NTNU

Faculty of Information Technology and Electrical Engineering

Department of Computer Science

IDATG2206 - Computer Vision

Project Report

Group project

Table of contents

Abstract

This project aimed to develop a simple system that could detect whether a fish is present in an image, with the goal of running the system on a raspberry Pi. The initial approach focused on traditional image processing techniques, starting with preprocessing steps and continuing with edge detection and template matching to identify the fish. However, the results showed that template matching did not work as we intended it to do. Because of the limitations we met, the group began exploring the possibility of using machine learning instead. While the project did not result in a working detection system, it provided valuable experience and highlighted which methods were less effective, helping the way forward for further development.

Use of AI

Python Code:

We got help from ChatGPT to fix an error in the Canny edge detection function. The feedback explained that the image needed to be in 8-bit format, so we added a line using np.uint8(gray_image * 255) to convert it which made the function work.

ChatGPT also helped modify the preprocessing code so that multiple images could be processed and automatically saved to a folder. Originally, the transformations were applied manually to a single image and saved, but we wanted to apply all preprocessing steps to both a cropped fish and a full tank image and save the results in a folder. ChatGPT helped us simplify this process into a loop that applies each method to all images and stores the outputs.

Contributions

The group project has been a group effort, where all members contributed actively in the planning and implementation process. In the first part of the project, each member participated in testing and developing different edge detection and template matching techniques. The group developed their own python code for the methods and worked collaboratively to combine all the ideas into a shared solution.

Near the deadline, when the team switched to focusing on machine learning, Sofiya was responsible for preprocessing the images using various methods. Hans Erik was responsible for the annotation and labelling of fish images in Pascal VOC format, and Simon was responsible for setting up the structure for future training and deployment of the model. The team maintained communication continuously and updated each other regularly.

Introduction

The goal of this project was to develop a simple method that could detect the presence of a fish in video frames, with the idea of classifying each frame as either “fish” or “no fish”. The plan was to be able to run the system on a Raspberry Pi, making it accessible for low-cost equipment. The initial approach focused on simple image processing techniques covered in the lectures of the IDATG2206 Computer Vision course. The work began by using traditional methods such as edge detection and template matching. The aim was to isolate the shape of the fish and distinguish it from background noise, so that the system could be able to recognize the fish by its outline. Several challenges occurred throughout the process, and near the deadline the plan had to change. The methods proved to be very limited, and as a result, the project shifted focus towards exploring a more robust solution with the use of machine learning.

Theory

Gradient-based edge detection

In gradient based edge detection, the change in intensity from one pixel to its neighbour, in both horizontal and vertical direction, is calculated (OpenCV. (2025a)). This is done by calculating the derivative of the pixel intensity in both pixels.
A threshold value is the selected value for what is to be determined as an edge in the image or not, filtering out what is considered noise. Gradient based edge detection is a method of outlining an object in the image.

Image diffusion

Image diffusion is a method of smoothening out an image. This is done by taking one specific pixel, then looking at the value of the neighbouring pixels in a predefine matrix size around the main pixel. Then taking the average of all the pixel intensities and giving the main pixel this value (OpenCV (2025b)). Doing this to all the pixels in an image will gradually smooth or blur out noise, while still preserving details such as edges, making it easier to detect the edges in a noisy environment.

Canny edge detection

Canny edge detection is an algorithm developed by John F. Canny that combines multiple methods and stages to detect edges. This process produces an outcome that highlights and sharpens continuous edges, with reduced noise (OpenCV. (2025c)). In a fish detection project, this algorithm can be used to highlight the outline of an object.

Contour detection

Contour detection is a method of grouping together continuous edges in an image, for example the edge of a fish (OpenCV. 2025d). The method follows an edge and groups the pixels together, based on either pixel intensity or colour. Contour detection can be used to recognizing objects in an image.

Template matching

Template matching is a technique where a smaller template is pulled over a larger image and the pixels are compered in order to find similarities (OpenCV. (2025e)). This is done in order to find the location of the template in the larger image. During this process, the template is compared to different parts of the larger image. An example is to take a template of an object, then search for a match in a larger image.

Image preprocessing

Image preprocessing is an important step when it comes to using machine learning. OpenCV has various techniques that can be applied to enhance data and make it more suitable for machine learning (OpenCV, 2025). Techniques such as colour space conversions, blurring, brightness adjustments and flipping or rotating the images. These steps improve model robustness by introducing variability and reducing irrelevant data, making preprocessing essential for reliable image analysis and training results (Patel, 2023).

Methods

Several preprocessing functions were implemented using OpenCV (OpenCV, 2025).

