight, LOW);
      digitalWrite(MotorUp, LOW);
      digitalWrite(MotorDown, HIGH);
  }
```

Fonte: Arquivo Pessoal, 2022.

2 ou 3. E cada dado desses, vem de um passo executado dentro do aplicativo.

Na Figura 39 temos a parte que fica responsável por acionar a bomba, ligar o giroflex e atuar no atuador linear.

No aplicativo, essa parte será enviado exatamente o que está sendo comparado. Se apertarmos o botão para "abrir"o atuador linear, ele irá aplicar um estado *HIGH* no *AtuadorJato*, botando assim o esguicho aberto para que saia um jato de água. Se apertarmos o botão para "fechar"o atuador linear, ele irá aplicar um estado *HIGH* no *AtuadorNevoa*, retraindo assim o esguicho, fechando-o para que saia uma névoa de água.

Para acionamento da bomba e do giroflex, informações vindas do aplicativo serão exatamente da forma como consta na comparação. Se for "F-500: true", devemos acionar o relé e fazer o acionamento da bomba. Se for "F-500: false", devemos manter desligado. Como igualdade, o giroflex também exercerá essa mesma forma.

Figura 39 – Função *Bluetooth*.

```
  if(inData == "Abre"){
    digitalWrite(AtuadorJato, HIGH);
  } else {
    digitalWrite(AtuadorJato, LOW);
  }
  if(inData == "Fecha"){
    digitalWrite(AtuadorNevoa, HIGH);
  } else {
    digitalWrite(AtuadorNevoa, LOW);
  }
  if(inData == "F-500: true"){
    digitalWrite(PortaReleUm, LOW);
  } else{
    digitalWrite(PortaReleUm, HIGH);
  }
  if(inData == "Giroflex: true"){
    digitalWrite(PortaReleDois, LOW);
  } else{
    digitalWrite(PortaReleDois, HIGH);
  }

