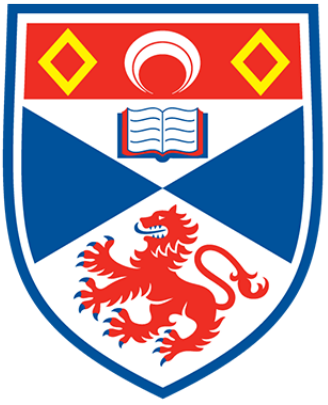


University of
St Andrews

CS4099 Major Software Project

Graph matching with Lobsters

April 6, 2018

Supervisor:
Kasim Terzić

Submitted By:
Sizhe Yuen

Abstract

The ability to measure lobsters is important to be able to monitor the size and health of the creatures. By being able to do this task with images and automatically with software, we can aid scientists in this field accomplish their work more quickly. There is currently an existing dataset of images where efforts have been made to determine the size and sex of the lobsters using global features such as total length.

This project aims extend that work by representing images of lobsters as graphs and use graph matching techniques to compare between them. We aim to use these techniques to discover properties of the lobster, for example its age, gender and health. The effectiveness of these techniques will be evaluated against the existing dataset to discover if graph matching is a suitable method for lobster recognition and characterisation. Extensions to this project would be to develop a new algorithm for create graphs rather than using existing ones and to try the same techniques on more complex images with lobsters in their natural environment.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is NN,NNN* words long, including project specification and plan.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. I retain the copyright in this work, and ownership of any resulting intellectual property.

Contents

1 Introduction

In this project, the application of subgraph matching and interest point detectors is explored to automatically identify different parts of a lobster such as the claw, body or tail. This was done by representing the body parts as a set of nodes connected by weighted edges to represent the distance between the parts.

TODO

1.1 Background

The fishing industry plays a large role in the Scottish economy. Many rules and regulations have been put into place to improve the management of inshore fisheries in Scotland for sustainable fishing. The minimum landing size is an example of a simple regulation to protect breeding stock, not allowing any catches below the minimum size to be landed. This insures a proportion of the animal population continues to grow and reproduce [10] and is used particularly for inshore shellfisheries. Lobsters are an example of a family of crustaceans affected by the application of this regulation.

Typically, catches have to be sorted and measured manually in order to adhere to regulations on minimum landing size. Illegal and unwanted products are then discarded back to the sea. Rochet [21] notes that discarding continues to be an important problem in world fisheries as discarded products often do not survive. This defeats the purpose of the regulation to preserve breeding stock and further hurts the industry as discarded products are lost without economic gain.

There has been some research in the area of applying computer vision techniques to automatically identify and detect species of fish [6, 8, 17] and research into automatic measurement of fish size [11, 22] which can be applied to reduce discarding. Much work is focused on fish due to the impractical nature of manual observation and measurement. Though there is little work that focuses on using such techniques for lobsters.

1.2 Objectives

Below the primary and secondary objectives of the project are listed.

1.2.1 Primary objectives

- Explore and create suitable graph representations for lobsters
- Measure similarity of lobster graphs with existing software
- Automatically detect interest points from images
- Evaluate this method of graph matching on lobsters against the existing dataset

All primary objectives were met, resulting in a complete method that can take an image of a lobster from the dataset and create a graph that detects and labels the different body parts of the lobster. The accuracy of the method is then analysed and evaluated.

1.2.2 Secondary objectives

- Extending existing or coming up with new algorithms for creating lobster graphs from images, knowing what we've learned so far.

- Apply this technique on more complex images with noise such as natural lobsters in their environment
- Apply this technique for video/real time instead of images to give properties and information on the lobster in real time.

Some of the secondary objectives were touched upon and investigated during the design and implementation of this project and the primary objectives, though none were explicitly explored. The final method developed is a new and novel way to create a graph representation of a lobster from an image and the ability to deal with noisy backgrounds was both noted and dealt with as an issue in the filtering (section **??**) and combination (section 4.7) stages respectively.

2 Context Survey

2.1 Previous work

The dataset of lobster images used in this project are taken from the master's thesis of Abdallah A. Abdallah [1]. In his work, Abdallah created the dataset consisting of images and features of lobsters with help and advice from the Scottish Oceans Institute. The lobsters were measured and categorised and segmentation and feature extraction techniques were applied to create a more diverse dataset with baseline results. Additionally, machine learning techniques of classification and regression were used to classify the category (juvenile or mature) and sex of the lobsters and to predict the carapace length.

The methods used in Abdallah's work were simple in that to be able to segment the lobsters from the image backgrounds, a high contrasting background was required, meaning the method was prone to noise. Further, his method of feature extraction using linear regression and principle component analysis was based mostly on the shapes of the segmented images, which were negatively affected by the posture of the lobster and asymmetric lobster shapes. This project uses a new, more complex approach with feature detectors and graph matching for a more accurate evaluation by automatically detecting the individual parts of a lobster to overcome issues of noise and shape.

2.2 Related work

2.2.1 Human pose recognition

The use of graphs in human pose estimation has been studied in the past [12, 24] where labelled nodes that represent important features such as hands and heads are used to build skeleton models.

Some of the methodology used in human pose estimation can be applied to our lobster matching problem. In particular, the paper on human pose estimation using a topological graph database by Tanaka et al. [12] uses a graph matching technique with an attributed graph to match skeletons to corresponding human postures. In their paper, example skeletons with different human topologies are developed into attributed graphs with manually assigned body part labels and stored in a model graph database. Any input skeletons can then also be converted to an attributed graph and matched with the examples in the database. Because their method of subgraph matching produces multiple results, further filters were needed to reduce the remaining graphs to one correct match.

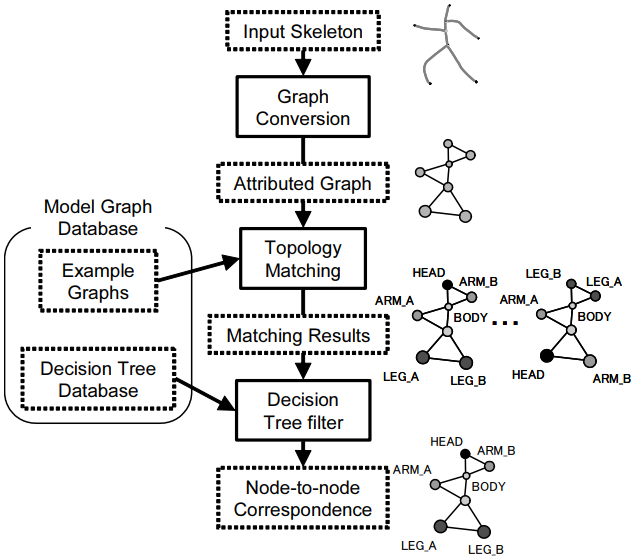


Figure 1: Body part identification algorithm using a model graph database (MGDB). Reproduced from [12].

The aspect of having labelled attributed graphs of human skeletons extends well to labelled graphs of lobsters. In the methodology of this project - which is explained in section 4 - some parts of Tanaka's method were followed. Most notably, the use of example lobster graphs stored in a database for input subgraphs to match to as an initial filter and the application of further techniques from the matched results to arrive at the final lobster graph were all inspired by Tanaka's paper.

2.2.2 Cattle identification

The concept of combining local invariant features (keypoints) and graph matching has also been studied for use in automatic cattle identification [19]. The application of feature extraction and graph matching in Monteiro's paper is different from the approach taken in this project, but it showed the concept of combining feature extraction with computer vision algorithms and graph matching could be used in complement as the graph structure is able to preserve the point neighbouring structure of extracted features. This is relevant to the problem of lobster part identification as different body parts have a clear neighbouring structure with other body parts.

In the paper, the goal of graph matching was to match images of the same muzzle as a means of identification (like human fingerprints). This is similar to classic computer vision matching with k-nearest neighbours matching between keypoints to match objects in a scene. The paper shows how graph matching with attributed graphs can take advantage of the associated attributes in each node and edge to represent the relationship between two features. Aspects of this are used in the final stages of graph creation (section 4.7) when building the full lobster graph from a pool of smaller labelled subgraphs.

2.3 Graph matching problem

The graph matching, or subgraph isomorphism problem is where, given two undirected graphs G_1 and G_2, it must be determined whether the graph G_1 contains a subgraph that is isomorphic to G_2 [7]. Cook showed in his paper that subgraph isomorphism was NP-complete with a reduction to the 3-SAT problem. This problem applies to the project from the use of subgraph matching to determine if a labelled subgraph is contained in a larger complete graph of a lobster.

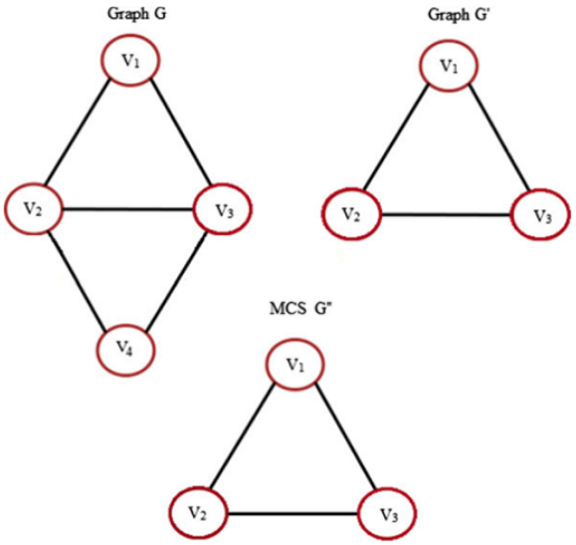


Figure 2: Example of matching two graphs G and G' and the resultant Maximum Common Subgraph (MCS). The MCS is simply a subgraph of both G and G' which contains the maximum number of nodes of all possible subgraphs of G and G' [23].