Gradient-based edge detection with diffusion

The focus was on classical edge detection techniques during the first part of the project. The group reused and adapted a gradient-based edge detection python code from a previous project in the course IMAG2024. It was a program that calculated the change in pixel intensity throughout the image. When re-using the program for the fish images, the program calculates the derivatives of the pixel in both the horizontal (f_x) and the vertical (f_y) directions of a grayscale image using NumPy. By identifying areas with the sharpest gradient changes, adjustable with a parameter, it was possible to highlight the edges in the image, which in this case was the outline of the fish. To further enhance the edges and reduce noise, diffusion was also applied to smooth out the image through multiple iterations.

This process was divided into three separate functions. The first function applied the gradient-based edge detection directly to the image, using a threshold parameter to adjust the highlighting of the edges. The second function performed diffusion, reducing noise while maintaining the important edges. The third function combined the two functions, by firstly applying diffusion and then running edge detection on the smoothed image.

Canny edge detection and contour detection

Other edge detection methods were also explored, using built in functions from OpenCV library, such as canny edge detection. The function was applied to a grayscale image with threshold values of 100 and 200 which allowed for enhanced edges. After using the gaussian blur and canny functions, the resulting image was then further processed using contour detection, which identified and isolated the outlines of the fish. These methods were useful and fast to implement because of the built-in functions in the OpenCV library.

Template matching

Template matching was implemented to test whether the system could recognize a fish by comparing an image with a fish outline to other images. The matchTemplate() function from OpenCV was used and various similarity metrics were tested to see if they could find a match. The method computed a match score at the target image to evaluate if the position was a good match score for the fish outline image.

Image Preprocessing

To improve and prepare images for possible machine learning, multiple preprocessing methods were applied using OpenCV. These included colour space conversions (to HSV and

YCbCr), Gaussian- and median blurring, contrast and brightness adjustments, horizontal- and vertical flipping, rotation, resizing, and Laplacian edge enhancement. The methods helped simulate variety for possible AI-model training. The methods and parameter choices were based on standard OpenCV documentation, with some adjustments inspired by course materials and example projects shared by the teaching assistant Omar El Hajj. The preprocessing was applied to full frame images of the fish, and cropped fish regions to ensure flexibility in the detection.

Results

In this project we tested a range of computer vision techniques to detect fish in images. Our primary focus was on edge detection methods such as gradient based detection with diffusion, canny edge detection and contour detection. The goal was to identify the outline of the fish and use this information in template matching to classify whether a fish was present in an image.

The gradient-based method, which was adapted from a previous course (IMAG2024), involved calculating intensity gradients in the x and y directions. To reduce noise from water particles, algae, and other background elements we applied diffusion before performing edge detection. This improved the visibility of the fish edges, but also preserved some of the background noise, making it difficult to distinguish the fish from the background noise.

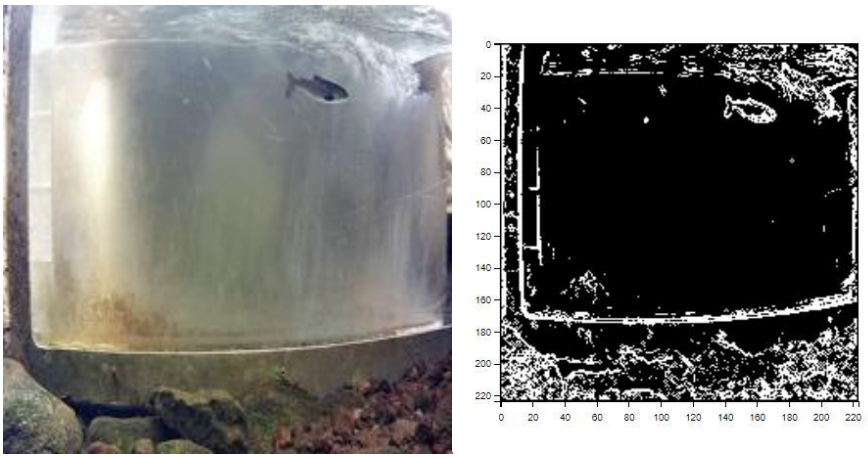

Figure 1: shows the original image used to the left and the image with gradient based edge detection

In parallel we tested Canny edge detection and combined it with Gaussian blur and contour detection using OpenCV's built-in functions. These methods provided cleaner results and showed clearer contours of the fish when applied to greyscale versions of the images. However, the contours of distracting background noise and objects were still sometimes equally prominent as the fish itself.