}
```

Fonte: Arquivo Pessoal, 2022.

4.4 APLICATIVO *MOBILE*

Dentro do sistema de controle remoto do canhão de água, o aplicativo possui o papel de interagir com o usuário, apresentando a possibilidade de, se em algum momento o *joystick* falhar ou não estar presente próximo ao operador, ele consiga suprir essa necessidade. Com isso, pretende-se auxiliar no controle do canhão ajudando no combate ao incêndio.

De forma geral, o fluxo completo de utilização do aplicativo se inicia com o usuário acessando o aplicativo instalado em seu *smartphone*. O aplicativo irá iniciar, e o usuário se conectará ao canhão de água, de forma que consiga fazer o controle e os acionamentos necessários.

Nesse capítulo, serão discutidos os assuntos a cerca da modelagem e implementação do aplicativo *mobile* do canhão de água do tanque Mata Fogo.

4.4.1 Modelagem do aplicativo

O aplicativo, como citado, foi desenvolvido em *Flutter*, um kit de desenvolvimento para aplicativos e sistemas *web*. A linguagem de programação por trás do *Flutter* é a linguagem *Dart*. Uma linguagem simples, direta, e que foi desenvolvida pelo *Google* buscando trazer uma dinâmica diferente para os desenvolvedores de aplicativos *mobile*. O *Flutter* permite que criemos aplicações para os sistemas operacionais mais comuns,

e também desenvolvimento de aplicativos para o sistema iOS e Android.

O desenvolvimento se dá através de construção de *widgets* que ficarão responsáveis por fazer a parte gráfica e a parte de interações com os usuários.

Ao iniciarmos o *app*, fazem-se duas ações simultâneas: inicialização dos *widgets* e conexão do *Bluetooth* com o módulo presente no tanque MF. Quando os *widgets* são renderizados na tela, eles podem sofrer alterações conforme o uso, ou seja, não são estáticos.

Os *widgets* nativos por si só podem ser usados para desenvolvimento de projetos, mas podem ser limitados, com alguns recursos faltantes, e comprometendo assim o decorrer de um projeto. Para contornar isso, buscam-se pacotes desenvolvidos por terceiros.

A ferramenta *Flutter* é desenvolvida pelo *Google* porém os pacotes, *packages*, são mantidos pela comunidade em geral. Qualquer desenvolvedor pode desenvolver bibliotecas que achar útil. Dessa forma, existem diversas bibliotecas para um problema, e o desenvolvedor de aplicativo pode usá-las baseado no pacote que melhor lhe agradar.

Nesse projeto em questão, precisamos importar algumas bibliotecas que serão úteis para o desenvolvimento de *widgets* que o usuário for interagir. É o caso da biblioteca *contro_pad*. Essa biblioteca imita um controle de *video-game*, podendo usar os botões direcionais para direcionamento de objetos.

A biblioteca *sliding_switch* é uma biblioteca usada para imitar o acionamento de chaves alavanca. Uma vez que clicada sobre o *widget* na tela, ele desliza demonstrando que acionou, e se clicado novamente, o botão desliza indicando que entrou em *standby*.

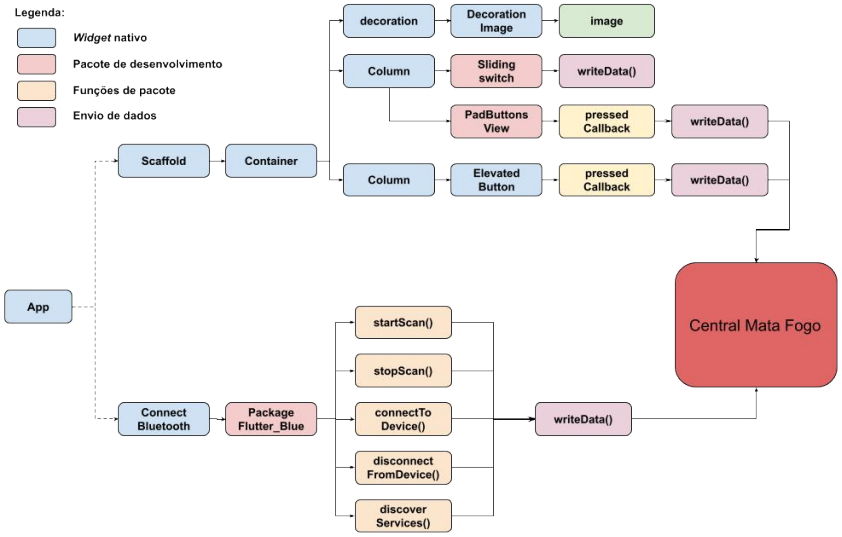
4.4.2 Componentes visuais

Os componentes do *Flutter* são construídos, como citado, a partir de componentes nativos e bibliotecas da comunidade. Utilizando desses componentes, fez-se o trabalho de organizar os componentes para que esses tivessem a disposição definida. Através de uma árvore de componentes, é possível descrever toda a estrutura do código.

A Figura 40 mostra um diagrama de blocos demonstrando o passo a passo utilizado para o desenvolvimento do *app*.

Inicializamos a ferramenta que está instalada no *smartphone*. Ao ser inicializada, os componentes visuais são renderizados na tela, podendo o usuário interagir com os mesmos. É importante salientar que teremos uma página apenas, e portanto, o usuário não precisará se preocupar com abertura de telas secundárias. Essa opção de apenas uma tela se dá pelo fato de que quando começa o incêndio nas lavouras, o desespero acaba tomando conta do usuário, e o que ele mais precisa nesse momento é de algo intuitivo e fácil de ser usado.

Figura 40 – Diagrama de blocos referente ao desenvolvimento do aplicativo.



Fonte: Arquivo Pessoal, 2022.

Na tela de uusuário, quando em desenvolvimento, o aplicativo deve seguir algumas regras para que apareçam os *widgets*. Dessa forma, queremos que seja renderizado ao lado esquerdo da tela os botões do tido *slide* e os controles para movimentação do canhão. E ao lado direito da tela, os botões para abertura e fechamento do esguicho do canhão.

Começando então desenvolvendo os botões de *slide*. A Figura 41 mostra o código que ficará responsável por renderizar na tela o objeto em questão.

Figura 41 – *Sliding Switch*.