In this project, the problem of graph matching extends beyond that of pure subgraph isomorphism, where only the number of vertices and its connections are relevant, though the normal subgraph isomorphism problem is also applied as part of the method pipeline. The extension comes from the fact that the shape of the lobster is a crucial part in determining successful matches. As was mentioned in section 2.2.2, the size of each node in the graph and the length (weight) of the edges are all important aspects in matching and are included during the final matching step. This leads to two different graph matching steps used in methodology of this project. First, the normal subgraph isomorphism problem is applied to get a list of matched subgraphs, then another complicated problem of rebuilding the complete graph from a list of subgraphs is solved to get to the final lobster graph, where the size of nodes and lengths of edges play a larger role.

3 Software Engineering Process

Many different tools and technologies were used in this project to help facilitate the research. The use of these tools and libraries allowed the project to be focused on its methodology without the need or difficulty to implement known algorithms and methods.

3.1 Existing software

3.1.1 Graph visualisation

In order to visualise what a graph representation of a lobster may look like, graph drawing software that could import and export into a graph data format was needed. Initially the popular Graphviz and its `.dot` graph format [26] was explored. The `dot` graph format had all the attributes needed such as size of nodes and weights of edges, however there was no readily available GUI tool for drawing graphs as Graphviz mostly works on rendering existing `.dot` files. This was not ideal as the initial stage of the project involved manual annotation of the dataset with graph drawing software, to be done before any automatic generation graph files.

The open source Gephi [2] tool was the next piece of software explored and it was exactly what was needed in terms of a graph drawing tool. It allowed a simple graph to be drawn with nodes and edges labelled, so this could be done on top of a lobster image from the dataset. Further, the software was able to import and export from and into various different file formats such as `.graphml`, `.gml`, `.gdf` and `.dl`. Unfortunately it did not handle the `.dot` file format, so a choice had to be made between choosing the drawing tool or more powerful file format.

Listing 1: Header formats for `.gdf` files showing the kind of node and edge data it could keep.

```
1 nodedef> name VARCHAR,label VARCHAR,width DOUBLE,height DOUBLE,x DOUBLE,y DOUBLE,color
      VARCHAR
2
3 edgedef> node1,node2,weight DOUBLE,directed BOOLEAN,color VARCHAR
```

From exploring the various file formats that Gephi could use, it was discovered that the `.gdf` format contained enough information for our purposes. Specifically it contained a label, width, height, x and y coordinates for nodes and weights for edges. As such the Gephi tool was used over the Graphviz library and graph format due to its ease of use for drawing and file formats that contained enough information for its intended purpose.

3.1.2 Graph matching

Different graph matching and graph querying software was explored to deal with subgraph matching without the need for our own implementation. What was needed was a tool that could find if a labelled subgraph was part of a larger graph in a database of pre-defined lobster graphs. The tool also had to be fast and able to query a large number ($> 100,000$) of subgraphs with sufficient speed. Three tools, nauty [18], GraphGrep [9] and APPAGATO [3] were looked at for this purpose.

nauty was the first program that was investigated, as it was popular in the area of subgraph isomorphism. Of the three tools, it was more powerful as it contained more functionality beyond the subgraph isomorphism problem and also contained support for an interactive tool and a C interface to program against. However, the additional strength comes with the disadvantage of being more complex to use. The interactive tool is not suitable for our purposes as an automatic process is needed and both the graph file format supported and C interface are difficult to program against. Since the added functionality of nauty is not required, the other tools were looked at next to see if they could offer reasonable performance and be easier to use.

Both the GraphGrep and APPAGATO tools offered similar functionality and used the same input/output formats for graphs and matches. They also allowed input graphs nodes to be labelled with names, which was convenient for labelling the different parts of a lobster. The advantage APPAGATO offered over GraphGrep was its parallel GPU implementation for large speed up. During the initial prototyping stage, both tools were tested for their speed and suitability to the problem. Example graphs and queries were created and ran with large numbers of queries to test the speed and scalability of the tools. There was little difference in speed between the two tools for the scale of subgraphs needed in this project as both performed very reasonably for up to 500,000 subgraphs. The one issue found with APPAGATO was that it did not support matching multiple target and query graphs easily as its file format only support one graph per file. GraphGrep on the other hand allowed multiple graphs to be defined in a single file for both its database and queries. For this reason, GraphGrep was chosen over APPAGATO.

Given the simpler GraphGrep tool was enough to accomplish what was needed without suffering performance issues, it was also chosen over nauty and used for the subgraph matching stage of the project.

3.2 Technologies used

3.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) [4] is an open source library which contains a vast number of functions and interfaces for computer vision algorithms. The library was used heavily to prevent the need for implementing classical computer vision algorithms such as SIFT [16] and to make use of its image processing functions such as drawing detected keypoints and calculating colour histograms. This allowed much of the project to focus on the application of these algorithms and the combination of these computer vision techniques with probabilistic graph matching models, rather than need to produce new or classic keypoint detection algorithms. Further, the use of OpenCV exposed many practical underlying concepts and approaches in computer vision which were explored and used.

As OpenCV supports C++, Python, Java and MATLAB, the choice of language used was between these languages.

3.2.2 Python

Python was chosen as the primary language used for development over the other languages support by OpenCV. It is popular in the scientific community thanks to its extensive library support and ease of writing. Although performance of the algorithm matters from a computational complexity point of view, the performance for scripting the experiments is negligible and not a big issue. Because of this, it is worth it to take the trade off for using a higher level language like Python to make it easier and faster to test, as well as prototype different methods without the need for heavy error checking and debugging. Using Python also gives access to powerful libraries such as NumPy and SciPy, providing useful functionality such as probability distributions and matrix manipulation out of the box. Additionally, before the use of OpenCV in the project, the prototyping for probabilistic models, subgraph creation and subgraph matching was already written in Python, making it natural to extend from the prototype after the introduction of OpenCV functions.

3.3 Process

3.3.1 Development environment

During the development process of this project, a Python virtual environment was set up. Libraries such as OpenCV and even Python itself comes in multiple versions and some combinations of versions such as Python2.7 with OpenCV 3 has missing or moved functionality. To ensure the developed Python scripts can be run in different environments, development was kept in a virtual environment which contains all packages and libraries used with their versions. This was important as evaluation experiments had to be run on the cluster node machines which had a different setup to the lab machines. The virtual environment allowed a simplified setup to download and use the correct versions of all packages and libraries.

3.3.2 Testing

testing

3.3.3 Python types and classes

Although Python is a dynamically typed language, newer versions (Python 3.5 and above) have added support for type hints and static type checking with `mypy` [15]. As the implementation of the project grew, the addition of type hints was added to keep the code organised and readable. This was especially helpful to express the types of data contained in lists and tuples.

Listing 2: Example of a function definition with added type hints. The types from Python's `typing` class are highlighted here in red.

```
1 def bf_graph(graph: Graph,
2            matches: List[Tuple[KeyLabel, ...]],
3            node_dis: Dict[str, LabelData],
4            edge_dis: Dict[Edge, LabelData]) -> List[Tuple[KeyLabel, ...]]:
```

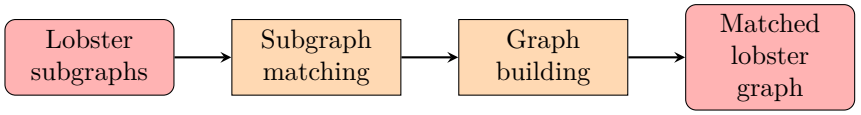
With type hints added to the Python code, a bash script was used to type check each Python file using `mypy`. This is useful because the type checking can check for errors that the Python interpreter does not catch. Often if there are mistakes in the code to do with mismatched types, they will not be detected until the code is run due to Python's interpreted nature, even if the Python code is compiled. This happens especially due to the types of variables being passed into function parameters becoming more complicated. Because scripted experiments can take some time to run, being able to type check beforehand and not run into errors mid or late into the experiment is very advantageous. Further, annotating the types of function parameters in the code makes it more readable as the type of the parameters can be immediately seen, rather than having to keep track of what is being passed manually or through comments.

4 Methodology

4.1 Overview

Figure 3 shows the key steps of the overall method used to go from an image of a lobster to a labelled graph of the lobster. To ensure this all works, an annotation step was done on a subset of the dataset to create a database of manually annotated attributed lobster graphs. This dataset is important for many aspects of the overall method as it is the set of complete graphs for subgraph matching. Further, the annotated attributes such as node labels, node sizes and edge weights contribute as a basis for the probability models used in both subgraph creation and rebuilding a complete lobster graph.

(a) Initial process of creating labelled subgraphs for matching.



(b) Process of taking the labelled subgraphs to creating a full lobster graph.

Figure 3: Flow chart of the whole matching process from getting keypoints to creation of the lobster graphs. The yellow steps in the pipeline represent processes and red steps represent inputs and outputs.

In the initial stages, keypoints are extracted and filtered from the image through the use of various OpenCV algorithms and techniques such as SIFT for keypoint detection and colour histograms for keypoint filtering. The remaining set of keypoints are then labelled using a probabilistic model and all permutations of possible subgraphs are created.

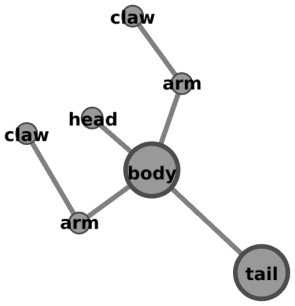
The second stage of taking a large set of subgraphs and matching them to rebuild the complete lobster graph consists of the following two main steps:

1. Run GraphGrep to find all subgraphs of valid configurations against the database of complete lobster graphs.
2. Use a probabilistic model to rebuild the matched subgraphs into the complete lobster graph.

