Universidade Estadual de Londrina

FERNANDO MORGADO PIRES NETO

CLASSIFICAÇÃO DE FLUXO DE DADOS PARA VISÃO COMPUTACIONAL EMBARCADA

LONDRINA
2023

FERNANDO MORGADO PIRES NETO

CLASSIFICAÇÃO DE FLUXO DE DADOS PARA VISÃO COMPUTACIONAL EMBARCADA

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Gilberto Fernandes Júnior
Coorientador: Prof. Dr. Guilherme Pina Cardim

LONDRINA
2023

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Neto, Fernando Morgado Pires.
CLASSIFICAÇÃO DE FLUXO DE DADOS PARA VISÃO COMPUTACIONAL EMBARCADA / Fernando Morgado Pires Neto. - Londrina, 2023.
56 f.

Orientador: Gilberto Fernandes Júnior.
Coorientador: Guilherme Pina Cardim.
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Graduação em Ciência da Computação, 2023.
Inclui bibliografia.

1. Classificação de Fluxo de Dados - TCC. 2. Visão Computacional Embarcada - TCC. 3. Processamento de Imagem - TCC. 4. Computação em tempo real - TCC. I. Fernandes Júnior, Gilberto. II. Pina Cardim, Guilherme. III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Graduação em Ciência da Computação. IV. Título.

CDU 519

FERNANDO MORGADO PIRES NETO

CLASSIFICAÇÃO DE FLUXO DE DADOS PARA VISÃO COMPUTACIONAL EMBARCADA

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Gilberto Fernandes Júnior
Universidade Estadual de Londrina

Prof. Dr. Guilherme Pina Cardim
Universidade Estadual Paulista – UNESP

Prof. Dr. Evandro Baccarin
Universidade Estadual de Londrina - UEL

Londrina, 29 de maio de 2023.

AGRADECIMENTOS

Começo agradecendo a minha familia, que desde sempre esteve ao meu lado, oferecendo apoio incondicional e assistência em todos os momentos da minha vida. Sem o amor e a compreensão proporcionada pelos meus familiares, eu não teria chegado até aqui.

Agradeço aos professores do curso pelos ensinamentos, em especial ao meu orientador, Guilherme Pina Cardim, que sempre se mostrou solícito e disposto a me ajudar no desenvolvimento desse trabalho, e ao professor Sylvio Barbon Junior, que me auxiliou com a concepção inicial do mesmo.

Agradeço aos meus amigos, por compartilharmos juntos os momentos felizes e difíceis que passamos ao longo desses anos de graduação.

Por fim, quero agradecer a todos os que, direta ou indiretamente, contribuíram para que eu pudesse chegar até aqui.

"A realidade existe na mente humana e em nenhum outro lugar.
(George Orwell)

PIRES NETO, F. M.. **Classificação de Fluxo de Dados para Visão Computacional Embarcada**. 2023. 56f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

Os veículos guiados automatizados (*automaded guided vehicles* - AGV) são uma categoria de robô industrial que ajudam a melhorar a logística e mobilidade do setor. Eles contêm sensores, como câmeras de profundidade, que os ajudam a enxergarem e se localizarem no ambiente. Os AGVs devem responder rapidamente aos eventos que acontecem ao seu redor e, para isso, necessitam processar os dados de seus sensores em tempo real. O presente projeto visa investigar e realizar experimentos com algoritmos na área de Visão Computacional Embarcada para classificação rápida da presença de obstáculos na cena, e almeja ser um ponto de partida para projetos futuros que queiram otimizá-los.

Palavras-chave: Classificação de Fluxo de Dados. Visão Computacional Embarcada. Processamento de Imagem. Computação em tempo real.

PIRES NETO, F. M.. **Data Stream Classification using Embedded Computer Vision**. 2023. 56p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2023.

ABSTRACT

Automated guided vehicles (AGV) are a type of industrial robot that helps to improve logistics and mobility in its area. They contain several sensors, such as depth cameras, that assist them to see and locate themselves in the environment. AGVs must have the ability to quickly respond to events happening in theirs surroundings, and to achieve that, they must process the data coming from their sensors in real time. This project seeks to investigate and experiment with Embedded Computer Vision algorithms for fast classification of the presence of obstacles in the scene, and aims to be a starting point for future projects that may want to optimize them.

Keywords: Data Stream Classification. Embedded Computer Vision. Image processing. Real-time computing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AGV	*Automated Guided Vehicle*
ARIMA	*Autoregressive Integrated Moving Average*
CPU	*Central Processing Unit*
CUSUM	*Cumulative Sum*
FN	Falso Negativo
FP	Falso Positivo
fps	*frames* por segundo
GPU	*Graphics Processing Unit*
ML	*Machine Learning*
OCC	*One Class Classification*
RGB	*Red, Green and Blue*
SVDD	*Support Vector Data Description*
SVM	*Support Vector Machines*
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo

SUMÁRIO

1 INTRODUÇÃO

Os desenvolvimentos tecnológicos modernos têm potencializado diversas áreas da vida cotidiana, sendo o setor industrial um dos mais beneficiados. As novas tecnologias que surgiram focaram em melhorias no processo produtivo – ponto-chave para o crescimento econômico das indústrias [2].

Dentre essas tecnologias, uma que se destaca são os *automated guided vehicles* (AGV). O AGV é um robô industrial que possui sensores capazes de situarem sua posição no espaço e, desta forma, fazer com que ele seja capaz de se movimentar de forma autônoma [3]. Assim, ele é capaz de trazer melhorias para a logística do ambiente onde ele é instalado, o que proporciona redução de custos e otimiza o processo de automação da indústria.

Devido aos requisitos de se trabalhar em um ambiente industrial, os AGVs devem ser precisos e seguros. Todas as suas ações devem ser processadas em tempo real, o que garante ao robô capacidade de responder rapidamente aos eventos que acontecem ao seu redor [4]. Esta peculiaridade torna esta categoria de robô um grande desafio tecnológico, devido à necessidade de otimizar cada um dos seus aspectos, tanto do lado do *hardware* como do *software*.

A maioria dos AGVs modernos utiliza uma câmera de profundidade para criar uma imagem virtual precisa do ambiente e identificar possíveis obstáculos. No entanto, essa configuração requer um *hardware* robusto para processar o fluxo de dados proveniente desse sensor. Por conta disso, o projetista do sistema deve saber balancear o desempenho do algoritmo de processamento com gasto energético do robô, para que seus outros sistemas continuem funcionando normalmente [3].

Objetiva-se então, através do presente projeto, realizar o estudo e a experimentação com algoritmos de *Machine Learning* para Visão Computacional Embarcada, e verificar seu uso em comparação com outras técnicas para classificação de obstáculos em tempo real.

2 COMPUTAÇÃO GRÁFICA - CAPTURA, ANÁLISE E PROCESSAMENTO DE IMAGENS

A Computação Gráfica é uma área da Ciência da Computação que trata da geração, manipulação e exibição de imagens e vídeos digitais [5]. Sua aplicação pode ser observada em uma variedade de campos como jogos, design industrial, arquitetura, medicina e robótica, entre outras que necessitem criar, armazenar e/ou manipular modelos de objetos e suas imagens via computador [6]. Suas subáreas podem ser divididas em três principais: Processamento de Imagem, Síntese de Imagem e Visão Computacional [7]. A representação do relacionamento entre elas pode ser exemplificada através do diagrama na Figura 1.

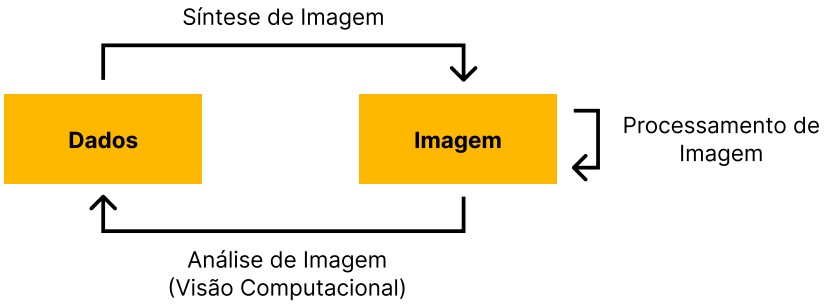


Figura 1 – Relação entre subáreas da Computação Gráfica.

Apesar de suas relações, cada uma das áreas possuem seus próprios objetivos, que serão detalhados nas subseções a seguir.

2.1 Processamento de Imagem

Processamento de imagem se concentra na análise, manipulação e geração de imagens digitais. Isso inclui técnicas como filtragem, transformações geométricas, segmentação e compressão de imagens. O objetivo geral é extrair informações úteis das imagens ou melhorar sua qualidade, garantindo melhor análise dos dados em etapas posteriores.

Uma imagem digital pode ser definida como uma função de intensidade luminosa bidimensional f, onde um valor de intensidade é atribuído a uma coordenada espacial (x, y) [7]. Os valores podem ser obtidos pelo processo de digitalização da imagem, onde dados oriundos da transformação de amostragem discreta de um sensor são quantizados para determinar a informação necessária para armazenar a imagem e, por fim, codificados para se obter sua representação digital.

Em termos computacionais, a função bidimensional pode ser representada como uma matriz, onde cada posição armazena o valor de intensidade da imagem, também chamado de *pixel* [8] (Figura 2). Essa forma de armazenamento é chamada de banda espectral, onde cada banda denota uma imagem monocromática (Figura 3). Da mesma forma, uma imagem multibanda representa uma sequência de imagens monocromáticas, ou seja, uma matriz de n dimensões [7].

