

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA**

JHONATAN DE FREITAS LANG

**INTEGRAÇÃO DE ROBÔ COM SISTEMA DE VIDEOMONITORAMENTO PARA
RONDAS DE VIGILÂNCIA REMOTAS**

FLORIANÓPOLIS, 2024

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE
SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA**

JHONATAN DE FREITAS LANG

**INTEGRAÇÃO DE ROBÔ COM SISTEMA DE VIDEOMONITORAMENTO PARA
RONDAS DE VIGILÂNCIA REMOTAS**

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro Eletrônico.

Orientador:
Prof. Me. Daniel Lohmann

FLORIANÓPOLIS, 2024

Ficha de identificação da obra elaborada pelo autor.

Lang, Jhonatan de Freitas

Integração de Robô com Sistema de Videomonitoramento para Rondas de Vigilância Remotas / Jhonatan de Freitas Lang; orientador, Prof. Me. Daniel Lohmann – Florianópolis, SC, 2024. 67 p.

Trabalho de Conclusão de Curso (TCC) – Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, Campus Florianópolis. Bacharelado em Engenharia Eletrônica. Departamento Acadêmico de Eletrônica.
Inclui Referências.

1. Robô para videomonitoramento. 2. Robotino. 3. Intelbras Defense IA. 4.ONVIF. 5. RTSP. I Lohmann, Daniel. II. Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina. Departamento Acadêmico de Eletrônica. III. Integração de Robô com Sistema de Videomonitoramento para Rondas de Vigilância Remotas

INTEGRAÇÃO DE ROBÔ COM SISTEMA DE VIDEOMONITORAMENTO PARA RONDAS DE VIGILÂNCIA REMOTAS

JHONATAN DE FREITAS LANG

Este Trabalho foi julgado adequado para obtenção do Título de Engenheiro Eletrônico em setembro de 2024 e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação Ciência, e Tecnologia de Santa Catarina.

Florianópolis, 27 de setembro, 2024.

Banca Examinadora:

Documento assinado digitalmente
 **DANIEL LOHMANN**
Data: 01/10/2024 11:20:55-0300
Verifique em <https://validar.iti.gov.br>

Prof. Daniel Lohmann. Msc. Eng.

Documento assinado digitalmente
 **JULIO FELLER GOLIN**
Data: 01/10/2024 10:12:34-0300
Verifique em <https://validar.iti.gov.br>

Prof. Julio Feller Golin Msc. Eng.

Documento assinado digitalmente
 **RENAN AUGUSTO STARKE**
Data: 01/10/2024 11:16:40-0300
Verifique em <https://validar.iti.gov.br>

Prof. Renan Augusto Starke Dr. Eng.

AGRADECIMENTOS

Gostaria de expressar minha imensa gratidão a todos que me apoiaram e, nos momentos em que me faltou propósito, não permitiram que eu desistisse.

À minha querida Amanda Acauan de Aquino, pelo amor e suporte incondicional.

Aos meus pais e familiares, cujo apoio foi fundamental para que eu chegasse até aqui.

Aos meus amigos Alexandro Gehlen, Augusto Rodrigues e Lucas Cleber, que sempre se interessaram e perguntaram "Como está o TCC?" nas horas certas.

Ao meu amigo Pedro Saraiva, por me mostrar o caminho.

Ao meu professor orientador Daniel Lohmann, por sua paciência e dedicação durante todos os semestres.

Ao Gregory Gomes e ao Departamento de Mecatrônica do IFSC, por disponibilizarem o Robotino, tornando este trabalho possível.

A todos vocês, meus mais sinceros agradecimentos.

RESUMO

O aumento da violência e da sensação de insegurança no Brasil, somados a falta de capacidade do poder público para lidar com a situação, tem impulsionado um mercado de segurança privada, com uma gama de serviços que objetivam preservar a vida, os bens e o patrimônio. Como protagonista deste tipo de trabalho, temos a figura do vigilante, sendo ele o responsável por operar sistemas de videomonitoramento, realizar escoltas, rondas de vigilância e intervenções necessárias. Porém ao atuar prevenindo riscos provenientes de ações criminosas, este profissional acaba expondo a própria vida, correndo o risco de ser surpreendido por agentes mal-intencionados durante o exercício de suas funções. Observando esta problemática, propõe-se neste trabalho, o desenvolvimento da integração entre um robô móvel com uma câmera embarcada e um sistema de videomonitoramento, para realização de rondas de vigilância de maneira remota, de forma que o vigilante possa visualizar a imagem e comandar o deslocamento do robô estando em um lugar seguro. Para o desenvolvimento deste trabalho, realizou-se inicialmente uma pesquisa bibliográfica com o estudo de métodos, protocolos e ferramentas necessários à implementação do protótipo, além da pesquisa aplicada na integração entre o robô móvel didático, Robotino da Festo e sistema de videomonitoramento Defense IA da Intelbras, analisando aspectos necessários para aplicabilidade da solução para cenários reais. Por fim, desenvolveu-se um software para interconexão dos componentes, convertendo uma transmissão de imagens via UDP em fluxo RTSP e em sequência em ONVIF, além de rotas API para comando dos motores do robô. Obteve-se um desempenho de acordo com parâmetros desejáveis em sistemas de videomonitoramento com a visualização de imagem em tempo real, com atraso previsível e constante menor do que um segundo, a uma taxa de quinze frames por segundo e controle de deslocamento via botões na interface, mas houveram dificuldades associadas com a capacidade de processamento do Robotino, com eventuais travamentos do sistema operacional.

Palavras-chave: Robô para videomonitoramento. Robotino. Intelbras Defense IA. ONVIF. RTSP.

ABSTRACT

The rise in violence and the increasing sense of insecurity in the country, combined with the public sector's inability to adequately address these issues, have driven the growth of the private security market. This market offers a variety of services aimed at safeguarding lives, assets, and property. At the forefront of these efforts is the security guard, responsible for operating video surveillance systems, conducting escorts, surveillance patrols, and taking necessary action when required. However, while mitigating risks from criminal activities, this professional often puts their own life at risk, potentially facing malicious agents while performing their duties. In response to this challenge, this paper proposes the development of an integrated solution combining a mobile robot equipped with an onboard camera and a video surveillance system. This allows remote patrols, enabling the security guard to view the footage and control the robot's movements from a safe location. The development process involved a literature review, studying methods, protocols, and tools required for the prototype's implementation, and applied research on integrating Festo's Robotino mobile robot with Intelbras' Defense IA video surveillance system. This exploratory qualitative research analyzed aspects necessary for real-world applicability of the solution. In this work, a software was developed to interconnect the components, converting image transmission via UDP into RTSP stream and subsequently to ONVIF, as well as API routes to control the robot's motors. As result, the system achieved good performance in image display with low delay and movement control via buttons on the interface, though challenges were encountered with Robotino's processing capacity with occasional operating system crashes

Keywords: Video surveillance robot, Robotino, Intelbras Defense IA, ONVIF, RTSP.

LISTA DE FIGURAS

Figura 1 - Topologia geral do sistema	18
Figura 2 - Central de monitoramento COR.....	19
Figura 3 - Defense IA	21
Figura 4 - Matriz sensor CMOS.....	22
Figura 5 - Campos de aplicação de câmeras.....	22
Figura 6 - Operação de Drones UAV e UGV.....	25
Figura 7 – Robotino	26
Figura 8 - Matriz de comparação de protocolos de vídeo	28
Figura 9 - Protocolo RTSP	31
Figura 10 - Topologia componentes RTSP	32
Figura 11 - Diagrama do processo de transcodificação com FFmpeg	34
Figura 12 - Formato de mensagem SOAP	36
Figura 13 - Métodos da API REST	37
Figura 14 - Página Web Robotino	42
Figura 15 - Atraso entre capturas de imagem	44
Figura 16 - Captura de pacotes de imagens do Robotino	45
Figura 17 - Cabeçalho .jpg na captura dos pacotes	46
Figura 18 - Recuperando imagem corrompida.....	47
Figura 19 - Fluxograma aquisição de imagens.....	48
Figura 20 - Topologia da transmissão de vídeo	49
Figura 21 - Serviço servidor ONVIF	50
Figura 22 - Adição de câmera no Defense IA.....	51
Figura 23 - Topologia completa.....	51
Figura 24 - Câmera do Robotino no Defense IA	52
Figura 25 - Configuração de botão em canal origem	53
Figura 26 - Vinculo de comando URL HTTP	53
Figura 27 - Botões de controle na interface de videomonitoramento	54
Figura 28 - Teste de ronda de vigilância	55

LISTA DE QUADRO

Quadro 1 - Especificações Robotino	27
Quadro 2 - Exemplo FFmpeg em linha de comando	34
Quadro 3 - Start WEB Server	41
Quadro 4 - Requisições da API HTTP REST do Robotino	42

LISTA DE SIGLAS

API	Application Programming Interface
ARQ	Automatic Repeat Request
CFTV	Circuito Fechado de Televisão
CMOS	Complementary Metal-Oxide-Semiconductor
COR	Centro de Operações do Rio
DVR	Digital Video Recorder
FEC	Forward Error Correction
FOV	Field of View
HD	High Definition
HDMI	High-Definition Multimedia Interface
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
IA	Inteligência Artificial (Artificial Intelligence - AI)
I/O	Input/Output
IoT	Internet of Things
IP	Internet Protocol
IR	Infrared
ISS	Intelligent Surveillance System
LAN	Local Area Network
NACK	Negative Acknowledgment
NVR	Network Video Recorder
ONVIF	Open Network Video Interface Forum
OpenCV	Open Source Computer Vision Library
PAN	Personal Area Network
PC	Personal Computer
PTZ	Pan-Tilt-Zoom
QoS	Quality of Service
RGB	Red, Green, Blue
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RIST	Reliable Internet Stream Transport
RPC	Remote Procedure Call

RTCP	Real-time Transport Control Protocol
RTMP	Real-Time Messaging Protocol
RTSP	Real-Time Streaming Protocol
RTP	Real-Time Protocol
SDRAM	Synchronous Dynamic Random-Access Memory
SIP	Session Initiation Protocol
SOAP	Simple Object Access Protocol
SRT	Secure Reliable Transport
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UDT	UDP-based Data Transfer Protocol
USB	Universal Serial Bus
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
URL	Uniform Resource Locator
VR	Virtual Reality
VGA	Video Graphics Array
WEB	World Wide Web
Wi-Fi	Wireless Fidelity
WSNs	Wireless Sensor Networks

SUMÁRIO

1	INTRODUÇÃO	14
1.1	Justificativa	15
1.2	Definição do Problema	16
1.3	Objetivo Geral.....	16
1.4	Objetivos Específicos	16
1.5	Estrutura do Trabalho.....	17
2	FUNDAMENTAÇÃO TEÓRICA.....	18
2.1	Sistema de videomonitoramento	19
2.1.1	Software Defense IA	20
2.1.2	Câmeras de vigilância	21
2.2	Robôs para monitoramento	24
2.2.1	Robotino da Festo	25
2.3	Protocolos de transmissão de vídeo sobre IP.....	27
2.3.1	Protocolo RTSP	30
2.3.2	Protocolo ONVIF	32
2.3.3	Framework FFmpeg	33
2.4	Software Middleware	35
2.4.1	Interface de Programação de Aplicativos (API).....	35
2.5	Requisitos de projeto.....	37
3	METODOLOGIA	39
4	DESENVOLVIMENTO	40
4.1	Seleção do Robô e Software para o sistema de vigilância remota.....	40
4.2	Comunicação com o Robotino	40
4.3	Estudo API REST do Robotino	42
4.4	Estudo API SOAP do Robotino	44
4.5	Transmissão de vídeo RTSP	48
4.6	Conversão de RTSP para ONVIF	49
4.7	Configuração Defense IA e adição de câmera ONVIF.....	50
4.8	Comandos dos motores via interface	52
4.9	Resultado final	54
5	CONSIDERAÇÕES FINAIS.....	55
5.1	Sugestões para trabalhos futuros	56

REFERÊNCIAS.....57
APÊNDICES60

1 INTRODUÇÃO

Segundo o Atlas da Violência 2023, em 2021 foram registrados no Brasil 33.039 homicídios por armas de fogo, sendo esse número 69,1% dos homicídios totais naquele ano. A pesquisa também indica existir uma relação direta entre o aumento da difusão de armas de fogo com as taxas de homicídios e latrocínios e ressalta que a resposta pública à violência tem enfrentado desafios significativos, incluindo cortes orçamentários e mudanças legislativas que podem ter exacerbado a situação da violência e o aumento de armas em circulação (IPEA; FBSP, 2023).

Essa percepção de insegurança e falta da capacidade do poder público para lidar com o aumento da violência tem impulsionado um mercado de segurança privada no país, para atender a demanda por aquilo que é uma necessidade básica.

De acordo com Santos e Melo (2021), as empresas de segurança privada atuam nos segmentos de segurança patrimonial, com o objetivo de preservar a vida, os bens e o patrimônio, agindo para prevenir os riscos proveniente de ações criminosas em comércios, indústrias, órgãos públicos e eventos. Elas operam também no segmento de escolta de pessoas, bens e valores.

Em ambos os segmentos mencionados, o protagonista é o vigilante, sendo ele o responsável por operar os sistemas de monitoramento, realizar escoltas, rondas e intervenções quando necessário. Ele é o sujeito que coloca a sua vida em risco ao se expor durante o exercício da sua profissão.

Com um olhar voltado para este mercado e buscando melhorar a segurança e eficiência no trabalho de segurança privada, os avanços na tecnologia têm trazido formas de atuação mais preventivas ao monitoramento, tais como o uso analíticos de vídeo nas centrais de monitoramento ou de drones e robôs em rondas.

O estudo de Huang *et al.* (2019), por exemplo, explora abordagens e estratégias utilizadas em sistemas de patrulhamento com robôs, substituindo os vigilantes em rondas. O estudo aponta que robôs podem ser enviados para patrulhar áreas que podem ser perigosas ou inacessíveis para humanos, como zonas de desastre, áreas contaminadas ou locais com risco de violência, minimizando o risco para a vida humana. Além disso, os robôs podem operar de maneira consistente e precisa, seguindo rotas programadas sem se distrair ou se cansar.

Diante dessa problemática e a fim de propor uma abordagem mais segura para o procedimento de ronda em segurança patrimonial, este trabalho tem como objetivo realizar a integração do robô de aprendizagem Robotino da Festo com o sistema de videomonitoramento Defense IA® da Intelbras, uma renomada empresa nacional do segmento de segurança, para as rondas comandadas remotamente. Para isso, foi realizada uma pesquisa qualitativa, explicativa e aplicada no laboratório LPDE do Departamento Acadêmico de Eletrônica, no Instituto Federal de Santa Catarina (IFSC) – Câmpus Florianópolis, no ano de 2024.

Este trabalho utiliza como base o Trabalho de Conclusão de curso “Aplicação de um Veículo Guiado Automatizado (AGV) no Contexto de IoT Aplicada a Inteligência Perimetral”, realizado em 2023, por Pedro E. Saraiva, dando sequência ao que foi desenvolvido. As contribuições aqui são a integração com a câmera e o controle dos motores de uma forma alternativa, utilizando botões na interface do software de videomonitoramento.

1.1 Justificativa

O Projeto de Lei nº 1043/2021, que dispõe sobre o Estatuto dos Vigilantes, propõe os limites de atuação, direitos e deveres do profissional vigilante, sendo enfático em relatar os perigos aos quais os vigilantes são submetidos e sobre a necessidade de pensarmos na vida desses trabalhadores.

Não podemos nos preocupar somente com os bens vigiados. Temos que pensar também na vida do trabalhador da segurança privada. Na vida do vigilante! Do exposto, é imprescindível trazer para o debate parlamentar a total falta de segurança que o vigilante enfrenta para desempenhar sua atividade. Chega de descaso com esse profissional que oferece sua vida na proteção dos bens e da integridade física de terceiros! VIDAS DE VIGILANTES IMPORTAM! (Brasil, 2021).

Propor novas abordagens mais seguras ao trabalho dos vigilantes é algo tão incentivado quanto vital para a vida desses profissionais. Portanto, a importância desta pesquisa está em implementar a integração entre um robô com uma câmera embarcada e um sistema de videomonitoramento, permitindo que os vigilantes executem as patrulhas remotamente, visualizando as imagens do perímetro sem sair da central de monitoramento, operando o robô à distância e em um local seguro, sem correr o risco de serem interceptado por criminosos em trechos da ronda.

