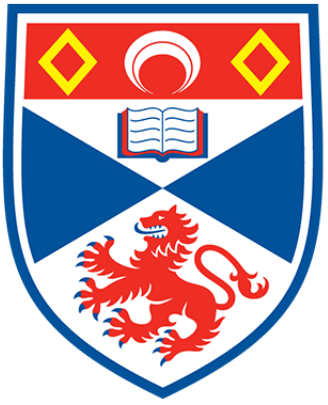


University of
St Andrews

CS4099 Major Software Engineering Project

Graph matching with Lobsters

March 30, 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 **4**
1.1 Objectives 4

2 Context Survey **5**
2.1 Background 5
2.2 Related work 5
2.3 Graph matching problem 6

3 Software Engineering Process **7**
3.1 Existing software 7
3.2 Technologies used 7

4 Methodology **9**
4.1 Overview 9
4.2 Annotation of dataset 9
4.3 Keypoint detection 10
4.4 Keypoint filtering 12
4.5 Subgraph creation 15
4.6 Graph matching 16

5 Implementation **18**

6 Results **19**
6.1 Keypoint identification 19
6.2 Keypoint labelling 19
6.3 Classification 19

7 Evaluation **21**

8 Conclusion **22**

Appendices **24**

A Comparison of feature detection algorithms **24**

1 Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

1.1 Objectives

1.1.1 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

1.1.2 Secondary 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

2 Context Survey

2.1 Background

2.1.1 Previous work

A large part of this project follows on from [1]. In his work, Abdallah created a dataset consisting of images and features of lobsters. The lobsters were measured and categorised and segmentation and feature extraction techniques were applied to create a more diverse dataset with baseline results. Additionally, classification and regression techniques were used to both classify the category of the lobster (juvenile or mature) and predict the carapace length.

This project is a continuation of Abdallah's work to apply computer vision techniques

2.2 Related work

2.2.1 Human pose recognition

The use of graphs in human pose estimation has been studied in the past [8] [16] where labelled nodes that represent important features such as hands and head 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 [8] by Tanaka et al. also 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 matches 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.

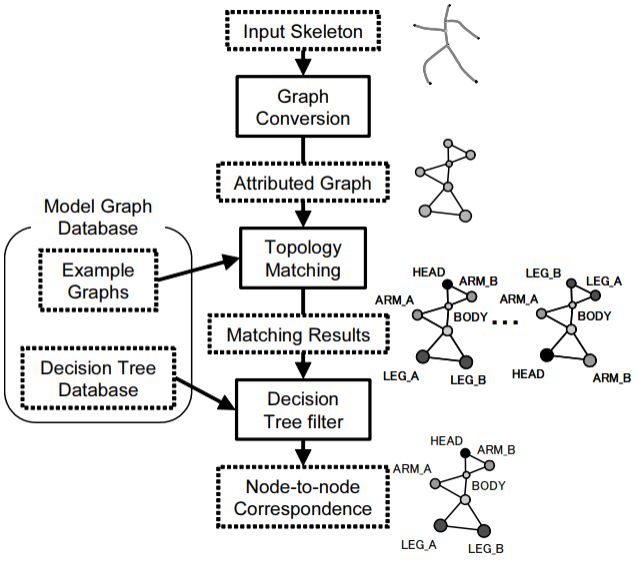


Figure 1: Body part identification algorithm using a model graph database (MGDB). Extracted from [8].

The methodology in this project, which is explained in section 4, follows some parts of Tanaka's method. Most notably the use of example lobster graphs stored in a database for input subgraphs to match to. The aspect of having labelled attributed graphs of human skeletons extends well to labelled graphs of lobster.

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 [14]. The application of feature extraction and graph matching in Monteiro's paper is different from the approach taken in this project, but there are a few similarities which show the concept of combining feature extraction and graph matching can give good results (TODO reword).

In the paper, the goal of graph matching was to match images of the same muzzle as a means of identification (like human fingerprints).

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 [6]. 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.

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. Because the shape of the lobster is a crucial part in determining successful matches, the size (weight) of each node in the graph and the length (weight) of the edges are all included as an important aspect in matching.

3 Software Engineering Process

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 [18] 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 too 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.

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. Two tools, GraphGrep [7] and APPAGATO [3] were looked at for this purpose.