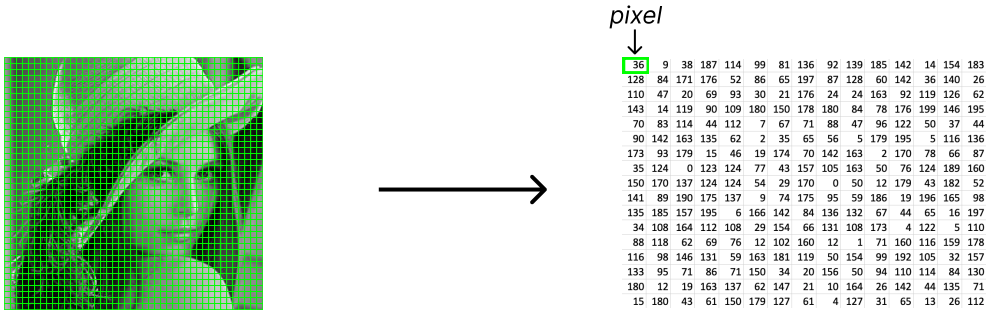


Figura 2 – Representação matricial da banda de uma imagem

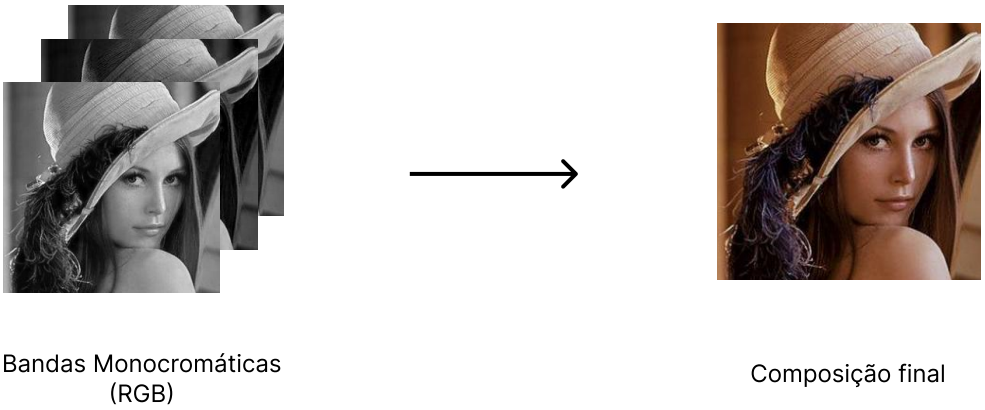


Figura 3 – Exemplo de composição final de uma imagem RGB

Cada dimensão pode ser responsável por armazenar um dado específico da imagem, como cores, distância, temperatura, e fica a cargo da Síntese de Imagens realizar sua apresentação conforme as características do cenário proposto, e da Visão Computacional por analisar o que a imagem representa [7].

2.1.1 Captura da informação

Existem diversos tipos de sensores utilizados para capturar a informação desejada, e sua escolha depende do tipo da aplicação a ser realizada. Para aplicação em AGV, as câmeras são as melhores opções. Elas são leves, não consomem muita energia, e possibilitam coletar uma quantidade abundante de informações acerca do ambiente [4]. Além disso, também existem as opções de câmeras que capturam distância, com imagem RGB, o que as tornam ideais para percepção detalhada do posicionamento dos objetos ao redor do robô [9].

Uma técnica para obtenção desse dado é através do sistema de visão estéreo, também chamado de estereoscopia, conforme exemplificado na Figura 4. Nele, duas câmeras são posicionadas paralelamente, em formação similar ao dos olhos humanos, e calculam a distância através da correlação dos pontos da imagem geradas pela câmera da esquerda com a da direita [10]. Opcionalmente, um projetor infravermelho auxilia na construção da imagem ao projetar um padrão de pontos, capturado por um sensor próprio, que contribui no alinhamento das imagens e visualização em ambientes com baixa luminosidade.

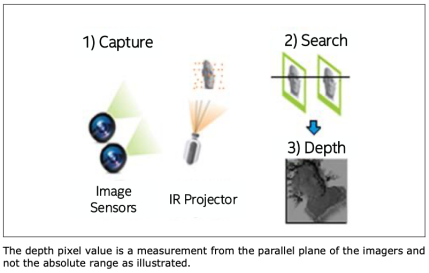


Figura 4 – Funcionamento de câmera com sistema de visão estéreo. (Fonte: Intel RealSense [1])

2.1.2 Tratamento dos dados

Com o intuito de extrair apenas as informações necessárias do *frame* e diminuir o custo computacional requerido para processar os dados, as técnicas de compressão, recorte e filtragem são as mais utilizadas [11]. Na compressão, o princípio é reduzir o tamanho da imagem sem perdas significativas de qualidade. Um exemplo desse processo é a redução de bandas de uma imagem agrupando-as em uma só, como está exibido na Figura 5.

Figura 5 – Exemplo de compressão de imagem RGB para Monocromática.

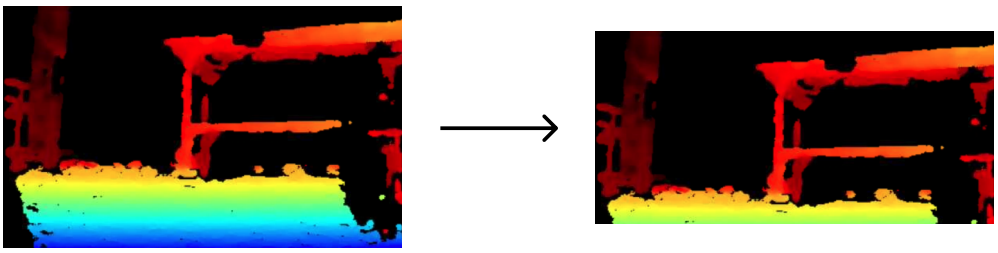
Esse processo pode ser feito realizando a média dos valores das camadas, resultando na imagem unidimensional, como mostrado na equação 2.1, resultante em uma matriz unidimensional, composta pela soma valor de cada *pixel* de camada i sobre o número de camadas n.

$$f(x,y)' = \frac{\sum_0^i f(x,y,i)}{n} \tag{2.1}$$

A técnica de recorte de imagem é um processo em que uma porção específica de uma imagem é selecionada e cortada, criando uma nova imagem que contém apenas essa porção. Ao remover informações irrelevantes da imagem, essa técnica beneficia o processamento de imagem em visão computacional criando de conjuntos de dados mais eficientes [12]. Dessa forma, o corte da imagem pode melhorar a eficácia de algoritmos de detecção de objetos e acelerar o processamento, resultando em resultados mais precisos e acurados.

Por exemplo, para analisar obstáculos em um *frame*, não se faz necessário a análise do chão, tendo em vista que os mesmos aparecem acima da linha do horizonte. Dessa forma, é possível recortar a imagem ao remover o piso (Figura 6) e realizar a análise apenas com a parte superior da cena, o que consequentemente reduz a quantidade de dados necessárias durante a etapa de processamento.

Outra técnica de processamento de imagem é a filtragem. Nela, o intuito é aplicar uma função de transformação sobre a imagem para corrigir, suavizar ou realçar suas característica. Na correção, ocorre uma remoção de informações indesejáveis na imagem. Já na suavização ou realce, os detalhes da imagem são atenuados ou amplificados, a depender da função de transformação aplicada.



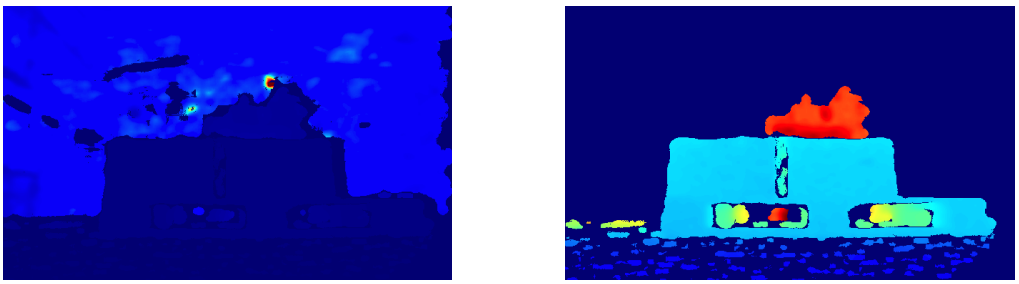
Frame original *Frame* segmentado

Figura 6 – Segmentação de *frame* de distância.

Na banda oriunda de uma câmera de profundidade, em alguns casos, pode haver dados indesejáveis que podem atrapalhar a análise dos algoritmos de Visão Computacional. Em sua maioria, estes se referem a valores de distância que estão fora do *range* ideal de operação da câmera, ou do cenário proposto para análise. Para corrigir essa informação, é aplicada uma função de transformação em cada pixel da banda de profundidade, filtrando apenas aqueles que pertencem ao *range* desejado. Um exemplo dessa função está descrito na equação 2.2, onde os valores de distância que estiverem fora de um *range* > 0 até n são atribuídos como 0.

$$f'(x,y) = \begin{cases} f(x,y), & \text{se } 0 < f(x,y) \leq n \\ 0, & \text{se } f(x,y) < 0 \text{ ou } f(x,y) > n \end{cases} \tag{2.2}$$

A Figura 7 exemplifica a aplicação da função exemplificada na equação 2.2 em um *frame*. O *frame* exibido é o mesmo utilizado para treino de classificação de obstáculo. É notável que, após a aplicação do filtro, os contornos do obstáculo se tornam mais evidentes, proporcionando uma melhor distinção do mesmo em relação ao fundo do ambiente.



(a) Imagem original (b) Imagem filtrada

Figura 7 – Aplicação da função de filtragem.

A técnica de filtragem também pode servir como compressão, tendo em vista que remove da imagem dados que não serão utilizados na etapa de análise. Dessa forma, ao utilizar as técnicas descritas, é possível destacar apenas as características importantes da imagem ao problema a ser solucionado, melhorando o processamento da mesma ao reduzir o tempo de processamento e garantir que o mesmo analise apenas as regiões necessárias da imagem.

2.2 Síntese de Imagem

A área de Síntese de Imagem se destina na conversão dos dados obtidos através dos processos anteriores em uma imagem digital a ser visualizada [8]. Seu uso pode ser conduzido desde a criação de uma imagem do zero, até à conversão e apresentação de dados de uma situação-problema de forma visual e mais compreensível ao humano. Essa etapa não é necessária para tomada de ação por parte do robô após a classificação da imagem, mas sua execução é importante para visualização dos dados de saída dos algoritmos e equipamentos de captura de utilizados.

Conforme descrito na seção 2.1, a câmera de profundidade retorna uma banda de dados contendo a distância do sensor a um objeto na cena em cada um dos pixeis da imagem. Uma análise desses dados de uma forma numérica pode ser considerada complexa. Dessa forma, pode-se utilizar da Síntese de Imagem para construir uma imagem para atribuir um padrão de cores correspondente ao valor de distância armazenado na matriz, como mostra a Figura 8.