1.2 Definição do Problema

O procedimento de ronda é um processo importante no monitoramento de segurança patrimonial para prevenção de perdas tangíveis e intangíveis. No entanto, ao executar a ronda de segurança de maneira tradicional, o vigilante pode colocar a sua própria vida em risco. Ao se expor ao desconhecimento do que lhe espera no trajeto, o vigilante pode ser surpreendido por agentes externos mal-intencionados. Considerando isso, é importante encontrar uma forma para que a ronda de segurança possa ser executada remotamente e sem exposição ao risco.

Dessa maneira, a questão-problema que orienta este Trabalho de Conclusão de Curso é: de que forma se pode integrar um robô a um sistema de videomonitoramento para executar uma ronda de segurança de forma remota?

1.3 Objetivo Geral

O objetivo geral deste trabalho é integrar o robô de aprendizagem Robotino com o sistema de videomonitoramento Defense IA, para a realização de rondas de segurança comandadas remotamente.

1.4 Objetivos Específicos

- a) Desenvolver uma aplicação que capture imagens da câmera embarcada no Robotino em tempo real;
- b) Converter a sequência de imagens capturadas em um stream de vídeo;
- c) Simular uma câmera IP, utilizando o stream de vídeo como fonte de entrada de imagem;
- d) Adicionar e configurar uma câmera IP virtual no sistema de videomonitoramento;
- e) Projetar e implementar rotas API para o comando dos motores do Robotino.

1.5 Estrutura do Trabalho

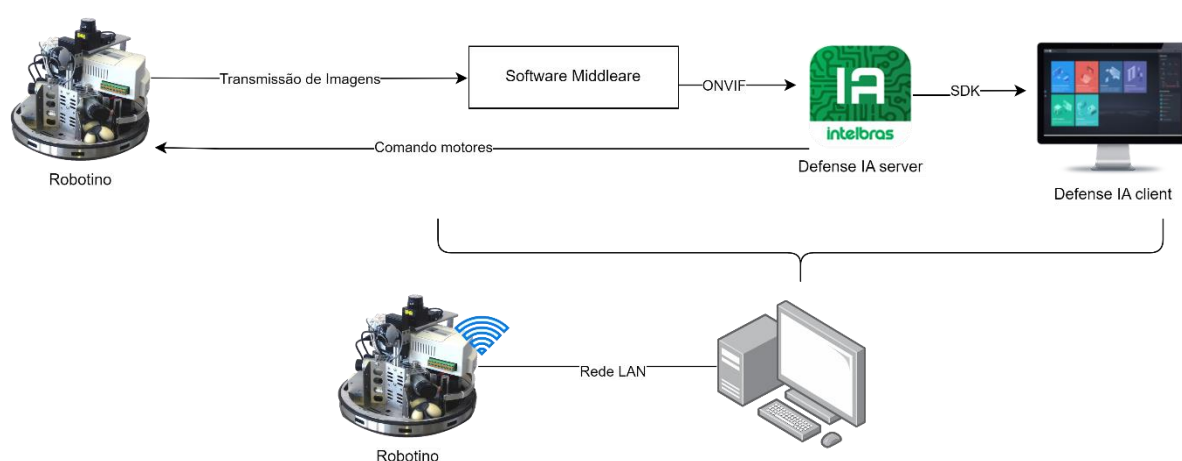
Este trabalho foi organizado da seguinte forma: no segundo capítulo, será apresentada uma revisão bibliográfica dos produtos, protocolos e *frameworks* necessários para o desenvolvimento da integração entre o robô e o sistema de videomonitoramento escolhidos para este trabalho. No capítulo seguinte, é indicada a metodologia utilizada na pesquisa, com a descrição da implementação de um protótipo em laboratório. O quarto capítulo explora os resultados dos testes do protótipo, comparando os resultados para a verificação do atendimento dos requisitos. Por fim, o capítulo cinco traz as considerações finais do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Para melhorar o entendimento deste trabalho, faz-se necessária a introdução de alguns tópicos conceituais. Primeiro sobre a definição de sistemas de videomonitoramento, para que servem e as diferenças de operações humanas e operações automatizadas em um sistema. Nessa seção também é aprofundado o estudo sobre a plataforma de segurança utilizada neste trabalho, o Defense IA, e câmeras de vigilância, destacando características de imagem necessárias para um sistema de videomonitoramento. Na seção seguinte são abordados casos onde robôs são utilizados para operações semelhantes a proposta deste trabalho, como se deu a escolha do Robotino para esta pesquisa e suas especificações técnicas. A terceira seção do capítulo explora os protocolos de transmissão de vídeo, em busca do mais aderentes à sistemas de videomonitoramento e que possam ser utilizados para transmitir as imagens do Robotino, e para integração com o Defense IA. Por fim, a quarta seção explica o conceito de software *middleware*, e das interfaces de comunicação utilizadas entre componentes do sistema.

A Figura 1 demonstra como os componentes do sistema se conectam de uma forma geral, sendo que toda aplicação desenvolvida é embarcada em um computador.

Figura 1 - Topologia geral do sistema



Fonte: Elaboração própria (2024)

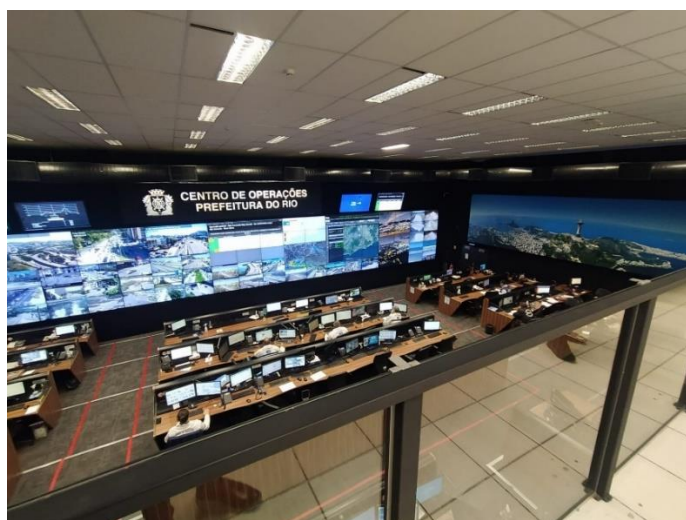
2.1 Sistema de videomonitoramento

Os sistemas de videomonitoramento em CFTV (Circuito Fechado de Televisão) são tecnologias utilizadas para a vigilância e a segurança de ambientes, tanto públicos quanto privados. Esses sistemas consistem em um conjunto de câmeras de vigilância, monitores, dispositivos de gravação de vídeo e softwares de análise que trabalham em conjunto para capturar, transmitir e armazenar imagens de determinadas áreas. Sistemas de videomonitoramento servem tanto para a visualização das imagens e operações de segurança em condomínios e empresas, quanto para aplicações de segurança pública, gestão de tráfego e gerenciamento de multidões (Vijeikis *et al.*, 2021).

De acordo com Kruegle (1995), um sistema de videomonitoramento em CFTV desempenha um papel crucial na proteção da vida humana em situações de emergência e desastre, possibilitando que oficiais de segurança monitorem locais remotos, avaliem em tempo real as prioridades nas áreas e deem a atenção imediata necessária, acionando a polícia e deslocando equipes de patrulhas.

A Figura 2 é uma foto tirada na central de monitoramento do COR (Centro de Operações Rio) da cidade do Rio de Janeiro (RJ), onde trabalham em conjunto órgãos como Defesa Civil, Guarda Municipal, Centro Integrado de Mobilidade, secretarias municipais, entre outros setores. Nessa central de monitoramento, os funcionários operam o sistema de videomonitoramento, assim como outros sistemas de gestão de crise e atendimentos.

Figura 2 - Central de monitoramento COR



Fonte: Foto do autor (2021)

A presença de câmeras de vigilância também atua como um forte fator dissuasivo. Muitas pessoas são menos propensas a cometer crimes quando sabem que estão sendo observadas. A possibilidade de serem identificadas e processadas aumenta a percepção de risco entre potenciais ofensores, levando a uma redução nas atividades criminosas. Logo, a instalação de câmeras em ambientes comunitários pode aumentar o sentimento de segurança (Myagmar-Ochir; Kim, 2023).

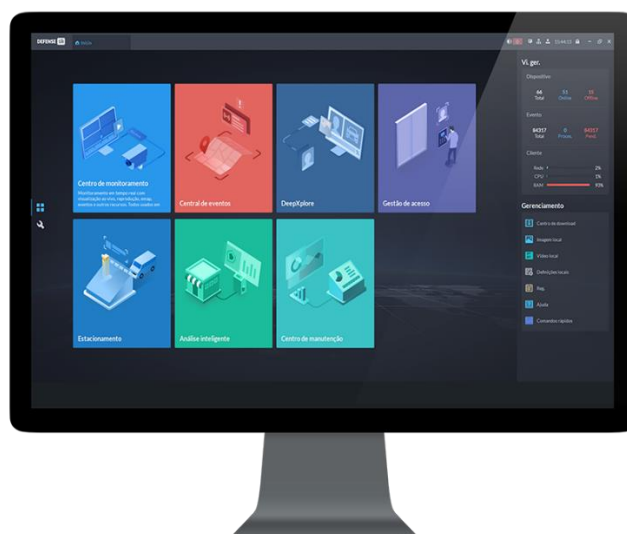
Conforme Vijeikis *et al.* (2021), o estado da arte para soluções empregadas em sistemas de videomonitoramento está fortemente ligado ao uso de tecnologias de inteligência artificial, que permitem a análise e a interpretação de dados em tempo real, detectando objetos, padrões e alvos em imagens e vídeos. Com a análise e interpretação sendo realizadas por algoritmos de inteligência artificial, a necessidade de operadores humanos para monitorar constantemente as câmeras é reduzida. Isso não apenas economiza recursos humanos, mas também permite que os operadores se concentrem em tarefas mais complexas que exigem julgamento humano.

A automação melhora a eficiência do sistema de segurança, pois pode detectar e responder a incidentes mais rapidamente do que um ser humano poderia. Por exemplo, se um comportamento anômalo é identificado, o sistema pode automaticamente alertar as autoridades ou iniciar protocolos de segurança sem a necessidade de um operador humano. Em resumo, essa capacidade de análise e interpretação em tempo real, proporcionada pela IA, transforma o videomonitoramento de um processo passivo, em que os humanos monitoram *feeds* de vídeo, para um sistema ativo que pode identificar e responder a situações de risco de forma autônoma e eficiente. Isso representa um avanço significativo na segurança e na gestão de riscos em ambientes urbanos e outros contextos.

2.1.1 Software Defense IA

O software Defense IA é um sistema desenvolvido pela Intelbras para gerenciamento centralizado de segurança em ambientes públicos e privados. Como demonstrado na Figura 3, o software dispõe de interfaces para monitoramento de vídeo, gestão de acesso, tratamento de alarmes de dispositivos com inteligência artificial, manutenção de dispositivos e análise de dados para gestão de negócios (Intelbras, 2024).

Figura 3 - Defense IA



Fonte: Elaboração própria (2024)

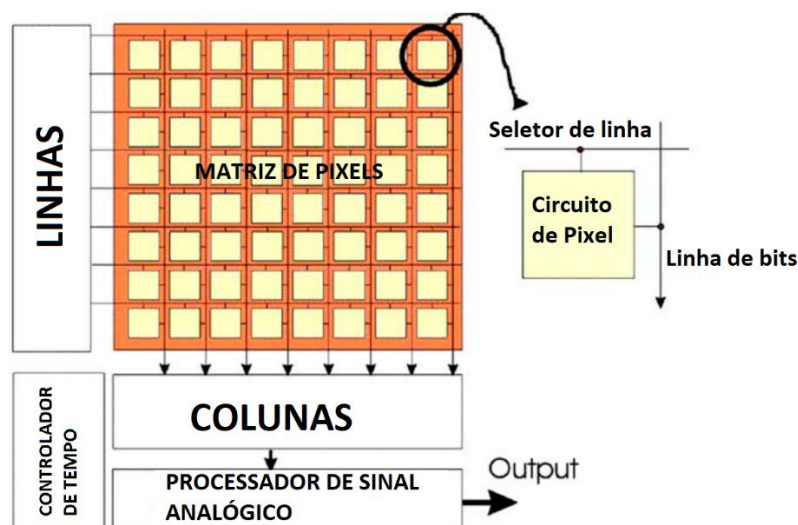
O Defense IA possui uma arquitetura cliente-servidor, sendo um sistema escalável e modular, que possibilita a adição de servidores de acordo com a necessidade de recursos, conexões de dispositivos ou conexões de aplicações clientes para operadores. Ele é compatível com diversos tipos de câmeras, através do protocolo Intelbras e protocolo ONVIF, além de possuir uma API aberta e interface para comandos HTTP (*Hypertext Transfer Protocol*) (Intelbras, 2024).

O Defense IA foi escolhido para este trabalho por ser um sistema de videomonitoramento profissional, além de disponibilizar uma licença de ativação da plataforma por um período de testes.

2.1.2 Câmeras de vigilância

As câmeras são dispositivos que capturam imagens, convertendo a luz incidente em elétrons. A tecnologia mais utilizada nos sensores atuais é a de sensores CMOS (Complementary Metal-Oxide-Semiconductor), que utilizam uma arquitetura de matriz de pixels, como mostra a Figura 4 (Sukhavasi *et al.*, 2021).

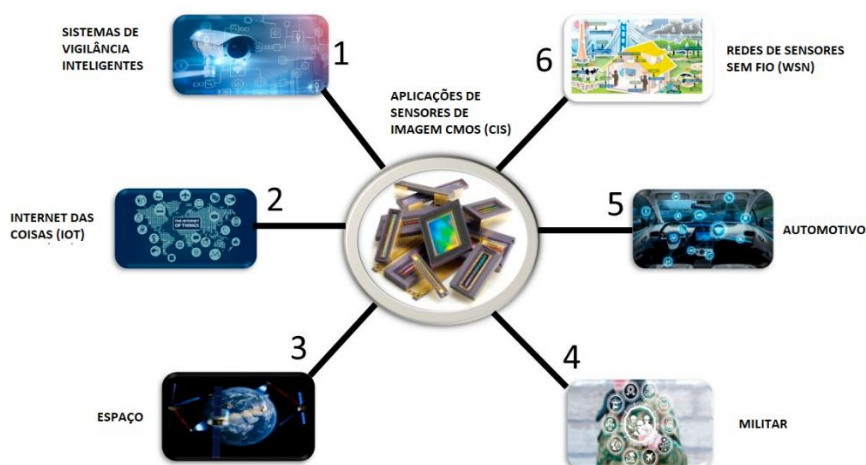
Figura 4 - Matriz sensor CMOS



Fonte: Adaptado de Sukhavasi *et al.* (2021)

Segundo a pesquisa de Sukhavasi *et al.* (2021), a aplicação de câmeras com sensores de imagem CMOS tem ajudado a humanidade em diferentes campos, dentre eles os setores automotivos, internet das coisas (IoT), sistemas de vigilância inteligentes (ISS), espacial, militar e redes de sensores sem fio (WSNs). Essas aplicações têm crescido rapidamente, dia após dia, como demonstrado na Figura 5.

Figura 5 - Campos de aplicação de câmeras



Fonte: Adaptado de Sukhavasi *et al.* (2021)

Dentro do contexto de CFTV, as câmeras são o primeiro e mais importante elemento na cadeia de captura de imagens, sendo essencial para a vigilância e monitoramento em diversas aplicações (Damjanovski, 2014).