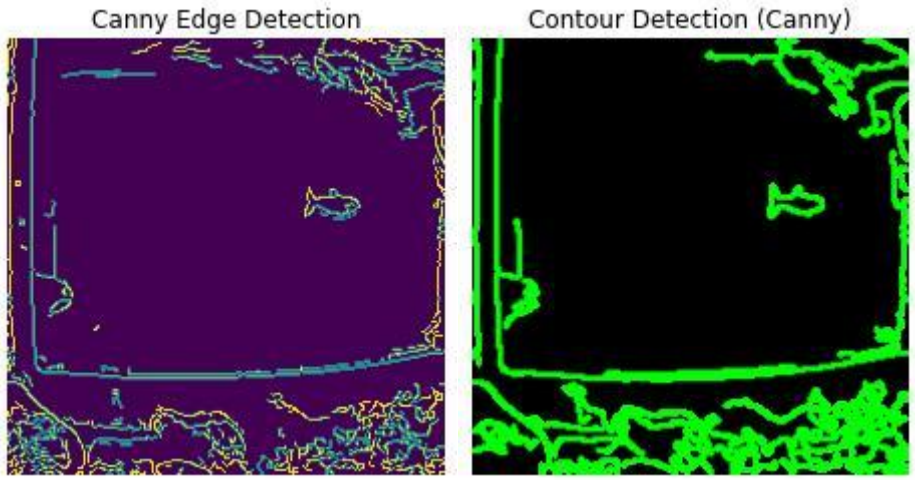

Figure 2: Shows images with Canny edge detection to the left and contour detection applied to the image to the right.

In order to test whether these detected outlines could be used for fish recognition, we implemented template matching. The method only worked when matching a cropped edge-detected fish image back to its original image. As soon as we tried matching it against other fish in images with similar shapes and preprocessing the method failed. This limitation is due to template matching being pixel-based and highly sensitive to scale, rotation, lighting, and position. The matching was only successful when the template was an exact sub-image of the target image.

To support future model development, we began systematic image preprocessing using OpenCV transformations such as brightness/contrast adjustments, HSV and YCbCr colour conversions, horizontal and vertical flipping, rotation, blurring, resizing and Laplacian edge enhancement. This was applied to both full frame images and to cropped fish images. These variations were created to simulate a variety in preparation for potential machine learning.

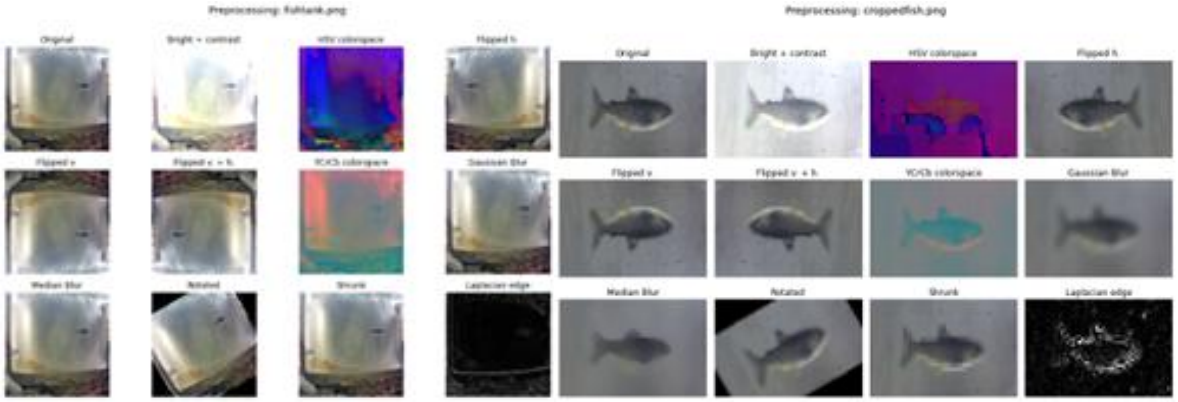

Figure 3: shows the preprocessing steps for machine learning

Although we did not achieve a working fish detection system, the results revealed important limitations in traditional methods and motivated further exploration of machine learning.

Discussion

Initial and main approach

The first part of the project focused on applying image processing techniques to detect fish in images. We experimented with different edge detection methods, including a gradient based approach we have worked on in a previous course, as well as Canny edge detection and contour detection using OpenCV.

The gradient-based method allowed us to manually tune thresholds and apply diffusion to reduce noise. In some cases, this provided a visible outline of the fish. However, despite using diffusion to smooth the image, the output still contained a significant amount of noise from background elements. This made it difficult to isolate the fish reliably, especially when other objects had similar contrasts and edge sharpness.

Canny edge detection, combined with contour detection gave slightly better results. The built-in functions were faster and easier to apply, and they highlighted the fish more clearly in several cases. However, these methods were also sensitive to changes in contrasts and sharp edges, and background noise was still often detected as strongly as the fish.