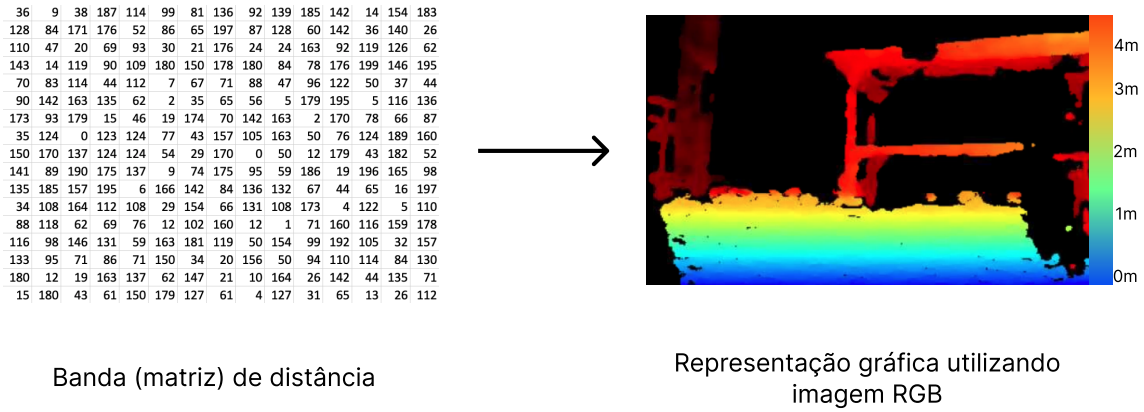


Figura 8 – Síntese de matriz de distância em imagem RGB.

2.3 Visão Computacional

A visão computacional é o campo de estudo que visa utilizar métodos estatísticos para analisar os dados obtidos de imagens através de modelos matemáticos [11]. Ela é uma área interdisciplinar, que combina teorias e técnicas de processamento de imagens, aprendizado de máquina e inteligência artificial. Na visão computacional, existe a ramificação para sua área embarcada, que se concentra em aplicar técnicas de visão computacional em dispositivos embarcados, como câmeras, drones, robôs e veículos autônomos.

Ainda nessa área, há também o estudo dos componentes embarcados, que possui um foco maior no hardware empregue na captação, transmissão e processamento da imagem. Dessa forma, ela tem em vista aumentar a eficiência dos componentes e sistemas utilizados ao analisar todo o fluxo de dados obtido através da aquisição da imagem, até as operações que serão realizadas [13]. A revisão literária dos algoritmos utilizados será trabalhada com mais detalhes na seção de Classificação de Fluxo de Dados.

Para que os algoritmos obtenham o resultado esperado, é necessário um estudo preliminar dos componentes que serão utilizados para a aquisição e processamento dos dados. Isso se faz necessário, pois os algoritmos, por mais generalistas que sejam, precisam ser adaptados para o tipo de dado que está sendo coletado, e otimizado para conseguirem processar os dados no hardware no qual foi implementado.

O estudo do equipamento para aquisição de imagens pôde ser visto na subseção 2.1.1. Em relação ao *hardware* para processamento dos dados, o ideal é o uso de um *hardware* dedicado para analisar o fluxo de dados oriundos da câmera [13]. Nesse sentido, se destaca o uso de GPUs para processamento dos dados.

As GPUs são elaboradas para processar os dados paralelamente, através do uso de mais núcleos de processamento e arquitetura projetada para cálculo de dados vetoriais. Dessa forma, elas conseguem processar dados com mais rapidez, e também suportam maior volume de dados sendo processados simultaneamente [14]. Aliado a isso, os núcleos de processamento das GPUs podem ser projetados para consumirem menos energia do que CPUs tradicionais [15], ponto importante quando se trata de um sistema embarcado, como o AGV, onde o consumo de energia é um fator crítico. Assim, seu uso se torna relevante para o processamento de dados proposto.

3 CLASSIFICAÇÃO DE FLUXO DE DADOS

A Classificação de Fluxo de Dados é uma categoria de aprendizado incremental para altas taxas de transmissão de dados, que resolve problemas com tempo de processamento e memória limitados, e onde só há a possibilidade de análise única do dado recebido [16] [17]. Além disso, ela deve ser adaptativa, tendo em vista que boa parte de suas aplicações são feitas em ambientes não-estacionários, nos quais os dados recolhidos podem mudar com o tempo [17]. Apesar de seu uso na literatura ser mais relacionado à gerência e tráfego de dados em redes de computadores, o uso dessa categoria torna-se oportuno, pois se encaixa com a problemática e cenário previstos para o processamento de dados requerido da visão computacional [18] [19].

Para a seleção dos algoritmos de classificação de fluxo de dados, as seguintes propriedades devem estar presentes [17][20]:

1. Processar apenas um exemplo e inspecioná-lo apenas uma única vez;
2. Utilizar uma quantidade limitada de memória;
3. Estar pronto para classificar a qualquer momento;
4. Estar preparado para se ajustar ao desvio de conceito[1].

Essas características garantem que o algoritmo se adapte a qualquer ambiente que ele for instalado, enquanto produz um resultado preditivo consistente. Na literatura, os algoritmos de *Machine Leraning* (ML) são os que mais se enquadram nessa categoria [19]. Contudo, devido à natureza do problema presente neste projeto, e a robustez apresentada pelos algoritmos de ML, seu uso nesta situação pode ser considerado *overkilling*[2]. Dessa forma, a utilização de técnicas estatísticas para a classificação do fluxo de dados pode ser uma alternativa aos modelos de ML [21]. Estes, bem como suas classes temática, serão detalhados nas subseções a seguir.

3.1 *Machine Learning*

Machine Learning, ou Aprendizado de Máquina, é uma técnica onde um computador adapta suas ações e decisões com base em um conjunto de dados, visando aumentar a precisão das suas previsões [22][23]. O conjunto de dados utilizado como forma de experiência para o processo de aprendizado é composto por variáveis, chamadas também

[1] Quando os dados analisados mudam com o tempo de maneira imprevista
[2] Uso a mais do que o necessário de algo para um determinado fim

de *features* (características). A qualidade e quantidade das informações contidas nesses dados são fundamentais para um melhor desempenho e acerto nas previsões realizadas pelo algoritmo.

Existem duas categorias principais de algoritmos de ML: os supervisionados e não-supervisionados [24]. Algoritmos supervisionados são aqueles que aprendem a partir de um conjunto de dados rotulados, onde as saídas desejadas para cada entrada já são conhecidas. Isso permite que o algoritmo aprenda a fazer previsões ou classificações precisas sobre novos conjuntos de dados, uma vez que ele foi treinado com exemplos de entrada e saída. Por outro lado, algoritmos não-supervisionados aprendem a partir de dados não rotulados, onde as saídas desejadas não são conhecidas. Esses algoritmos são usados para encontrar padrões ou estruturas nos dados, como agrupamentos, tendências ou relações.

A categoria que mais se enquadra na proposta deste trabalho é a de algoritmos supervisionados, tendo em vista em que é possível treiná-los com cenas padrões (imagens com obstáculos) e sua saída é conhecida (*frame* com ou sem obstáculo). Para tal temática, dá-se o nome de classificação [25] e, dentre os algoritmos que se adéquam a classificação de fluxo de dados, evidencia-se o *One Class Classification* (OCC) [26] [27].

3.1.1 *One Class Classification*

O algoritmo *One Class Classification* (OCC) é um tipo de algoritmo de ML utilizado para identificar exemplos de uma única classe em um conjunto de dados, sem a necessidade de informações sobre outras classes [28]. Ele é amplamente utilizado em tarefas de detecção de anomalias, onde o objetivo é identificar exemplos que sejam significativamente diferentes dos exemplos normais, e produz uma resposta binária. Por conta disso, ele possui um rápido processamento [19] e, devido ao seu grande uso em aplicações e na literatura, já existem diversas implementações em bibliotecas e *frameworks* [29].

Seu funcionamento é baseado na construção de um modelo representativo da classe normal, usando exemplos positivos (da classe desejada), sendo então utilizado para classificar novos exemplos como sendo normais ou anômalos. Diversas técnicas são utilizadas para construir esse modelo, sendo uma delas o algoritmo para *clustering* [30]. Essa técnica procura agrupar exemplos similares juntos e, em seguida, utiliza o agrupamento mais denso como o modelo representativo da classe normal.

O seguinte pseudocódigo, proposto em [31], descreve o algoritmo de OCC de forma geral, sendo adequado para o cenário proposto pelo presente projeto.

1. Receber conjunto de dados com exemplos positivos (cena sem obstáculos)
2. Inicializar centroides dos agrupamentos
3. Treinar modelo usando conjunto de dados positivos

4. Para cada novo *frame* a ser classificada, realizar *clustering*:
    a) Calcular distância entre o novo *frame* e os centroides dos agrupamentos
    b) Atribuir *frame* ao agrupamento cujo centroide é mais próximo
    c) Se o *frame* estiver no agrupamento mais denso, classificar como normal
    d) Caso contrário, classificar como anormal
5. Atualizar centroides dos agrupamentos com base nos *frames* atribuídos
6. Iterar passo 4 e 5 até que o agrupamento não mude ou até alcançar o número máximo de iterações
7. Retornar classificações

Para realização da clusterização, os algoritmos *k-means* e *Support Vector Machine* (SVM) são os mais utilizados [32] [33] [34] [35]. O algoritmo *k-means* é geralmente mais rápido do que a SVM, pois possui baixo custo computacional, e pode ser usado para encontrar o *cluster* com a maior densidade e classificar qualquer ponto fora dele como uma anomalia. Já a SVM é baseada no conceito de vetores de suporte e, apesar de conseguir modelar limites de decisão mais complexos, requer mais poder computacional a depender das características do problema a ser resolvido.

Tendo em vista a problemática presente no projeto, o algoritmo SVM é o que mais se adéqua, pois, apesar de exigir mais computacionalmente, consegue trabalhar com modelos mais complexos, como nas imagens de profundidade. Seu uso se enquadra no Passo 4 do pseudocódigo apresentado anteriormente, e seu conceito busca encontrar uma hiperesfera que minimiza seu volume entre diferentes classes de dados [36].