Em um contexto de videomonitoramento, podemos encontrar inúmeros tipos de câmeras, com tecnologias de transmissão analógicas, IP e Wi-Fi. A seguir, apresenta-se uma lista de tipos de câmeras que encontramos no catálogo da empresa fabricante de câmeras (Intelbras, 2024):

- a) Câmeras fixas: são câmeras com lente fixa, ou seja, filmam sempre o mesmo ângulo assim que instaladas;
- b) Câmeras *Speed Dome*: são câmeras equipadas com um conjunto de lentes que permitem zoom óptico e motores que movimentam o bloco óptico com função PTZ (*Pan/Tilt/Zoom*). O termo “Pan” se refere ao movimento panorâmico ou horizontal, “Tilt” indica o movimento vertical de inclinação e “Zoom” corresponde à aproximação e ao afastamento da imagem;
- c) Câmeras Panorâmicas: são câmeras projetadas para capturar imagens em um amplo campo de visão, frequentemente de 180 ou 360 graus. Possuem lentes e sensores que trabalham juntos para criar uma imagem panorâmica contínua;
- d) Câmeras Fisheye: são câmeras equipadas com lentes ultra grande angulares que possuem um campo de visão extremamente amplo, geralmente de 180 graus ou até mais. Normalmente, esse tipo de imagem é planificado no software de videomonitoramento;
- e) Câmeras térmicas: são câmeras que possuem sensores de radiação de corpo negro ao invés de sensores CMOS. Elas podem montar imagens a partir da radiação emitida pelo calor dos corpos, mesmo que não haja luz;
- f) Câmeras Veiculares: são câmeras adaptadas para instalação em veículos. Normalmente, são utilizadas em viaturas, veículos públicos e frotas de caminhão;
- g) Bodycams: são câmeras móveis corporais, usadas, principalmente, por policiais, agentes de segurança e outros profissionais em áreas de segurança pública e privada. Essas câmeras são geralmente montadas no uniforme ou em uma parte do corpo do usuário, como o peito ou o ombro, permitindo que capturem áudio e vídeo do ponto de vista da pessoa que a usa.

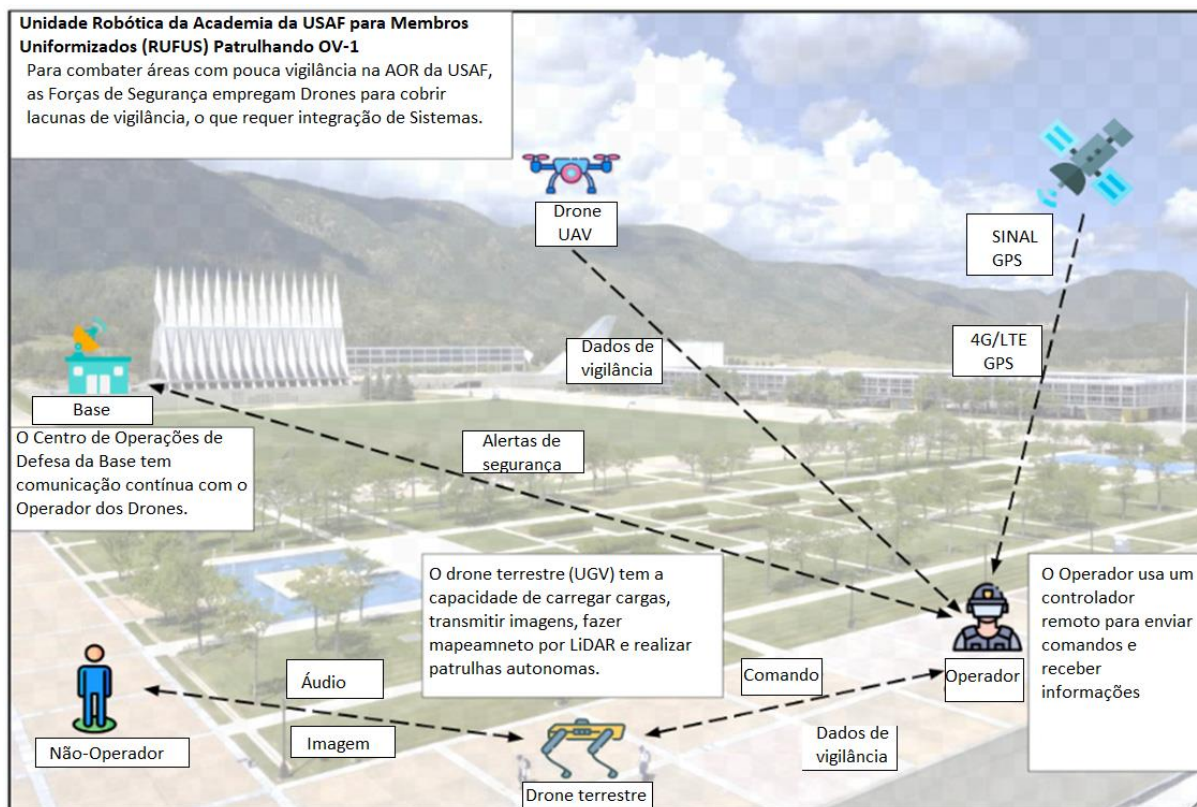
Além do tipo de câmera para cada cenário de aplicação, existem outros parâmetros que também devem ser levados em conta na escolha do dispositivo. Sendo os principais:

- a) Resolução: Resoluções mais altas fornecem imagens mais nítidas, essenciais para capturar detalhes. O mínimo desejado em uma aplicação de segurança é que a resolução seja suficiente para identificação de rostos, placas de veículos.
- b) Taxa de Quadros (FPS – *Frames Per Second*): Influencia a fluidez do vídeo. De acordo com o *datasheet* dos gravadores de vídeo encontrados no portfólio da Intelbras (2024), existe um padrão de mercado quanto à taxa de quadros dos dispositivos de segurança, que vai de 7 FPS a 30 FPS.
- c) Iluminação Infravermelha (IR): Utilizada em cenários que precisam de visão noturna. Câmeras que possuem LED infravermelhos e filtro de lente são capazes de capturar imagem em ambientes de baixa iluminação.
- d) Transmissão em tempo real: O dispositivo deve ser capaz de enviar imagens dentro de restrições de tempo previsíveis e específicas, ou seja, por mais que exista um atraso na imagem, o atraso deve ser invariável, para que exista uma previsibilidade da cena. Caso o atraso entre quadros seja maior que o máximo esperado, ocorrerá uma degradação na taxa de quadros por segundo.

2.2 Robôs para monitoramento

Segundo o trabalho de Avila *et al.* (2024), robôs para monitoramento são dispositivos autônomos ou semiautônomos, equipados com tecnologias avançadas que permitem a coleta de dados em tempo real, contribuindo para a tomada de decisões estratégicas em operações de defesa. Também conhecidos como drones, possuem características de mobilidade em terrenos desafiadores e a capacidade de realizar patrulhas autônomas que aumentam a segurança das operações, reduzindo a exposição de pessoal humano a situações de risco. A Figura 6 mostra um exemplo de operação com drones UAV (Veículo Aéreo Não Tripulado) e UGV (Veículo Terrestre Não Tripulado) em uma base de operações.

Figura 6 - Operação de Drones UAV e UGV



Fonte: Adaptado de Avila *et al.* (2024)

Assim como outros dispositivos que surgiram para o uso militar e depois se popularizaram, o avanço tecnológico tem encontrado espaço crescente para robôs em ambientes civis. O projeto desenvolvido por López *et al.* (2013) é um exemplo disso, o WatchBot é um sistema que realiza tarefas de manutenção predial, vigilância e patrulhamento de edifícios, utilizando robôs autônomos guiados por uma combinação de tecnologias, tais como RFID (*Radio Frequency Identification*), Lidar e sonar. Os robôs da WatchBot utilizam câmeras para monitorar atividades e detectar eventos anômalos.

2.2.1 Robotino da Festo

Este trabalho, utilizou o Robotino, demonstrado na Figura 7, como componente robô da proposta, devida a disponibilidade do equipamento.

“Robotino é uma plataforma robótica móvel para pesquisa e educação. Com motores omnidirecionais, sensores, interfaces e extensões específicas de aplicação, Robotino pode ser usado de forma muito flexível” (Festo Didatic, 2024, tradução nossa).

Desenvolvido pela Festo, o Robotino é um dispositivo versátil para projetos acadêmicos com finalidades de uso em espaços internos.

Figura 7 – Robotino



Fonte: Festo (2024)

O Robotino é capaz de se locomover utilizando três unidades de acionamento omnidirecional, posicionadas com uma angulação de 120° entre si, o que permite ao robô se mover em qualquer direção. Conforme demonstra o Quadro 1, a unidade de controle do Robotino é equipada com um processador PC 104, operando a 300 MHz e possui 128 MB de memória SDRAM, além de um sistema operacional Linux. O dispositivo também conta com um cartão de memória flash de 256 MB, que inclui uma API e um ponto de acesso (access point) para uma rede Wireless LAN, configurada para criar uma rede Wi-Fi chamada “Robotino”, facilitando a conexão e o acesso ao sistema do robô (Festo, 2007).

Além disso, o Robotino da Festo é compatível com câmeras USB, sendo capaz de capturar e fornecer imagens (Festo, 2007).

Quadro 1 - Especificações Robotino

Especificações técnicas	
Processador	PC 104 300 MHz
Sistema Operacional	Linux Robotino 2.6.28.9 (base Ubuntu)
Câmera	Câmera USB 320x240p
Interface USB	2 portas USB
Interface de rede	Roteador Wifi embarcado / 1 Porta Ethernet
Drivers de motor	3 Drivers Omnidirecionais < 120º com encoder
Sensores de distancia	9 Sensores de distancia IR
Sensor anti-colisão	1 Sensor Fita
Portas I/O	8 Portas de input Digital
	8 Portas de input Analógico
	2 Portas de output Digital
	2 Relés de estado sólido
Bateria	2 baterias de chumbo ácido 12V 7Ah

Fonte: Elaboração própria (2024)

2.3 Protocolos de transmissão de vídeo sobre IP

Tratando-se de câmeras de vigilância digitais, que comunicam via protocolos de rede, o termo que se adota para designá-las é IP (Protocolo de Internet). Nesse caso:

Os protocolos de transporte de vídeo baseiam-se em dois protocolos básicos de distribuição de rede: User Datagram Protocol (UDP) e Transmission Control Protocol (TCP). UDP é um serviço de rede simplificado onde os pacotes são entregues o mais rápido possível, mas correm o risco de ser perdidos (dropados). O UDP oferece suporte à entrega um-para-muitos usando multicast. Por outro lado, o TCP fornece controle de fluxo e garante a entrega, mas o custo disso é a entrega limitada. Além disso, o TCP é somente unicast. Uma fonte TCP deve replicar o fluxo para todos os destinatários (Kauffmann, 2023, p. 2).

A Figura 8 demonstra quais são os protocolos de rede mais utilizados para transmissão de vídeo.

Figura 8 - Matriz de comparação de protocolos de vídeo

Alta resiliência à perda de pacotes		RIST SRT	HLS/DASH RTMP/RTSP
Média resiliência à perda de pacotes		RTP + FEC	
Sem resiliência à perda de pacotes	RTP/RTSP UDP		
	Baixa Latência	Média Latência	Alta Latência

Base-TCP
Base-UDP

Fonte: Adaptado de Kauffmann (2023)

A seguir, é apresentada a descrição de Kauffmann (2023) sobre os protocolos:

- UDP Puro: este protocolo transmite o vídeo diretamente no payload dos pacotes UDP, sem qualquer latência de protocolo. No entanto, ele não oferece nenhuma possibilidade de recuperação de perdas de pacotes;
- RTP: o Protocolo de Transporte em Tempo Real (RTP) é uma extensão baseada em UDP. Ele adiciona uma camada sobre o UDP que inclui *timestamps* (carimbo de data e hora) e números de sequência nos pacotes, facilitando a sincronização e a ordenação dos dados;
- RTP + FEC: “RTP com FEC (*Forward Error Correction*) conforme o padrão SMPTE 2022” é uma variante do RTP, também baseada em UDP. Nesse protocolo, o vídeo é transmitido utilizando o RTP padrão, enquanto pacotes adicionais contendo Códigos de Correção de Erros (FEC) são enviados simultaneamente. Em caso de perda de pacotes, o receptor pode reconstruir os dados perdidos usando os pacotes recebidos e os pacotes FEC. O protocolo FEC permite ajustes em termos de *overhead* (*sobrecarga*), latência e capacidade de recuperação. No entanto, o seu uso na Internet pode ser arriscado, pois depende da capacidade adequada dos Provedores de Serviço, baixo nível de congestionamento e outros fatores fora de controle;
- RTSP (UDP): o Protocolo de Streaming em Tempo Real (RTSP) foi inicialmente desenvolvido como um protocolo de controle para servidores de vídeo, oferecendo uma interface de usuário para funções como iniciar, pausar

e parar a reprodução. A interface de controle RTSP é baseada em TCP, enquanto o streaming de dados reais utiliza o RTP, que envia fluxos de vídeo e áudio em portas separadas. Geralmente, não há mecanismos de recuperação de perda de pacotes no nível RTP;

e) RTSP (TCP): o RTSP também possui um modo em que os dados RTP são transmitidos pela conexão de controle TCP, proporcionando a mesma resiliência que o TCP. Esse modo é, frequentemente, utilizado em câmeras de vigilância, devido à sua interface de controle para iniciar, pausar, parar a reprodução e reproduzir de forma acelerada. O RTSP sobre TCP é compatível com alguns decodificadores de software;

f) SRT: o Transporte Confiável e Seguro (*Secure Reliable Transport*, SRT) é um protocolo baseado em UDT (protocolo de transferência de dados que utiliza UDP) projetado para a transferência rápida de arquivos via UDP. Ele confirma os pacotes recebidos corretamente de maneira similar ao TCP e utiliza uma confirmação negativa explícita (*NACK*) para pacotes descartados, solicitando ao remetente a retransmissão dos pacotes perdidos ou não confirmados;

g) RIST: o Transporte de Stream de Internet Confiável (*Reliable Internet Stream Transport*, RIST) é uma especificação publicada pelo *Video Services Forum* (VSF) voltada para a contribuição e distribuição de vídeo de baixa latência. Ele recupera pacotes perdidos utilizando uma variante da retransmissão seletiva, conhecida como ARQ (*Automatic Repeat Request*). A transmissão de mídia é feita usando RTP/UDP padrão e os pacotes recebidos corretamente não são retransmitidos. O receptor solicita a retransmissão de pacotes perdidos por mensagens RTCP padrão. O RIST também foi projetado para ser compatível com *firewalls*;

h) RTMP: o protocolo de Mensageria em Tempo Real (*Real-Time Messaging Protocol*) foi originalmente desenvolvido pela Macromedia e agora mantido pela Adobe, sendo usado, principalmente, para a transmissão de streams ao vivo na internet. Utiliza TCP, tornando-o resiliente à perda de pacotes e a sua limitação é suportar apenas um tipo de mídia. Embora tenha sido amplamente utilizado para transmitir vídeo em Flash, está se tornando obsoleto e sendo substituído por tecnologias como HLS;

i) HLS: o protocolo de Transmissão de Vídeo Ao Vivo via HTTP (*HTTP Live Streaming*) desenvolvido pela Apple, é um protocolo que divide o vídeo em

pequenos segmentos transmitidos por HTTP, permitindo *streaming* adaptativo e maior robustez. Com suporte nativo em dispositivos Apple e Android, o HLS oferece alta escalabilidade e resiliência, mas possui latência relativamente alta, geralmente variando entre 2 e 30 segundos. É amplamente utilizado para transmissão de vídeo ao vivo na internet.

2.3.1 Protocolo RTSP

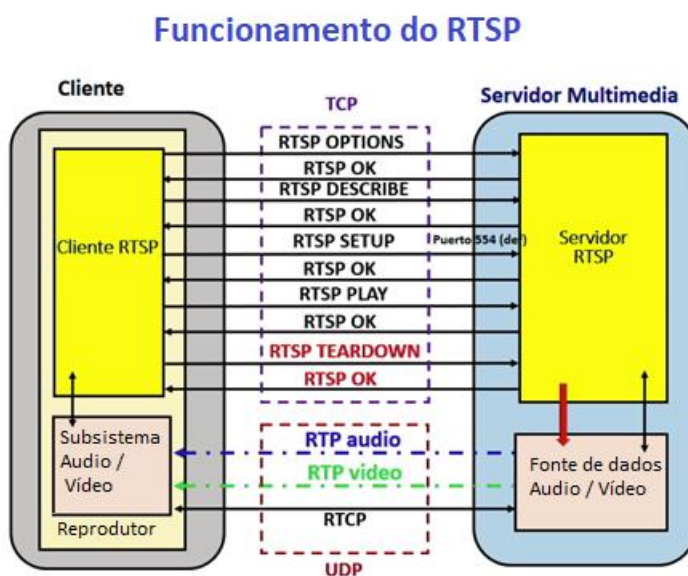
Visto que dentre os protocolos apresentados o protocolo RTSP é o mais empregado em câmeras de vigilância, essa subseção aprofunda-se na explicação deste protocolo.