Both tools were similar, using the same input/output formats for graphs and matches. They 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

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 [12] 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.

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. It is popular in the scientific community thanks to its extensive library support and ease of writing. Since performance was not a major issue, using Python would make it easier and faster to test and prototype different methods without the need for heavy error checking and debugging. Using Python gives us 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 subgraph creation and matching was already written in Python, making it natural to extend from the prototype after the introduction of OpenCV functions.

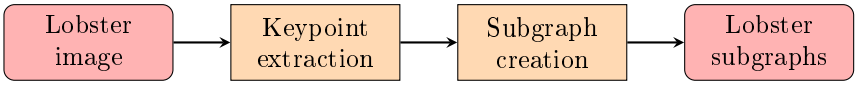
Although Python is a dynamically typed language, newer versions (Python 3.5 and above) have added support for type hints and static type checking with `mypy` [11]. As the implementation of the project grew, the addition of type hints was added to keep the code organised and readable. This was especially helpful to express types of data contained in lists and tuples.

TODO Gephi prototype lobster graph

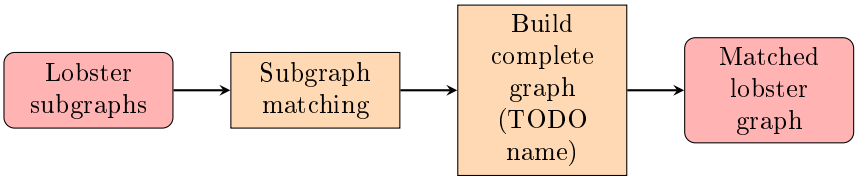
4 Methodology

4.1 Overview

Figure 2 shows the key steps of overall method used to go from an image of a lobster to a labelled graph of the lobster. 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 Bayes' theorem and all permutations of possible subgraphs are created. Next, the subgraphs are matched using GraphGrep [7] and all matches combined using probability models to get the final 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 2: Flow chart of the whole matching process from getting keypoints to creation of the lobster graphs.

A subset of the dataset was taken 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 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.

4.2 Annotation of dataset

The dataset provided by [1] was tagged with information on each image such as the lobster's sex, length of the carapace and width of the tail. 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 prototyped using Gephi to explore different graph configurations. Later, with the introduction of applying OpenCV keypoint detection algorithms, the annotations were amended with the detected keypoints to include node sizes and edge lengths.

In the first lobster graph prototyping done with Gephi,

Figure 3: Example of annotated lobster image with nodes and edges of the graph perfectly matched.

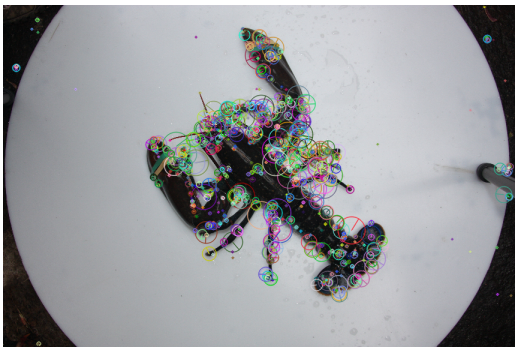
4.3 Keypoint detection

First, to identify important parts from our lobster images, keypoints or areas of interest must be identified. OpenCV [4] provides a host of different algorithms for feature detection such as Harris and Shi-Tomasi corner detectors and SIFT, SURF, ORB keypoint detectors [17]. 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

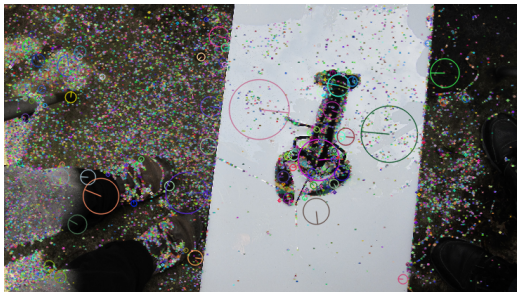


(d) SURF keypoint detection

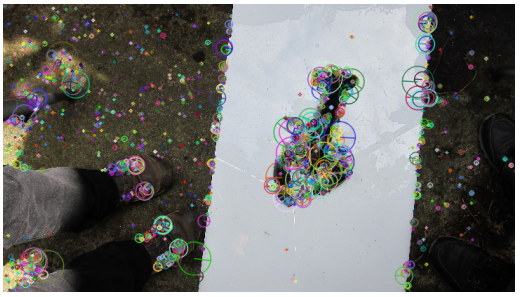
Figure 4: Comparison of different feature detection algorithms. The images have been scaled down after applying the detection to more clearly show the keypoints. Further comparison of the different detection algorithms on more images can be found in appendix A