Para tal, o SVM transforma os dados de entrada em um espaço de dimensão superior. Com isso, ele separa todos os pontos de dados a partir da origem utilizando uma hiperesfera. Os pontos que estiverem fora da hiperesfera são aqueles classificados como anomalias. Esse método também é chamado de *Support Vector Data Description* (SVDD).

O OCC utilizando esse método busca, então, utilizar um algoritmo de otimização que minimiza a função objetivo, descrita na equação 3.1, onde w é o vetor de pesos, $\phi(x)$ é uma função de mapeamento não-linear para um espaço de características de alta dimensão, ν é um parâmetro que controla o tamanho da região de suporte, n é o número de pontos de dados, ξ_i são variáveis de folga que medem a violação das restrições e ρ é a distância do hiperplano de separação até o centro da região normal.

$$\min_{w,\xi,\rho} \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \tag{3.1}$$

A equação em 3.1 pode ser detalhada da seguinte forma:

- A primeira parte, $\frac{1}{2}||w||^2$, é uma penalidade pela magnitude dos pesos w, minimizada para evitar *overfitting*.
- A segunda parte, $\frac{1}{\nu n}\sum_{i=1}^{n}\xi_i$, é a penalidade pelo número de pontos de dados fora da região normal e é minimizada para maximizar a abrangência da região normal.
- O parâmetro ν controla o *trade-off* entre a abrangência da região normal e o número de pontos de dados fora dela.

É importante ressaltar que essa função objetivo do OCC-SVM é restrita pela inequação 3.2, que impõe que os pontos de dados devem estar dentro da região normal com uma margem de pelo menos ξ_i.

$$w^T\phi(x_i) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n \tag{3.2}$$

A restrição $w^T\phi(x_i) \geq \rho - \xi_i$ é equivalente a dizer que x_i está dentro da região normal se sua projeção no espaço de características é maior ou igual a $\rho - \xi_i$. A restrição $\xi_i \geq 0$ garante que as variáveis de folga sejam positivas ou zero, e a restrição $\sum_{i=1}^{n}\xi_i \leq \nu n$ limita o número total de pontos de dados fora da região normal em νn.

Através da aplicação desses conceitos, e da parametrização inicial dos valores de ξ_i e ν, o algoritmo consegue então minimizar a hiperesfera para conter em seu interior apenas os que pertencem à classe principal, e insere os que não se enquadram nessa classificação fora do hiperplano. Ao considerar o cenário proposto por este trabalho, o agrupamento interior é considerado a classe padrão (com obstáculos), e os exemplos que não estão nessa densidade são classificados como anormais (sem obstáculos).

3.2 Técnicas Estatísticas

As técnicas estatísticas podem ser consideradas um substituto simples de *Machine Learning*, pois ambos os métodos buscam modelar relações entre variáveis e prever valores futuros [21]. Além disso, elas são geralmente mais simples e requerem menos recursos computacionais, o que as torna ideais para sistemas embarcados com limitações de potência e capacidade de armazenamento [37].

O uso da análise estatística é feito com base em séries estatísticas, por exemplo, séries temporais. Estas se caracterizam como um conjunto de dados que contêm informações sobre eventos que ocorrem ao longo do tempo [38]. Os modelos se utilizam de suas informações de tendência e sazonalidade para conseguirem prever valores futuros. Métodos como o ARIMA (*Autoregressive Integrated Moving Average*) e CUSUM (*Cummulative*

Sum) [39][40][41], consolidados na literatura, resultam em uma resposta para detecção de anomalia similar ao algoritmo de ML apresentado anteriormente, e terão suas aplicações detalhadas nas subseções a seguir.

3.2.1 *Autoregressive Integrated Moving Average* (ARIMA)

O modelo estatístico ARIMA é utilizado para prever séries temporais [42]. O algoritmo utiliza informações sobre a tendência e a sazonalidade passadas da série temporal para prever valores futuros, combinando três componentes principais [43]:

1. Autorregressiva: se refere à utilização dos valores anteriores da série temporal como variáveis explicativas

2. Média móvel: se refere à utilização da média dos valores anteriores como previsão

3. Diferenciação: se refere ao processo de diferenciação da série temporal para torná-la estacionária

É importante ressaltar que, para o uso com ARIMA, a série temporal precisa ser estacionária, isto é, sua média e variância são constantes ao longo do tempo [44]. Tal fato é importante, pois, caso contrário, o algoritmo não conseguirá capturar as características estáveis da série temporal, o que o impediria de prever com precisão valores futuros.

Através do teste de raiz unitária é possível determinar se uma série temporal é estacionária, sendo o procedimento elaborado por Dickey-Fuller um dos mais utilizados [45] [46]. Para a aplicação no cenário proposto pelo presente processo, a análise é feita com os *frames* do fluxo de dados contendo os dados de distância capturados na imagem. Isso requer, contudo, que o algoritmo seja previamente inicializado com um conjunto de dados, similar à categoria de algoritmo de ML supervisionado vista anteriormente.

Essa análise [47] está representada na equação 3.3, onde y_t é o valor observado na posição temporal t, α é uma constante relacionada ao valor médio esperado da série temporal, ϕ é o coeficiente de autorregressão, e ϵ_t é o erro de previsão no tempo t.

$$y_t = \alpha + \phi * y_{t-1} + \epsilon_t \tag{3.3}$$

Se o valor resultante em ϕ for menor que 1, a série temporal é estacionária, caso contrário, ela é não estacionária. No cenário proposto, classifica-se uma série temporal estacionária como aquela que possui *frames* sem obstáculos. Após essa verificação, é possível, então, realizar o cálculo do ARIMA, que está descrito na equação geral 3.4, e resumida na equação do modelo em 3.5.

$$(1 - \phi_1 L - \phi_2 L^2 - ... - \phi_p L^p)(1 - L)^d y_t = (1 + \theta_1 L + \theta_2 L^2 + ... + \theta_q L^q)\epsilon_t \tag{3.4}$$

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d y_t = (1 + \sum_{i=1}^{q} \theta_i L^i)\epsilon_t \tag{3.5}$$

As terminologias da equação 3.5 se descrevem como:

- L é o valor de diferenciação, ou defasagem, da série temporal, tal que $L^i y_t = y_{t-i}$;
- p é a ordem do termo autorregressivo;
- d é a ordem de diferenciação;
- q é a ordem do termo de média móvel;
- $\phi_1, ..., \phi_p$ são os coeficientes autorregressivos;
- $\theta_1, ..., \theta_q$ são os coeficientes de média móvel;
- ϵ_t é o erro aleatório em t.

A diferenciação da série temporal na equação 3.5 ocorre no resultado em $(1-L)^d y_t$, que é a aplicação da operação de diferenciação na série temporal y_t d vezes. Os coeficientes $\phi_1, ..., \phi_p$ e $\theta_1, ..., \theta_q$ são estimados a partir dos dados históricos da série temporal. Dessa forma, o modelo consegue, então, prever os valores futuros da série temporal.

Para aplicação no projeto, o algoritmo do ARIMA é utilizado para prever um próximo *frame* com base na sua série temporal. Através disso, é possível comparar com o *frame* capturado para identificar se ele é similar ao previsto. Caso não for, conclui-se que o mesmo é uma anomalia, sendo classificado como obstáculo ou não de acordo com o estado inicial de saída no qual o algoritmo foi iniciado. Um pseudocódigo genérico, baseado na aplicação realizada por [48], para uso do algoritmo pode ser escrito da seguinte maneira:

1. Carregar série temporal dos dados
2. Verificar se a série temporal é estacionária usando teste de raiz unitária
    a) Caso não for, reajustar a série temporal realizando nova captura de *frames*
3. Ajustar os valores iniciais de constantes do algoritmo
4. Realizar previsão usando o modelo ajustado
5. Verificar a qualidade da previsão usando métricas de erro

6. Repetir os passos 3-5 para diferentes valores de constantes iniciais e até ajuste dos melhores valores conforme as métricas de erro

7. Realizar previsões de acordo com o modelo final ajustado e comparar o resultado previsto com o *frame* capturado

Através desses passos é possível garantir o melhor ajuste das constantes do algoritmo e, então, prosseguir para a análise de obstáculos nos *frames* capturados através da comparação entre o previsto e capturado para definir se há ou não a presença dos mesmos na cena.

3.2.2 *Cumulative Sum* (CUSUM)

O método CUSUM é um modelo estatístico que detecta mudanças em uma série temporal [41]. Ele é uma alternativa com aplicabilidade simples para detecção de mudanças repentinas em grandes quantidades de dados. Seu processo é baseado na comparação da soma dos desvios entre a observação atual e a média esperada, com a soma dos desvios anteriores. Se a soma dos desvios atuais ultrapassar um limite predefinido, é considerado que houve uma mudança na série temporal [49].

Para isso, utiliza-se de duas terminologias, CUSUM positivo (C_+) e negativo (C_-). O primeiro é utilizado para detectar aumentos na série temporal, e o segundo diminuições. Da mesma forma do que no ARIMA, este algoritmo também precisa ser inicializados com valores de referência, sendo eles o limiar de detecção e média esperada. Assim, para cada *frame* capturado e analisado, o desvio entre médias é calculado e adicionado ao CUSUM positivo ou negativo. Se algum dos dois ultrapassar o limiar de detecção, é considerado que houve anomalia.

Um pseudocódigo [41] que descreve a aplicação do algoritmo na resolução do problema proposto por este projeto pode ser escrito como:

1. Inicializar os valores assumidos de média esperada (μ) e limite de detecção (h).

2. Inicializar as variáveis CUSUM positivo (C_+) e CUSUM negativo (C_-) como 0.

3. Para cada novo *frame* analisado:

   a) Calcular o desvio $d = x - \mu$

   b) $C_+ = max(0, C_+ + d)$

   c) $C_- = min(0, C_- + d)$

   d) Se $C_+ >$ h ou $C_- <$ -h, houve mudança na série temporal, logo, indicar a presença de obstáculo no *frame*

É importante ressaltar que os valores de média esperada e limite de detecção são, geralmente, estabelecidos com base em conhecimentos prévios sobre o processo. Além disso, o limite de detecção deve ser estabelecido com base no nível de sensibilidade desejado para a detecção de mudanças. Essas definições podem ser feitas por experimentação dos resultados obtidos da análise dos *frames* sem obstáculos capturados.