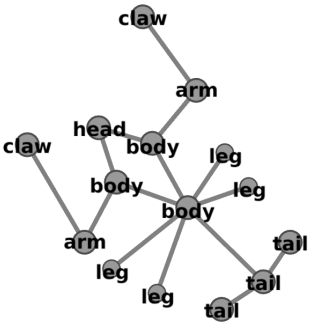
4.2 Annotation of dataset

The dataset provided by [1] was tagged with information for each image such as the lobster's sex, length of the carapace, width of the tail etc. To be able to apply the probability models to the dataset and match to complete lobster graphs, the dataset had to be further annotated as attributed graphs. This was initially done and mocked up using Gephi to explore different graph configurations and test with prototype probability models, but only with dummy values for the edge weights and size of the nodes. Later, with the introduction of applying OpenCV keypoint detection algorithms, the annotations were amended with the detected keypoints to include the real node sizes and edge lengths.

(a) Original lobster image



(b) Simple representation



(c) More complex representation

Figure 4: Examples of different configurations of lobster graphs that were explored using Gephi on the same lobster image.

In the initial lobster graph prototyping done with Gephi, different configurations and representations were explored to see how more or less information from the lobster graphs could influence the matching process. At this stage, the use of OpenCV keypoint detectors were not yet used, as the goal was to prototype the graphs and probability models for matching.

TODO why chose the representation that we did

After the use of computer vision algorithms was explored, the annotated step was revisited to update the graphs to correspond to keypoints that could be detected in the images. This improved the annotated dataset by adding real data for the size of nodes and lengths of edges which would be used. The annotations are all stored as `.gdf` graph files named after the images to be used later. This annotation took a few steps to complete:

1. Run SIFT to detect all keypoints on the image, then output all the keypoints as nodes in `.gdf` format.
2. By looking at the corresponding images, manually annotate the edge connections between relevant keypoints and remove all other keypoints.
3. Translate the annotated graphs in `.gdf` format to `.gfu` format for graph matching tools to use. The `.gdf` graph formats are still kept as they are used for the probabilistic models.

Figure 5: Example of annotated lobster image with nodes and edges of the graph perfectly labelled using keypoint detectors.

Additionally, the annotations were split between mature lobsters and juvenile lobsters. This was done as juvenile lobsters are smaller and the sizes of the nodes matter in later probabilistic models. The images from Abdallah's dataset were all taken from a tripod at the same distance, so the difference in size between the lobsters in the image reflects their actual size. Moreover, separate mature and juvenile models can be developed from the two categorised annotations. It would then be expected that the mature model matches better to mature lobsters and the juvenile model matches better to juvenile lobsters.

Because the process of manual annotation could have human errors, the final completed annotations are drawn as nodes and edges on top of the raw image again to visually check for mistakes made during the annotation of edges and removal of other keypoints.

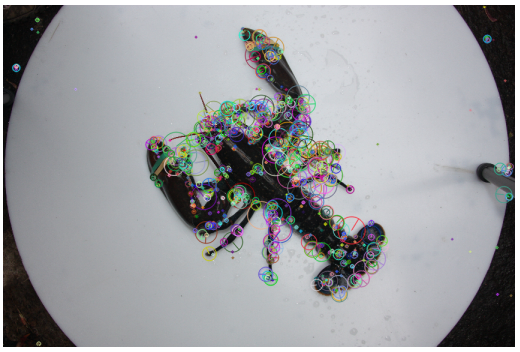
4.3 Keypoint detection

First, to identify important parts from our lobster images, keypoints or areas of interest must be identified. OpenCV provides a host of different algorithms for feature detection such as Harris and Shi-Tomasi corner detectors and SIFT, SURF, ORB keypoint detectors [25]. All these algorithms were tried and tested on a small subset of the dataset to see if any would provide both useful and consistent features that can be used.

(a) Harris corner detection

(b) Shi-Tomasi corner detection

(c) SIFT keypoint detection

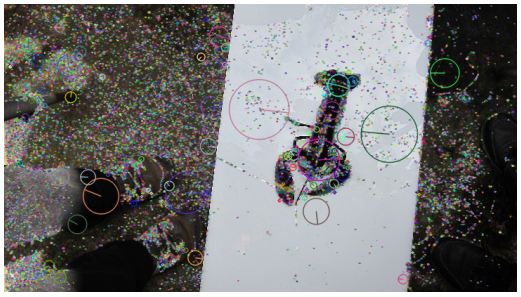


(d) SURF keypoint detection

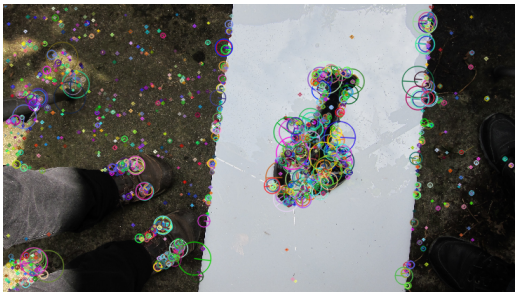
Figure 6: Comparison of different feature detection algorithms. The images have been scaled down after applying the detection to more clearly show the keypoints. Further comparison of the different detection algorithms on more images can be found in appendix **??**

(a) Harris corner detection

(b) Shi-Tomasi corner detection



(c) SIFT keypoint detection



(d) SURF keypoint detection

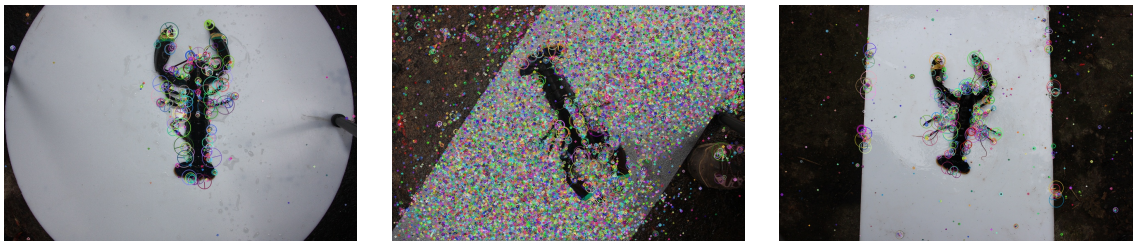
Figure 7: Comparison of different feature detection algorithms on an image with more noise.

From visually seeing the effects of each algorithm, it can be seen that the corner detectors do not work very well for our purpose. This is to be expected, as they are meant to detect corners and not feature points. Although corner detectors have applications in image matching [5], our goal from feature detection is to be able to extract graph like objects or features to apply graph matching on and so corner detection is unsuitable as we are not looking to match the different images to each other directly and the comparisons in figure 7 and 6 demonstrate why.

The keypoint detectors are able to provide better results for our objective, as we can see keypoints of important body parts being identified, such as the body, tail and claws. These keypoints are further able to be consistently identified from multiple images, showing that use of these algorithms are promising for automatically detecting the various parts of the lobster for construction of a graph. There is still a lot of irrelevant keypoints detected due to feature points in the backgrounds around the lobster, but we shall see later in section 4.4 that different methods can be applied to filter out unwanted keypoints. It is also non-trivial to create a graph from an outline of corners to represent the shape of a lobster, whereas the keypoints naturally translate well as nodes of a graph, with edges connecting them to form the shape and pose of a lobster. Because of these reasons, the keypoint detection algorithms in SIFT, SURF and ORB were further investigated while the corner detectors were discarded.

(a) (b) (c)

Figure 8: SIFT on different images



(a) (b) (c)

Figure 9: SURF on different images

(a) (b) (c)

Figure 10: ORB on different images

The three different keypoints detectors each detected different features in the images. ORB was

able to detect many edges and legs for the lobster, but did not consistently detect the claws or tail. Many of the features it finds are focused on areas near the head or sides rather than near the body of the lobster. In comparison, SIFT and SURF are able to find the more relevant body parts more consistently, but the trade off is a lot of small irrelevant keypoints are also detected. SURF still detects a lot of edges and legs like ORB, which is less useful, while SIFT is able to detect keypoints on top of lobsters, especially getting larger body and tail keypoints clearly. Between the different keypoint detection algorithms, SIFT was chosen as it gave consistent results for many images and gave the kind of useful keypoints that are needed and very applicable to mapping a graph structure on top of the image.

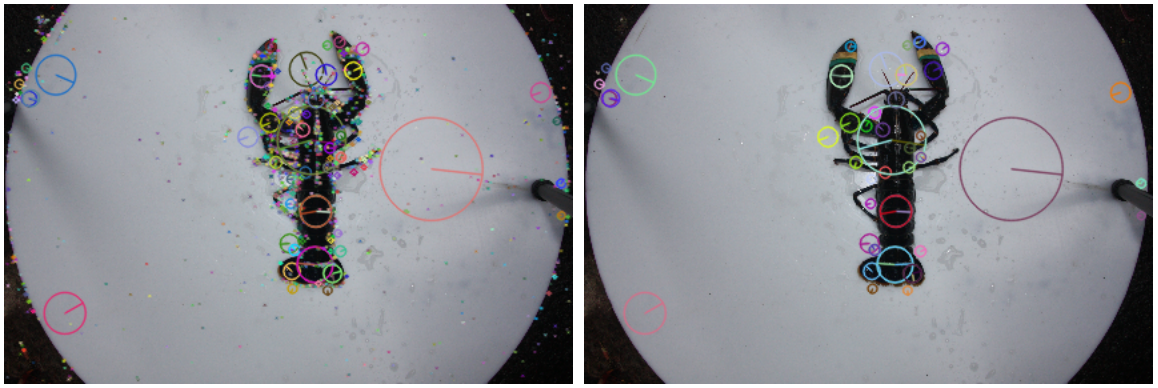
4.4 Keypoint filtering

From just running a SIFT detector on the lobster images, it can be seen that there are a lot of small keypoints that are unimportant for our purposes. There are also many keypoints around the lobster that we would like to filter out, as we want all our keypoints to be on the lobster. First, the smaller keypoints that are irrelevant have to be filtered away. Both the size of the keypoints and SIFT octaves were looked at for this step. Next, the typical computer vision approach for feature matching using the keypoint descriptors [16] was tried, but the results obtained were surprisingly poor. Because of this, a more novel approach was taken to filter out remaining keypoints that are not on the lobster using a colour histogram method where the colour histograms of the keypoints are compared and any below a certain difference threshold are filtered away.

