INSTITUTO FEDERAL
São Paulo
Câmpus Presidente Epitácio

MOSTRA CIENTÍFICA, CULTURAL E TECNOLÓGICA DE 2021

Protocolo e Plataforma de Software para Veículos Remotamente Controlados

Luan N. Sibinel[1], André L. Olivete[2]

1. Discente do Curso Bacharelado em Ciência da Computação – IFSP – Câmpus Presidente Epitácio; 2. Docente – IFSP – Câmpus Presidente Epitácio, Área de Computação.
E-mails: luan.negroni@aluno.ifsp.edu.br, olivete@ifsp.edu.br

(Área: A – Ciências Exatas e da Terra)

Introdução

Os veículos remotamente controlados são utilizados a bastante tempo para acessar locais que oferecem perigo ao ser humano. O Colossus, robô criado pela empresa francesa Shark Robotics em 2017 foi utilizado pelos bombeiros para auxiliar no combate ao incêndio de 2019 na catedral de Notre Dame (IEEE, 2020).

Em Silva et al. (2015) é apresentado o ARCODE, Automóvel Remotamente Controlado para Obtenção de Dados de Exploração, um sistema para aquisição de dados de exploração em áreas florestais, como imagens de animais e florestas, quantidade de luminosidade, temperatura e umidade do ambiente. O sistema é composto de um veículo terrestre não tripulado e um dispositivo de controle que pode ser um smartphone ou computador pessoal, e a comunicação entre eles é realizada utilizando *bluetooth*.

O objetivo desse projeto é o desenvolvimento de um protocolo de comunicação entre um veículo remotamente controlado e uma controladora remota, de modo que a controladora consiga controlar a movimentação do veículo e requisitar fotos, vídeo e leituras de possíveis sensores.

Metodologia

Para o desenvolvimento da comunicação, foi escolhida a criação de uma biblioteca em C++ para o veículo e uma em Java para a estação de controle.

No desenvolvimento destas bibliotecas, foi usado o Java 16 e o software IntelliJ IDEA para a implementação da biblioteca em Java. Na biblioteca em C++ para o veículo, foi usada a versão do C++ 11 com as bibliotecas do Arduino e os softwares Arduino e Visual Studio Code.

Para a formulação do protocolo, foram criadas 42 mensagens, separadas em 4 categorias, sendo estas: Configuração com 13 mensagens, Dados com 15, Movimentação com 6 e Erros com 8 mensagens.

As mensagens de configuração iniciam e terminam a comunicação, assim como a mantém através do pacote *Heartbeat* que é trocado periodicamente entre as partes para confirmar que as duas estão conseguindo receber e enviar pacotes.

Após a confirmação da conexão, as duas partes poderão trocar mensagens de Dados, Movimentação ou Erros. Sendo estas para:

- Dados: leitura de possíveis sensores, requisição de fotos e início ou término de uma *stream* de vídeo.
- Movimentação: pacotes enviados pela controaldora para o veículo, requisitando uma ação nos motores.
- Erros: pacotes enviados pelo veículo como resposta a alguns pacotes de controladora que podem gerar erros ou não executar como devido, como por exemplo mudar a resolução da câmera ou iniciar uma *stream* de vídeo.