4 METODOLOGIA DE AVALIAÇÃO

A avaliação do resultado de um algoritmo é uma etapa importante para determinar qual, dentre as opções, será o escolhido para aplicação prática. Para analisar o resultado obtido dos algoritmos, e compreender seu impacto no sistema, se faz necessário uma metodologia que estude a relação do *software* com o *hardware*. Com essa finalidade, o estudo de *Four-Way Performance Trade-Off* [50] é o que mais se aproxima da comparação pretendida por este projeto. Essa análise consiste em obter os dados de desempenho preditivo, consumo de energia, consumo de memória, e custo temporal de um algoritmo.

O desempenho preditivo do algoritmo pode ser considerado o aspecto mais importante de avaliação, quando considerado o cenário de classificação de obstáculos pelo AGV. Isso pois, caso haja uma classificação errônea, acidentes poderiam acontecer ocasionados por esse resultado. Para tal análise, é considerada a acurácia do algoritmo.

Uma das formas mais consolidadas na literatura para cálculo da acurácia é através da matriz de confusão [51]. Ela é uma tabela usada para avaliar o desempenho de um modelo de classificação, e mostra o número de vezes que cada classe foi classificada corretamente ou incorretamente pelo modelo. A matriz de confusão é composta por quatro elementos principais:

1. Verdadeiro Positivo (VP): quando uma amostra é classificada como positiva e ela realmente é positiva.

2. Falso Positivo (FP): quando uma amostra é classificada como positiva e ela realmente é negativa.

3. Verdadeiro Negativo (VN): quando uma amostra é classificada como negativa e ela realmente é negativa.

4. Falso Negativo (FN): quando uma amostra é classificada como negativa e ela realmente é positiva.

A partir desses elementos, é possível calcular diversas métricas de avaliação do modelo, como acurácia. Ela é uma métrica que indica a proporção de amostras classificadas corretamente pelo modelo em relação ao total de amostras, conforme exibido na equação 4.1.

$$acurácia = \frac{VP + VN}{VP + FP + VN + FN} \tag{4.1}$$

O consumo de energia é um fator importante quando se considera que um AGV opera utilizando energia proveniente de uma bateria. Nesse contexto, o uso excessivo e descontrolado de energia pode levar a uma rápida descarga da bateria, reduzindo sua capacidade de operação. Portanto, é essencial que o impacto energético do algoritmo operando não seja alto, a fim de maximizar o tempo de uso do robô.

Apesar de sua relevância, o sistema de visão embarcada não é o único a estar executando em um AGV, sendo que outros sistemas, como movimentação e conectividade, executam em paralelo para garantir seu funcionamento. Dessa forma, o algoritmo de classificação de obstáculos deve desempenhar seu papel utilizando apenas o necessário para sua execução, sem ocupar o espaço de outras aplicações. Para isso, seu consumo de memória deve ser considerado para evitar problemas de paralisação no processamento dos dados.

Por fim, na classificação de fluxo de dados, o custo temporal é um fator influente no resultado das métricas anteriores. Isso porque, diferente dos algoritmos de ML tradicionais, ao se trabalhar com esse tipo de classificação as etapas de treino e predição ocorrem em paralelo. Ao se aumentar o tempo de treino, o algoritmo pode ter um desempenho preditivo melhor, contudo também aumenta o consumo de energia e memória requeridos para realizar essa tarefa. Assim, sua análise é importante para entender o balanço requerido para balancear o algoritmo nas necessidades almejadas.

Através desses parâmetros é possível, então, não apenas comparar qual algoritmo possui o melhor desempenho geral, mas também compreender qual é o impacto no consumo dos recursos de *hardware* pelo *software*.

5 DESENVOLVIMENTO

Nesta seção estão descritos os procedimentos utilizados para captura, pré-processamento e classificação dos dados, bem como os resultados obtidos após a execução e análise de cada um dos algoritmos.

5.1 Base de dados

Para compor a base de dados, objetivou-se a captura dos *frames* em ambiente relevante relacionado à atividade operacional do AGV, utilizando *hardware* com características próximas às ideais propostas na fundamentação teórica. Desta forma, a aquisição dos dados ocorreu em um galpão industrial, através da câmera Intel Real-Sense D435i [9][52], capaz de captar tanto imagens RGB, como também imagens com informação de profundidade. As especificações técnicas desse modelo em relação às câmeras de profundidade e RGB se encontram na Tabela 1 e 2, respectivamente.

Tabela 1 – Características da câmera de profundidade. (Fonte: Intel Real-Sense [1])

Tecnologia de Captura	Estereoscopia
Range ideal	.3 até 3m
Resolução	Até 1280 x 720
FPS	Até 90 FPS
Acurácia	<2% em 2m

Tabela 2 – Características da câmera RGB. (Fonte: Intel Real-Sense [1])

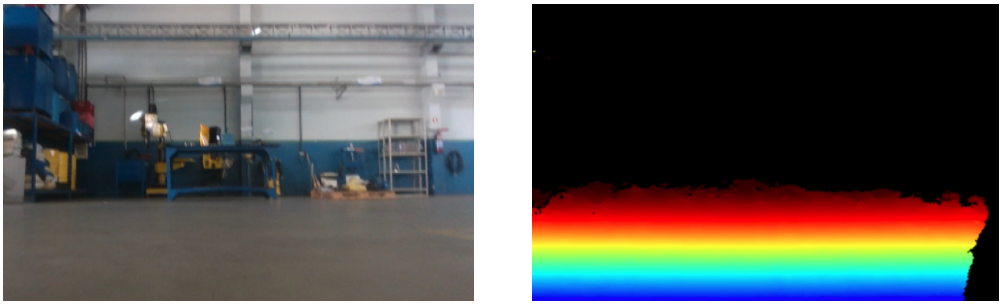
Resolução do frame	Até 1920 x 1080
Resolução do sensor RGB	2 MP
FPS	Até 30 FPS

Com o intuito de padronizar a análise dos algoritmos selecionados, foram gravados vídeos de rotas diferentes percorridas por um AGV em ambientes com e sem obstáculos, para garantir que seja possível comparar diretamente o resultado dos testes entre eles. Cada *frame* dos vídeos foi classificado como *"com obstáculo"* e *"sem obstáculo"*, de acordo com a seguinte definição:

- *Frame* sem obstáculo: captura onde não há presença de impedimentos no caminho do robô, ou que os obstáculos que surjam não sirvam de barreira para sua movimentação.

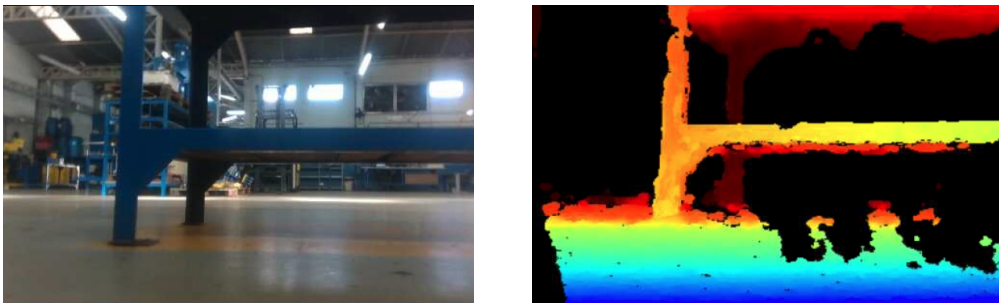
- *Frame* com obstáculo: captura onde há presença de itens que obstruam o caminho do robô, impedindo sua movimentação.

Essas descrições estão exemplificadas, respectivamente, nas Figuras 9 e 10. Na primeira, é possível observar, através da banda de profundidade, que o *frame* capturado não possui obstáculos, e que colunas e demais itens ao fundo, visíveis na banda RGB, não impõem um impedimento direto ao robô. Já na Figura 10, é possível observar que a bancada vista na imagem se mostra como uma barreira direta ao robô, considerada o aumento evidente de dados visualizados na síntese da banda de profundidade.



(a) Imagem RGB (b) Imagem de Profundidade

Figura 9 – Exemplo de *frame* sem obstáculo



(a) Imagem RGB (b) Imagem de Profundidade

Figura 10 – Exemplo de *Frame* com obstáculo

Cada uma das rotas percorridas pelo robô foram gravadas em 4 vídeos de 2 minutos e 30 segundos, à 10 *frames* por segundo (FPS), compondo no final 1500 *frames* por vídeo. Os arquivos de gravação são compostos das bandas RGB e da banda contendo os dados de profundidade, com resolução de 1920 × 1080 e 1280 × 720 *pixels*, respectivamente. Apenas os *frames* de profundidade são considerados para a classificação dos dados, sendo a banda RGB utilizada apenas como referência para análise posterior. A proporção de *frames* com e sem obstáculos está presente na Tabela 3.

Esses vídeos, por sua vez, são apresentados para os algoritmos de forma a simular sua operação em tempo real. Fora eles, também foram capturados outros 120 frames com obstáculos utilizados na etapa de pré-treino dos algoritmos, que está detalhada na seção 5.3.

Tabela 3 – Quantidade e relação percentual ao total de *frames* em cada gravação

	***Frames* com obstáculo**	%	***Frames* sem obstáculo**	%
Gravação 1	895	59,6	605	30,4
Gravação 2	1019	67,9	481	32,1
Gravação 3	326	21,7	1174	78,3
Gravação 4	912	60,8	588	39,2

5.2 *Hardware* e *frameworks* utilizados

Para processamento dos dados adquiridos, o hardware escolhido foi o Nvidia Jetson Nano [52][53], com 2 GB de memória RAM e GPU dedicada, e um *thermal design power*[1] (TDP) de 10W. Neste hardware, foi acoplado na entrada de alimentação um voltímetro e amperímetro digital, que forneceram os dados necessários para o cálculo de consumo de energia.