Definido na RFC 2326 por Schulzrinne *et al.* (1998), o *Real Time Streaming Protocol*, ou RTSP, é um protocolo de nível de aplicação para controle sobre a entrega de dados com propriedades em tempo real. O RTSP fornece uma estrutura extensível para permitir entrega controlada e sob demanda de dados em tempo real, como áudio e vídeo. As fontes de dados podem incluir transmissões de dados ao vivo e clipes armazenados. Esse protocolo tem como objetivo controlar várias sessões de entrega de dados, fornecendo um meio para escolher canais de entrega como UDP, UDP multicast e TCP, bem como um meio para escolher mecanismos de entrega com base em RTP (RFC 1889).

O protocolo possui uma arquitetura cliente-servidor, no qual o cliente envia comandos ao servidor para controlar a transmissão em tempo real (Segui, 2022)

Essa sequência de comandos de controle e transmissão é exemplificada na Figura 9.

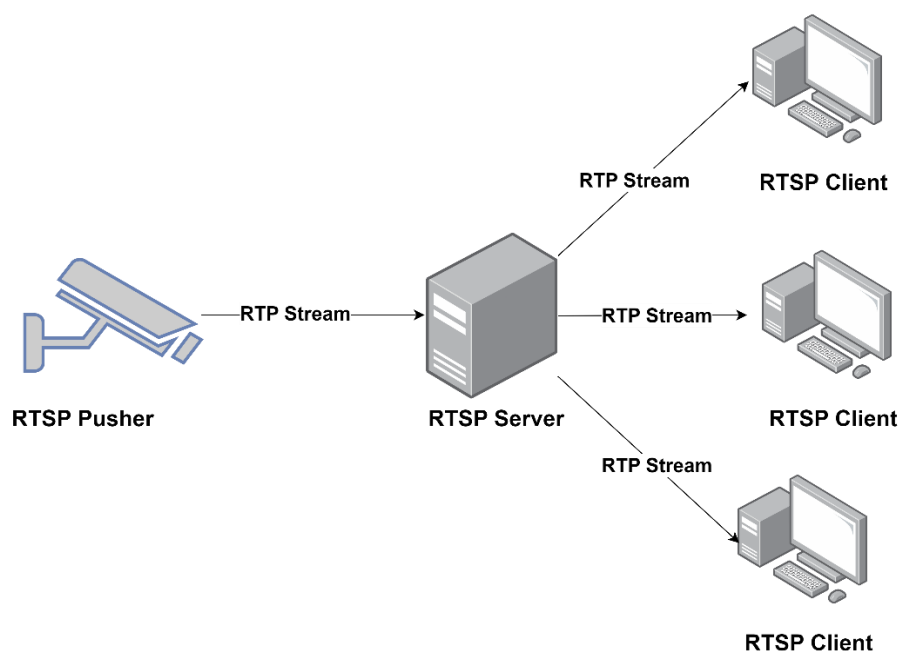
Figura 9 - Protocolo RTSP



Fonte: Adaptado de Segui (2022)

Em muitos casos se encontram a aplicação servidor RTSP embarcada no mesmo dispositivo no qual a imagem é adquirida. No entanto, também é usual encontrar em serviços de transmissão de mídia o componente RTSP *pusher* (impulsor). Quando a mídia não reside no servidor, o RTSP *pusher* envia ao servidor RTSP a mídia que poderá ser retransmitida (Schulzrinne *et al.*, 1998). A topologia está representada na Figura 10.

Figura 10 - Topologia componentes RTSP



Fonte: Elaboração própria (2024)

2.3.2 Protocolo ONVIF

O ONVIF (Open Network Video Interface Forum) é um protocolo de comunicação de padrão industrial significativo na área de vigilância por vídeo. É um protocolo que estabelece padrões de configuração, gerenciamento, transmissão de vídeo e outros recursos entre dispositivos. Compatível com mais de 8.000 câmeras IP (também conhecidas como ONVIF NVT) e gravadores de vídeo em rede (ONVIF NVS) de diferentes fornecedores listados no site oficial do ONVIF (Lin *et al.*, 2018): “Fundada em 2008, a ONVIF tem como missão prover e promover interfaces padronizadas para interoperabilidade efetiva de produtos e serviços de segurança física baseados em IP” (ONVIF, 2024, tradução nossa).

Os critérios de aceitação para homologação de produtos no ONVIF estão descritos de acordo com os perfis em uma tabela oficial do site. Os perfis variam de acordo com as funções homologadas. Os perfis mais importantes para vídeo são: Perfil S para streaming de vídeo básico; Perfil T para streaming de vídeo avançado e Perfil G para armazenamento e recuperação de borda (ONVIF, 2022).

2.3.3 Framework FFmpeg

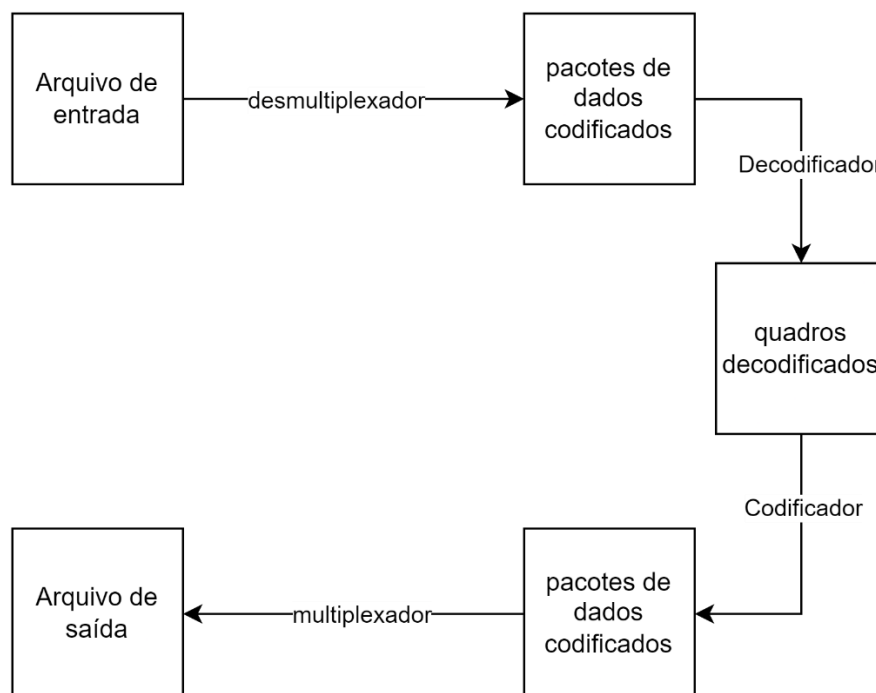
O FFmpeg é um importante *framework open-source* para multimídia, capaz de decodificar, codificar, transcodificar, multiplexar, demultiplexar, transmitir, filtrar e reproduzir, praticamente, qualquer formato de mídia de áudio e vídeo utilizada hoje em dia. Possui uma estrutura portátil, compatível com Linux, Mac OS X, Microsoft Windows, BSD, Solaris, entre outras (FFMPEG, 2024a).

O FFmpeg é um software que processa um número arbitrário de arquivos de entrada, os quais podem ser de diferentes tipos, incluindo arquivos regulares, pipes, fluxos de rede e dispositivos de captura. Esses arquivos são especificados utilizando a opção `-i`. O FFmpeg também grava em um número arbitrário de arquivos de saída, identificados por uma URL de saída. Qualquer argumento na linha de comando que não possa ser interpretado como uma opção é tratado como uma URL de saída. Cada URL de entrada ou saída pode, em teoria, conter múltiplos fluxos de diversos tipos, tais como vídeo, áudio, legendas, anexos e dados. O número e os tipos permitidos de fluxos podem ser restringidos pelo formato do contêiner utilizado. A seleção dos fluxos provenientes das entradas e a sua correspondência com as saídas é realizada automaticamente pelo FFmpeg ou pode ser controlada pelo usuário pela opção `-map`. Para referenciar arquivos de entrada nas opções, deve-se utilizar os seus índices, baseados em zero. Por exemplo, o primeiro arquivo de entrada é indicado como 0, o segundo como 1 e assim por diante. Da mesma forma, os fluxos dentro de um arquivo são referenciados pelos seus índices. Por exemplo, 2:3 se refere ao quarto fluxo no terceiro arquivo de entrada.

Em geral, as opções são aplicadas ao próximo arquivo especificado na linha de comando, tornando a ordem dos argumentos relevante. É possível repetir a mesma opção múltiplas vezes e cada instância será aplicada ao próximo arquivo de entrada ou saída. No entanto, existem exceções para esta regra, sendo as opções globais aquelas que não seguem essa norma de aplicação (FFMPEG, 2024b).

A Figura 11 demonstra o fluxo de transcodificação com FFmpeg.

Figura 11 - Diagrama do processo de transcodificação com FFmpeg



Fonte: Adaptado de FFmpeg (2024b)

O Quadro 2 também traz um exemplo de uso do FFmpeg, neste caso combinando uma fonte de entrada de vídeo e uma fonte de entrada de imagem, codificando-os com os *codecs* apropriados e transmitindo para uma URL RTSP usando o protocolo UDP.

Quadro 2 - Exemplo FFmpeg em linha de comando

Exemplo de uso fo ffmpeg em linha de comando:

```
ffmpeg -i "input_video.mp4" -i "input_audio.mp3" -map 0:v:0 -map 1:a:0 -c:v libx264 -c:a aac -f rtsp -rtsp_transport udp rtsp://127.0.0.1:8554/mystream
```

Explicação:

1. **-i "input_video.mp4"**: Vídeo de entrada.
2. **-i "input_audio.mp3"**: Áudio de entrada.
3. **-map 0:v:0**: Mapeia o fluxo de vídeo do arquivo de vídeo de entrada.
4. **-map 1:a:0**: Mapeia o fluxo de áudio do arquivo de áudio de entrada.
5. **-c:v libx264**: Codifica o vídeo com **H.264**.
6. **-c:a aac**: Codifica o áudio com **AAC**.
7. **-f rtsp**: Define o formato de saída como **RTSP**.
8. **-rtsp_transport udp**: Define o transporte **UDP** para a transmissão RTSP.
9. **rtsp://127.0.0.1:8554/mystream**: URL RTSP para o fluxo de saída.

Fonte: Elaboração própria (2024)

2.4 Software Middleware

O *Middleware* é um software intermediário, sendo uma camada de software que facilita a comunicação entre sistemas heterogêneos, permitindo a integração eficiente de diferentes programas, linguagens de programação e hardware. A sua principal função é harmonizar a comunicação entre sistemas distribuídos, criando uma ponte independentemente das tecnologias usadas, simplificando a integração de aplicações diversas e promovendo interoperabilidade (Amazon, 2024a).

Segundo a Amazon (2024a), o *middleware* abstrai o processo de comunicação subjacente entre componentes. Logo, as aplicações das pontas se comunicam apenas com o *middleware* e não precisa aprender a linguagem de outros componentes. Alguns dos *frameworks* (estrutura de software) comuns são serviços de mensageria e APIs.

2.4.1 Interface de Programação de Aplicativos (API)

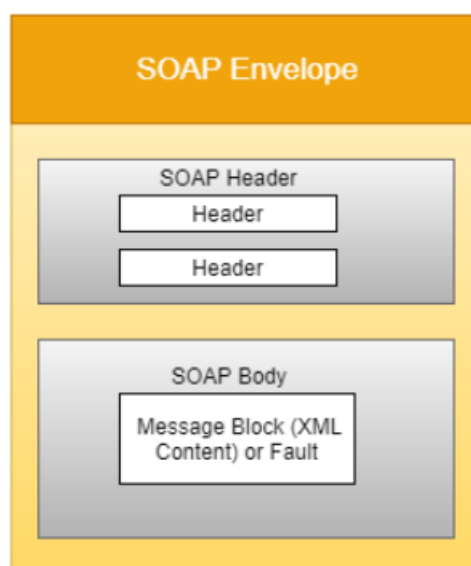
Uma Interface de Programação de Aplicativos (API) é uma interface que permite a comunicação entre dois programas de software. É um conjunto de regras e convenções que os desenvolvedores devem seguir para que as aplicações possam se comunicar de forma eficaz. A interface pode ser interpretada como um contrato de serviço entre duas aplicações, com a definição de como elas devem trocar informações usando solicitações e respostas (Amazon, 2024b).

Segundo Amazon (2024b), a arquitetura de uma API é explicada, geralmente, em termos de cliente e servidor. Existem quatro maneiras diferentes pelas quais as APIs podem funcionar:

- a) APIs SOAP: a composição do Protocolo de Acesso a Objetos Simples (SOAP) é um protocolo de mensagens que usa o formato de mensagens Linguagem de Marcação Extensível (XML) para fornecer comunicação pela rede. O SOAP é um componente essencial da Arquitetura Orientada a Serviços (SOA) e dos requisitos de serviços *web* relacionados à SOA. A Figura 12 representa o formato de uma mensagem SOAP, sendo: o Envelope a estrutura

que determina a formatação, caracterizando o começo e fim de arquivo; o *Header* (cabeçalho), no qual são inseridos elementos opcionais tais como autorização; o *Body* (corpo), que é onde de fato estão os dados de solicitação e resposta, e o *Fault* (Falha), que contém o código de erro respondido para quando a mensagem não é enviada corretamente. Os métodos de uma API SOAP são definidos no seu WSDL (Web Services Description Language) (Soni; Ranga, 2019);

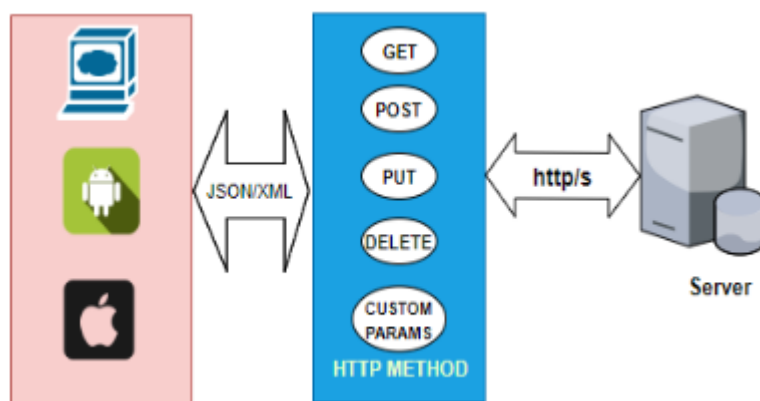
Figura 12 - Formato de mensagem SOAP



Fonte: Soni e Ranga (2019, p. 667)

b) APIs REST: a API de Transferência de Estado Representacional (REST) é uma convenção para serviços de rede *web* com uma arquitetura cliente-servidor, com transmissão *Stateless* (Sem Estado), ou seja, sem memória de transações anteriores pois nenhuma requisição é salva. Pode utilizar os formatos JSON (Notação de Objeto JavaScript) e XML, na qual o cliente envia solicitações à URL (*Uniform Resource Locator* do servidor), também conhecida como *Endpoint* (ponto de acesso específico em uma interface de comunicação), contendo dados, e o servidor utiliza essas entradas para executar funções internas, retornando posteriormente os dados de saída ao cliente. Como demonstra a Figura 13, utiliza os métodos definidos: GET para obter dados; Post para criar dados; PUT para atualizar dados e DELETE para deletar dados;

Figura 13 - Métodos da API REST



Fonte: Soni e Ranga (2019)

- c) APIs RPC: “essas APIs são conhecidas como Remote Procedure Calls (Chamadas de Procedimento Remoto). O cliente conclui uma função (ou um procedimento) no servidor e o servidor envia a saída de volta ao cliente” (Amazon, 2024b);
- d) APIs WebSocket: a API WebSocket utiliza JSON para transmitir dados. Uma API WebSocket oferece suporte à comunicação bidirecional entre aplicativos do cliente e o servidor. O servidor pode enviar mensagens de retorno de chamada a clientes conectados (Amazon, 2024b).

2.5 Requisitos de projeto

A escolha do sistema Defense IA e do robô Robotino para este projeto de integração foi baseada na disponibilidade dos produtos e capacidade dessas tecnologias em oferecer soluções adequadas, mesmo considerando as limitações de hardware do Robotino.

