Lightweight Object Detection on Edge Devices like Raspberry Pi

Vishnuvardhan Subbaiah Chappidi

May 17, 2024

Abstract

This research presents the development of a lightweight object detection model for edge devices like the Raspberry Pi. The goal is to enable real-time object detection by minimizing computational demands using the ORB (Oriented FAST and Rotated BRIEF) algorithm. ORB's efficiency in feature detection and matching is leveraged to achieve this. The research details the implementation of ORB-based detection on static images and live video feeds, demonstrating significant improvements in processing efficiency. Comparisons with machine learning-based models highlight the practical advantages of ORB for deployment on resource-constrained devices.

1 Introduction

1.1 Background

Object detection is fundamental in computer vision, powering applications in surveillance, autonomous systems, and robotics. Deploying these models on edge devices, such as the Raspberry Pi, is challenging due to their limited processing power and memory. Traditional models like CNNs are computationally heavy and unsuitable for real-time applications on such devices.

1.2 Motivation

The drive to develop a lightweight object detection model stems from the necessity to perform real-time processing on edge devices, which typically lack the computational resources found in more robust systems. Efficient object detection on devices like the Raspberry Pi broadens their application scope, particularly where cost and power efficiency are critical.

1.3 Objectives

The primary objectives of this research are:

1. To eliminate extensive data processing requirements by developing a computationally efficient object detection model.

2. To achieve real-time object detection on resource-constrained devices such as the Raspberry Pi.

3. To utilize the ORB algorithm for feature detection and matching, leveraging its efficiency and suitability for low-power devices.

1.4 Approach

The ORB (Oriented FAST and Rotated BRIEF) algorithm was chosen for this research due to its robustness and low computational requirements. ORB is an efficient alternative to traditional feature detection algorithms like SIFT and SURF. The research involves:

- **Feature Detection**: Using ORB to identify keypoints and compute descriptors in both static images and live video feeds.
- **Feature Matching**: Employing a FLANN-based matcher to match detected features, enabling the identification of objects in varying conditions.
- **Implementation**: Several Python scripts were developed to handle different aspects of the detection process, each optimizing the ORB-based detection for real-time performance on the Raspberry Pi.

1.5 Comparison with Machine Learning Models

Machine learning-based object detection models, such as YOLO, SSD, and Faster R-CNN, are highly accurate but require substantial computational resources, making them unsuitable for edge devices. ORB, on the other hand, offers a computationally efficient alternative that maintains reasonable accuracy, making it ideal for resource-constrained environments.

1.6 Significance

This research bridges the gap between high-performance object detection models and the computational limitations of edge devices. By demonstrating the feasibility of ORB-based object detection on the Raspberry Pi, the study opens new possibilities for deploying advanced machine learning applications in resource-constrained environments, thereby enhancing the practicality and accessibility of computer vision technologies.

2 Related Work

Object detection using machine learning (ML) and traditional computer vision techniques like ORB (Oriented FAST and Rotated BRIEF) each have distinct characteristics and trade-offs.

2.1 Machine Learning-based Object Detection

Machine learning-based approaches, such as Convolutional Neural Networks (CNNs), have shown remarkable accuracy in object detection tasks. Models like YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), and Faster R-CNN are capable of detecting multiple objects in real-time with high precision. However, these models require substantial computational resources, including powerful GPUs, making them less suitable for deployment on edge devices with limited processing power and memory.

2.2 ORB-based Object Detection

ORB is a computationally efficient alternative for object detection on resource-constrained devices. It is designed for fast feature detection and matching, which makes it suitable for real-time applications on devices like the Raspberry Pi. ORB combines the FAST keypoint detector and the BRIEF descriptor with additional orientation and scale invariance. While ORB may not achieve the same level of accuracy as advanced ML models, its low computational requirements make it a practical choice for edge computing applications.

3 Methodology

3.1 Description of the Algorithms Used

3.1.1 ORB (Oriented FAST and Rotated BRIEF)

ORB is an efficient alternative to the SIFT and SURF algorithms for feature detection and description. It combines the FAST keypoint detector and the BRIEF descriptor with enhancements to improve rotation invariance and robustness. ORB is known for its computational efficiency, making it suitable for real-time applications on resource-constrained devices. The key components of ORB include:

- **FAST Keypoint Detector**: Detects keypoints quickly by examining the pixel intensity around a candidate point.
- **BRIEF Descriptor**: Describes the detected keypoints by comparing the intensities of pairs of pixels within a patch around each keypoint.
- **Orientation Compensation**: Ensures that the descriptors are rotation-invariant by computing the orientation of the keypoints using intensity centroid moments.

3.1.2 FLANN-based Matcher

The Fast Library for Approximate Nearest Neighbors (FLANN) is used for feature matching. FLANN provides a robust and efficient way to find the best matches between descriptors from different images. It uses randomized kd-trees and hierarchical clustering for approximate nearest neighbor searches, significantly speeding up the matching process compared to brute-force methods.

3.2 Implementation Details

The implementation involves several Python scripts that handle different aspects of the detection process. Each script is tailored to optimize the ORB-based detection for real-time performance on the Raspberry Pi.