4.4.1 Octave filtering

The method of filtering by the actual size of the keypoints was first looked at before looking at octaves. It was found to be less robust and less general than using octave levels. There are a few issues involved in using size of the keypoint for filtering, namely how to choose a suitable threshold. The size threshold must be constant across all images, otherwise the method will not be able to generalise to unseen images. The size of the keypoints is directly related to the size of the original image, so any size filter threshold must be calculated based on the size of the image. This is not an issue, as a constant size threshold can be relative to the size of the image. However, with different sizes of lobsters, an aggressive threshold may remove important keypoints that we wanted to keep. Conversely a more conservative threshold would not remove enough keypoints and cause a large combinatorial explosion, a problem explained later in section 4.5.1 that we wish to avoid. This makes it difficult to set a good threshold as it would have to be arbitrarily defined and based solely on manual inspection of the images and keypoints sizes of the dataset. Furthermore, this seems to be quite a crude method with little support in existing research and doesn't take advantage of any properties in the SIFT algorithm.

Octaves in SIFT are created by continually blurring an image. The idea behind this is to emulate looking at the image from different distances to get a varied set of keypoints. This means different features may be found at different octave levels. The high resolution of the images in our dataset causes many small keypoints to be found in the first few octave levels. These keypoints show many details that are irrelevant as we are concerned with the overall pose and size of the lobster.



(a) Before octave filtering (b) After octave filtering

Figure 11: Before and after applying a filter on keypoints based on the octave level the keypoints were found in.

From this observation, we can apply a filter on all keypoints found below a certain octave level so that we are only left with the larger keypoints that capture the features we are looking for. A filter for all keypoints found below octave level 3 was used. This octave level threshold is highly dependent on the size and resolution of the original image. An image with lower size and resolution may need a lower threshold or none at all as the lowered resolution effectively emulates octave levels applying their blur to the original image. This method is better grounded than using keypoint sizes as it is a part of the SIFT algorithm.

4.4.2 SIFT descriptors

In computer vision, keypoint descriptors obtained from detectors like SIFT and SURF are often used for feature matching [14]. Lowe's paper [16] on the SIFT detector states that keypoints descriptors are highly distinctive, allowing a single feature to be correctly matching with good probability in a large database of features. The descriptors are calculated by taking a region around the keypoint and split into subregions. Orientation histograms are then created for each subregion and the values are stored in the form of a vector.

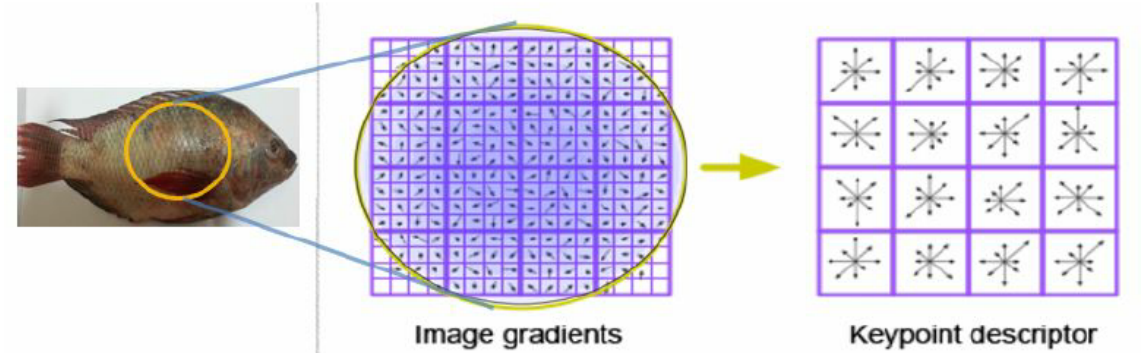


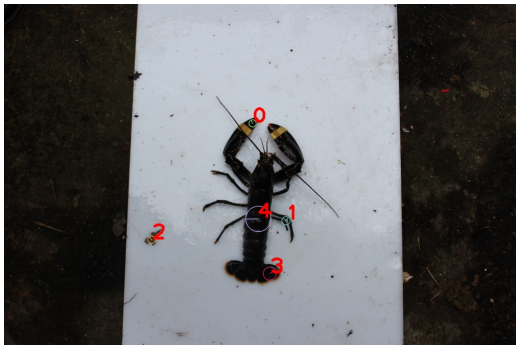
Figure 12: Process in calculated keypoint descriptors, taken from [8]

This is exactly what we want, as we wish to extract the different features of a lobster (tail, claws, head). Because we do not have a dataset for individual parts, to test if this method is viable, a specific keypoint was chosen and its descriptor calculated. Next, additional images were processed where the keypoints whose descriptors have the closest to the descriptor calculated previously are kept. Distance here is defined as difference norm of the two descriptors, as they are represented as vectors. It was ensured in these tests that the detection on the additional images tested contained

keypoints of the chosen feature. For example, if the *tail* feature is chosen, the descriptor for the tail keypoint is calculated for one image and used for comparison. The keypoint is identified by hand at this stage. Then images of other lobsters are processed where every keypoint descriptor is calculated and matched to the tail descriptor from the first image. This allows us to see the few best keypoints that were matched for new images based on the descriptor from one image. All keypoints of the additional images were printed to make sure a tail keypoint was detected and the goal is for the descriptor matching to automatically find that keypoint.

(a) Starting image where the *tail* descriptor is taken.

(b) Image 1424



(c) Image 4844

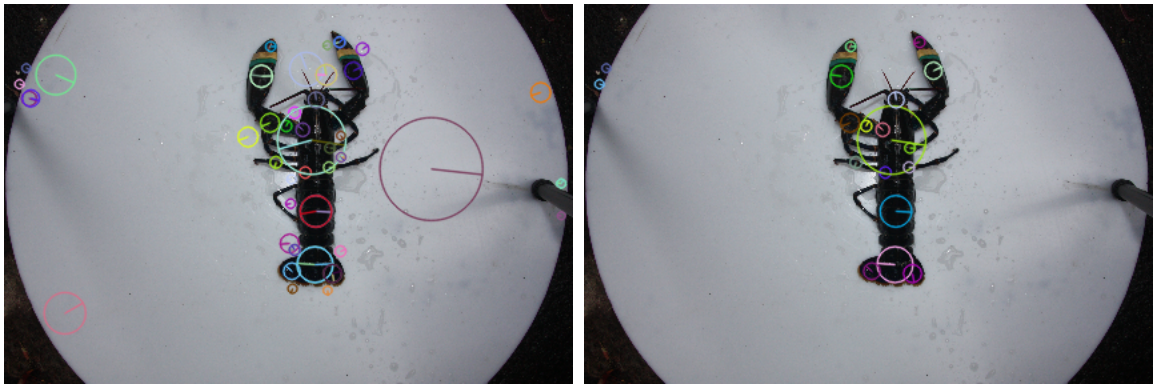
(d) Image 4785

Figure 13: Example of calculating a keypoint descriptor on one image and finding the closest 5 keypoints from other images. The found keypoints are numbered 0 to 4 where 0 is the closest distance and 4 is the furthest.

Figure 13 shows that the use of keypoint descriptors as a means of matching or filtering was not very reliable. After the tail descriptor was calculated, three other images are tested. Only one of the three could correctly detect the tail again, but it is not even the best matched keypoint. As the traditional method of descriptors proved unreliable for our means, a slightly more novel method was needed to filter out keypoints for labelling.

4.4.3 Colour histogram filtering

After applying the octave filter, there still remains some irrelevant keypoints that need to be removed. Most notably are the keypoints found on the white background of the images. There have been studies [13, 20] that show applying a colour filter to eliminate unwanted feature points can be quite effective, especially if the background is very different from the target of the image. Following from this, colour histograms of each keypoints were calculated and compared to a set of pre-defined histograms. The difference between the histograms was compared and only keypoints whose difference is above a certain threshold are kept.



(a) Octave filtering (b) Octave + colour histogram filtering

Figure 14: Difference between applying only octave filtering and applying both octave and colour histogram filtering.

The pre-defined histograms are taken from one of the lobster images from the dataset. A keypoint is taken and a mask applied to the image so only the area of the keypoint remains, then the histogram is calculated and saved as a NumPy array. The saved array can then be loaded later and compared to any keypoint to find the difference between them. Initially only one histogram was used, which was taken from the body keypoint of one image. However, this was not very effective as often the other keypoints had a large difference which meant the threshold had to be set to very low to capture important keypoints. This is not desirable as the lowered threshold causes more noisy keypoints to also be captured. To alleviate this issue, a separate histogram was calculated and saved for each different body part. This allowed higher thresholds to be applied with less likelihood of losing relevant keypoints. The actual threshold used for filtering is found experimentally and detailed in section 5.

Figure 15: Keypoints detected after filtering, showing the weakness of colour histogram filtering against noise.

A potential issue with this method is the application of this approach to noisy backgrounds. The colour histograms are effective due to the large contrast between the white background and the

lobster. However, this means that keypoints detected in noisy backgrounds that have similar dark colours are not filtered out. Figure 15 shows an image with more in the background and it can be seen that multiple keypoints from the background not belonging to the lobster still are kept. The probabilistic models applied in the later stages were designed with this issue in mind and help alleviate this issue, as incorrect keypoints detected far away from the lobster will have low probabilities during graph matching and be filtered out.

