University of Bath

MEng Project Report

Template Matching using Best Buddies Similarity and its Application to Cloud Tracking

Daniel Martinez Amigo
Candidate Number: 11069

supervised by:
Dr. Adrian Evans

I certify that I have read and understood the entry in the Student Handbook or the Department of Electronic and Electrical Engineering on Cheating and Plagiarism and that all material in this assignments is my own work, except where I have indicated with appropriate references. An electronic copy of this work has been uploaded to Moodle to allow submission to the University's Plagiarism Detection Service.

Number of words: 11,232

May 18, 2020

Abstract

A common method used to track the movement of clouds is template matching. This is normally done by comparing a section of an image to another using a similarity metric such as the Cross Correlation Coefficient (CCC) or Sum of Absolute Differences. In 2015, a novel method of template matching called Best Buddies Similarity (BBS) was presented and proven to be able to greatly adapt to a variety of difficult matching scenarios for object tracking including large deformations of objects. In this project, different implementations of the BBS method were coded and applied to different satellite images of clouds to test the performance of this new template matching metric for cloud tracking. Implementations of other commonly used similarity metrics for this application were coded and their performance was compared to that of BBS. An implementation of a relaxation labelling framework was created to assess the performance of BBS compared to other methods as the input layer for the relaxation method in creating smooth and accurate optical flow fields of cloud movement. Overall, it was determined that BBS is not a suitable replacement for commonly used CCC for cloud tracking in single channel greyscale sequences, yet it showed potential as an alternative to Ordinal Measures κ coefficient for estimating motion in impulse noise corrupted sequences. Further work may be done to determine the effectiveness of BBS in cloud tracking for multi-channel sequences or artificially coloured cloud image sequences.

Contents

Abbreviation Table

Abbreviation	**Definition**
BBS	Best Buddies Similarity
BBPs	Best Buddies Pairs
CCC	Cross Correlation Coefficient
SAD	Sum of Absolute Differences
SSD	Sum of Square Differences
OMκ	Ordinal Measures kappa coefficient
MSG	Meteosat Second Generation
SEVIRI	Spinning Enhanced Visible and Infrared Imager
NWP	Numerical Weather Prediction
IR	Infrared

1 Introduction

Tracking the movement of clouds is an area of interest within digital image processing for scientific and meteorological applications. A common method of estimating cloud motion is through the use of template matching, a digital image processing technique which works by taking a section of an image and comparing it to equal sized candidate windows in a another image using a chosen similarity/dissimilarity measure to find the area that most closely resembles the original template. By splitting an image into evenly sized templates and performing template matching on each templates the overall optical flow of cloud satellite images can be estimated. However, due to occlusions, depth disparities and deformations caused by the non-rigid motion of clouds, commonly used similarity metrics such as the Cross Correlation Coefficient (CCC) at times fail to correspond the top match in their correlation surface to the correct match for a template [1] and therefore a probabilistic relaxation framework is often used in conjunction with template matching to select the best possible motion vectors from a correlation surface and create smooth optical flows of large non-rigid bodies [2] such as clouds. At the core of the motion estimation process is the choice of measure used to assess the similarity between template and candidate windows. Recently a novel method of template matching called Best Buddies Similarity (BBS) was presented in [3]. This method was proven to consistently outperform other popular metrics in a variety of different matching scenarios for object tracking. However even though the method has shown great results for object tracking, its performance for remote sensing applications such as cloud tracking has not yet been investigated. Some of the stated properties of BBS in [4] make it a promising alternative to common metrics used for these applications such as its robust performance when matching templates with large deformations. This project will focus on implementing the BBS method and analysing its performance for cloud motion estimation.

1.1 Project aims

The following project aims were set for the project based on the plan made in the interim report [5]:

- Implement different template matching methods.
- Test performance of different template matching method applied to cloud tracking.
- Implement BBS method and investigate the effects of different parameters for cloud tracking applications.
- Create an implementation of a relaxation labelling framework.
- Assess the viability and performance of the different methods including BBS as the input layer for relaxation labelling.