(a) Harris corner detection

(b) Shi-Tomasi corner detection



(c) SIFT keypoint detection



(d) SURF keypoint detection

Figure 5: 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. The shape of the lobster is not fully detected reliably. For images with a noisier background, much of the background such as slabs with sharp contrasting corners are also detected, as shown in figure 5. Furthermore, 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.

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 noise in the detected keypoints, 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.

TODO fig of keypoint detector algorithm comparison

Figure 6: Comparison of different keypoint detection algorithms on multiple lobster images. See comparison of more images in appendix A

Between the different keypoint detection algorithms, SIFT was chosen as it gave the most consistent results and the kind of useful keypoints that are needed. TODO

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. Initially, the classic vision approach for feature matching using the keypoint descriptors [12] was tried, but the results obtained were surprisingly poor. Because of this, a more novel approach was taken for filtering. The small keypoints are filtered out by specifying an octave where all keypoints coming from that octave or above are kept. Finally, remaining keypoints that are not on the lobster are filtered with 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 SIFT descriptors

In computer vision, keypoint descriptors obtained from detectors like SIFT and SURF are often used for feature matching [10]. Lowe's paper [12] 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. This is exactly what we want, as we wish to extract the different features of a lobster (tail, claws, head). The only difference is we do not have a dataset of the lobsters, but not of dataset of individual lobster parts.

Because we do not have a dataset for individual parts, it made more sense to do the opposite of matching. Instead of using the distance between descriptors to match a detected keypoint with a known keypoint, the distance can be used as filter out keypoints that do not match closely to known ones. This distance can then be used as a threshold to filter more or less keypoints away.

As a test, the descriptor for the important body keypoint was taken from one image and calculated.

The descriptor was then matched to the closest other keypoints on another image to see if the body could be identified again.

TODO keypoint descriptor matching other image

Figure 7: The image on the left shows the keypoint that the descriptor was calculated from and the image on the right is the closest TODO matched keypoints to that descriptor.

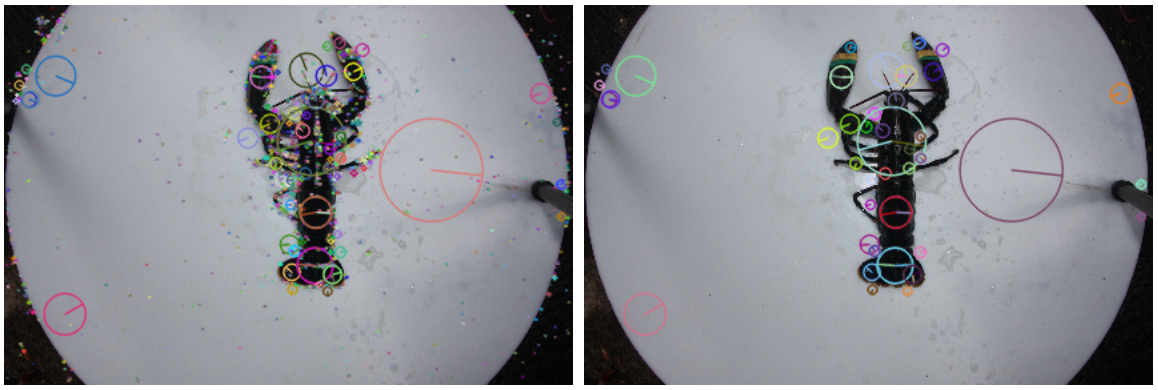
Figure 7 shows that the use of keypoint descriptors as a means of matching or filtering was not very reliable. TODO

As the traditional method of descriptors proved unreliable for our means, a slightly more novel method was needed to filter out keypoints. (TODO sentance wording)

4.4.2 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 filter, 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 TODO.