4.5 Subgraph creation

With a set of keypoints extracted and filtered from the image, the next step was to label the keypoints and create permutations of subgraphs to be matched. To reduce the problems with combinatorial explosion when creating subgraph permutations, the number of labels assigned to each point and the size of the subgraphs have to be kept low. The idea of starting from creating subgraphs rather than try to create the entire lobster graph is to reduce the initial complexity of the problem. With a set of matched subgraphs, the larger lobster graph can be built incrementally. This method would also be more reliable than trying the whole lobster graph at once as more subgraphs can be taken into account during the rebuilding stage. Additionally, errors would be more easily accountable as the errors would only occur per mismatched subgraph rather than the whole lobster graph being mismatched in one step, making it difficult to find out what caused the error.

4.5.1 Node labelling

A simple probabilistic model was used to determine how to label the keypoints through the use of Bayes' Theorem. For each keypoint, a probability is assigned for each label and a threshold applied to remove label probabilities that are too low.

$$P(\text{label} \mid \text{size}) = \frac{P(\text{size} \mid \text{label})P(\text{label})}{P(\text{size})} \tag{1}$$

The three probabilities $P(\text{size} \mid \text{label})$, $P(\text{label})$, $P(\text{size})$ are defined as follows:

- $P(\text{size} \mid \text{label})$ - This probability was found using a normal distribution for all sizes of the particular label found in the annotated dataset.
- $P(\text{label})$ - This is defined as how often that label appears in the annotated dataset relative to the total number of all labels. For example, if there were a total of 100 labels and the *body* label appears 20 times, then $P(\text{label} = \text{body}) = \frac{20}{100}$.
- $P(\text{size})$ - The probability of keypoint sizes again uses a normal distribution for all keypoint sizes in the annotated dataset.

This method means multiple labels can be applied to the same keypoint and each separate labelling must be treated as a separate node during subgraph creation. The threshold for labelling represents the trade off between computation required and completeness. If the threshold is too high, then we may miss many potential labels. This is important because the size of the annotated dataset is quite small, so there is a high standard deviation in the distributions for label sizes. On the other hand, if the threshold is too low, then too many labels are applied, leading to a combinatorial explosion when creating the permutations of subgraphs. Similar to the threshold for colour histograms, the threshold for labelling was explored experimentally to see what combination of thresholds give the better results. The output of the labelling process is a list of tuples containing a keypoint and its label. Having all the keypoints labelled, the next step is to create subgraphs where the nodes are represented by labelled keypoints and edges are defined between the nodes to create the subgraph.

4.5.2 Subgraph permutations

To create labelled subgraphs from the labelled keypoints, all possible permutations are produced. The size of the subgraph to create (and therefore the size of the permutation) is an important consideration in this step. The equation for the number of permutations of size k from n labelled keypoints is as follows:

$$P(n,k) = \frac{n!}{(n-k)!} \tag{2}$$

A large k would make the matching step easier as a larger subgraph conveys more information about the node labels and connectivity, so there would be fewer but more distinct matches. However, this would lead to considerably more permutations, for example, given 20 labelled nodes, there would be 6840 possible permutations for $k = 3$ while $k = 4$ would give 116280 permutations. Having this growing combinatorial explosion is undesired as the computational complexity of the problem increases significantly. Though a single permutation cannot contain the same keypoint with two different labels, each permutation of a single keypoint and its multiple labels must be explored. Further these are permutations so the order matters as it directly represents the connection of the nodes to edges.

(a) Permutation (body, claw, tail) (b) Permutation (claw, body, tail)

Figure 16: The order of the permutations matters as it determines the edges between the nodes.

A subgraph size of 3 was chosen in the end as it was a strong middle ground between retaining enough information for effective subgraph matching and not causing a large number of permutations to match. A size of 2 is ineffective for matching, it leads to too many successful matches because it is easy to find subgraphs of two nodes and one connecting edge as long as the labels are correct. However using 4 nodes as the subgraph led to too many permutations being produced, leading to a slow process in matching. This is why subgraphs of size 3 were chosen.

4.6 Subgraph matching

All subgraph permutations created are written as a query for GraphGrep to match. The reason for this matching step is to filter out subgraph configurations that are not possible based on the annotated dataset. Any invalid configurations will not have appeared in the database of complete lobster graphs and therefore will not be matched. A single subgraph can match to multiple database graphs and can also match multiple times to a single database graph. This does not matter as extra matches are redundant information, though it is an area for optimisation if needed.

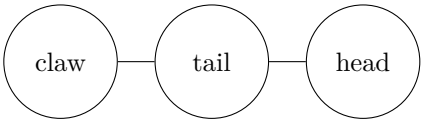


Figure 17: An example of an invalid subgraph configuration that is filtered out by subgraph matching.

Listing 3: Example of matching output from GraphGrep, which specifies the graph ids that were matched and which node from the query corresponded to which node from the database graph.

```
475:1:{(0,3),(1,1),(2,2)}
475:2:{(0,0),(1,3),(2,4)}
```

The output of running GraphGrep gives a list of graph ids of the subgraphs and database graphs that were matched. Any subgraph that appears in the list is taken as a matched permutation. After, there is a further step where the lengths of the edges between each node is taken into account. This acts as a late filter step for keypoints that may not be on the lobster, not allowing graph configurations where the points are too far apart to make sense. For example if the distance between an *arm* node and *claw* node is the same as the distance between a *body* and *tail*, the *arm*/*claw* permutation will be discarded as it is too long. This uses a probability distribution from known edge distances. This is helpful in discarding keypoints that are far away from the lobster, for example on the edges of the background. Furthermore, this filters out subgraphs where the labelling was valid, but the distances between the nodes were not, which acts as a premature step to remove mislabelled subgraphs, for example if the tail or back was labelled as a claw and connected to a correctly labelled arm.

4.7 Graph building

The final stage of the whole matching process is to take the pool of valid, matched and labelled subgraphs and piece together the complete lobster graph. Often there are many more subgraphs than are needed to rebuild the graph, so the best subgraphs and configurations should be chosen. To choose whether a subgraph was better than another, a naïve Bayes probability model is used. The probability of each subgraph is calculated as the product of the probability of each node labelling and probability of each edge.

$$P(\text{subgraph}) = \prod_{i=0}^{n} P(\text{node}_i) \cdot \prod_{j=0}^{m} P(\text{edge}_j) \tag{3}$$

where n is the number of nodes and m is the number of edges in the subgraph. The probability of each node labelling is already defined as it was used to assign the labels during the creation of the subgraph. It is the same as explained for equation 1.

$$P(\text{node}) = P(\text{node.label} \mid \text{node.size}) \tag{4}$$

The probability of the edges is computed using Bayes' Theorem with a probability distribution for each type of edge based on the nodes it connects and its length.

$$\begin{aligned} P(\text{edge}) &= P(\text{edge.length} \mid n_1 \wedge n_2) \\ &= \frac{P(n_1 \wedge n_2 \mid \text{edge.length}) \cdot P(\text{edge.length})}{P(n_1) \cdot P(n_2)} \end{aligned} \tag{5}$$

Each type of edge is uniquely defined by the two nodes it connects and so the probability of the edge takes that into account. To prevent floating point issues during implementation, the sum of logs is used rather than the product of the probabilities.

As stated before, a single keypoint can also be contained in multiple subgraphs. These cases have to be treated carefully as it is valid for those more than one of those multiple subgraphs to appear in the final lobster graph. An easy example of this is a body keypoint which has multiple subgraphs. One subgraph could contain the tail and back while another contains the arm and claw. In this case they are both valid because they could both be part of the final graph. Because of this, subgraphs in the pool cannot be discarded simply because they contain the same keypoint and label as another. Further, it is inevitable that multiple subgraphs with the same keypoint are used to be able to build a graph that is fully connected, otherwise additional rules will be need to define how two subgraphs should be connected.

With these considerations in mind, three different methods were used to build up the lobster graph from subgraphs, one of which gave poor results and was not used in the final evaluation.

(a) Keypoint method (b) Label method (c) Graph method

Figure 18: Comparison of the three different methods for rebuilding the complete lobster graph from matched subgraphs.

The three different methods are as follows:

1. **Keypoint method** - The initial method explored was to ensure every keypoint is used when rebuilding the subgraph. This will make sure that no keypoints are missed. For every keypoint that was detected and not filtered away, the subgraph with the highest combined probability is chosen. Some chosen subgraphs are removed to prevent overlaps where the same keypoint is given multiple labels. This removal is again based on probability where the subgraph with the highest probability is kept. The main issue with this method is it is very reliant on good keypoint detection and filtration. Any incorrectly detected keypoints will be included this process which leads to errors of labelling many irrelevant points. As such, more sophisticated methods were explored that do not rely so heavily on the keypoints detected.
2. **Label method** - In this method, a list of canonical required labels is used to choose the subgraphs for the final lobster graph. This ensures that the complete set of labels needed for the complete lobster is used and found where possible and very few additional labels past the canonical ones will be included. To build the complete graph, the most probable subgraph is chosen for each label in the canonical list of labels. Then, overlap is checked by comparing the keypoints and labels in each subgraph. This is done to make sure no keypoint has two different labels but will allow one keypoint to be contained in multiple subgraphs. Further, when a probably subgraph is chosen, the labels it contains are removed from the canonical list. The final output is the set of most probable subgraphs that contain as many of the canonical labels as possible without repetition.
3. **Graph method** - The final method explored takes into account graph properties of how nodes are connected rather than just choosing based on canonical labels. The weakness of the label method is not taking into account any relations between the nodes. Here, a canonical *graph* is defined which specifies both the labels and the connections between them. For each edge in the graph, matched subgraphs are chosen based on whether they contain the same node connections as the canonical edge. Again the same method of choosing the subgraph with the highest probability and removing overlaps was used to prevent duplicate labelling.