A linguagem de programação Python [54] em sua versão 3.9.6 foi utilizada para desenvolvimento e aplicação dos algoritmos de captura, pré-processamento e análise dos dados. Dessa forma, as principais bibliotecas utilizadas foram::

- Intel RealSense SDK 2.0[2]: utilizada na captura e pré-processamentos dos dados
- scikit-learn[55]: utilizada aplicação do algoritmo de OCC através da classe One-ClassSVM, para *clusterização* e classificação dos *frames*. Além disso, a biblioteca foi configurada para utilizar aceleração via GPU da Nvidia.
- statsmodels[56]: aplicação do algoritmo de ARIMA

[1] Medida de referência que indica a quantidade máxima de calor que um processador (CPU) ou placa de vídeo (GPU) gera e precisa dissipar para operar dentro dos limites de temperatura especificados.
[2] https://www.intelrealsense.com/sdk-2/

- PyCUSUM[57]: aplicação do algoritmo de CUSUM

- matplotlib[58] e opencv2[59]: para visualização de gráficos e síntese de imagens, respectivamente

- psutil: para cálculo do consumo de memória do algoritmo

5.3 Classificação dos Dados

Os algoritmos de OCC, ARIMA e CUSUM possuem peculiaridades no que tange a forma com que cada um lida com a classificação. Contudo, para garantir o fator comparativo entre eles, a entrada dos dados nos algoritmos foi padronizada para simular as condições reais nas quais eles teriam que realizar para classificar o fluxo de dados em tempo real. Uma das etapas adicionadas para garantir isso é o pré-treino, onde o algoritmo é treinado com *frames* contendo obstáculos. Isso é feito para que ele possa realizar a classificação desde o primeiro *frame* analisado, e essa etapa recebe o nome de *"fase de aquecimento"*.

Após o treino, inicia-se a fase *"online"*, na qual o algoritmo começa a ser alimentado pelo vídeo gravado e, a cada *frame* lido, é realizado o pre-processamento dos dados e, após isso, realizada sua classificação. O resultado é, então, comparado com o *label* do *frame* inserido durante a etapa de classificação prévia da base de dados para gerar a acurácia. A cada *frame* avaliado como obstáculo, o mesmo é treinado novamente no modelo para adaptá-lo ao desvio de conceito que pode acontecer na classificação de fluxo de dados.

A arquitetura utilizada para desenvolver esse fluxo, representada na Figura 11, é uma forma genérica para adaptar a entrada e saída dos dados nos algoritmos selecionados. As subseções a seguir oferecem informações mais aprofundadas como cada algoritmo foi implementado para realizar a classificação de fluxo de dados de acordo com suas características.

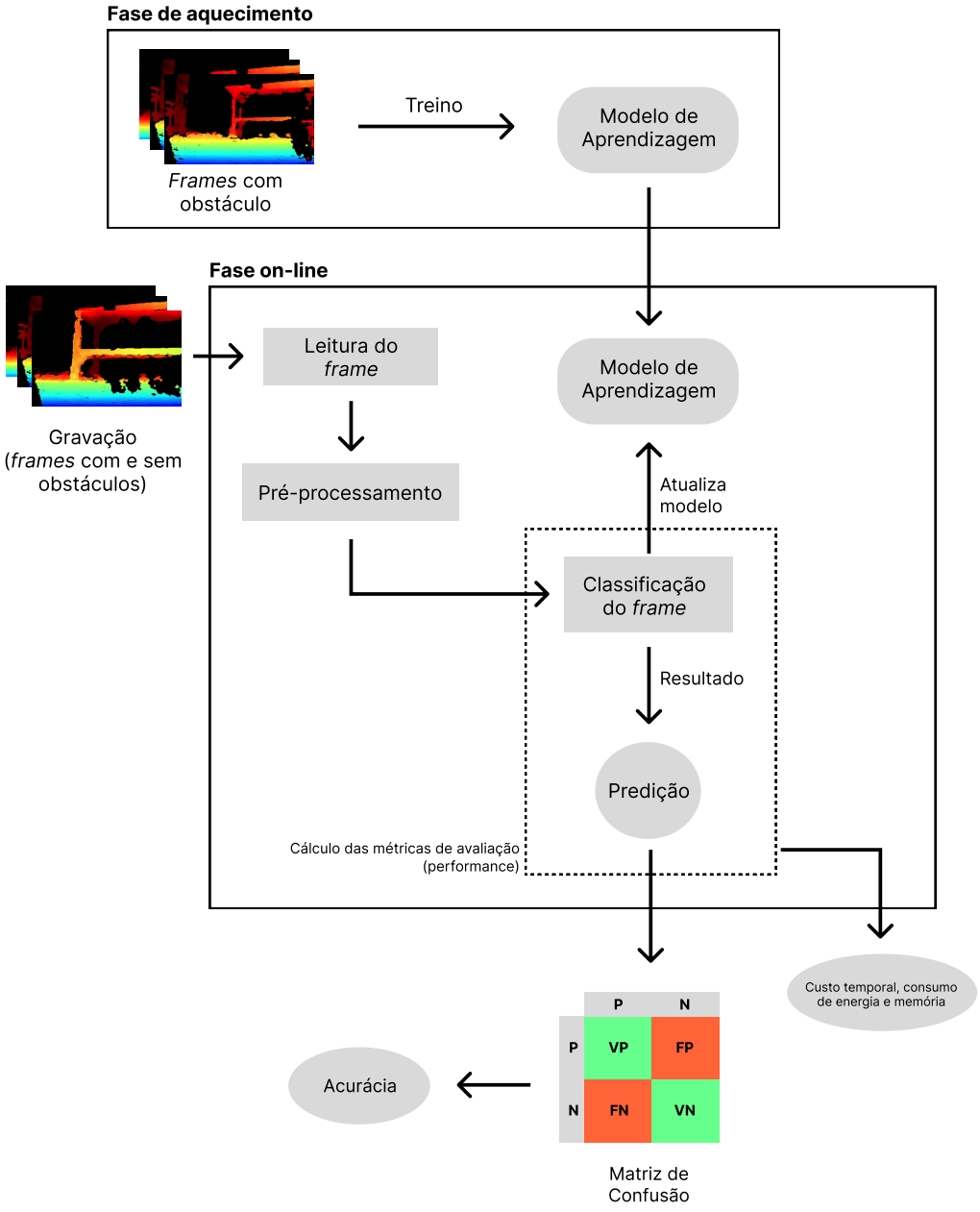


Figura 11 – Fluxograma genérico para aplicação do algoritmo de classificação

5.3.1 Utilizando OCC

Para uso do algoritmo de *One Class Clsasification*, a biblioteca *scikit-learn* foi a escolhida. A função para classificação com OCC foi inicializada com o *kernel* em função de base radial (rbf), conforme sugerido pela biblioteca, e com o parâmetro γ inicializado com valor 0.1. Este último foi ajustado de forma arbitrária durante a etapa de ensaios com o algoritmo de forma a atingir maiores níveis acurácia.

Durante a etapa de pré-treino, os *frames* foram introduzidos no modelo utilizando o método *fit* da biblioteca. Nessa aplicação, cada *pixel* do *frame* é considerado uma *feature* do modelo, e o mesmo realiza a clusterização com base nos valores de distância obtidos em cada cena. Após o treino, o algoritmo começa a ser testado, *frame* a *frame*, utilizando o método *predict*.

5.3.2 Utilizando ARIMA

A biblioteca *statsmodels* possui uma implementação direta do método ARIMA, similar à presente na *scikit-learn*. Os componentes iniciais dos valores de ordem autorregressivo, média móvel e diferenciação foram inicializados como 1, 1, e 0, respectivamente, conforme sugerido pela biblioteca. Além disso, a biblioteca também implementa a análise do teste de raiz unitária de Dickey-Fuller para verificar se a série temporal indicada é estacionária ou não.

Como o ARIMA realiza sua predição conforme a série temporal indicada, o algoritmo é inicializado com os *frames* contendo obstáculos utilizando o método *fit* da biblioteca. O treino acontece em cada *pixel* do *frame*, para que o algoritmo seja capaz de montar uma imagem predizendo como ela seria se houvesse um obstáculo. Após o treino, começa a etapa de predição, onde cada *frame* lido é comparado com o previsto, e com base nisso é feita a classificação do *frame* utilizando a função *forecast*.

5.3.3 Utilizando CUSUM

O algoritmo de CUSUM disponível na biblioteca *PyCUSUM* possibilita o uso direto dos cálculos desse modelo estatístico. Primeiramente, ele é inicializado com os valores de limiar de detecção e média esperada, sendo eles para essa aplicação 500 e 100, respectivamente. Assim como sugerido durante investigação na literatura, esses valores foram ajustados arbitrariamente, durante a etapa de ensaios do algoritmo com a base de dados, para maximizar a acurácia do algoritmo.

Similar ao método do ARIMA, o CUSUM também avalia os *frames pixel* a *pixel*, através da função *update* para realizar a análise de desvio na série temporal. Porém, diferente das bibliotecas que implementam OCC e ARIMA, esta não possui método de treino, sendo detecção feita de acordo com os últimos *frames* armazenados na série temporal.

Nessa implementação, o algoritmo foi adaptado para iniciar com os primeiros *frames* que contêm obstáculos, definindo assim o estado inicial de saída. A cada novo *frame* lido e analisado, o *frame* mais antigo é descartado. O *buffer* que armazena a série temporal tem um tamanho de 120 *frames*, o que corresponde a 5 segundos, determinado pelo tempo de resposta do robô e limitações de recursos computacionais. Dessa forma, cada vez que uma anomalia é detectada, o estado de saída do algoritmo é alterado, permitindo a classificação dos *frames*.

6 ANÁLISE E DISCUSSÃO DOS RESULTADOS

6.1 Apresentação dos Resultados

Os algoritmos foram executados 3 vezes em cada um dos 4 vídeos gravados que compõem a base de dados. Para calcular o resultado de cada uma das execuções, foi feita uma média entre os valores obtidos na matriz de confusão, desempenho preditivo, consumo de energia, consumo de memória e tempo de execução. Todos os valores mensurados foram obtidos apenas enquanto o algoritmo de classificação estava sendo executado, descartando etapas anteriores como, por exemplo, as de pré-treino.