```
SlidingSwitch(
  value: false,
  width: 150,
  onChanged: (bool valorF500) {
    writeData('F-500: $valorF500');
  },
  height: 75,
  animationDuration: const Duration(milliseconds: 100),
  onTap: () {},
  onDoubleTap: () {},
  onSwipe: () {},
  textOn: 'F-500\n  On',
  textOff: 'F-500\n Off',
  colorOn: Colors.green,
  colorOff: Colors.red,
  background: Colors.redAccent,
  inactiveColor: Colors.grey,
),  // SlidingSwitch
```

Fonte: Arquivo Pessoal, 2022.

Cada *widget* tem seus componentes, e dessa forma, tem as suas funções. Por isso, é importante verificar qual se encaixa perfeitamente no caso de uso, e quais as funções que serão utilizadas. Muitas vezes, muitas dessas funções não precisarão ter efeito algum no aplicativo, mas precisam ser citadas durante a construção do *widget*, com risco de não renderizar o componente.

Começamos setando o valor do *widget* em *false*, pois quando houver a troca de estado com o botão indo para o estado *ON*, precisamos que envie um sinal *booleano* de *true*, para indicar o acionamento. Esse *widget* possui a função *onChanged*, que nos retorna a alteração, quando houver, do *value* citado acima. Podemos ainda ajustar o duração da animação de deslize do botão, conforme for necessário perante o projeto. As funções *onTap*, *onDoubleTap* e *onSwipe* não precisaremos utilizar nesse projeto. E após isso, é apenas configuração de como será renderizado o *widget* na tela. É importante verificar, que o que realmente nos interessa esteja dentro de *onChanged*, pois é essa a informação que será coletada e enviada via *Bluetooth* para o tanque MF.

Nesse projeto, precisaremos ter dois *widget* de *slide*, pois precisamos acionar a bomba e acionar o giroflex presentes no canhão do tanque MF.

Dando sequência, precisamos renderizar na tela os botões que imitarão o controle remoto. Por isso utilizamos a biblioteca *control_pad*, e a Figura 42 demonstra a utilização dessa biblioteca na aplicação.

Figura 42 – *Pad Button*.

```
PadButtonsView(
  padButtonPressedCallback: padButtonPressedCallback,
  buttonsPadding: 5,
  buttons: const [
    PadButtonItem(
      supportedGestures: [Gestures.TAP],
      index: 0,
      backgroundColor: Colors.redAccent,
      pressedColor: Colors.orangeAccent,
      buttonIcon: Icon(
        Icons.arrow_forward,
        size: 30,
      ), // Icon
    ), // PadButtonItem
    PadButtonItem(
      supportedGestures: [Gestures.TAP],
      index: 1,
      backgroundColor: Colors.redAccent,
      pressedColor: Colors.orangeAccent,
      buttonIcon: Icon(
        Icons.arrow_downward,
        size: 30,
      ), // Icon
    ), // PadButtonItem
```

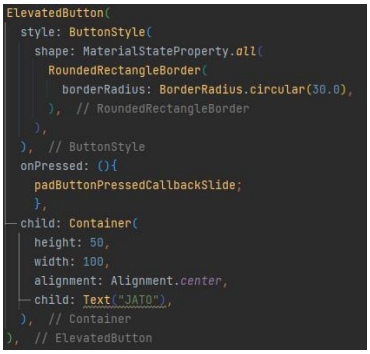
Fonte: Arquivo Pessoal, 2022.

Para renderizar, cria-se um *widget* pai, chamado *PadButtonsView*, invocando a biblioteca, e dentro desse *widget* cria-se os botões do controle. Essa biblioteca permite criar quantos botões forem necessários. Porém, precisamos renderizar apenas quatro, indicando o controle vertical e horizontal do canhão. Para isso, é criado o *PadButtonI-*

tem, que fará a função do botão. Na Figura 42 aparecem apenas dois, mas no total são quatro, e diferem entre si apenas o *index* de cada botão, pois esse *index* será lido pela central do tanque e identificado o número correspondente. Dentro do *PadButtonsView*, temos uma função denominada *Gestures*. Os *Gestures* são, principalmente, uma maneira de um usuário interagir com o aplicativo baseado em toques. São geralmente definidos como qualquer ação/movimento físico de um usuário na intenção de ativar um controle específico do dispositivo (POINT, 2020).

Por fim, faz-se os botões com elevação para indição do modo *JATO* ou *NÉVOA*. A Figura 43 mostra a criação dos botões. No geral, os botões possuem uma função denominada *onPressed*, que designa a alguma ação quando o botão for pressionado. No caso do desenvolvimento do aplicativo, quando um botão é pressionado ele chama a função de envio de dados, informando qual botão foi pressionado.

Figura 43 – *Elevated Button*.



Fonte: Arquivo Pessoal, 2022.

4.4.3 Função *Bluetooth*

Para que o *app* possa se comunicar com o Arduino acoplado no tanque, precisa enviar os dados em forma de *bytes*, ou seja, uma unidade de informação por vez, devido a porta *serial* presente no Arduino, em que o módulo *Bluetooth* está conectado.

Mas aqui também precisamos levar em conta alguns fatores para o desenvolvimento do aplicativo e a sua conexão. O módulo escolhido para ser o receptor de sinal, é um módulo de baixo consumo de energia, e isso é interessante pois em muitos locais que durante o combate ao incêndio venha a ocorrer, são locais remotos, distante de fazendas e afins, tornando o uso da bateria do celular essencial para o bom funcionamento do *app*. E também é importante tornar o uso o mais fácil possível, e para isso podemos tornar o aplicativo funcional ao ponto de ao abri-lo, ele conectar automaticamente com o módulo presente no tanque.

O desenvolvimento do *Bluetooth* dentro do aplicativo se dá através de funções da linguagem *Dart*. Mas antes de dar continuidade, precisamos saber que os módulos

Bluetooth trazem de fábrica algumas pré configurações de serviços, características e nome. Cada módulo possui um *ID* de serviço diferente, bem como um *ID* de característica diferente. O nome, permanece o mesmo para cada módulo.

O módulo *BLE1010* possui um *ID* definido demonstrado na Figura 44. Esses *ID's* são necessários para que quando o *app* ser aberto, ele conecte direto com o módulo.

Figura 44 – Função *Bluetooth*.