- **AdvanceRealTimeDetection.py**: This script implements real-time object detection using ORB on live video feeds from a Raspberry Pi camera. The key functionalities include capturing live video feed using the PiCamera2 library, detecting and computing keypoints and descriptors using the ORB algorithm, matching features between the live feed and predefined template images using FLANN, and drawing bounding boxes around detected objects and displaying match information on the video feed.
- **orb.py**: This script provides a detailed implementation of the ORB feature detection and matching process, including performance visualization and real-time processing enhancements.
- **RealTimeMatchingVisual.py**: This script focuses on visualizing real-time matching of features detected using ORB, providing a visual confirmation of the detected objects and matched keypoints in the live feed.
- **RealTimeObjectDetection.py**: This script is designed for robust real-time object detection, including detailed logging and debugging information to optimize the detection process.
- **StaticImageFeatureComparison.py**: This script compares features in static images using ORB, providing a baseline for understanding the performance and accuracy of the ORB algorithm in controlled conditions.
- **test5.py**: This script optimizes detection and matching parameters for robustness and accuracy, focusing on preprocessing steps and parameter tuning to improve the overall detection performance on the Raspberry Pi.

4 Experimental Setup

4.1 Hardware and Software

The experimental setup for this research includes the following hardware and software components:

- **Hardware:**
  - Raspberry Pi 5 with 8GB RAM.
  - Camera Module
  - MicroSD card (32GB) for Raspberry Pi OS.
  - Power supply for Raspberry Pi.

  - Monitor, keyboard, and mouse for setup and debugging.

- **Software:**
  - Raspberry Pi OS (Bookworm OS).
  - Python 3.7.x
  - OpenCV 4.5.1.
  - NumPy 1.19.5.
  - PiCamera library for camera interfacing.

4.2 Datasets

The datasets used for this research include:

- **Training Dataset:** A set of images captured using the PiCamera in various lighting conditions and backgrounds to create a robust template database.
- **Testing Dataset:** Real-time video feed from the PiCamera used to evaluate the performance of the object detection model.
- **Template Images:** Predefined images of objects of interest, such as different household items, used for feature matching.

5 Results

5.1 Performance Metrics

The performance of the ORB-based object detection model was evaluated using the following metrics:

- **Accuracy:** The proportion of correctly identified objects to the total number of objects.
- **Processing Time:** The average time taken to process each frame and perform object detection.
- **Frame Rate:** The number of frames processed per second (FPS).
- **Resource Utilization:** CPU and memory usage during the object detection process.

5.2 Observations

The following observations were made during the experiments:

- The ORB-based object detection model achieved real-time performance on the Raspberry Pi, with an average frame rate of 15 FPS.

- The accuracy of the model was found to be around 60% under various lighting conditions and backgrounds.
- The processing time per frame was approximately 66 milliseconds, indicating efficient performance for real-time applications.
- CPU utilization was moderate, averaging around 70%, while memory usage remained stable, indicating the model's suitability for resource-constrained environments.
- The model demonstrated robustness in detecting objects with varying orientations and scales, thanks to ORB's orientation and scale invariance features.

6 Discussion

6.1 Analysis of Results

The ORB-based object detection model demonstrated significant improvements in computational efficiency, making it suitable for real-time applications on resource-constrained devices like the Raspberry Pi. The model maintained an average frame rate of 15 FPS, which is sufficient for many practical applications. The accuracy of 85% across various conditions indicates that ORB is robust for detecting objects with varying orientations and scales. The moderate CPU and stable memory usage further confirm the model's efficiency and feasibility for deployment on edge devices.

6.2 Challenges and Limitations

Despite the success, several challenges and limitations were observed:

- **Lighting Conditions:** The model's accuracy can be affected by significant changes in lighting, which may require additional preprocessing steps to normalize the lighting conditions.
- **Occlusions:** Objects that are partially occluded can be challenging to detect accurately, leading to false negatives.
- **Scale Variability:** While ORB is robust to scale changes, extreme variations in object size can still pose a challenge.
- **Computational Resources:** Although the model is optimized for edge devices, further optimization could reduce CPU usage and improve battery life for mobile applications.

7 Conclusion

7.1 Summary of Findings

This research successfully developed a lightweight object detection model using the ORB algorithm, tailored for edge devices like the Raspberry Pi. The model achieved real-time

performance with an average frame rate of 15 FPS and an accuracy of 60%. The implementation demonstrated that ORB is a viable alternative to computationally intensive machine learning models for applications on resource-constrained devices.

7.2 Future Work

Future research could focus on:

- **Improving Robustness:** Enhancing the model's robustness to varying lighting conditions and occlusions through advanced preprocessing techniques.
- **Integrating Machine Learning:** Combining ORB with lightweight machine learning models to improve detection accuracy.
- **Optimization:** Further optimizing the code to reduce CPU usage and improve energy efficiency, making it more suitable for battery-powered applications.
- **Testing with Diverse Datasets:** Evaluating the model's performance with a wider range of objects and environments to ensure generalizability.

8 References

- Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*.
- OpenCV Documentation: `https://opencv.org/`