Para cálculo do consumo de memória, foi utilizada a biblioteca nativa *psutil*, onde a cada um segundo era feita uma coleta do valor de memória consumido pelo processo do algoritmo de classificação, sendo o valor final uma média do consumo durante o intervalo de tempo de execução. O custo temporal foi adquirido utilizando a implementação nativa de coleta de *timestamp* do Python através da biblioteca *time*, com o cálculo médio de processamento de cada *frame*. Tendo em vista a captura de 10 *frames* por segundo, o custo deve ficar abaixo de $\frac{1}{10}s$ para ser possível dar uma resposta em quase tempo real.

Já o consumo de energia foi obtido através da leitura em tempo real dos dados do voltímetro e amperímetro acoplados na entrada de energia do Jetson Nano. É importante ressaltar que, apenas com esses valores, não é possível inferir o consumo de energia do dispositivo. Esses valores combinados resultam na potência elétrica, a qual é a multiplicação entre tensão elétrica (V, em *volts*) e a corrente elétrica (I, em ampere) que flui através do dispositivo, equação esta descrita em 6.1.

$$P = V \cdot I \tag{6.1}$$

A energia consumida é, então, a potência consumida pelo dispositivo em um intervalo de tempo, conforme equação 6.2. O algoritmo de teste realiza o cálculo dessa fórmula ao final do término da etapa de classificação.

$$E = P \cdot t \tag{6.2}$$

A matriz de confusão resultante da classificação está exibida nas Tabelas 4, 6, 8 e 10, que mostram o percentual dos *frames* em cada categoria em relação à quantidade total de *frames*. Os resultados dos testes de cada gravação se encontram nas Tabelas 5, 7, 9 e 11, e o resultado da média final entre as execuções está presente na Tabela 12. Para a

acurácia, quanto mais perto de 1, melhor. Para os valores de tempo, energia e memória, quanto menor, melhor.

Tabela 4 – Matriz de confusão na gravação 1

OCC			**ARIMA**			**CUSUM**		
	P	**N**		**P**	**N**		**P**	**N**
P	55,4%	13,3%	**P**	57,3%	8,1%	**P**	58,1%	8,3%
N	4,3%	27%	**N**	2,4%	32,%	**N**	1,5%	32,1%

Tabela 5 – Resultado após execução na gravação 1

	Acurácia	**Tempo (s)/*Frame***	**Memória (MB)**	**Energia (J)**
OCC	0,824	0,0116	740	233,77
ARIMA	0,895	0,0429	1148	459,89
CUSUM	0,902	0,0369	724	392,30

Tabela 6 – Matriz de confusão na gravação 2

OCC			**ARIMA**			**CUSUM**		
	P	**N**		**P**	**N**		**P**	**N**
P	65,9%	9,3%	**P**	67,4%	7,1%	**P**	67,3%	7,2%
N	2%	22,8%	**N**	0,5%	25%	**N**	0,6%	24,9%

Tabela 7 – Resultado após execução na gravação 2

	Acurácia	**Tempo (s)/*Frame***	**Memória (MB)**	**Energia (J)**
OCC	0,887	0,0172	752	379,32
ARIMA	0,924	0,0473	1163	560,03
CUSUM	0,922	0,0384	743	414,59

Tabela 8 – Matriz de confusão na gravação 3

OCC			**ARIMA**			**CUSUM**		
	P	**N**		**P**	**N**		**P**	**N**
P	19,5%	18,7%	**P**	20,1%	17,2%	**P**	21,2%	14,3%
N	2,3%	59,6%	**N**	1,7%	61,1%	**N**	0,5%	64%

Tabela 9 – Resultado após execução na gravação 3

	Acurácia	**Tempo (s)/*Frame***	**Memória (MB)**	**Energia (J)**
OCC	0,791	0,0123	752	211,03
ARIMA	0,811	0,0422	1163	475,19
CUSUM	0,852	0,0357	743	414,71

Tabela 10 – Matriz de confusão na gravação 4

OCC			**ARIMA**			**CUSUM**		
	P	**N**		**P**	**N**		**P**	**N**
P	58,1%	14,5%	**P**	60,1%	9,1%	**P**	60,5%	8,8%
N	2,7%	24,7%	**N**	0,7%	30,1%	**N**	0,3%	30,4%

Tabela 11 – Resultado após execução na gravação 4

	Acurácia	**Tempo (s)/*Frame***	**Memória (MB)**	**Energia (J)**
OCC	0,828	0,0125	740	245,63
ARIMA	0,901	0,0423	1148	502,25
CUSUM	0,909	0,0405	724	448,11

Tabela 12 – Média final dos resultados

	Acurácia	**Tempo (s)/*Frame***	**Memória (MB)**	**Energia (J)**
OCC	0,833	0,0134	746	267,44
ARIMA	0,883	0,0437	1.156	499,34
CUSUM	0,896	0,0379	734	417,43

6.2 Análise dos resultados

Os resultados obtidos através dos testes revelaram informações importantes sobre como os algoritmos se desempenharam em relação à base de dados testada. Além disso, a análise individual dos parâmetros mensurados pode proporcionar interpretações diferentes do que quando se analisado junto ao coletivo.

Através dos valores de acurácia, é notável que o ARIMA e CUSUM tiveram um desempenho superior em relação ao OCC na detecção de obstáculos. Em uma análise específica das gravações, houve uma redução não-significativa na acurácia de todos os algoritmos da Gravação 3, que apresentava mais *frames* sem obstáculos do que com obstáculos. Isso pode ser explicado pelo fato de que os algoritmos foram treinados inicialmente considerando apenas *frames* com obstáculos, o que pode ter limitado sua capacidade de detecção nos ambientes sem obstáculos, gerando um aumento de falsos-positivos, conforme visto na matriz de confusão na Tabela 8.

Esse fato também é corroborado quando se analisa os resultados da Gravação 2, que apresentava uma proporção maior de *frames* com obstáculos, onde os algoritmos tiveram um desempenho melhor. Esse resultado aproximou a acurácia OCC dos algoritmos de ARIMA e CUSUM, que se mantiveram estáveis, indicando que o OCC está melhor treinado para classificar *frames* com obstáculos. Na média final, o CUSUM apresentou um desempenho melhor do que o ARIMA, e ambos superaram o OCC. Dessa forma, isso

indica que o CUSUM e ARIMA possuem um melhor desempenho teórico na detecção de anomalias dentro do cenário proposto. A menor acurácia do OCC em relação aos outros pode ser explicada se considerado que as informações contidas em um frame de profundidade podem não ser suficientes para uma diferenciação fina quando se trata do OCC.

Quando se analisa o tempo de processamento, em segundos, por *frame*, todos apresentaram desempenho satisfatório, ficando abaixo de 0,1 segundos, que seria o valor ideal para gerar uma saída em quase tempo real na configuração proposta. É interessante ressaltar o menor tempo de processamento do OCC em relação ao ARIMA e CUSUM. Isso ocorre por conta da aceleração proporcionada pelo uso da GPU no algoritmo de OCC habilitada junto ao uso da biblioteca *scikit-learn*, que não estava presente nos outros dois algoritmos. Este menor tempo de processamento teve impacto direto nos valores finais de consumo de energia, onde OCC necessitou de cerca da metade da energia consumida pelo ARIMA e CUSUM para processar os mesmos *frames*.

Ainda na análise de custo temporal, o algoritmo ARIMA foi o que demorou mais para realizar o processamento de um *frame*, tendo um custo de cerca de 3.2x maior em relação ao OCC, e 1.15x maior em relação ao CUSUM. Esse tempo superior de processamento pode estar relacionado ao consumo de memória onde, enquanto OCC e CUSUM obtiveram um consumo médio similar, o ARIMA utilizou cerca de 1.56x mais memória para realizar o processamento dos *frames*. Tanto o tempo superior de processamento como o consumo maior de memória do ARIMA não estavam previstos em literatura, sendo necessária uma investigação mais detalhada para compreender a relação entre esses resultados.

Dessa forma, é notável que, através da análise dos resultados, a escolha do *"melhor algoritmo"* depende muito da necessidade do sistema e respostas esperadas pelo projetista do mesmo. Se considerado apenas a acurácia como fator determinante, logicamente o algoritmo de CUSUM seria o escolhido, por possuir o melhor resultado. Porém, quando se considera que o algoritmo terá que ser executado em um sistema embarcado, onde as limitações de processamento e consumo de energia são determinantes, o projetista do sistema poderia escolher fazer um *trade-off* na acurácia para obter um melhor custo temporal e menor consumo de energia, escolhendo, então, o OCC.

7 CONCLUSÃO E TRABALHOS FUTUROS

A área de computação é marcada por momentos em que determinadas técnicas ou abordagens se tornam mais populares que outras, geralmente impulsionadas por avanços tecnológicos, demandas de mercado ou mesmo modismos. O uso massivo de técnicas de aprendizado de máquina, como o *Machine Learning*, é um exemplo disso. No entanto, é importante lembrar que nem todos os problemas exigem o uso dessas técnicas, e muitas vezes soluções mais simples e eficazes podem ser encontradas fora do *hype* tecnológico.

Nesse sentido, é compreensível que desenvolvedores tentem usar as soluções do momento para resolver quaisquer problemas que apareçam. Contudo, essas técnicas não são a solução para todos os problemas e é preciso avaliar cuidadosamente qual é a técnica mais adequada para cada situação. Isso se torna mais evidente quando se trata da aplicação em um sistema embarcado, que envolve a avaliação de diferentes fatores, como a disponibilidade de dados, a complexidade do problema, as restrições de processamento e energia, entre outros.

Este trabalho mostra que, quando aplicado no cenário proposto, o uso de técnicas de ML pode ser considerado *overkilling*. Tal conclusão se revela quando os algoritmos de ARIMA e CUSUM, que são algoritmos de detecção de anomalias mais simples, obtiveram resultados de acurácia superiores ao OCC nos testes realizados. Isso sugere que, em algumas situações, técnicas mais simples e diretas podem ser mais adequadas do que abordagens mais complexas, mesmo que estas estejam em alta no momento.