```
final String SERVICE_UUID = "0000FFE0-0000-1000-8000-00805F9B34FB";
final String CHARACTERISTIC_UUID = "0000FFE1-0000-1000-8000-00805F9B34FB";
final String TARGET_DEVICE_NAME = "Soft AT";

FlutterBlue flutterBlue = FlutterBlue.instance;
late StreamSubscription<ScanResult> scanSubScription;

late BluetoothDevice targetDevice;
late BluetoothCharacteristic targetCharacteristic;

String connectionText = "";

@override
void initState(){
  super.initState();
  startScan();
}

startScan(){
  setState(() {
    connectionText = "Iniciando Scaneamento";
  });

  scanSubScription = flutterBlue.scan().listen((scanResult) {
    if(scanResult.device.name == TARGET_DEVICE_NAME){
      print('Device found');
      stopScan();
      setState(() {
        connectionText = "Dipositivo de destino encontrado";
      });
      targetDevice = scanResult.device;
      //connectToDevice();
    }
  }, onDone: () => stopScan());
}
```

Fonte: Arquivo Pessoal, 2022.

O nome, também citado na figura, é essencial que seja esteja definido no código do *app*, pois será necessário para fazer a comparação, para quando o aplicativo localizar o *Bluetooth*.

O *Bluetooth* precisa ser iniciado através da função *FlutterBlue.instance*, e essa função é pré definida pelo pacote *flutter_blue*.

A inicialização de dispositivo *Bluetooth* se dá pela função *BluetoothDevice*. É importante criar uma lista de dispostivos pois precisamos aguardar, mesmo que por pouco tempo, que os dispositivos sejam localizados e conectados. Enquanto faz a varredura de aplicativos próximos, o *app* vai guardando esses dispositivos em uma lista a ser acessada posteriormente.

A função de escaneamento de dispositivos é iniciada na função *startScan*. É aqui que vai sendo gerado os dispositivos que serão guardados numa lista para posterior acesso. Se encontrar o dispositivo que queremos conectar, o escaneamento é parado através da função *stopScan*, é informado isso ao sistema, mas o usuário não vê essa informação, apenas vê na hora de utilização do aplicativo que está conectado.

Na Figura 45 vemos a continuidade da função *Bluetooth*. Aqui, temos a parte de parada de escaneamento, que é chamada na hora que localiza o dispositivo.

O *Dart* trabalha com funções assíncronas, e isso é essencial que se tenha. A

função que é apontada como assíncrona, indica que o aplicativo pode continuar executando outras tarefas sem se preocupar com elas. Nesse caso, enquanto estamos fazendo a varredura de dispositivos, não temos nada conectado, e a função *connectToDevice*, que faz a conexão com o dispositivo, permanece esperando. O *await* serve para determinar que o aplicativo deve esperar uma resposta de uma função antes de continuar a execução. Aqui também temos a função para desconectar o *Bluetooth*, quando solicitado pelo usuário. Então, ao ser invocado, apenas faz uso da função *disconnect* presente no pacote do *flutter_blue*.

Figura 45 – Função *Bluetooth*.