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

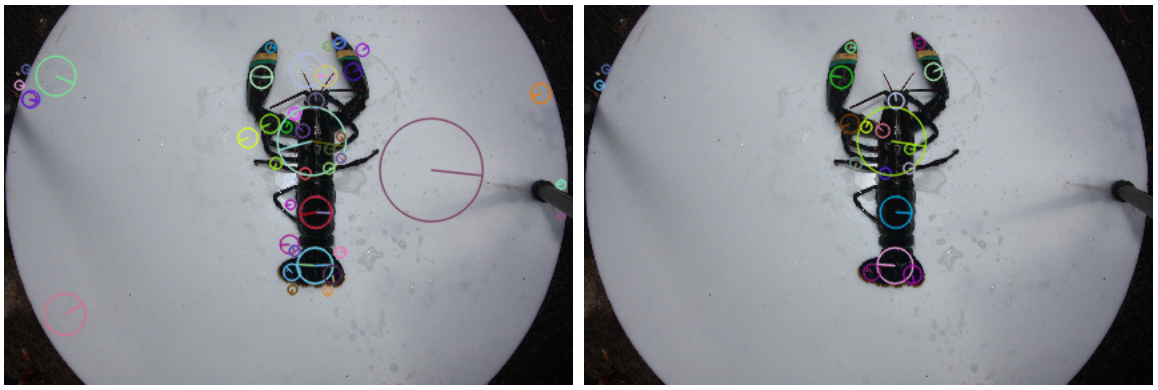
Figure 8: Before and after applying a filter on keypoints based on the octave level the keypoints were found in.

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 (TODO reason).

4.4.3 Colour histogram filtering

After applying the octave filter, there still remains some noisy keypoints that need to be removed. Most notably are the keypoints found on the white background of the images. There have been studies [15] [9] 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 different is above a certain threshold are kept.



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

Figure 9: Difference between applying only octave filtering and applying both octave and colour histogram filtering.

The pre-defined histograms TODO

Figure 10: Keypoints detected after filtering, showing the weakness of colour histogram filtering against noise.

A potential issue that was not fully explored 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 10 shows an image with more in the background and it can be seen that multiple keypoints from the background not belonging to the lobster are kept. (TODO ref return to issue in results)

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 problems with combinatorial explosions when creating subgraph permutations, the number of labels assigned to each point and the size of the subgraphs have to be kept low.

4.5.1 Node labelling

A simple probabilistic model was used to determine how to label the keypoints through the use of Bayes' Theorem.

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

The three probabilities $P(size|label)$, $P(label)$, $P(size)$ are defined as follows:

- $P(size|label)$ - This probability was found using a normal distribution for all sizes of the particular label found in the annotated dataset.
- $P(label)$ - 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 $P(label = body) = \frac{20}{100}$.
- $P(size)$ - The probability of keypoint sizes again uses a normal distribution for all keypoint sizes in the annotated dataset.

With the probability calculated for each possible labelling for every keypoint, the labels can be applied based on a threshold. This 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. The result 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 combinatorial explosion is unsustainable. 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 11: 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 Graph matching

The crucial step of taking a large list of subgraphs and matching them to rebuild the complete lobster graph consists of two main steps:

1. Run GraphGrep to find all subgraphs of valid configurations against the database of complete lobster graphs.
2. Use the probability of each remaining subgraph to piece together the complete lobster graph.

4.6.1 Subgraph matching

After the matching step with GrapGrep, the list of matches is used to pick out the matched permutations. During this step, 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. This is helpful in discarding keypoints that are far away from the lobster, for example on the edges of the background.

4.6.2 Subgraph rebuilding

5 Implementation

6 Results

To evaluate how well this method of recreating lobster graphs performs, precision and recall metrics are used. The evaluation 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. For example, if a keypoint has been incorrectly identified, then how is the labelling of the keypoint dealt with? The label cannot be correct, but only because the keypoint itself is wrong and there is not defined correct label for them.

6.1 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.