2 Background work

2.1 Template matching

Template matching is a digital image processing technique which aims to find a section in an image that most closely resembles a chosen template image. This technique has a range of applications such as feature extraction in medical imaging and object tracking [6], [7]. The basic algorithm for any template matching application is shown below in Figure 2.

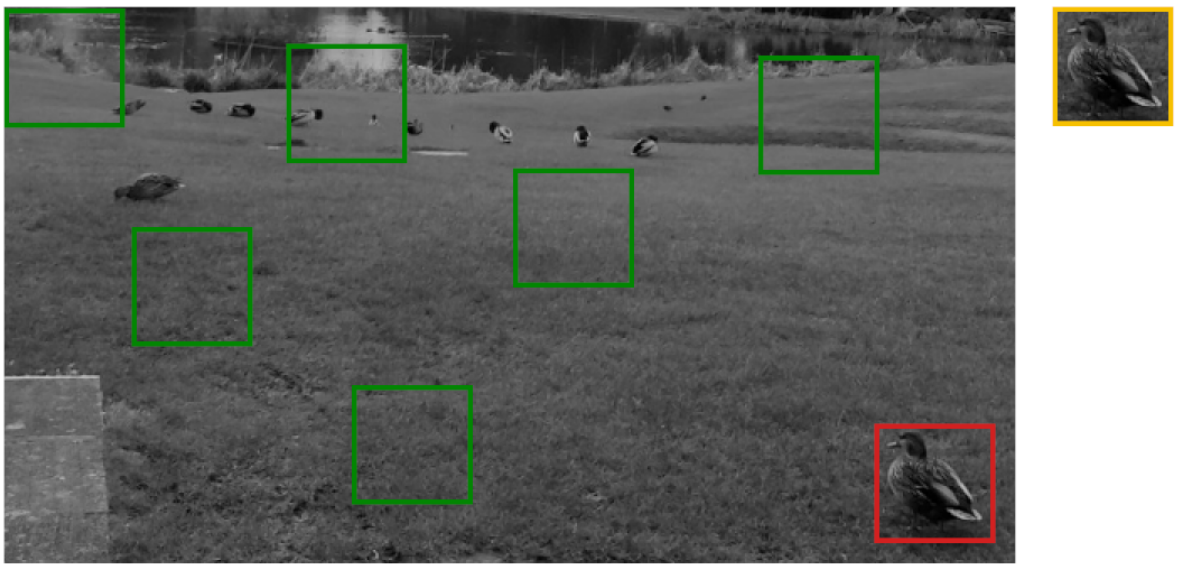


Figure 1: Template matching example. In yellow the template to be found within the image. The green squares are example candidate windows. Red square shows the best matched template

A key part of template matching is the selection of potential candidate windows. If the template matching application involves finding the best match for a template in an image with no previous information of the image i.e. the previous location of the template in the image is unknown, then the the entire image must be checked for the template. However in some applications such as motion estimation a limited search size of candidate windows may be used instead of an exhaustive search of the entire image. Reducing the search size of candidate windows to a set number of pixels around the original template's location in the image sequence can heavily reduce the computational cost but risks not missing the correct match if the search size is too small in relation to the movement in the image sequence.

At the core of most template matching application is the similarity/dissimilarity measure used to select which candidate window most closely resembles the template. Multiple different measures exist to compare the template window to the candidate windows and each have different advantages and disadvantages when it comes to computational complexity, accuracy of matches and robustness.

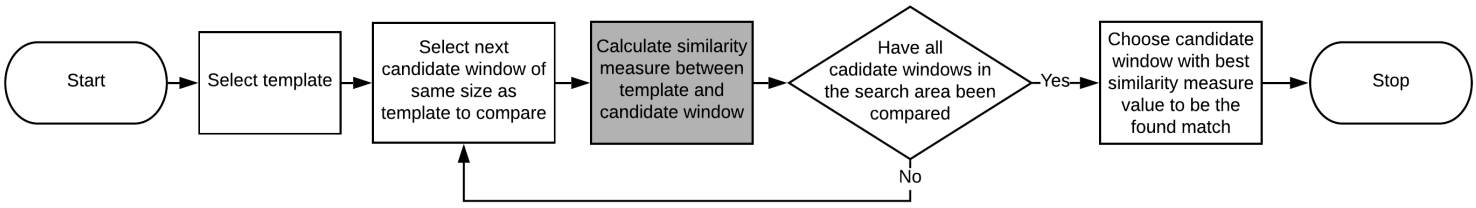


Figure 2: Template matching basic algorithm flowchart. Outlined in grey is the step at which the different similarity measured can be implemented for different results of the template matching process.