```
stopScan(){
  scanSubScription.cancel();
  scanSubScription;
}

connectToDevice() async {
  if(targetDevice == null) return;

  setState(() {
    connectionText = "Dispositivo conectando";
  });

  await targetDevice.connect();
  print('Dispositivo Conectado');

  setState(() {
    connectionText = "Dispositivo Conectado";
  });
  discoverServices();
}

disconnectFromDevice(){
  if(targetDevice == null) return;
  targetDevice.disconnect();

  setState(() {
    connectionText = "Dispositivo desconectado";
  });
}
```

Fonte: Arquivo Pessoal, 2022.

A Figura 46 mostra a função que faz o descobrimento de serviços presente no aplicativo. É aqui que será testada as *ID's*, e verificadas se estão de acordo com o que foi passado. Como citado, o *app* se conectará diretamente com o módulo indicado, não alterando sua rota.

Na parte final do código temos a função *writeData*, que será responsável por enviar os dados para o dispositivo no canhão. Essa função espera as características retornarem, e só será enviado algo ao dispositivo se realmente houver algo para ser enviado, senão a funçõa permanece esperando ser chamada. As funções *padButtonPressedCallback* e *padButtonPressedCallbackSlide* são responsáveis por identificar quando os botões do controle e do *slide* são alterados. Quando alterados, informam esses valores para a função *writeData* para ser enviado.

Figura 46 – Função *Bluetooth*.

Fonte: Arquivo Pessoal, 2022.

4.4.4 Identidade Visual

O aplicativo ainda sofrerá bastante alterações no que diz respeito ao visual. Mas para um projeto piloto, está funcional e obedecendo o que precisamos. Assim, na Figura 47, temos uma imagem do aplicativo em seu estado funcional.

Figura 47 – Tela inicial do aplicativo.

Fonte: Arquivo Pessoal, 2022.

A intenção é melhorar mais o visual do arquivo, para que seja bem intuitivo para o cliente. Alguns problemas durante a parte de desenvolvimento visual aconteceram, e não puderam ser resolvidos, mas o aplicativo funcionou, atendeu as expectativas que se esperavam, e por isso os próximos passos serão em melhorias nele.

5 RESULTADOS

No capítulo em questão, apresenta-se uma análise dos resultados obtidos no projeto desenvolvido. Primeiramente, faz-se a verificação quanto ao cumprimento dos requisitos do sistema definidos na seção 4.1. Em seguida, expõem-se os testes realizados e as conclusões obtidas através deles.

5.1 CUMPRIMENTO DOS REQUISITOS

Uma forma de julgar os resultados obtidos no projeto é revisitando os requisitos do sistema definidos na sua fase inicial e verificando se eles foram de fato concluídos.

O controle sobre o canhão foi totalmente satisfeito, uma vez que o tanque foi submetido a testes e tivemos total controle sobre o produto. A comunicação entre controle remoto e comunicação central presente no tanque nos deu uma resposta muito satisfatória em questão de tempo, e isso era buscado também nos requisitos. A resposta ficou na faixa informada pelo prórpio fabricante dos componentes, em torno de 400 ms de velocidade de resposta.

O controle ficou intuitivo para o uso do cliente, pois com a ligação de *LED's* pudemos fazer com que o cliente olhasse e entendesse o que estava sendo realizado no momento do clique do botão.

O acionamento da bomba e do giroflex pode ser totalmente controlado, uma vez que quando acionado os botões no controle remoto, as informações eram rapidamente enviadas a comunicação central e processadas pela mesma.

A distância de testes se deu em duas formas: frontal ao tanque e lateral ao tanque.

Quando os testes se iniciaram defronte ao tanque, tivemos uma área totalmente limpa para comunicação, sem obstáculos perto, ou entre a comunicação de controle e tanque. E isso nos deu uma rede boa de controle, podendo atingir até aproximadamente 50 metros de distância.