Constatou-se a necessidade de desenvolvimento de um software *middleware*, capaz de estabelecer conexão com o Robotino utilizando API, converter o sinal de imagens do mesmo em RTSP utilizando o *framework* FFmpeg e consecutivamente em ONVIF. Uma vez que, o Defense IA suporta o protocolo ONVIF para adição de dispositivos, e o RTSP é um protocolo de transmissão de vídeo eficiente para sistemas de videomonitoramento. Devido necessidade de performance em uma aplicação de transmissão de vídeo, este middleware deveria executado no mesmo computador onde o servidor do Defense IA fosse instalado.

Também foram levantados requisitos para imagem e transmissão de vídeo, tais como a necessidade de que a imagem possua resolução o suficiente para identificação de pessoas na cena, taxa de quadros suficiente para que seja uma imagem fluida e transmissão em tempo real para que não ocorra a degradação da taxa de quadros por segundo.

A integração de comando dos motores do Robotino deve partir diretamente do Defense IA, através das chamadas em API.

3 METODOLOGIA

A metodologia adotada neste trabalho seguiu uma abordagem de pesquisa aplicada, bibliográfica e qualitativa, com caráter exploratório. A pesquisa aplicada se caracteriza pela busca de soluções práticas para problemas específicos, como a integração de tecnologias em um contexto real. Segundo Severino (2007), esse tipo de pesquisa visa resultados que possam ser diretamente utilizados em aplicações concretas, como é o caso da utilização de um robô móvel em um sistema de videomonitoramento.

A pesquisa bibliográfica, por sua vez, consistiu na análise de material previamente publicado, como artigos científicos, livros e documentos técnicos, com o objetivo de fundamentar teoricamente o desenvolvimento do método proposto. Gil (2008) afirma que a pesquisa bibliográfica é essencial para garantir uma base sólida de conhecimento sobre o tema abordado, permitindo ao pesquisador explorar diferentes perspectivas e soluções já existentes.

A pesquisa qualitativa foi escolhida por permitir uma análise mais profunda e interpretativa dos dados. De acordo com Minayo (2010), esse tipo de pesquisa é útil quando se busca compreender fenômenos complexos e subjetivos, como a interação entre o operador e o sistema robótico. A abordagem qualitativa possibilita explorar não apenas os aspectos técnicos da integração, mas também as percepções e experiências dos usuários envolvidos.

Por fim, a natureza exploratória do estudo reflete o objetivo de investigar um campo ainda pouco explorado e ampliar o conhecimento sobre a integração de robôs móveis com sistemas de videomonitoramento. Conforme Gil (2008), a pesquisa exploratória é indicada quando o tema abordado carece de estudos prévios aprofundados, permitindo ao pesquisador formular hipóteses e delinear novas direções para futuras investigações.

4 DESENVOLVIMENTO

Neste capítulo serão discutidas as escolhas do robô, sistema de videomonitoramento e demais componentes necessários para implementação de um sistema de vigilância remoto. Além dos testes de validação de cada etapa, é apresentado o desenvolvimento da aplicação e os resultados.

4.1 Seleção do Robô e Software para o sistema de vigilância remota

Durante a pesquisa por robôs que pudessem atender às necessidades de protótipo, constatou-se que o Robotino da Festo, sendo um robô móvel de caráter didático, com uma câmera embarcada, sistema de locomoção e API para comunicação, seria adequado para os objetivos propostos. É importante destacar que este trabalho dá continuidade a um tema abordado em um Trabalho de Conclusão de Curso anterior, no qual o Robotino foi o recurso escolhido, sendo disponibilizado pelo Departamento de Mecatrônica do IFSC Florianópolis.

De maneira similar, o software de videomonitoramento Defense IA, da Intelbras, atende aos requisitos necessários para vigilância e possui capacidade de integração para envio de comandos via API. O uso deste software foi viabilizado por uma parceria com a Intelbras, que disponibilizou licenças de demonstração para o autor deste projeto.

4.2 Comunicação com o Robotino

De acordo com o trabalho de Saraiva(2023), quando ligado, o Robotino inicializa o sistema operacional e habilita uma rede *Wi-Fi* com SSID “Robotino 4.0”, permitindo acesso à sua rede. Ao conectar um computador a essa rede, é possível acessar remotamente o Robotino, por SSH (*Secure Shell*), um protocolo de rede com autenticação e criptografia forte que permite a comunicação segura entre dois dispositivos, e Telnet, que também é um protocolo de comunicação utilizado entre dispositivos, porém mais simples, baseado em texto.

No início das atividades práticas sobre o funcionamento do Robotino, foi realizada uma conexão via SSH para mapear as pastas e arquivos presentes no sistema. Durante o processo, foram encontrados arquivos WSDL que contêm os

métodos da API SOAP, sendo eles “Camera.wsdl”, “DeviceManager.wsdl” e “IOControl.wsdl”. Além disso, também foram identificados arquivos da API REST e o programa responsável por inicializar o servidor web. Esse último foi executado no sistema Linux pelo comando “webserverd startservice()”, conforme demonstrado no Quadro 3.

Quadro 3 - Start WEB Server

```

1 root@robotino: /usr/local/OpenRobotinoAPI/2/server/bin
2
3 servidor web      servidorweb.sh
4 root@robotino:/usr/local/OpenRobotinoAPI/2/server/bin# cat webserverd.sh
5 #!/bin/sh
6
7 SERVICEPATH=/usr/local/OpenRobotinoAPI/2/servidor/bin
8 SERVIÇO=servidorweb
9 PARAM=
10
11 startservice ()
12 {
13 start-stop-daemon--background --start--quiet --chdir $SERVICEPATH --exec
$SERVICEPATH/$SERVICE $PARAM
14     if [ $? -eq 0 ]; then
15         echo $SERVICE started
16     fi
17 }
18
19 stopservice ()
20 {
21     start-stop-daemon --stop --retry 10 --signal INT --name $SERVICE
22     if [ $? -eq 0 ]; then
23         echo $SERVICE stopped
24     fi
25 }
26
27 case "$1" in
28     start)
29         startservice
30         ;;
31
32     stop)
33         stopservice
34         ;;
35
36     restart)
37         stopservice
38         startservice
39         ;;
40     *)

```

```

41      echo "Usage: $0 {start|stop|restart}"
42      exit 1
43      ;;
44 esac
45
46 root@robotino:/usr/local/OpenRobotinoAPI/2/server/bin# webserverd

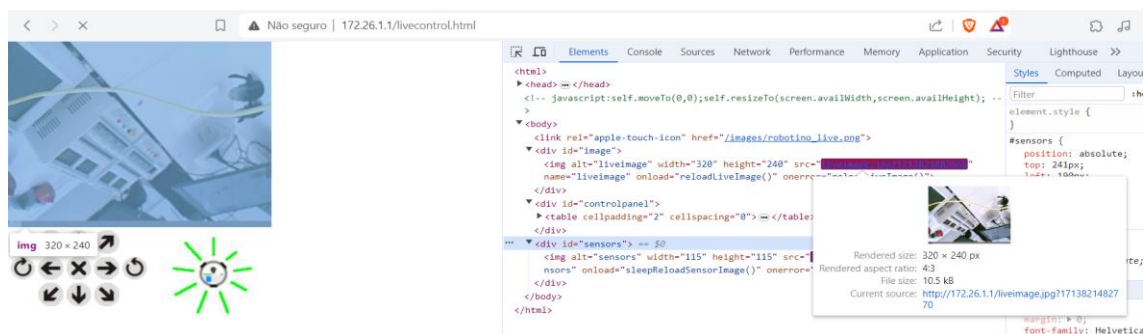
```

Fonte: Elaboração própria (2024)

4.3 Estudo API REST do Robotino

Quando iniciado o serviço web da API REST do Robotino, além de disponibilizar rotas HTTP, é habilitada a página web demonstrada na Figura 14, no qual é possível visualizar a câmera do Robotino, assim como controlar os motores e visualizar o status dos sensores de distância e colisão.

Figura 14 - Página Web Robotino



Fonte: Elaboração própria (2024)

Inspecionando os elementos desta página, foi possível mapear as rotas HTTP, descritas no Quadro 4, sendo o parâmetro "<ip>" o endereço IP (*Internet Protocol*) do Robotino.

Quadro 4 - Requisições da API HTTP REST do Robotino

Comando	Requisição
Obter imagem da câmera	GET: <code>http://<ip>/sensors.jpg?(Unix_Timestamp)</code>
Cessar movimento dos motores	GET: <code>http://<ip>/controlpanel.ai?op=x</code>
Ativar motores no sentido norte	GET: <code>http://<ip>/controlpanel.ai?op=n</code>
Ativar motores no sentido sul	GET: <code>http://<ip>/controlpanel.ai?op=s</code>
Ativar motores no sentido oeste	GET: <code>http://<ip>/controlpanel.ai?op=w</code>

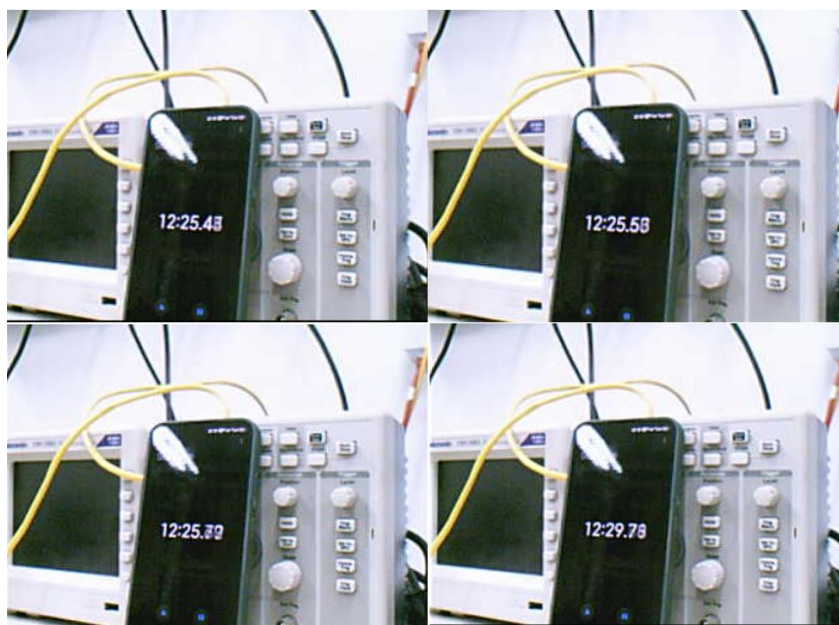
Ativar motores no sentido leste	GET: http://<ip>/controlpanel.ai?op=e
Ativar motores no sentido noroeste	GET: http://<ip>/controlpanel.ai?op=nw
Ativar motores no sentido nordeste	GET: http://<ip>/controlpanel.ai?op=ne
Ativar motores no sentido sudoeste	GET: http://<ip>/controlpanel.ai?op=sw
Ativar motores no sentido sudeste	GET: http://<ip>/controlpanel.ai?op=se
Ativar motores no sentido horário	GET: http://<ip>/controlpanel.ai?op=cl
Ativar motores no sentido anti-horário	GET: http://<ip>/controlpanel.ai?op=ccl

Fonte: Elaboração própria (2024)

Para a validação da API com as rotas mapeadas, utilizou-se o Postman (2024), uma ferramenta para desenvolvimento e testes de API, muito utilizada para APIs REST. Sempre que era executado um comando referente à ativação dos motores, o Robotino se movia na direção desejada por 10 segundos, com velocidade constante, sendo a unidade de medida de velocidade $v = 1$. Quando mais de um comando era enviado em sequência, os incrementos de velocidade se somavam durante as suas respectivas janelas de intervalo, tornando o movimento mais rápido ou cessando nos casos em que o incremento era no sentido oposto.

Para a captura de imagens, foi desenvolvido o código em Python do APÊNDICE A, e executado em um computador. Após a validação da captura, foi criado o código em Python localizado no APÊNDICE B para capturar e salvar as imagens em loop. O objetivo deste código foi obter um fluxo de capturas com uma taxa de frames constante. Para isso, diversas opções de atraso foram testadas no loop do código, entre elas: nenhum atraso, atraso de 1/15 segundos, atraso de 1 segundo e atraso de 5 segundos. Porém, em todos os casos, o atraso entre imagens capturadas foi inconstante. Como demonstra a Figura 15, o inconstante atraso entre pacotes torna a taxa de transmissão de frames imprevisível.

Figura 15 - Atraso entre capturas de imagem



Fonte: Elaboração própria (2024)

Levando em consideração a necessidade de controle em tempo real do robô, essa abordagem sem o devido sincronismo de transmissão não seria o ideal.

4.4 Estudo API SOAP do Robotino

Os arquivos WSDL “Camera.wsdl”, “DeviceManager.wsdl” e “IOControl.wsdl” foram abertos com o software SoapUI (2024), uma ferramenta voltada para testes e desenvolvimento de APIs SOAP.

Dentre os métodos presentes no arquivo “DeviceManager.wsdl”, utilizou-se o comando “Request_List” para mapear os dispositivos e seus respectivos IDs. O ID 1 corresponde à câmera e o ID 2 ao controlador de IO do Robotino. Para iniciar a conexão com qualquer um dos dispositivos e enviar comandos, é necessário solicitar um *token* (código chave de acesso). O *token* é dinâmico, incremental, possui tempo de duração limitado e pode ser revalidado por meio de um comando *keepAlive* (comando para revalidar conexão entre sistemas e mantê-la ativa). No envio de comandos o *token* deve ser inserido como parâmetro.

O código do APÊNDICE C demonstra como realizar a conexão e comando *keepAlive* do *token*.

A API do Robotino também apresenta um método para transmissão de imagens via protocolo UDP, chamado “startStreamingToAddress”. Quando requisitado, este comando configura o controlador do Robotino para dar início ao envio constante de pacotes contendo imagens da câmera USB para o IP e porta configurados como parâmetros no corpo da requisição. Desta forma, o Robotino foi configurado para transmitir para um computador, onde foi possível capturar os pacotes utilizando o software Wireshark (2024), como demonstra a Figura 16. O objetivo da captura foi verificar o conteúdo dos pacotes e a taxa de envio entre os pacotes. Constatou-se que o Robotino é capaz de enviar os pacotes em uma taxa de 15 quadros por segundo.