Em conclusão, espera-se, então, que os dados coletados no presente projeto sirvam como ponto de partida para a análise aprofundada e individual dos algoritmos de visão computacional embarcada e classificação de fluxo de dados. Isso pode permitir uma melhor compreensão das vantagens e limitações de cada técnica e auxiliar na escolha mais adequada para cada situação. Além disso, o estudo pode contribuir como base para futuros projetos que desejarem otimizar esses algoritmos para tais aplicações, buscando maximizar a acurácia e minimizar o consumo de recursos de *hardware*.

Com base nos resultados apresentados neste trabalho, uma possível linha de pesquisa futura seria o aprimoramento dos algoritmos selecionados de técnicas estatísticas buscando melhor performance e consumo de recursos. Além disso, uma abordagem interessante seria a possibilidade de embarcar esses algoritmos em *chips* dedicados, o que pode aumentar significativamente o desempenho e reduzir o consumo de energia em relação a soluções baseadas em software. Dessa forma, trabalhos futuros podem contribuir para o desenvolvimento de soluções mais eficientes e robustas para aplicações em sistemas embarcados, como aqueles aplicados nos AGVs, e outros dispositivos IoT.

REFERÊNCIAS

[1] INTEL. *Intel® RealSense™ D400 Series Product Family*. [S.l.], 2019. Rev. 5.

[2] INDUSTRY, B. N. C. of. Industry 4.0: A new challenge for brazilian industry. *CNI Indicators*, v. 17, n. 2, p. 1–16, 2016.

[3] KANG, J. et al. An application of parameter extraction for agv navigation based on computer vision. In: IEEE. *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. [S.l.], 2013. p. 622–626.

[4] BEECK, K. V. et al. Real-time embedded computer vision on uavs. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2018. p. 3–10.

[5] SCALCO, R. *Introdução a computação gráfica*. [S.l.]: Roberto Scalco, 2005.

[6] CARDOSO, A.; LAMOUNIER, E. A. *Computação Gráfica*. 2007.

[7] GONZALEZ, R. C.; WOODS, R. E. *Processamento de imagens digitais*. [S.l.]: Editora Blucher, 2000.

[8] SCURI, A. E. Fundamentos da imagem digital. *Pontifícia Universidade Católica do Rio de Janeiro*, p. 13, 1999.

[9] PIRES, M. A. *Natural navigation solutions for amrs and agvs using depth cameras*. Tese (Doutorado) — Universidade de Coimbra, 2021.

[10] SINZ, F. H. et al. Learning depth from stereo. In: SPRINGER. *Joint Pattern Recognition Symposium*. [S.l.], 2004. p. 245–252.

[11] FORSYTH, D.; PONCE, J. *Computer vision: A modern approach*. [S.l.]: Prentice hall, 2011.

[12] LIU, S. et al. Towards industrial scenario lane detection: Vision-based agv navigation methods. In: *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*. [S.l.: s.n.], 2020. p. 1101–1106.

[13] KISACANIN, B.; BHATTACHARYYA, S. S.; CHAI, S. *Embedded computer vision*. [S.l.]: Springer Science & Business Media, 2008.

[14] OWENS, J. D. et al. Gpu computing. *Proceedings of the IEEE*, IEEE, v. 96, n. 5, p. 879–899, 2008.

[15] ABE, Y. et al. Power and performance analysis of {GPU-Accelerated} systems. In: *2012 Workshop on Power-Aware Computing and Systems (HotPower 12)*. [S.l.: s.n.], 2012.

[16] GOMES, H. M. et al. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 50, n. 2, p. 1–36, 2017.

[17] BRZEZINSKI, D.; STEFANOWSKI, J. Stream classification. In: _____. [S.l.: s.n.], 2016.

[18] LIMA, M. C. de et al. A comprehensive analysis of the diverse aspects inherent to image data stream classification. *Knowledge and Information Systems*, Springer, v. 64, n. 8, p. 2215–2238, 2022.

[19] WANKHADE, K. K.; DONGRE, S. S.; JONDHALE, K. C. Data stream classification: a review. *Iran Journal of Computer Science*, Springer, v. 3, p. 239–260, 2020.

[20] Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. Moa: Massive online analysis. *Journal of Machine Learning Research*, n. 11, p. 1601–1604, 2010.

[21] GABER, M. M.; ZASLAVSKY, A.; KRISHNASWAMY, S. Mining data streams: A review. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 34, n. 2, p. 18–26, jun. 2005. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/1083784.1083789>.

[22] JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015.

[23] MITCHELL, T. M.; MITCHELL, T. M. *Machine learning*. [S.l.]: McGraw-hill New York, 1997. v. 1.

[24] RUSSELL, S. J. *Artificial intelligence a modern approach*. [S.l.]: Pearson Education, Inc., 2010.

[25] AGGARWAL, C. C. Data classification. In: SPRINGER. *Data mining*. [S.l.], 2015. p. 285–344.

[26] MAHESH, B. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, v. 9, p. 381–386, 2020.

[27] SINGH, A.; THAKUR, N.; SHARMA, A. A review of supervised machine learning algorithms. In: IEEE. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. [S.l.], 2016. p. 1310–1315.

[28] MOYA, M. M.; HUSH, D. R. Network constraints and multi-objective optimization for one-class classification. *Neural networks*, Elsevier, v. 9, n. 3, p. 463–474, 1996.

[29] KHAN, S. S.; MADDEN, M. G. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, Cambridge University Press, v. 29, n. 3, p. 345–374, 2014.

[30] ARABIE, P.; HUBERT, L.; SOETE, G. D. *Clustering and classification*. [S.l.]: World Scientific, 1996.

[31] CABRAL, G. G.; OLIVEIRA, A. L.; CAHÚ, C. B. Combining nearest neighbor data description and structural risk minimization for one-class classification. *Neural Computing and Applications*, Springer, v. 18, n. 2, p. 175–183, 2009.

[32] ROKACH, L.; MAIMON, O. Clustering methods. In: *Data mining and knowledge discovery handbook*. [S.l.]: Springer, 2005. p. 321–352.

[33] MADHULATHA, T. S. An overview on clustering methods. *arXiv preprint arXiv:1205.1117*, 2012.

[34] LIKAS, A.; VLASSIS, N.; VERBEEK, J. J. The global k-means clustering algorithm. *Pattern recognition*, Elsevier, v. 36, n. 2, p. 451–461, 2003.

[35] NOUMIR, Z.; HONEINE, P.; RICHARD, C. On simple one-class classification methods. In: IEEE. *2012 IEEE International Symposium on Information Theory Proceedings*. [S.l.], 2012. p. 2022–2026.

[36] TAX, D. M.; DUIN, R. P. Support vector data description. *Machine learning*, Springer, v. 54, p. 45–66, 2004.

[37] PALIWAL, M.; KUMAR, U. A. Neural networks and statistical techniques: A review of applications. *Expert systems with applications*, Elsevier, v. 36, n. 1, p. 2–17, 2009.

[38] WOOLDRIDGE, J. M. *Introductory econometrics: A modern approach*. [S.l.]: Cengage learning, 2015.

[39] HAMILTON, J. D. *Time series analysis*. [S.l.]: Princeton university press, 2020.

[40] MONDAL, P.; SHIT, L.; GOSWAMI, S. Study of effectiveness of time series modeling (arima) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications*, Academy & Industry Research Collaboration Center (AIRCC), v. 4, n. 2, p. 13, 2014.

[41] GRANJON, P. The cusum algorithm-a small review. 2013.

[42] HYNDMAN, R. J.; ATHANASOPOULOS, G. *Forecasting: principles and practice*. [S.l.]: OTexts, 2018.

[43] BOX, G. E. et al. *Time series analysis: forecasting and control*. [S.l.]: John Wiley & Sons, 2015.

[44] NEWBOLD, P. Arima model building and the time series analysis approach to forecasting. *Journal of forecasting*, Wiley Online Library, v. 2, n. 1, p. 23–35, 1983.

[45] AGIAKLOGLOU, C.; NEWBOLD, P. Empirical evidence on dickey-fuller-type tests. *Journal of Time Series Analysis*, Wiley Online Library, v. 13, n. 6, p. 471–483, 1992.

[46] NAU, R. The mathematical structure of arima models. *Duke University Online Article*, v. 1, n. 1, p. 1–8, 2014.

[47] DICKEY, D. A.; FULLER, W. A. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association*, Taylor & Francis, v. 74, n. 366a, p. 427–431, 1979.

[48] ALBERG, D.; LAST, M. Short-term load forecasting in smart meters with sliding window-based arima algorithms. *Vietnam Journal of Computer Science*, Springer, v. 5, n. 3, p. 241–249, 2018.

[49] RYAN, T. P. *Statistical methods for quality improvement*. [S.l.]: John Wiley & Sons, 2011.

[50] LOPES, J. F. et al. Evaluating the four-way performance trade-off for data stream classification in edge computing. *IEEE Transactions on Network and Service Management*, IEEE, v. 17, n. 2, p. 1013–1025, 2020.

[51] LIANG, J. Confusion matrix: Machine learning. *POGIL Activity Clearinghouse*, v. 3, n. 4, 2022.

[52] ZHANG, H. et al. Ros based framework for autonomous driving of agvs. *Proceedings of the IPS6-04, ICMEMIS, Kiryu, Japan*, p. 4–6, 2019.

[53] MOLNAR, S.; ROMULUS, L. A.; TAMAS, L. Embedded gpu based autonomous robot use cases. In: IEEE. *2022 30th Mediterranean Conference on Control and Automation (MED)*. [S.l.], 2022. p. 462–467.

[54] SUBASI, A. *Practical Machine Learning for Data Analysis Using Python*. [S.l.]: Academic Press, 2020.

[55] GÉRON, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. [S.l.]: "O'Reilly Media, Inc.", 2022.

[56] PERSSON, I.; KHOJASTEH, J. Python packages for exploratory factor analysis. *Structural Equation Modeling: A Multidisciplinary Journal*, Taylor & Francis, v. 28, n. 6, p. 983–988, 2021.

[57] ZAT'KO, P. Change point detection in network traffic time series.

[58] MORUZZI, G. Plotting with matplotlib. In: *Essential Python for the Physicist*. [S.l.]: Springer, 2020. p. 53–69.

[59] SIGUT, J. et al. Opencv basics: a mobile application to support the teaching of computer vision concepts. *IEEE Transactions on Education*, IEEE, v. 63, n. 4, p. 328–335, 2020.