Quando os testes foram feitos através da lateral do tanque, começamos a ter a diminuição da distância efetiva, uma vez que estando ao lado do tanque, temos uma estrutura de ferro entre a comunicação central e o controle, e isso faz com que além da grossa espessura do tanque e do material, também tenha o fator do tipo de material. Por ser metálico, houve um desvio de sinal, e isso nos reduziu o sinal para aproximadamente 25 metros.

Quanto ao uso do aplicativo, pudemos notar os mesmos problemas de comunicação em relação ao local de disparo de informações. Quando era usado diretamente de frente, o aplicativo nos dava um controle efetivo grande, podendo fazer a comunicação a 50 metros de distância como exímia qualidade. Porém, quando colocado lateralmente ao tanque, diminuía de fato o seu alcance, diminuindo assim seu poder

de controle.

6 CONCLUSÃO

O desenvolvimento do projeto foi sempre buscando entender quais eram as necessidades do cliente final. Pois é ele quem irá usufruir das vantagens e tecnologias que serão empregadas no produto.

Os desafios enfrentados para a realização desse projeto se deram na parte de desenvolvimento físico da ferramenta. Os tanques de água já estavam em produção, muitos deles saindo para entrega aos clientes, e o canhão eletrônico que viria a fazer parte dos tanques, só começaram a chegar na empresa depois de um certo tempo em que se estava fazendo a parte de programação. O que trouxe uma certa dificuldade para entender os mecanismos que habitavam o canhão eletrônico. Além disso, é uma empresa de implementos agrícolas, e se espera que o foco da empresa seja justamente esse, o que acarretou em alguns atrasos para o começo do desenvolvimento do projeto físico pois esse precisava de componentes eletrônicos que até então a empresa não possuía.

Ao longo de um período de aproximadamente um mês e meio, houve apenas contato com as ferramentas de programação, tentando entender o uso da ferramenta física e quais seriam as consequências desse uso.

A parte de programação de componentes aconteceu de forma fluída, com alguns percalços referentes a soldagem de componentes, ao entendimento de quais componentes seriam usuais, e tentando reaproveitar o pouco que existia dentro da fábrica.

Porém os resultados foram satisfatórios, uma vez que com as ferramentas certas em mãos, e os componentes corretos podemos fazer a ferramenta e desenvolver o sistema.

Os testes iniciais se deram com cada um dos componentes separados, e tentando acoplar separadamente esses itens no tanque de água. Todos os testes foram bem sucedidos, e todas as ferramentas que utilizamos atenderam o que era esperado.

Quando os testes de componentes foram validados, se deu a testagem de componentes todos acoplados num único sistema, e posterior a isso, testagem do controle remoto via *wireless*. Esse então foi o teste para validar todo o projeto e identificar possíveis erros e *bug's* do sistema, o que não foi encontrado. Pequenos problemas que ocorreram foram por causa de *hardware* não conectado corretamente, ou até mesmo por falta de carga na bateria do controle, o que fazia oscilar muito o sistema de sinal *wireless*.

Com tudo ajustado, os testes pularam para a parte de distância, e pudemos notar que o que se esperava foi muito bem atendido. No caso da rede *wireless* conseguimos manter uma distância considerada efetiva muito boa, uma vez que conseguimos controlar o canhão de água em uma distância aproximada de 50 metros com

muito sucesso. Na rede *Bluetooth*, o controle foi ainda maior, com mais de 50 metros de distância.