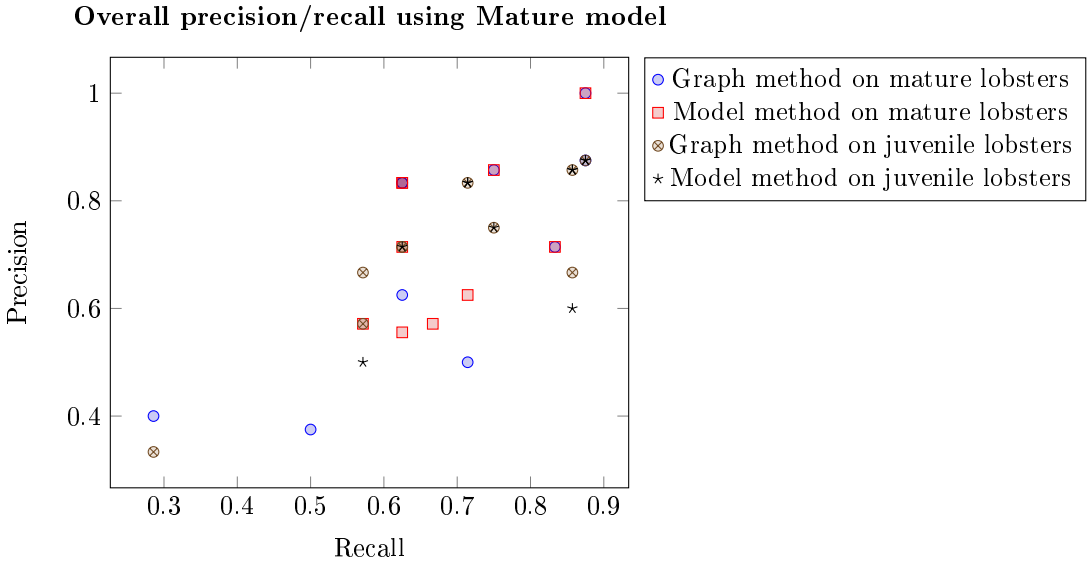


Figure 12: Precision Recall for different graph building methods

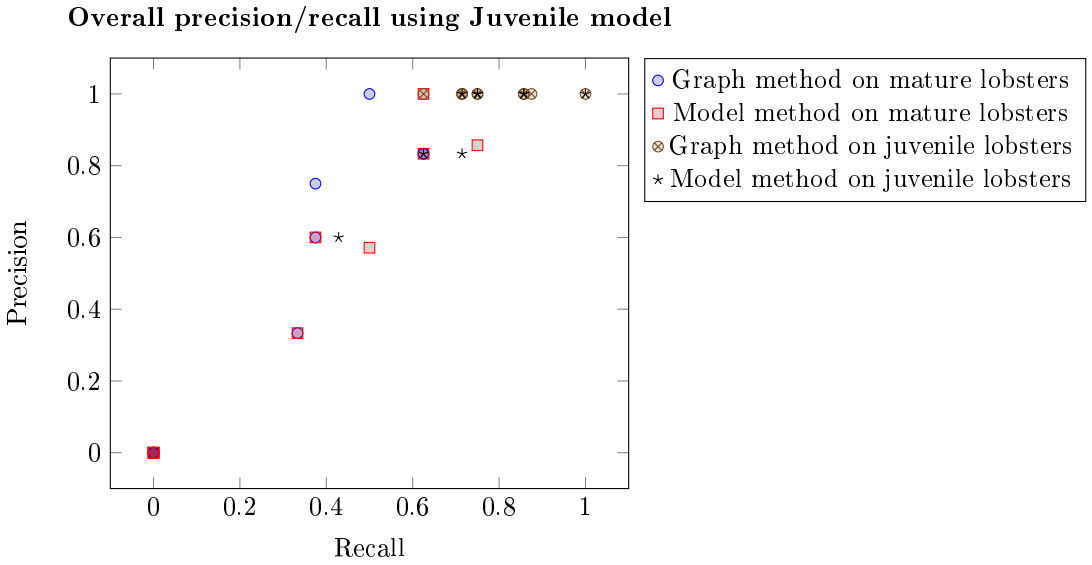


Figure 13: Precision Recall for different graph building methods

6.2 Keypoint labelling

6.3 Classification

results why overlap classify [13]

7 Evaluation

8 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] 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.

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

[8] 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.

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

[10] 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).

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

[12] 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.

[13] K. Mikolajczyk et al. "A comparison of affine region detectors". In: *International Journal of Computer Vision* Vol 65 (2005).

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

Appendices

Appendix A Comparison of feature detection algorithms

TODO imgs

TODO more imgs