In section 5, the results of using both the labelling and graph methods are compared and analysed to see how well they work.

5 Results and evaluation

The developed models and methods

(a) Example of good match (b) Example of bad match

Figure 19: Example of matches

To evaluate how well this method of recreating lobster graphs performs, precision and recall metrics were initially used. The goal of using precision and recall is too see what kind of trade-off there is between the relevant keypoints detected in all detected keypoints (precision) and the relevant keypoints detected in all relevant keypoints (recall). Next, the F1 score is computed to get the harmonic mean of precision and recall. This allows the best labelling and colour histogram thresholds to be found. Finally, by applying a (metric TODO) on the models, images from the rest of the dataset are classified as either mature or juvenile and the results compared to Abdallah's results [1].

They are also a two other parameters that are explored in parallel with the evaluation metrics:

1. As explained in section 4.7, three methods were investigated for rebuilding the matched subgraphs into the final lobster graph. Although the keypoint method was not evaluated due to its shortcomings, the label and graph methods are both used during the evaluation to test which is more effective.
2. In section 4.2, it was described how the additional annotation of attribute graphs to the lobster images were split into mature and juvenile categories based on the original dataset. Different probabilistic models can be created based on the two sets of annotations and their performance on both mature and juvenile lobsters can be compared. It is also interesting to see if any model performs better with a particular method or for a particular label.

5.1 Precision and recall

The evaluation for precision and recall is split into two evaluations, one for the performance of identifying correct keypoints and the second for the performance of keypoint labelling. The evaluations are split up into two parts because defining false positives and false negatives when evaluating both aspects together is a difficult problem, the key issue being an overlap between false positives and false negatives for correctly detected but mislabelled keypoints. For example, if a keypoint has been correctly identified and labelled as a *claw* but is actually an *arm*, then it should be a false positive because the labelling is incorrect. However, it should also be a false negative because correct *arm* labelling was missed. Because of these difficult definitions, it made sense to split the precision and recall evaluation into two separate parts where the false positives and false negatives could be clearly defined.

In order to create a precision-recall curve, the probability threshold for labelling keypoints and the threshold for colour histogram filtering is incrementally changed. This shows how the two thresholds affect the precision and recall of the lobster graphs created. Note that results for very low thresholds were not obtained due to the large time taken because of the combinatorial explosion.

5.1.1 Keypoint identification

First, the precision and recall metrics were calculated for keypoint identification to see how many keypoints from the annotated images could be re-identified in the final graph. The labels of each

keypoint and the edges between them are not taken into account for these results. The precision and recall for this evaluation is defined as follows:

$$\text{Precision} = \frac{\text{Correctly detected keypoints}}{\text{Correctly detected keypoints} + \text{Incorrectly detected keypoints}} \tag{6}$$

$$\text{Recall} = \frac{\text{Correctly detected keypoints}}{\text{Correctly detected keypoints} + \text{Missed keypoints in annotation}} \tag{7}$$

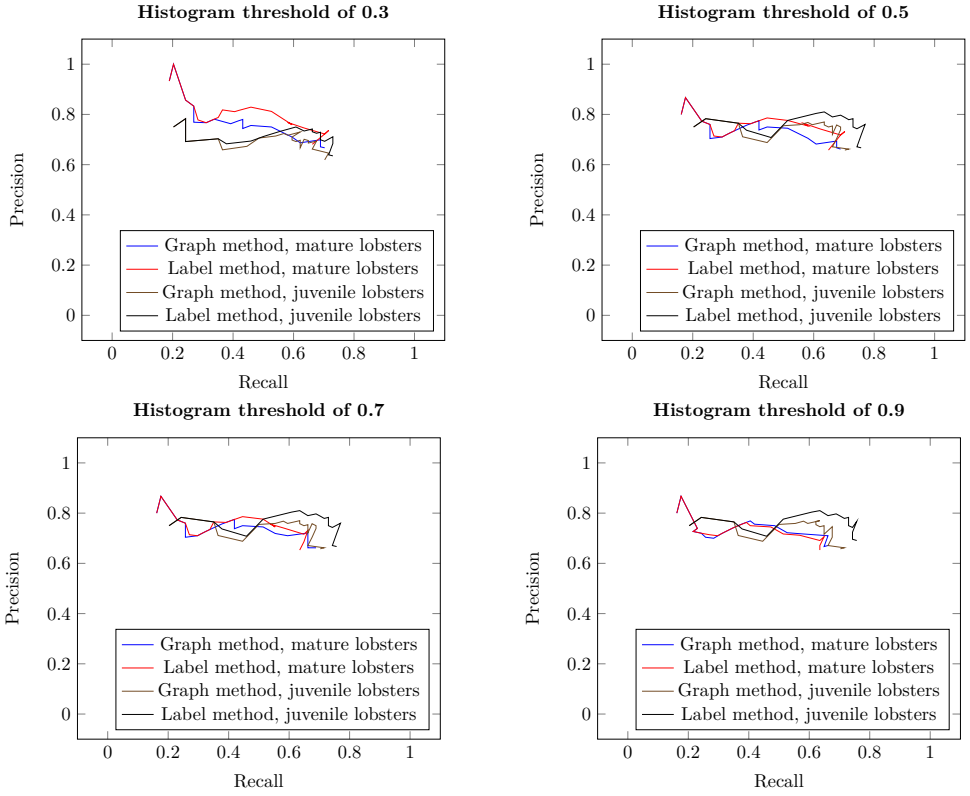


Figure 20: Precision/recall graphs with the mature model with varying label thresholds.

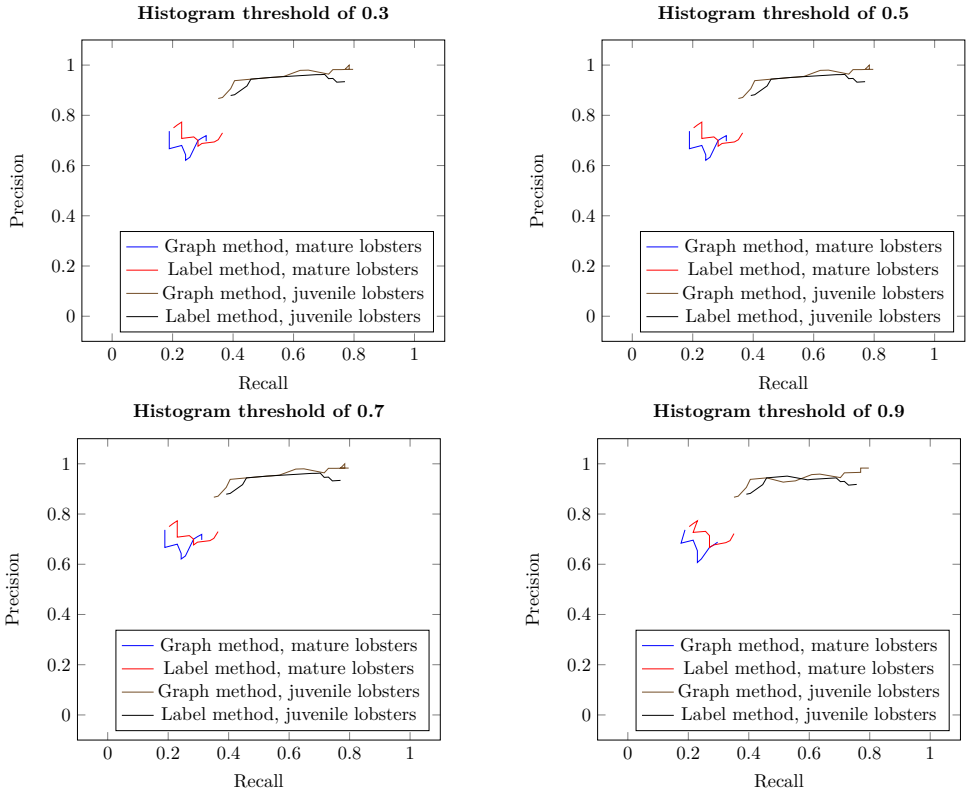


Figure 21: Precision/recall graphs with the juvenile model with varying label thresholds.

It can be seen from figure 21 and 20 that despite the label and histogram thresholds being varied, there is not a clear precision-recall curve that shows the trade-off between the two metrics. The change in the labelling threshold clearly affects the recall, but precision seems independent of both the recall and thresholds as it says relatively the same as recall improves dramatically. This suggests there is either no inherent trade-off between precision and recall in this problem, or the threshold that is being changed is not suitable to the metrics.

The label and histogram filtering thresholds that were varied may not be very suitable as they only indirectly affect the final graph that is built. The two thresholds changes what labels and keypoints are kept in the stages leading up to the creation of the final graph, for example a high label threshold would mean less subgraphs to explore as less labels could be applied. However it is the probabilistic model of choosing subgraphs which ultimately affects the lobster graph that is created. The issue here is there is no threshold that can be applied to this probabilistic model, as the most probable subgraphs are chosen incrementally. For this reason, looking at the precision and recall metrics is not the most suitable to evaluate performance.

Another interesting thing to note here is the difference between using a mature lobster model (figure 20) and using a juvenile lobster model (figure 21). There is a clear difference in both precision and recall when applying the juvenile model to the two categories of images, however, the difference is less obvious when using the mature model. This may come from the increased variance in the mature lobster images for sizes of keypoints and length of edges, whereas the juvenile lobster images are more consistent.