No que diz respeito as disciplinas cursadas durante a graduação e o envolvimento delas com esse projeto, vale destacar a disciplina DAS5102 - Fundamentos da Estrutura da Informação, onde pude aplicar um pouco sobre programação orientada a objetos dentro do projeto. Também vale destaque para a disciplina DAS5332 - Arquitetura e Programação de Sistemas Microcontrolados, onde toda a parte de programação de microprocessadores e microcontroladores foi utilizada se baseando nos tipos e formatos de instruções, modos de endereçamento, linguagem *C++*, barramento padrões, entre outros itens.

Agora, o projeto continuará em observação e continuará sendo melhorado. Os próximos passos serão desenvolver um circuito impresso, com todos os componentes, para poder assim trazer maior comodidade na hora da fábrica implementar o produto junto ao tanque, uma vez que entrado em produção e disponibilizado junto ao tanque será necessário uma equipe ou pessoa apenas para fazer essa soldagem de componentes. Será também feito um projeto de carcaça de controle, ainda a ser definido, para acoplar todos esses itens dentro dele e dar um melhor visual ao controle remoto. Também será desenvolvido uma placa de ciruito impresso para a central que ficará no tanque. Essa central precisa ser desenvolvida tomando alguns cuidados pois ficará exposta a chuva, ao vento, e as mais diversas condições climáticas. Por isso, o projeto ainda irá melhorar.

O aplicativo também entrará em produção e por isso tomará uma identidade visual melhorada. O cerne do projeto continuará o mesmo: fazer o controle do tanque MF com os componentes que hoje estão expostos no aplicativo. O que irá mudar é apenas a identidade visual, com outras fontes, outra imagem de fundo e afins.

Ter desenvolvido esse projeto junto a São José Industrial foi bastante desafiador. Uma vez que os processos para desenvolver e aplicar produtos dentro de uma fábrica devem seguir padrões. E principalmente, devem trazer melhoria para o produto buscando reduzir custos. E isso conseguimos entregar com o projeto de controle do canhão de água para o tanque MF.

Foi um grande prazer dividir meus dias com os colegas da empresa, onde pude aprender com cada um sobre cada produto comercializado. Além do mais, pude me interar dos projetos que a empresa está desenvolvendo, e qual o rumo que irá tomar, vendo a indústria 4.0 e a agricultura 4.0 avançar cada vez mais.

REFERÊNCIAS

101, Components. **Resistor**., 2019. Disponível em: https://components101.com/resistors/resistor. Acesso em: 30 abr. 2022.

ALVARO, Alexandre. **Utilizando botão de forma correta com Arduino.**, 2020. Disponível em: https://inteligenciadascoisas.com/post/20200517-pullup/. Acesso em: 20 abr. 2022.

ARDUINO. **Arduino IDE 1.8.19**., 2022a. Disponível em: https://www.arduino.cc/en/software. Acesso em: 15 mai. 2022.

ARDUINO. **Arduino Mega 2560 R3**., 2022b. Disponível em: https://docs.arduino.cc/static/b96faf9f7c3b64ff83e6efb96eb3d0f5/A000067-datasheet.pdf. Acesso em: 30 abr. 2022.

ARDUINO. **Arduino Nano**., 2014. Disponível em: https://docs.arduino.cc/static/a11aea39d1f2eb8da83934f041cba5cc/A000005-datasheet.pdf. Acesso em: 30 abr. 2022.

CALAZANS, André. **TP4056 – Carregando Baterias de Lítio Corretamente**., 2020. Disponível em: https://blog.eletrogate.com/tp4056-o-carregando-baterias-de-litio-corretamente/. Acesso em: 20 abr. 2022.

CIRCUITO, Curto. **O que é um LED?** São Paulo, 2021. Disponível em: https://www.curtocircuito.com.br/blog/eletronica-basica/o-que-e-um-led. Acesso em: 20 abr. 2022.

DEVELOPERS, Google. **Conheça o Android Studio**., 2022. Disponível em: https://developer.android.com/studio/intro. Acesso em: 1 mai. 2022.

ELETRÔNICA, Soft. **Módulos Bluetooth Low Energy BLE-1010**. Brasília/DF, 2018. Disponível em: https://www.filipeflop.com/img/files/download/Datasheet-BLE-1010-MPCBA-Modulo-Bluetooth.pdf. Acesso em: 30 abr. 2022.

ENGINEERS, Last Minute. **How nRF24L01+ Wireless Module Works** & **Interface with Arduino**., 2018. Disponível em: https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/. Acesso em: 20 abr. 2022.