Figura 16 - Captura de pacotes de imagens do Robotino

No.	Time	Source	Destination	Protocol	Length	Info
7	0.034096	172.26.1.1	172.26.1.100	UDP	1192	37026 → 37777 Len=7070
8	0.062690	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=713d) [Reassembled in #12]
9	0.062690	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=713d) [Reassembled in #12]
10	0.064431	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=713d) [Reassembled in #12]
11	0.066169	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=713d) [Reassembled in #12]
12	0.066169	172.26.1.1	172.26.1.100	UDP	1188	37026 → 37777 Len=7066
13	0.096227	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=713e) [Reassembled in #17]
14	0.096227	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=713e) [Reassembled in #17]
15	0.098882	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=713e) [Reassembled in #17]
16	0.098882	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=713e) [Reassembled in #17]
17	0.099716	172.26.1.1	172.26.1.100	UDP	1180	37026 → 37777 Len=7058
18	0.127009	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=713f) [Reassembled in #22]
19	0.127009	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=713f) [Reassembled in #22]
20	0.127009	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=713f) [Reassembled in #22]
21	0.128988	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=713f) [Reassembled in #22]
22	0.128988	172.26.1.1	172.26.1.100	UDP	1226	37026 → 37777 Len=7104
23	0.158956	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=0, ID=7140) [Reassembled in #27]
24	0.158956	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=1480, ID=7140) [Reassembled in #27]
25	0.158956	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=2960, ID=7140) [Reassembled in #27]
26	0.161328	172.26.1.1	172.26.1.100	IPv4	1514	Fragmented IP protocol (proto=UDP 17, off=4440, ID=7140) [Reassembled in #27]

Destination Address: 172.26.1.100

[5 IPv4 Fragments (7074 bytes): #8(1480), #9(1480), #10(1480), #11(1480), #12(1154)]

[Frame 8, payload: 0-1479 (1480 bytes)]

[Frame 9, payload: 1480-2959 (1480 bytes)]

[Frame 10, payload: 2960-4439 (1480 bytes)]

[Frame 11, payload: 4440-5919 (1480 bytes)]

[Frame 12, payload: 5920-7073 (1154 bytes)]

[Fragment count: 5]

[Reassembled IPv4 length: 7074]

[Reassembled IPv4 data [truncated]: 90a293911ba2a68d000000010000008b00001b82000000000001b82ffdfdb008400100b0c0e0c0a100e0d0e121101318281a181616183123251d283a333d3c3933383740485c4e404457453738506d515]

User Datagram Protocol, Src Port: 37026, Dst Port: 37777

Source Port: 37026

Destination Port: 37777

Length: 7074

Checksum: 0xa68d [unverified]

[Checksum Status: Unverified]

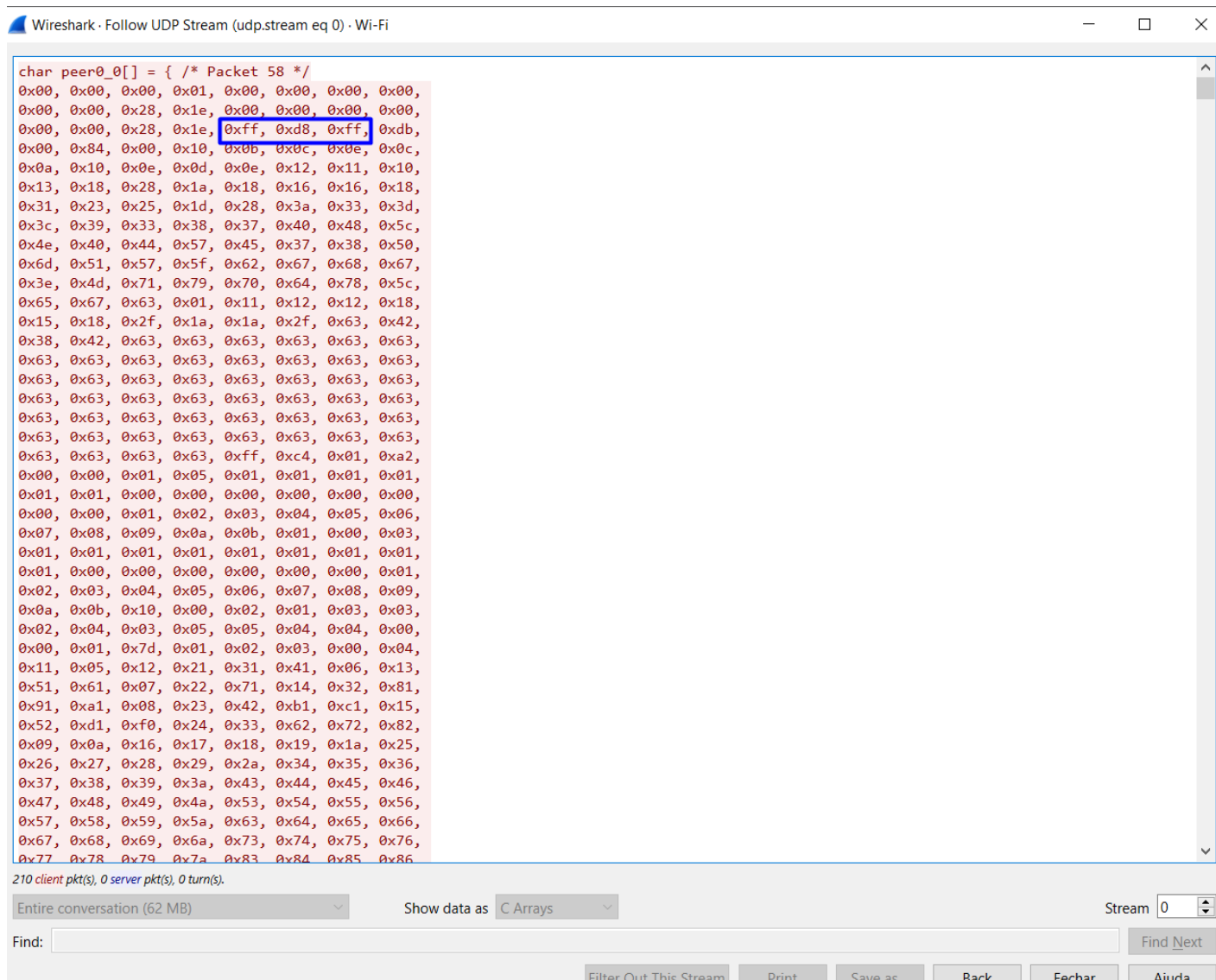
Fonte: Elaboração própria (2024).

A transmissão de pacotes UDP capturados ocorria a uma taxa constante de 15 frames por segundo.

Na sequência foi desenvolvido um código em Python para receber esses pacotes, por meio de um `socket` UDP, e salvar as imagens. No entanto, uma dificuldade enfrentada foi a falta de documentação sobre o formato do pacote. Sem o conhecimento da composição do cabeçalho do pacote, ao salvar as imagens sem a

devida separação dos campos encapsulados, os arquivos resultantes estavam corrompidos. Programas de visualização e edição de imagem não conseguiam abrir o arquivo.

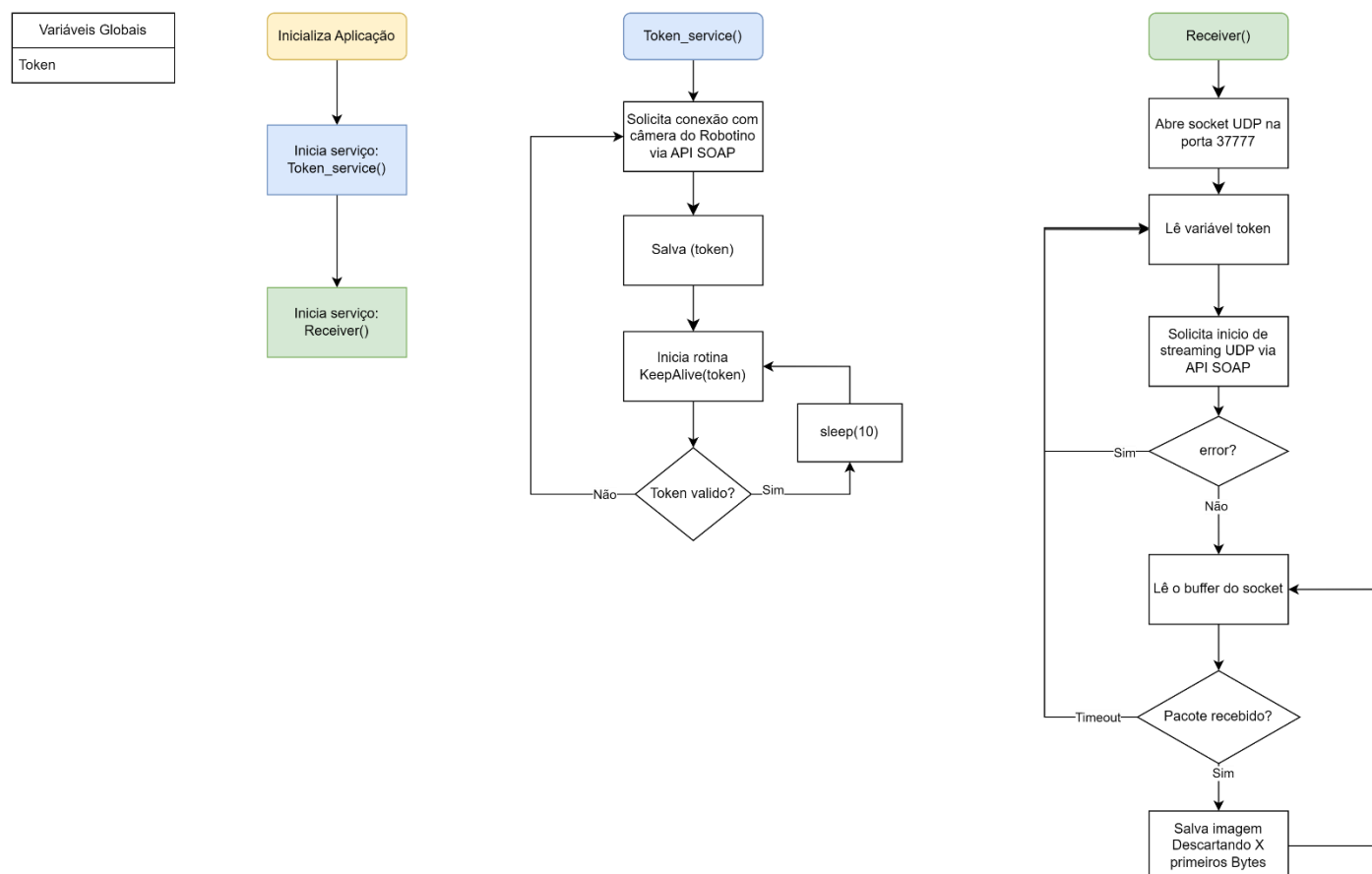
Figura 17 - Cabeçalho .jpg na captura dos pacotes



Fonte: Elaboração própria (2024).

Embora as informações do cabeçalho não tenham sido descobertas, observou-se, na Figura 17, que após os campos iniciais sempre havia o identificador de início de arquivo do formato “.jpg”. Então foram realizados testes utilizando a plataforma de edição hexadecimal HexEd.it (2024), editando os *bytes* do cabeçalho, e deletando-os. Ao descartar os primeiros bytes foi possível recuperar a imagem, como mostra a Figura 18.

Figura 19 - Fluxograma aquisição de imagens



Fonte: Elaboração própria (2024).

4.5 Transmissão de vídeo RTSP

Superada a etapa de recebimento das imagens, o passo seguinte no desenvolvimento foi buscar uma forma de transformá-las em vídeo e transmiti-las. Para isso, foi desenvolvido o código do APÊNDICE E, utilizando a biblioteca OpenCV para converter as imagens em vídeos e a biblioteca FFmpeg para transmitir o vídeo utilizando o protocolo RTSP UDP.

O primeiro problema encontrado nessa etapa foi a incompatibilidade da função *write* da biblioteca OpenCV com imagens no formato .jpg. Quando as imagens foram convertidas para o formato .png a biblioteca apresentou um melhor desempenho.

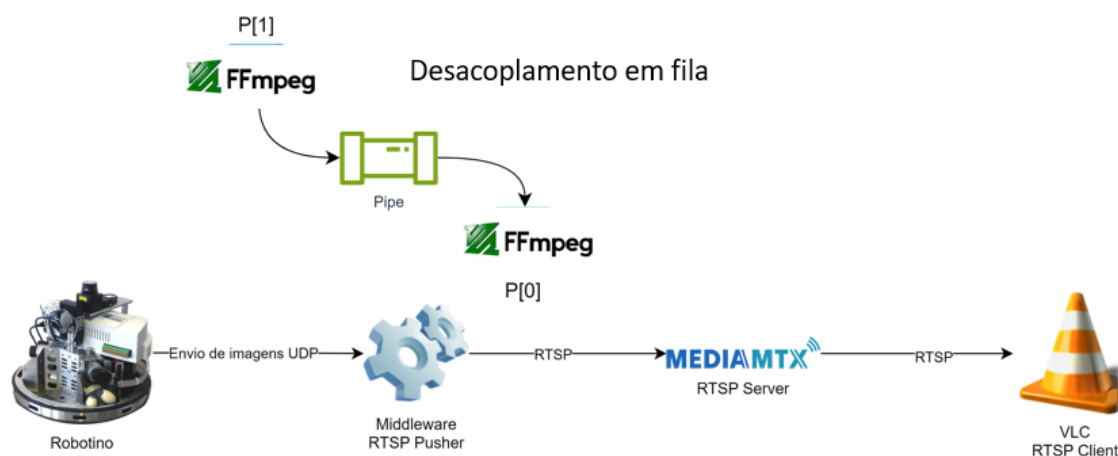
O segundo, e mais significativo, problema desta etapa foi que o cliente RTSP, no caso o software VLC media player (2024), não reproduziu a transmissão

utilizando a URL diretamente. Algo que não está claramente explicado em documentações oficiais do protocolo RTSP é a separação dos papéis de *pusher* RTSP e servidor RTSP. As funções implementadas na biblioteca FFmpeg para transmissão de vídeo via RTSP transformam a aplicação em um *pusher* RTSP, sendo ainda necessário um servidor RTSP entre a origem e o destino do vídeo. Neste caso, utilizou-se o servidor RTSP MediaMTX (2024), sendo este um aplicativo que pode ser baixado e executado localmente.

Até este ponto havia sido possível compilar arquivos de vídeo a partir de arquivos de imagens e, em sequência, transmiti-los via RTSP. No entanto, a aplicação necessitava de transmissão em tempo real. Para isso foi implementada a estrutura do FFmpeg chamada *pipe*. O código desta implementação está demonstrado no APÊNDICE F.

A Figura 20 mostra como ficou a topologia do sistema implementado para transmissão de vídeo RTSP.

Figura 20 - Topologia da transmissão de vídeo



Fonte: Elaboração própria (2024).

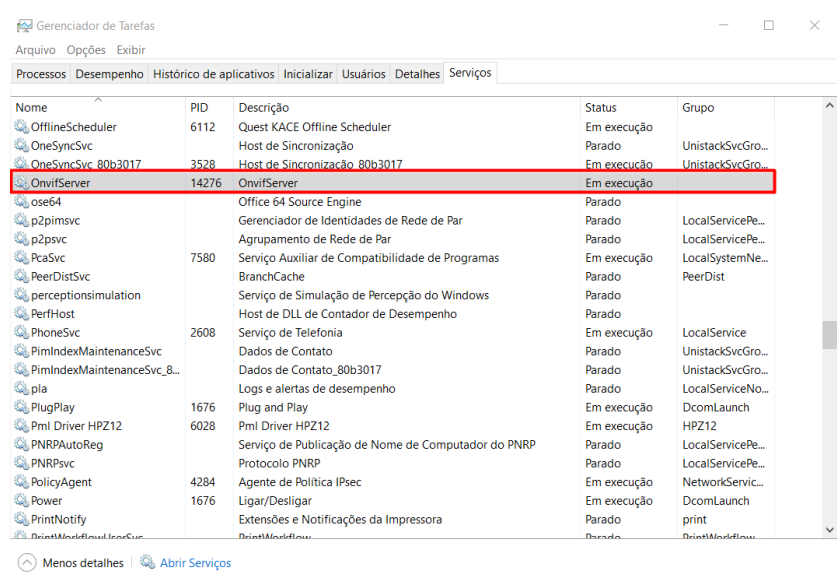
4.6 Conversão de RTSP para ONVIF

O servidor do Defense IA não reconhece um stream RTSP como entrada, apenas dispositivos IP. Por isso foi necessário simular uma câmera, utilizando o

software Happytime Soft (2024), que converte o sinal RTSP em um dispositivo ONVIF virtual com canais de vídeo.

O programa foi instalado localmente como um serviço do sistema, vide Figura 21, configurado para receber o stream RTSP do servidor RTSP MediaMTX, e virtualizar uma câmera ONVIF com o IP de *localhost*.

Figura 21 - Serviço servidor ONVIF



The screenshot shows the Windows Task Manager 'Serviços' (Services) tab. A table lists various system services. The 'OnvifServer' service is highlighted with a red box, showing it is running ('Em execução') with PID 14276. Other services like 'OneSyncSvc' and 'PcaSvc' are also running, while many others are stopped ('Parado').