5.1.2 Keypoint labelling

For the evaluation of keypoint labelling, the precision-recall calculations are altered slightly as the labels on the detected keypoints matter here, rather than the detection itself. Additionally, the

evaluation can be focused on each specific label (claw, tail, etc.) to see if there are any strengths of weaknesses for specific body parts.

$$\text{Precision} = \frac{\text{Correctly labelled keypoints}}{\text{Correctly labelled keypoints} + \text{Incorrectly labelled keypoints}} \tag{8}$$

$$\text{Recall} = \frac{\text{Correctly labelled keypoints}}{\text{Correctly labelled keypoints} + \text{Missed labels in annotation}} \tag{9}$$

- False negative - Missed labels, eg if 2 claws and only 1 was found/labelled TODO remove

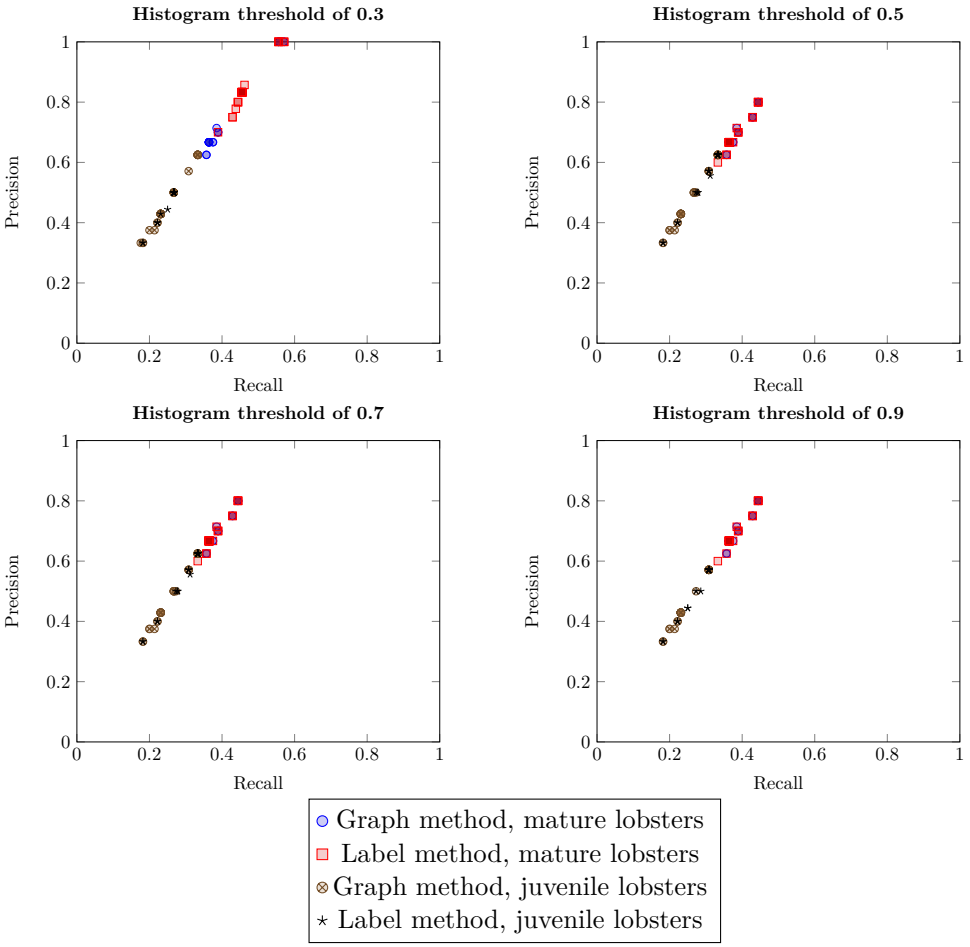


Figure 22: Precision/recall graphs with the mature model for label *claw*.

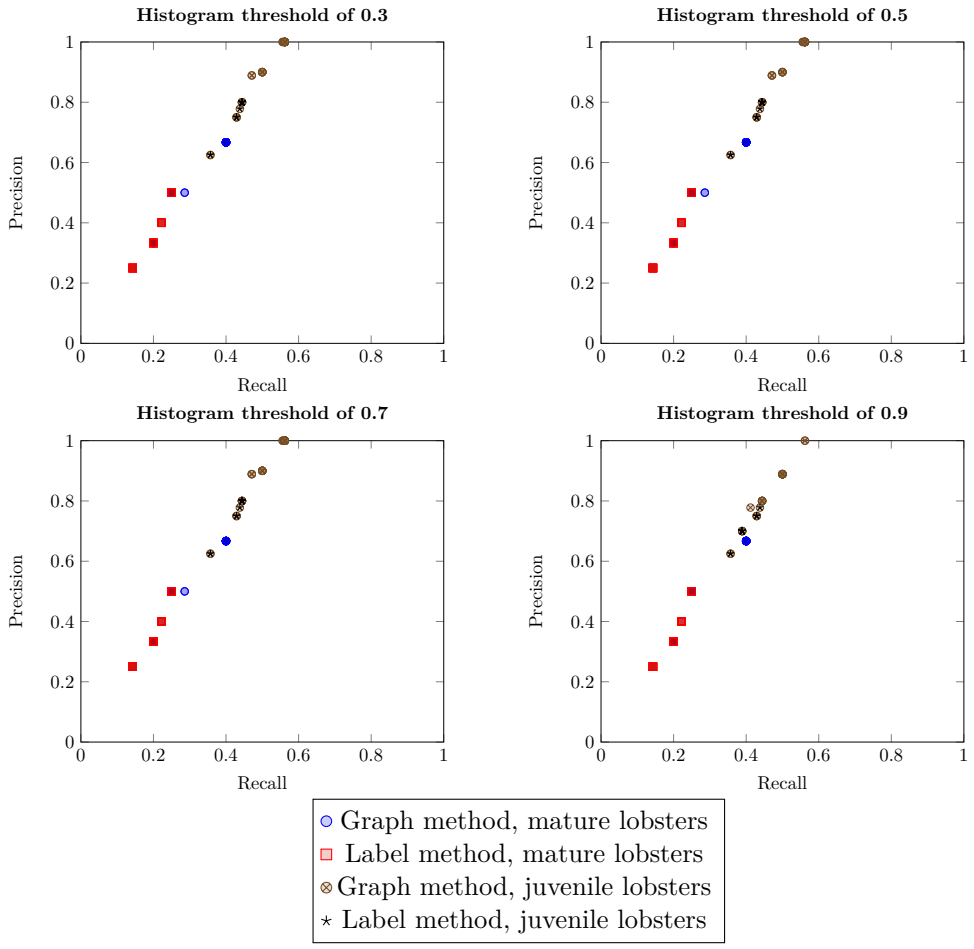


Figure 23: Precision/recall graphs with the juvenile model for label *claw*. Further graphs of all other labels can be found in appendix TODO

The precision-recall graphs for labels confirm that precision and recall are not very suitable metrics due to the lack of a good threshold to alter. There seems to be something missing in the curves, as the linear relationship from varying label thresholds and different methods and models is not expected from a precision-recall curve. Potentially each point in the graphs represent a separate precision-recall curve, but we are unable to find the threshold to vary that gives the trade-off and the rest of the curve.

Problem with recall for labelling is false negatives very low as only 1 or 2 labels are in annotated set.

5.2 F1 score

Although precision and recall did not give much insight into the models and methods used, the metrics can still be used to empirically find the best labelling and histogram filtering thresholds that should be used for classification. The F1 score can be used as a weighted average of precision and recall where both contribute equally. The best thresholds are therefore the ones where the F1 score is the highest. The F1 score also gives a reasonable look into which models and methods are the most effective. The identification and labelling evaluations continue to be kept separate for the same reason as before.

5.2.1 Keypoint identification

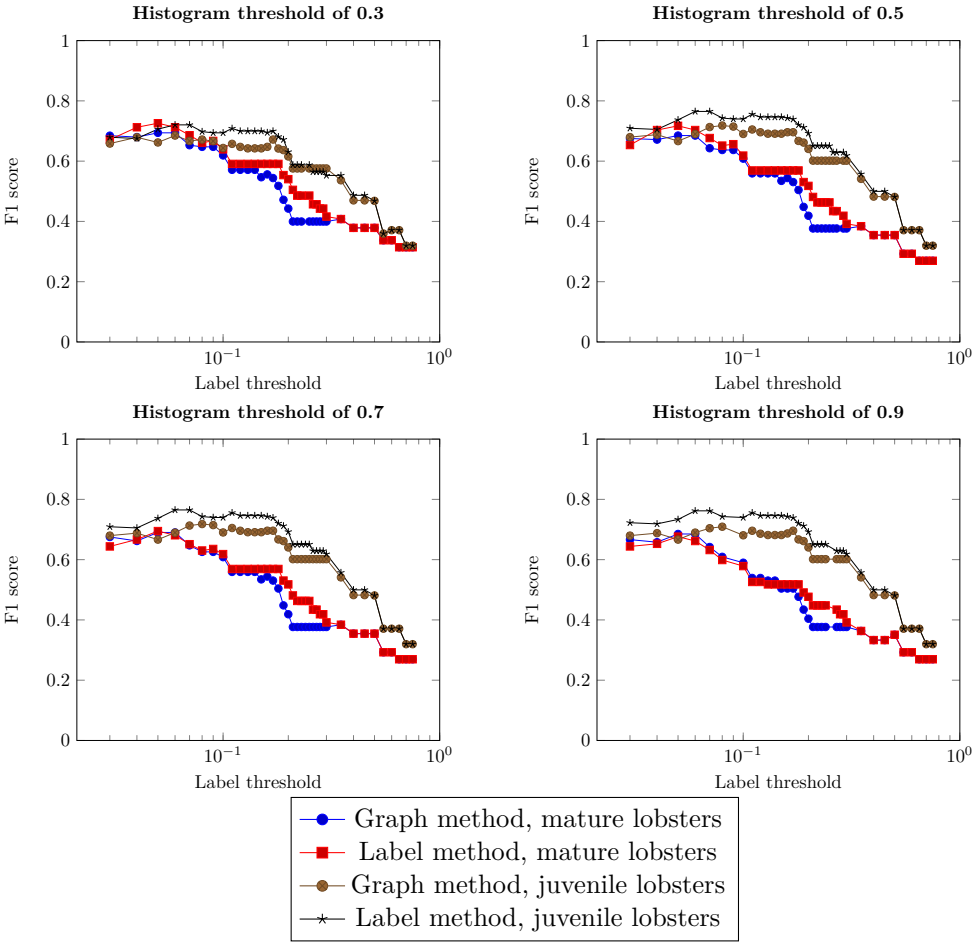


Figure 24: F1 score for varying thresholds with the mature model.