ENGINEERS, Last Minute. **Interface Two Channel Relay Module with Arduino**., 2019. Disponível em: https://lastminuteengineers.com/two-channel-relay-module-arduino-tutorial/. Acesso em: 30 abr. 2022.

FLUTTER. **Build apps for any screen.**, 2015. Disponível em: https://flutter.dev/. Acesso em: 14 abr. 2022.

GOOGLE. **Dart Programming Language**., 2022. Disponível em: https://dart.dev/. Acesso em: 1 mai. 2022.

INDUSTRIAL, São José. **Calcareadeiras e Distribuídores de Sólidos.** Sao Jose do Inhacorá/RS, 2022a. Disponível em: https://saojoseindustrial.com.br/detalhe-produto/36. Acesso em: 14 abr. 2022.

INDUSTRIAL, São José. **Carretas Graneleiras.** Sao Jose do Inhacorá/RS, 2022b. Disponível em: https://saojoseindustrial.com.br/detalhe-produto/32. Acesso em: 14 abr. 2022.

INDUSTRIAL, São José. **Empresa desenvolvedora de implementos agrícolas.** Sao Jose do Inhacorá/RS, 2022c. Disponível em: https://saojoseindustrial.com.br/. Acesso em: 11 abr. 2022.

INDUSTRIAL, São José. **Escarificadores e Grades.** Sao Jose do Inhacorá/RS, 2022d. Disponível em: https://saojoseindustrial.com.br/detalhe-produto/27. Acesso em: 14 abr. 2022.

INDUSTRIAL, São José. **Produtos.** São José do Inhacorá/RS, 2022e. Disponível em: https://saojoseindustrial.com.br/produtos. Acesso em: 14 abr. 2022.

INDUSTRIAL, São José. **Tanque Mata Fogo.** Sao Jose do Inhacorá/RS, 2022f. Disponível em: https://saojoseindustrial.com.br/detalhe-produto/37. Acesso em: 14 abr. 2022.

INDUSTRIAL, São José. **Tanques de Chorume.** Sao Jose do Inhacorá/RS, 2022g. Disponível em: https://saojoseindustrial.com.br/detalhe-produto/8. Acesso em: 14 abr. 2022.

N.MICHELIN. **Solução para combate a incêndio.** Erechim/RS, 2022. Disponível em: https://www.nmichelin.com.br/. Acesso em: 18 abr. 2022.

NERY, Gustavo. **Guia Definitivo de uso da Ponte H L298N**., 2020. Disponível em: https://blog.eletrogate.com/guia-definitivo-de-uso-da-ponte-h-l298n/. Acesso em: 30 abr. 2022.

OLIVEIRA, Euler. **Como usar com Arduino – Módulo Joystick KY.**, 2020. Disponível em: https://www.curtocircuito.com.br/blog/eletronica-basica/o-que-e-um-led. Acesso em: 20 abr. 2022.

POINT, Tutorials. **Flutter - Introduction to Gestures.**, 2020. Disponível em: https://www.tutorialspoint.com/flutter/flutter_introduction_to_gestures.htm. Acesso em: 20 jul. 2022.

REACT. **Uma biblioteca JavaScript para criar interfaces de usuário**., 2022. Disponível em: https://pt-br.reactjs.org/. Acesso em: 1 mai. 2022.

REFERENCE, Arduino. **Libraries**., 2022. Disponível em: https://www.arduino.cc/reference/en/libraries/. Acesso em: 15 mai. 2022.

SAFETY, Eco. **Calda Mata Fogo F500.** Eco Safety, 2022. Disponível em: https://www.ecosafety.com.br/f-500-encapsulator-agent. Acesso em: 14 abr. 2022.

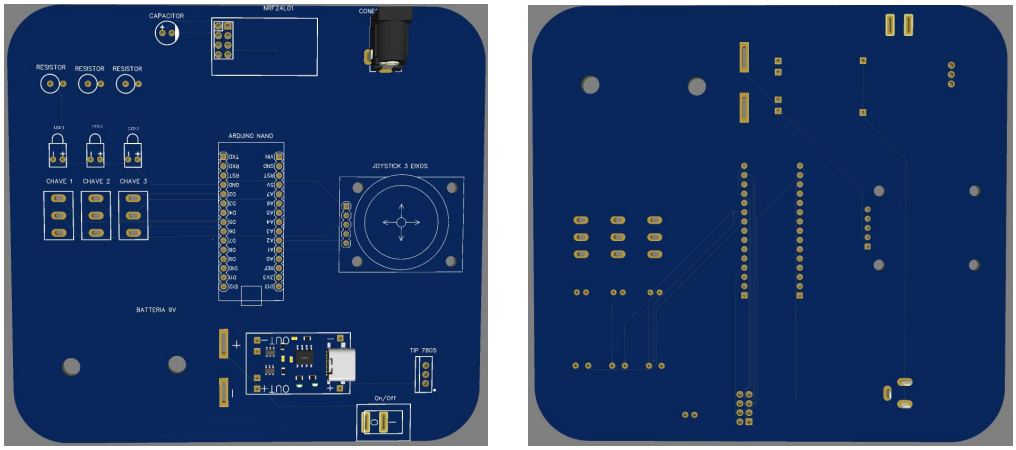
APÊNDICE A – DESENVOLVIMENTO DE CIRCUITO IMPRESSO

Como citado na 6, o projeto deve tomar mais forma e ser mais robusto com o desenvolvimento de placas de circuito impresso. Essas placas podem ocupar um papel importante dentro do projeto, pois agora o passo seguinte é colocar em produção e disponibilizar para o cliente o controle remoto, sendo um adicional ao tanque.

As placas de circuito impresso foram desenhadas para trazer um acabamento mais moderno ao projeto, e evitar assim que acabe acumulando cabos desnecessários tanto dentro do controle remoto quanto da central presente no tanque. As placas irão acoplar os componentes que são necessários para o projeto e irão desempenhar com maestria o controle sobre o tanque MF.

A Figura 48 mostra a placa de circuito impresso que estará presente nos controles, com os componentes que foram utilizados durante o projeto também.

Figura 48 – Placa de circuito interno do controle - frente e verso.



Fonte: Arquivo Pessoal, 2022.

Dessa forma, teremos a substituição dos fios mantendo o conceito original projetado.