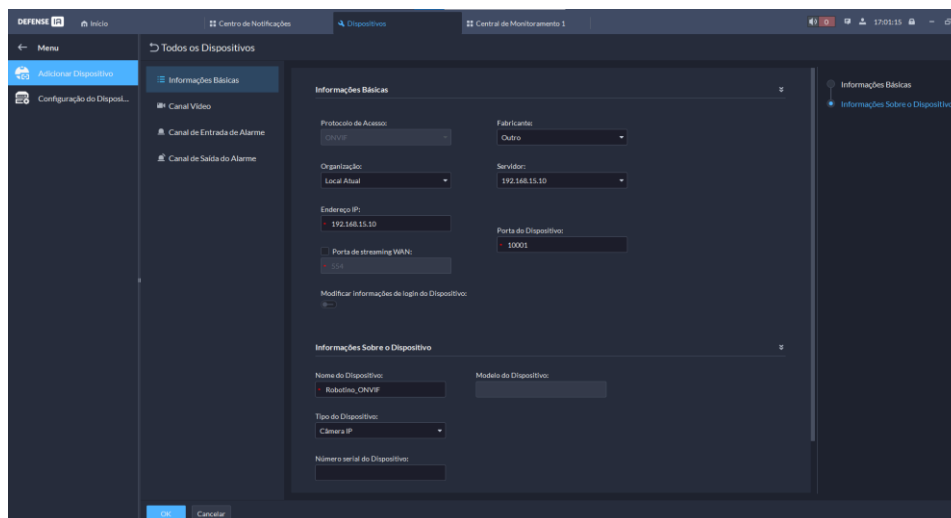
Nome	PID	Descrição	Status	Grupo
OfflineScheduler	6112	Quest KACE Offline Scheduler	Em execução	
OneSyncSvc		Host de Sincronização	Parado	UnistackSvcGro...
OneSyncSvc_80b3017	3528	Host de Sincronização_80b3017	Em execução	UnistackSvcGro...
OnvifServer	14276	OnvifServer	Em execução	
ose64		Office 64 Source Engine	Parado	
p2pimsvc		Gerenciador de Identidades de Rede de Par	Parado	LocalServicePe...
p2psvc		Agrupamento de Rede de Par	Parado	LocalServicePe...
PcaSvc	7580	Serviço Auxiliar de Compatibilidade de Programas	Em execução	LocalSystemNe...
PeerDistSvc		BranchCache	Parado	PeerDist
perceptionsimulation		Serviço de Simulação de Percepção do Windows	Parado	
PerfHost		Host de DLL de Contador de Desempenho	Parado	
PhoneSvc		Serviço de Telefonia	Em execução	LocalService
PimIndexMaintenanceSvc	2608	Dados de Contato	Parado	UnistackSvcGro...
PimIndexMaintenanceSvc_8...		Dados de Contato_80b3017	Parado	UnistackSvcGro...
pla		Logs e alertas de desempenho	Parado	LocalServiceNo...
PlugPlay	1676	Plug and Play	Em execução	DcomLaunch
Pml Driver HPZ12	6028	Pml Driver HPZ12	Em execução	HPZ12
PNRPAutoReg		Serviço de Publicação de Nome de Computador do PNRP	Parado	LocalServicePe...
PNRPsvc		Protocolo PNRP	Parado	LocalServicePe...
PolicyAgent	4284	Agente de Política IPsec	Em execução	NetworkServic...
Power	1676	Ligar/Desligar	Em execução	DcomLaunch
PrintNotify		Extensões e Notificações da Impressora	Parado	print
PrintModiflow		PrintModiflow	Parado	PrintModiflow

Fonte: Elaboração própria (2024).

4.7 Configuração Defense IA e adição de câmera ONVIF

Para configuração da aplicação, o servidor e o cliente do Defense IA foram instalados na mesma máquina. O sistema foi ativado com uma licença de demonstração e a câmera ONVIF virtual foi adicionada conforme a Figura 22.

Figura 22 - Adição de câmera no Defense IA



Fonte: Elaboração própria (2024).

Desta forma a conexão dos elementos do sistema pode ser vista na Figura 23.

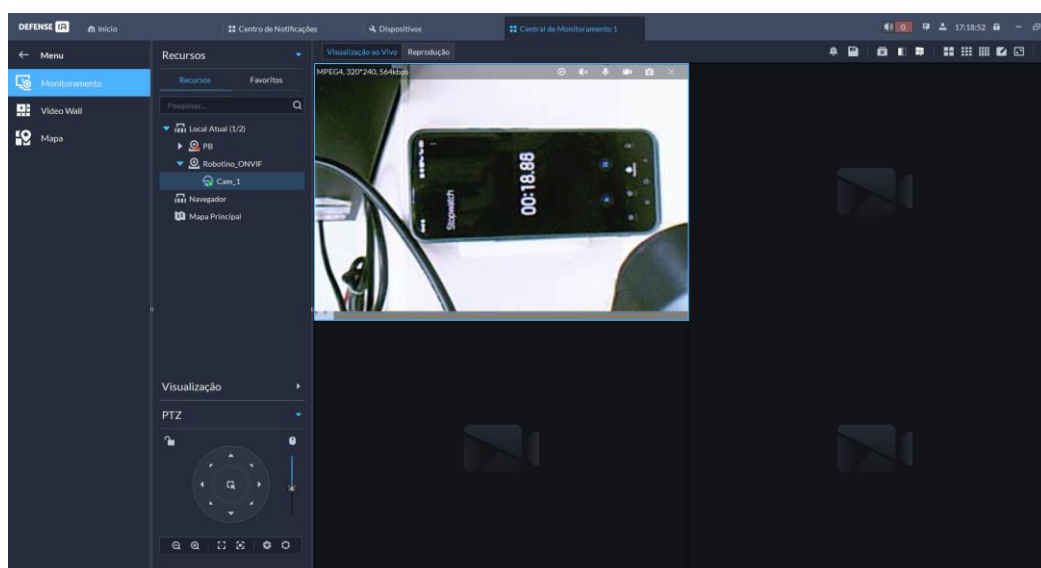
Figura 23 - Topologia completa



Fonte: Elaboração própria (2024).

Ao executar simultaneamente o método de transmissão da API do Robotino, software *middleware*, servidor RTSP, servidor ONVIF e Defense IA, foi possível visualizar o vídeo da câmera do Robotino em tempo real. O resultado disso pode ser observado na Figura 24.

Figura 24 - Câmera do Robotino no Defense IA

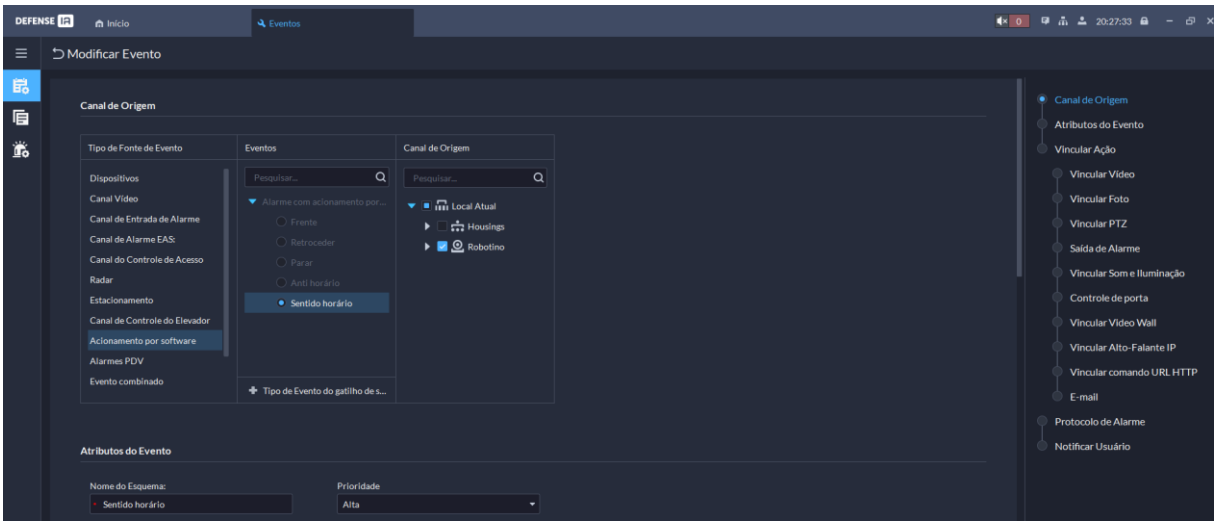


Fonte: Elaboração própria (2024).

4.8 Comandos dos motores via interface

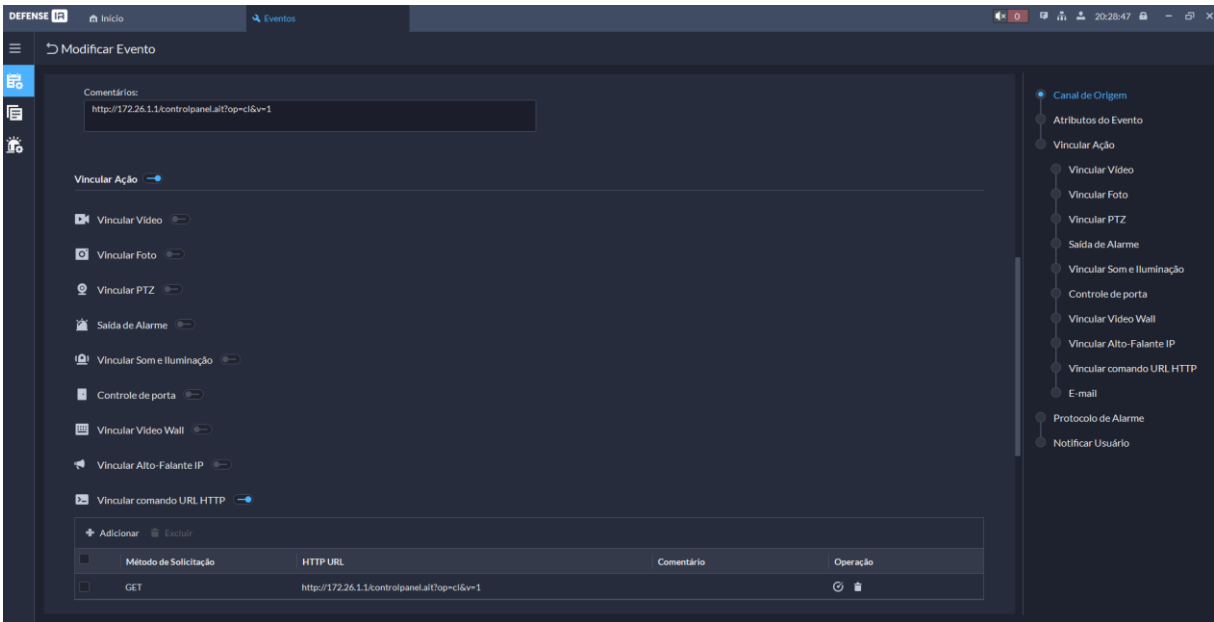
Como validado na etapa de estudo da API REST do Robotino, os motores podem ser controlados pelos métodos apresentados no Quadro 4. Para implementar estes comandos na interface de videomonitoramento, foram configurados no Defense IA gatilhos de eventos para botões associados ao canal de vídeo da câmera virtual, de acordo com as imagens exibidas nas Figura 25 e Figura 26, que quando pressionados disparam requisições HTTP para o endereço IP do Robotino.

Figura 25 - Configuração de botão em canal origem



Fonte: Elaboração própria (2024).

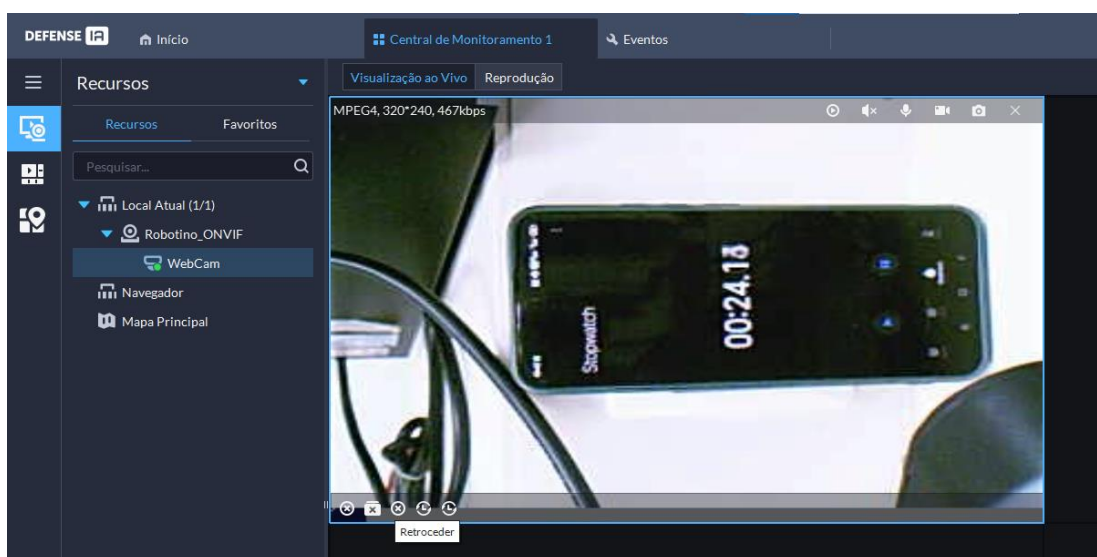
Figura 26 - Vinculo de comando URL HTTP



Fonte: Elaboração própria (2024).

A Figura 27 demonstra os botões na interface, com os comandos para : Avançar; Retroceder; Cessar Movimento; Girar no sentido horário; Girar no sentido anti-horário.

Figura 27 - Botões de controle na interface de videomonitoramento



Fonte: Elaboração própria (2024).

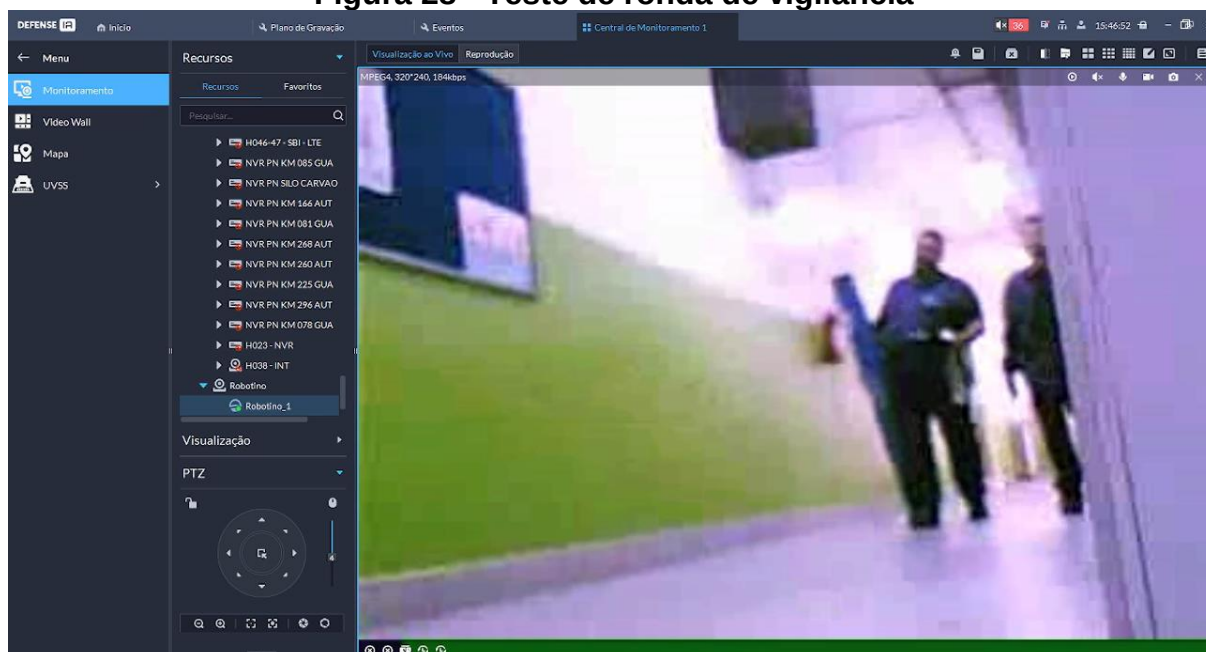
Utilizando os botões configurados foi possível controlar o deslocamento do Robotino.

4.9 Resultado final

Houve êxito em cumprir os objetivos específicos propostos, sendo que o software middleware desenvolvido foi capaz de capturar as imagens da câmera embarcada do Robotino, converter a sequência de imagens em transmissão de vídeo RTSP utilizando o FFmpeg, simular uma câmera IP com o servidor ONVIF, configurar a câmera no Defense IA, visualizar a imagem e comandar os motores via botões na interface.

Com relação ao teste do Robotino como um robô de vigilância, o mesmo foi controlado se movendo pelos corredores do IFSC, dentro do alcance do sinal de sua rede WIFI, como pode-se ver na Figura 28.

Figura 28 - Teste de ronda de vigilância



Fonte: Elaboração própria (2024).

Por fim, o protótipo apresentou um desempenho suficiente para fins didáticos, mas insatisfatório a finalidade de realizar rondas remotamente em uma aplicação real. Os pontos negativos incluíram a baixa qualidade da imagem da câmera, com resolução de 320 por 240 pixels, que permitiu identificar pessoas no corredor, mas não as reconhecer. Além disso, ocorreram eventuais travamentos na unidade de controle do Robotino, que possui baixo poder de processamento para a transmissão de vídeo, fazendo com que o sistema parasse de responder devido ao estouro de memória no sistema operacional. Esses travamentos só foram contornados com a reinicialização da unidade de controle.