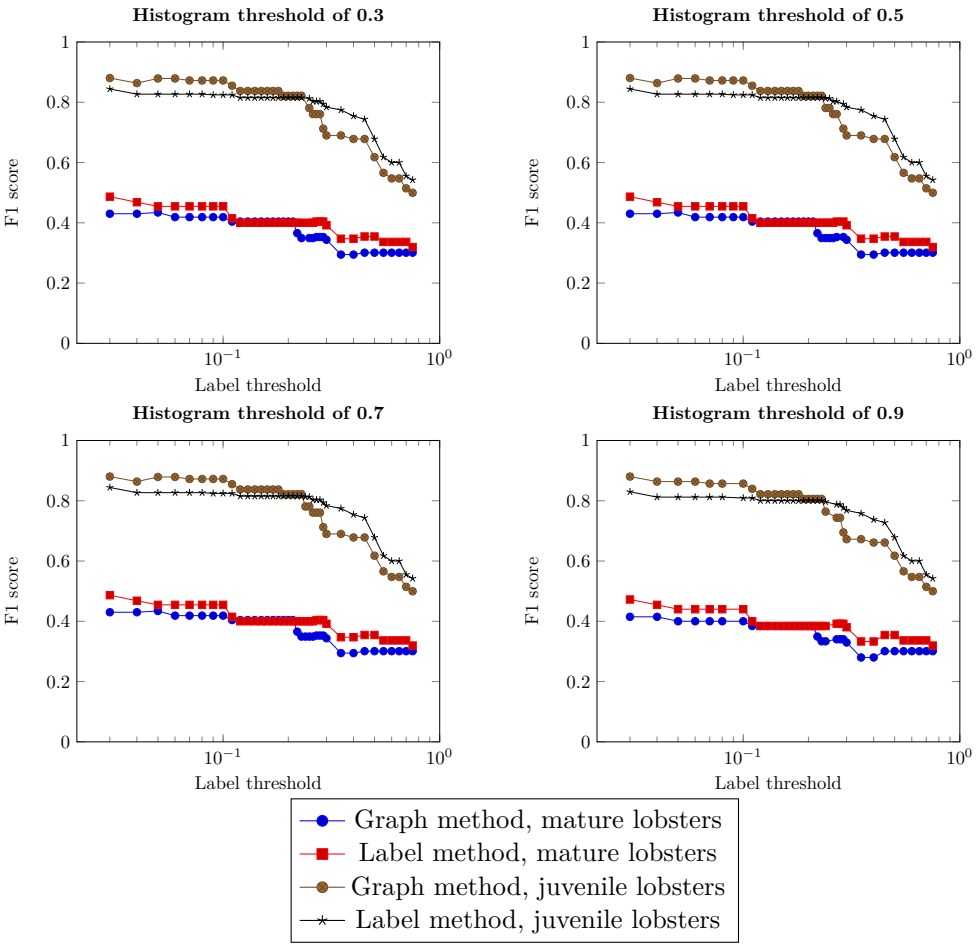


Figure 25: F1 score for varying thresholds with the juvenile model.

Method	Model	Category	Label threshold	Histogram threshold	F1 score
Graph	Juvenile	Juvenile	0.03	0.3-0.9	0.880
Label	Juvenile	Juvenile	0.03	0.3-0.7	0.844
Label	Mature	Juvenile	0.07	0.5-0.7	0.765
Label	Mature	Mature	0.05	0.3	0.726
Graph	Mature	Juvenile	0.08	0.5-0.7	0.718
Graph	Mature	Mature	0.06-0.05	0.3	0.694
Label	Juvenile	Mature	0.03	0.3-0.7	0.487
Graph	Juvenile	Mature	0.05	0.3-0.7	0.434

Table 1: Best F1 score and thresholds for different combinations of methods and models.

In general, it is shown clearly that the label threshold has a substantial impact on performance. As the label threshold increases, the F1 score decreases because it becomes less probable for any particular label to be applied to a keypoint. The decreased number of labelled keypoints then cause less subgraphs to be matched and so reduces the number of final keypoints. Moreover, at high label thresholds, it becomes more likely that a keypoint will not be labelled at all and so additional relevant information is lost.

The graphs of figure 24 and 25 further demonstrate juvenile lobsters being better detected in general, regardless of the model or method used, when compared to mature lobsters.

5.2.2 Keypoint labelling

5.3 Classification

From the results of precision, recall and F1 score, it was shown that there was a much clearer performance difference applying the juvenile model to mature and juvenile lobsters. Further juvenile lobsters still had high F1 scores even when applied to mature lobsters. For this reason, only the juvenile model is used to classify the unseen images of the dataset.

6 Evaluation

7 Conclusion

References

[1] A. A. Abdallah. "Machine Learning with Lobsters". MSc Thesis. University of St Andrews, 2017.

[2] M. Bastian, S. Heymann, and M. Jacomy. "Gephi: An Open Source Software for Exploring and Manipulating Networks". In: (2009). URL: http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

[3] V. Bonnici et al. "APPAGATO: an APproximate PArallel and stochastic GrAph querying TOol for biological networks". In: *Bioinformatics* Vol 32, Issue 14 (2016).

[4] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[5] J. Chen et al. "The Comparison and Application of Corner Detection Algorithms". In: *Journal of Multimedia* Vol. 4 Issue 6 (), pp. 435–441.

[6] M.-C. Chuang, J.-N. Hwang, and K. Williams. "A Feature Learning and Object Recognition Framework for Underwater Fish Images". In: *IEEE Transactions on Image Processing* Vol 25, Issue 4 (2016).

[7] S. A. Cook. "The Complexity of Theorem-proving Procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. Shaker Heights, Ohio, USA: ACM, 1971, pp. 151–158.

[8] M. Fouad et al. "Automatic Nile Tilapia fish classification approach using machine learning techniques". In: *13th International Conference on Hybrid Intelligent Systems, HIS 2013*. Dec. 2013.

[9] R. Giugno and D. Shasha. "GraphGrep: A Fast and Universal Method for Querying Graphs". In: *International Conference on Pattern Recognition* (2002).

[10] J. Hambrey et al. *Evidence Gathering in Support of Sustainable Socttish Inshore Fisheries: Integrating Stock Management Considerations with Market Opportunities in the Scottish Inshore Fisheries Sector - a Pilot Study*. Tech. rep. WP6. 2015.

[11] M. Hao, H. Yu, and D. Li. "The measurement of Fish Size by Machine Vision - A Review". In: *Computer and Computing Technologies in Agriculture IX: 9th IFIP WG 5.14 International Conference*. Beijing, China, 2015.

[12] T. Hidenori, N. Atsushi, and T. Haruo. "Human Pose Estimation from Volume Data and Topological Graph Database". In: *Computer Vision – ACCV 2007*. Springer Berlin Heidelberg, 2007, pp. 618–627.

[13] S. Hirai. "Color Filter in SIFT Matching". In: *Proceedings of the 2013 JSME Conference on Robotics and Mechatronics*. Tsukuba, Japan, 2013.

[14] E. Karami, S. Prasad, and M. S. Shehata. "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images". In: *Computer Vision and Pattern Recognition* abs/1710.02726 (2015).

[15] J. Lehtosalo and G. van Rossum. *mypy*. URL: http://www.mypy-lang.org.

[16] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *Int. J. Comput. Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 0920-5691.

[17] F. Martin-Rodriguez and M. Barral-Martinez. "iObserver: Species Recognition via Computer Vision". In: *7th International Workshop on Marine Technology*. SARTI, 2016, pp. 37–39.

[18] B. D. McKay and A. Piperno. "Practical graph isomorphism, {II}". In: *Journal of Symbolic Computation* 60.0 (2014), pp. 94–112. ISSN: 0747-7171. URL: http://www.sciencedirect.com/science/article/pii/S0747717113001193.

[19] F. C. Monteiro. “Automatic Cattle Identification Using Graph Matching Based on Local Invariant Features”. In: *Proceedings of the 13th International Conference on Image Analysis and Recognition*. Povoa de Varzim, Portugal, 2016, pp. 792–800.

[20] C. F. Olson and S. Zhang. “Keypoint Recognition with Histograms of Normalized Colors”. In: *2016 13th Conference on Computer and Robot Vision (CRV)*. Victoria, BC, Canada: IEEE.

[21] M.-J. Rochet, T. Catchpole, and S. Cadrin. “Bycatch and discards: from improved knowledge to mitigation programmes”. In: *ICES Journal of Marine Science* 71(5) (2014).

[22] A. Rodriguez et al. “Fish Monitoring and Sizing Using Computer Vision”. In: *Bioinspired Computation in Artificial Systems*. 2015, pp. 419–428.

[23] H. Sharma et al. “A review of graph-based methods for image analysis in digital histopathology”. In: *Diagnostic Pathology* (2015).

[24] M. Straka et al. “Skeletal Graph Based Human Pose Estimation in Real-Time”. In: *Proceedings of the British Machine Vision Conference*. 2011, pp. 69.1–69.12.

[25] opencv dev team. *Feature Detection and Description*. URL: `https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html`.

[26] *The DOT Language*. URL: `https://www.graphviz.org/doc/info/lang.html`.