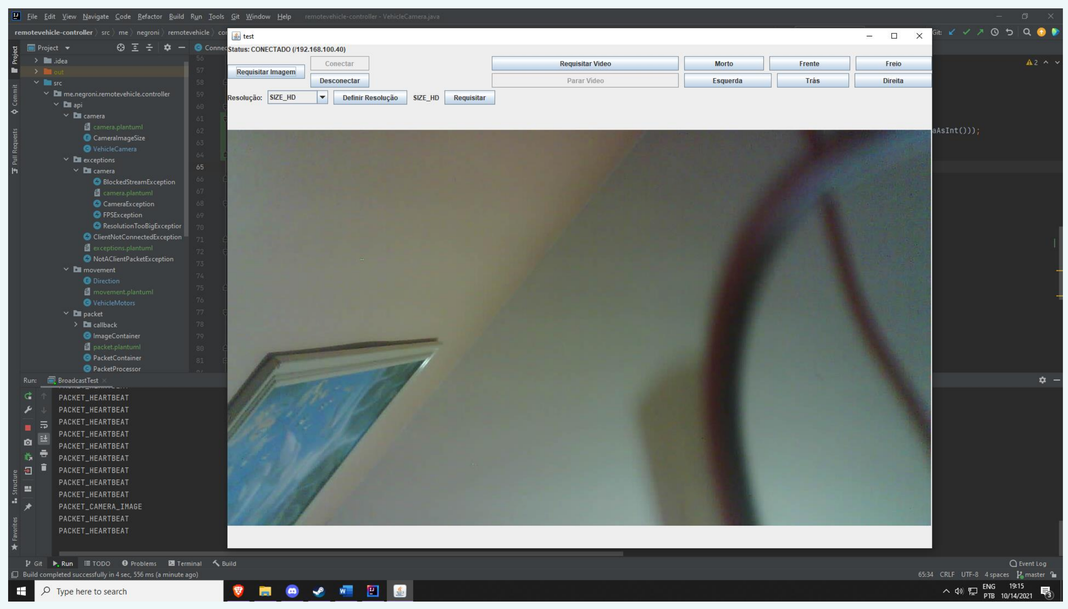
Foi separado o protocolo em 3 *sockets* (interfaces que conectam entre dois pontos e permitem a troca de mensagens), sendo estes, o *socket* de comandos usando TCP, imagens usando TCP e o de *broadcast* usando UDP.

O *socket* de *broadcast* do veículo, assim que ele conecta a uma rede WiFi, começa a transmitir em intervalos regulares um *broadcast* na rede em uma porta específica (por padrão 6888). Quando a controladora recebe este pacote, ele pode enviar um pacote de requisição de conexão para o veículo (usando o IP que enviou o pacote UDP) no *socket* de comandos (por padrão na porta 6887) e depois trocar mensagens de dados e movimentação no mesmo *socket*. Imagens são trocadas no *socket* de imagens (por padrão 6889), erros e mensagens de configuração são trocados no *socket* de comandos.

Resultados

Foram implementadas bibliotecas em Java e em C++ (para uso na IDE do Arduino). Também foi implementada uma interface de teste em Java para o desenvolvimento do protocolo, conforme apresentado na FIGURA 1.

Figura 1. Interface de teste.



Fonte: próprio autor.

A implementação do veículo permite mais configurações para o funcionamento, como bloquear o início de uma *stream* de vídeo, tempo para conectar ao WiFi, SSID e senha da rede entre outras. Já a implementação em Java é mais orientada e eventos, oferecendo métodos onde o programador chama a função que ele deseja e oferece uma função anônima ou não que será chamada assim que for obtida uma resposta da chamada ao veículo.

Conclusões

A definição do protocolo de comunicação e a implementação de bibliotecas facilita o futuro desenvolvimento de projetos parecidos, já apresentando bibliotecas que o programador pode apenas importar, fazer uma mínima configuração e usar no projeto, já possibilitando a conexão, movimentação e troca de pacotes de informação como imagens, vídeos e leituras de sensores.

Bibliografia

BRANDÃO, Raphael. **ESP32CAM Pin Notes**. Disponível em: <https://github.com/raphaelbs/esp32-cam-ai-thinker/blob/master/docs/esp32cam-pin-notes.md>. Acesso em 9 set. 2021.

IEEE. **Colossus** – ROBOTS: Your Guide to the World of Robotics. Disponível em: <https://robots.ieee.org/robots/ colossus/>. Acesso em 24 Agosto, 2021.

SILVA, Jadson S.; ARANCIBIA, Josilene C. R.; OLIVEIRA, Yuri S.; ALVAREZ, Ana B. **Veículo terrestre não tripulado controlado remotamente para obtenção de dados de exploração**. In: Anais da 5ª Mostra Nacional de Robótica (MNR 2015), Uberlândia, MG, p. 569-572, 2015.