5 CONSIDERAÇÕES FINAIS

Neste trabalho, foi possível realizar a integração do robô móvel de aprendizagem Robotino com o sistema de videomonitoramento Defense IA viabilizando a realização de rondas de vigilância remotamente, com a visualização de imagem da câmera embarcada do robô em tempo real, sem atraso significativo, taxa de frames constante e comandando os motores de deslocamento remotamente, conforme planejado.

Embora o Robotino tenha se mostrado uma ferramenta didática útil para o desenvolvimento deste projeto, suas limitações de hardware, como a unidade de processamento com baixo poder computacional e a câmera de baixa resolução, impactaram o desempenho geral da solução. Essas restrições limitam a aplicação do sistema em cenários não acadêmicos, que possuem como premissa a exigência de que o robô nunca trave, e que possua qualidade de imagem suficiente para obter mais detalhes visuais.

5.1 Sugestões para trabalhos futuros

Para trabalhos futuros, recomenda-se a utilização de uma unidade de processamento com maior capacidade, o que permitiria maior eficiência no processamento de dados em tempo real. Além disso, a adoção de uma câmera com maior resolução melhoraria significativamente a qualidade da imagem capturada, aumentando a precisão na identificação de objetos e pessoas.

Outra sugestão é explorar a implementação de algoritmos de inteligência artificial (IA) para gerar alertas automaticamente ao detectar pessoas ou comportamentos suspeitos durante as rondas de segurança. Também seria relevante desenvolver rotinas de locomoção automatizadas para que o robô pudesse realizar patrulhas autônomas, minimizando a necessidade de intervenção humana. Por fim, um campo promissor para futuras pesquisas seria a integração de periféricos adicionais acoplados ao robô, como sensores de movimento, microfones ou até sistemas de comunicação, expandindo o escopo de suas funcionalidades em aplicações de segurança e monitoramento.

Com essas melhorias, será possível desenvolver soluções mais robustas e adaptadas às necessidades reais de segurança, utilizando tecnologias emergentes de automação e IA para aprimorar o monitoramento em tempo real e a eficácia na resposta a eventos.

REFERÊNCIAS

AMAZON. **What is API**. 2024b. Disponível em: <https://aws.amazon.com/pt/what-is/api/>. Acesso em: 28 ago. 2024.

AMAZON. **What is middleware**. 2024a. Disponível em: <https://aws.amazon.com/pt/what-is/middleware>. Acesso em: 28 ago. 2024.

AVILA, Daniel *et al.* Simulation in Model Based Systems Engineering: Estimating Robot Range and Battery Performance. *In: Proceedings of the Annual General Donald R. Keith Memorial Conference*. Nova York: West Point, 2024. p. 1-6

BRASIL. **Projeto de Lei n. 1043, de 2021**. Brasília: Senado Federal, 2021. Disponível em: <https://www.senado.leg.br>. Acesso em: 28 jul. 2024.

CERQUEIRA, Daniel; BUENO, Samira (coord.). **Atlas da violência 2023**. Brasília: Ipea; FBSP, 2023.

DAMJANOVSKI, Vlado. **CCTV Cameras**. 2014. Disponível em: <https://ebook.intelbras.com.br/catalogo-intelbras/>. Acesso em: 28 jul. 2024.

FESTO DIDACTIC. **Robotino® 3** - Overview. 2024. Disponível em: <https://ip.festo-didactic.com/InfoPortal/Robotino3/Overview/EN/index.html>. Acesso em: 28 ago. 2024.

FESTO. **Robotino® Manual 73770**. Denkendorf: Festo, 2007.

FFMPEG. **About FFmpeg**. 2024a. Disponível em: <https://ffmpeg.org/about.html>. Acesso em: 28 ago. 2024.

FFMPEG. **FFmpeg Documentation**. 2024b. Disponível em: <https://ffmpeg.org/ffmpeg.html>. Acesso em: 28 ago. 2024.

GIL, A. C. **Métodos e técnicas de pesquisa social**. 6. ed. São Paulo: Atlas, 2008. Disponível em: <<https://books.google.com.br/books?id=T3BwPgAACAAJ>>.

HEXEDIT. **Hexadecimal Edit Platform**. Disponível em: <https://hexed.it/>. Acesso em: 06 nov. 2024.

HUANG, Li *et al.* A survey of multi-robot regular and adversarial patrolling. **IEEE/CAA Journal of Automatica Sinica**, v. 6, n. 4, p. 894-903, jul. 2019.

INTELBRAS. **Manual Defense IA**. 2024. Disponível em: manuais.intelbras.com.br/manual-software-de-seguranca-eletronica-defense-ia/. Acesso em: 28 ago. 2024.

KAUFFMANN, Boris. **Protocolos de Transmissão de Vídeo sobre IP com Compressão**. 2023. Disponível em: ad-digitaltecnologia.com.br. Acesso em: 28 ago. 2024.

KRUEGLE, Herman. **CCTV surveillance**: video practices and technology. Newton, MA: Butterworth-Heinemann, 1995.

LIN, Chia-Feng *et al.* A fault-tolerant ONVIF protocol extension for seamless surveillance video stream recording. **Computer Standards & Interfaces**, Taiwan, v. 55, p. 55-72, jan. 2018.

LÓPEZ, J. *et al.* WatchBot: A building maintenance and surveillance system based on autonomous robots. **Robotics and Autonomous Systems**, System Engineering and Automation Department, University of Vigo, Campus Lagoas-Marcosende, 36200 Vigo, Spain, v. 61, n. 12, p. 1559-1571, Dez 2013.

MINAYO, M. C. S. **O desafio do conhecimento: pesquisa qualitativa em saúde**. 9. ed. São Paulo: Hucitec, 2006, p. 406. Disponível em: <https://www.scielo.br/j/csc/a/FgpDFKSpjsybVGMj4QK6Ssv/#>

MYAGMAR-OCHIR, Y.; KIM, W. Video surveillance systems in smart cities: functions, challenges, and future directions. **Electronics**, Computer Engineering Department, Gachon University, Seongnam 1342, Republic of Korea, v. 12, n. 3, p. 567, ago. 2023.

ONVIF. **About ONVIF**: mission. Disponível em: <https://www.onvif.org/about/mission/>. Acesso em: 28 ago. 2024.

ONVIF. **About ONVIF**: mission. Disponível em: <https://www.onvif.org/about/mission/>. Acesso em: 28 ago. 2024.

ONVIF. **ONVIF profile feature overview**. [s.l.]: [s.n.], 2022. Disponível em: <https://www.onvif.org/wp-content/uploads/2022/04/onvif-profile-feature-overview.pdf>. Acesso em: 28 ago. 2024.

POSTMAN. **Postman API Platform**. Disponível em: <https://www.postman.com/>. Acesso em: 06 nov. 2024.

PREFEITURA DA CIDADE DO RIO DE JANEIRO. **Centro de Operações Rio**. 2024. Disponível em: <https://cor.rio/>. Acesso em: 18 ago. 2024

SANTOS, Yan Gomes dos; MELO, Felipe Pereira de. Segurança privada no Brasil: um olhar acadêmico sob a ótica da revisão sistemática. *In*: XII Encontro de Pesquisas em Ciências Contábeis, 12., 2021, Maringá. **Anais...** Maringá: UNICESUMAR, 2021. p. 1-9

SARAIVA, Pedro E. **Aplicação de um Veículo Guiado Automatizado (AGV) no Contexto de IoT Aplicada a Inteligência Perimetral**. Jun, 2023

SCHULZRINNE, H. (ed.). *et al.* **Real-Time Streaming Protocol Version 2.0**. 2016. Disponível em: <https://www.rfc-editor.org/info/rfc7826>. Acesso em: 28 ago. 2024.

SCHULZRINNE, H.; RAO, A.; LANPHIER R. **Real Time Streaming Protocol (RTSP)**. 1998. Disponível em: <https://www.rfc-editor.org/info/rfc2326>. Acesso em: 28 ago. 2024.

SEGUI, F. Boronat. **Análisis del Protocolo RTSP con Wireshark**. 2022. Disponível em: <http://hdl.handle.net/10251/183944>. Acesso em: 28 ago. 2024.

SEVERINO, A. J. **Metodologia do trabalho científico**. 23. ed. São Paulo: Cortez, 2007. Disponível em : https://edisciplinas.usp.br/pluginfile.php/3480016/mod_label/intro/SEVERINO_Metodologia_do_Trabalho_Cientifico_2007.pdf

SOAPUI. **SoapUI Open Source**. Disponível em: <https://www.soapui.org/downloads/soapui/>. Acesso em: 06 nov. 2024.

SUKHAVASI, S. B. *et al.* CMOS image sensors in surveillance system applications. **Sensors**, USA, v. 21, n. 488, jan. 2021.

VIJEIKIS, R.; RAUDONIS, V.; DERVINIS, G. Towards Automated Surveillance: A Review of Intelligent Video Surveillance. *In*: ARAI, K. (ed.). **Intelligent Computing**. Cham, Suíça: Springer, 2021. p. 784-803

APÊNDICES

APÊNDICE A - Requisição de imagem via API REST do Robotino

```
1 import requests
2 import time
3 from PIL import Image
4
5 # Get the current time in seconds since epoch
6 current_time = int(time.time())
7
8 # Construct the URL with the current time
9 url = f"http://172.26.1.1/liveimage.jpg?{current_time}"
10
11 response = requests.get(url)
12
13 with open("liveimage.jpg", "wb") as f:
14     f.write(response.content)
15
16 image = Image.open("liveimage.jpg")
17 image.save("liveimage2.jpg")
```

APÊNDICE B - Requisição de imagens via API REST do Robotino

```
1 import requests
2 import time
3 from PIL import Image
4
5 # Define the URL and filename
6 url = "http://172.26.1.1/liveimage.jpg"
7
8 # image name
9 image_str_initial = "liveimage"
10
11 # Get the current time
12 start_time = time.time()
13
14 time.sleep(5)
15
16 # Loop 15 times, getting an image every second
17 for i in range(15):
18     # Get the current time and format it as a string
19     current_time = int(time.time())
20
21     # Construct the full URL with the current time
22     full_url = f"http://172.26.1.1/liveimage.jpg?{current_time}"
23     print(full_url)
24
25     # Get the image from the URL
26     response = requests.get(full_url)
27
28     # Save the image as a JPEG file
29     with open(f"image_sequence/image{i:04d}.jpg", "wb") as f:
30         f.write(response.content)
31
32     print("Images saved")
33
34     time.sleep(0.066)
35
36 # Print a message indicating that the images have been saved
37 print("Images saved successfully!")
```

APÊNDICE C - Login nos dispositivos e keep alive do token

```

1 import requests
2 import time
3 from xml.etree import ElementTree as ET
4
5 def login_camera():
6     url = 'http://172.26.1.1:80'
7     body = """
8     <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dev="urn:roboticsec.de/DeviceManager">
9         <soapenv:Header/>
10        <soapenv:Body>
11            <dev:open>
12                <dev:id>1</dev:id>
13                <dev:accessMode>3</dev:accessMode>
14            </dev:open>
15        </soapenv:Body>
16    </soapenv:Envelope>
17    """
18    headers = {
19        'Content-Type': 'text/xml'
20    }
21    response = requests.post(url, data=body, headers=headers)
22    # print(response.content)
23    # response.raise_for_status() # Verifica se houve erro na
    requisição
24
25    root = ET.fromstring(response.content)
26    token_element =
    root.find('.://{urn:roboticsec.de/DeviceManager}token')
27    if token_element is None:
28        raise ValueError("Elemento 'token' não encontrado na resposta
    XML")
29    token_value= token_element.text
30    # print(" --- ")
31    # print(root)
32    # print(token_element)
33    print(("Token solicitado para câmera"))
34    print("Valor do token:" + str(token_value))
35
36    return token_value
37
38 def keep_alive(token):
39     url = 'http://172.26.1.1:80'
40     body = """
41     <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dev="urn:roboticsec.de/DeviceManager">
42        <soapenv:Header/>
43        <soapenv:Body>
44            <dev:keepAlive>
45                <dev:token>{}</dev:token>
46            </dev:keepAlive>
47        </soapenv:Body>
48    </soapenv:Envelope>
49    """.format(token)
50
51    headers = {
52        'Content-Type': 'text/xml'

```

```
53     }
54     response = requests.post(url, data=body, headers=headers)
55
56
57     root = ET.fromstring(response.content)
58
59     return response.content
60
61 if __name__ == "__main__":
62
63     token = login_camera()
64     while True:
65         response = keep_alive(token)
66         print("Resposta do keepAlive:", response)
67         time.sleep(10) # Solicita o keepAlive a cada 10 segundos.
```

APÊNDICE D - Recebimento das imagens do stream UDP

```
1 import cv2
2 import numpy as np
3 import socket
4 from PIL import Image
5 import os
6
7 # Endereço IP e porta do Robotino
8 ROBOTINO_IP = '172.26.1.100'
9 VIDEO_PORT = 37777
10
11 print("aqui")
12 # Configurar socket UDP
13 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
14 sock.bind((ROBOTINO_IP, VIDEO_PORT))
15
16 # Pasta para salvar as imagens recebidas
17 output_folder = "imagens_recebidas"
18
19 # Verifica se a pasta de saída existe, se não, a cria
20 if not os.path.exists(output_folder):
21     os.makedirs(output_folder)
22
23 while True:
24     print("aqui 1")
25     # Receber os dados do stream
26     data, addr = sock.recvfrom(65535) # 65535 é o tamanho máximo dos
    dados
27
28     print("DATA: " + str(data))
29
30     # Verifica se os dados são uma imagem JPEG
31     if b'\xff\xd8\xff' in data:
32         # Gera um nome de arquivo único para a imagem
33         filename = os.path.join(output_folder,
34 f"imagem_{len(os.listdir(output_folder))}.jpg")
35
36         # Salva a imagem no disco
37         with open(filename, 'wb') as f:
38             f.write(data[20:])
39     else:
40         print("Dados recebidos não contêm o cabeçalho de uma imagem
    JPEG.")
41
42 sock.close()
```

APÊNDICE E – Conversão de imagens e transmissão de vídeo

```
1 import glob
2 import os
3 import cv2
4
5
6 if __sendImg__ == '__main__':
7
8
9     # Create a video writer object
10    # fourcc = cv2.VideoWriter_fourcc(*'mp4v') # valor número de MP4
    = 0x7634706D
11    video_writer = cv2.VideoWriter('output_png.mp4', 0x7634706D, 25,
    (320, 240))
12
13    image_path = 'image_sequence_png/*.png'
14
15    # Get a list of all the images in the sequence
16    images = glob.glob(image_path)
17
18    # Sort the images by name
19    images.sort()
20
21
22    # Write the images to the video
23    for image in images:
24        frame = cv2.imread(image)
25        video_writer.write(frame)
26
27    # Release the video writer object
28    video_writer.release()
29
30    # Create a ffmpeg command to convert the video to RTSP stream
31    command = f'ffmpeg -i output.mp4 -f rtsp
    rtsp://localhost:8554/feed1.sdp'
32
33    # Execute the ffmpeg command
34    os.system(command)
```

APÊNDICE F – Transmissão em pipeline

```
1 import socket
2 import subprocess
3
4 # CONFIGURAÇÕES
5 ROBOTINO_IP = '172.26.1.100'
6 VIDEO_PORT = 37777 # Porta UDP para recebimento de imagens do
Robotino
7
8 RTSP_URL = "rtsp://127.0.0.1:8554/mystream" # URL RTSP para
transmitir o vídeo
9 FFMPEG_CMD = [
10     "ffmpeg",
11     "-f", "image2pipe",
12     "-i", "pipe:",
13     "-f", "rtsp",
14     "-rtsp_transport", "udp",
15     RTSP_URL
16 ]
17
18 # Configurar socket UDP
19 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
20 sock.bind((ROBOTINO_IP, VIDEO_PORT))
21
22 # Inicia o processo do ffmpeg
23 ffmpeg_process = subprocess.Popen(FFMPEG_CMD, stdin=subprocess.PIPE)
24
25
26 while True:
27     print("Início do loop")
28     # Receber os dados do stream
29     data, addr = sock.recvfrom(65535) # 65535 é o tamanho máximo dos
dados
30     ffmpeg_process.stdin.write(data[20:])
31
32     print("DATA: " + str(data))
33
34
35 # Limpa os recursos
36 sock.close()
37 ffmpeg_process.stdin.close()
38 ffmpeg_process.wait()
```