Template matching was the main strategy used to identify fish in processed images. This method was initially promising when we used a cropped fish image and matched it back to the same original image. However, this only worked in that specific case. When we attempted to use the same template on other fish images, the matching failed. What we tried to do is make a set of pre-processed images that a fish could be matched back to, but this was not possible as far as we searched for methods to make this possible. The main reason for this not working is that template matching is very sensitive to scale, rotation and small variations in shape and location for the fish. This is because it compares pixel values directly, it struggles to generalize across different input images.

Overall, the work we have done so far has helped us understand the strengths and weaknesses of the used image processing methods. While we were able to highlight fish outlines to some extent, we did not achieve consistent fish detection across varied images. The results revealed the limitations of our approach and shows us that more robust methods may be necessary for solving the task reliably.

Looking into and experimenting with Machine Learning

In our second attempt to solve the problem, we contacted our teacher who directed us to one of the subject teaching assistants. During the meeting with our teaching assistant, we came to a collective discussion to switch our focus over to machine learning. This was done both as a group decision and because we were recommended to approach the problem using machine learning by our teaching assistant, even though we only had a few days left until the project deadline.

During the meeting with our teaching assistant, we found a machine learning model that supports running on a Raspberry Pi. This model was the SSD MobileNet, which requires manual labelling of a handful of the pictures it where to train on. We also had to do some preprocessing to some of the images to add randomness and noise, increasing the robustness of the trained model.

After the meeting, we divided the machine learning project into three tasks, one for each group member. One where to do manual preprocessing, one had to label a sample of the images, and the last one where to set up and train the model on a local GPU.

We saw the potential during our research phase when looking into machine learning but did not have enough time to learn and test a model. In further work and development, we would have used more time to set up and test a model built with machine learning.

Conclusion

In this project, we set out to develop a system that could detect the presence of fish in images using different image processing methods. We explored a variety of edge detection methods such as gradient-based methods, Canny edge detection and contour detection. These methods helped highlight the shape of the fish in several images, and we gained practical experience working with OpenCV and preprocessing techniques.

Despite these efforts, we were not able to achieve reliable fish detection. Our implementation of template matching only worked when matching a cropped fish image back to the exact same original image and failed when applied to other images or when using other fish as inputs. This revealed clear limitations to our approach.

Although the project did not result in a functioning fish detection system, it provided valuable insight into what methods do not work in this context, The work has also laid the foundation

for further experimentation, including better preprocessing and the possibilities of using more robust methods, like machine learning in the future.

References

References of python tutorials used for the python- codes in the appendix, as well as references for the report:

OpenCV. (2025a). OpenCV-Python Tutorials: Image Gradients. Version 4.x. Retrieved 14. April 2025, from: https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html

OpenCV. (2025b) OpenCV-Python Tutorials: Smoothing Images. Version 4.x. Retrieved 14. April 2025, from: https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

OpenCV. (2025c) OpenCV-Python Tutorials: Canny Edge Detection. Version 4.x. Retrieved 14. April 2025, from: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

OpenCV. (2025d). OpenCV-Python Tutorials: Contours - Getting Started. Version 4.x. Retrieved 14. April 2025, from: https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html

OpenCV. (2025e). OpenCV-Python Tutorials: Template Matching. Version 4.x. Retrieved 14. April 2025, from: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

OpenCV. (2025). *OpenCV-Python Tutorials: Image Processing*. Version 4.x. Retrieved 14. April 2025, from: https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html

Patel, M. (23. October 2023). *The Complete Guide to Image Preprocessing Techniques in Python*. Medium. Retrieved 14. April 2025, from: https://medium.com/@maahip1304/the-complete-guide-to-image-preprocessing-techniques-in-python-dca30804550c

Roboflow. (2025). *Rotate Images with cv2.rotate.* Retrieved 17. April 2025, from: https://roboflow.com/use-opencv/rotate-images-with-cv2-rotate

Nair, A. (2023, August 29). *Guide to Adding Noise Images with Python and OpenCV*. AskPython. Retrieved 17. April 2025, from: https://www.askpython.com/python/examples/adding-noise-images-opencv

scikit-image team. (2025). *Tinting gray-scale images*. scikit-image. Retrieved 17. April 2025, from: https://scikit-image.org/docs/stable/auto_examples/color_exposure/plot_tinting_grayscale_images.html

Mahato, A. (14. Mars 2023). *Getting started with image processing using OpenCV*. Analytics Vidhya. Retrieved 17. April 2025, from: https://www.analyticsvidhya.com/blog/2023/03/getting-started-with-image-processing-using-opencv/