2.1.1 SAD/SSD

The Sum of Absolute Differences(SAD) and Sum of Squared Difference(SSD) are two similar and very commonly used metrics measured in template matching. These methods work by calculating the L_1 and L_2 distances respectively for SAD and SSD between the pixels in the template window and the pixels in each candidate window. The L_p norm of a vector of length N is given by

$$\|\vec{X}\|_p = (|x_0|^p + |x_1|^p + \cdots + |x_{N-1}|^p)^{1/p} \tag{1}$$

Therefore for a template window whose pixels have been placed into a vector $\vec{X_t}$ and a candidate window whose pixels have been placed into a vector $\vec{X_w}$ the SAD and SSD measures are given by:

$$SAD = \|\vec{X_t} - \vec{X_w}\|_1$$

and

$$SSD = \|\vec{X_t} - \vec{X_w}\|_2$$

[8]. The lower the value of the SAD/SSD, the better the match will be with a result of 0 being a perfect match between the template and candidate window. Both of these methods are very fast computationally and effective, however they are not robust and are affected by outliers such as impulse noise.

2.1.2 CCC

The Cross Correlation Coefficient is another commonly used matching metric as it corresponds to the optimal signal-to-noise ratio estimation [9] and avoids the issue that simple correlation has of obtaining higher value results from windows with a higher intensity which can result in brighter windows having a better correlation value than a windows that more closely match the template. This method compares each candidate window to the template window by giving an output between -1 and 1, where 1 is a perfect match, 0 is no match and -1 is the inverse of the template. The CCC calculation is given by:

$$\frac{\sum_{x=1}^{N}\sum_{y=1}^{N}\left(f(x,y)-\bar{f}\right)\times\left(g(x+u,y+v)-\bar{g}(x+u,y+v)\right)}{\left(\sum_{x=1}^{N}\sum_{y=1}^{N}\left(f(x,y)-\bar{f}\right)^2\right)^{\frac{1}{2}}\times\left(\sum_{x=1}^{N}\sum_{y=1}^{N}\left(g(x+u,y+v)-\bar{g}(x+u,y+v)\right)^2\right)^{\frac{1}{2}}} \tag{2}$$

where $f(x,y)$ is the template window, $g(x+u,y+v)$ is the template shifted by (u,v) with respect to its starting location and $\bar{f}$ and $\bar{g}$ are the corresponding window means. [2].

2.1.3 OMκ

Ordinal Measures(OM) are a type of template matching measures that calculate a matching coefficient based on the relative rank of intensities within the template instead of using the value of the intensity as is the case with the CCC and SAD/SSD. The OM correlation coefficient κ proposed in [10] is calculated by the following process:

OMκ definition. *Given windows I_1 and I_2 create a ranking of intensities within each window termed π_1, π_2 respectively. A composition permutation s is then calculated by:*

$$s^i = \pi_2^k, \quad k = (\pi_1^{-1})^i \tag{3}$$

where π_1^{-1} is the inverse permutation of π_1. s is the ranking of I_2 with respect to I_1 and under perfect positive correlation s is identical to the identity permutation $u = (1, 2, 3, ..., n)$. Once s is obtained a deviation value d_m^i that measures the distance between u and s is calculated:

$$d_m^i = i - \sum_{j=1}^{i} J(s^j \leq i) \quad where \quad J(B) = \begin{cases} 1 & if\ B\ is\ true \\ 0 & otherwise \end{cases} \tag{4}$$

Finally the measure of correlation κ is defined as:

$$\kappa(I_1, I_2) = 1 - \frac{2max_{i=1}^{n} d_m^i}{\lfloor \frac{n}{2} \rfloor} \tag{5}$$

An example of this computation taken from [10] can be seen below in Figure 3

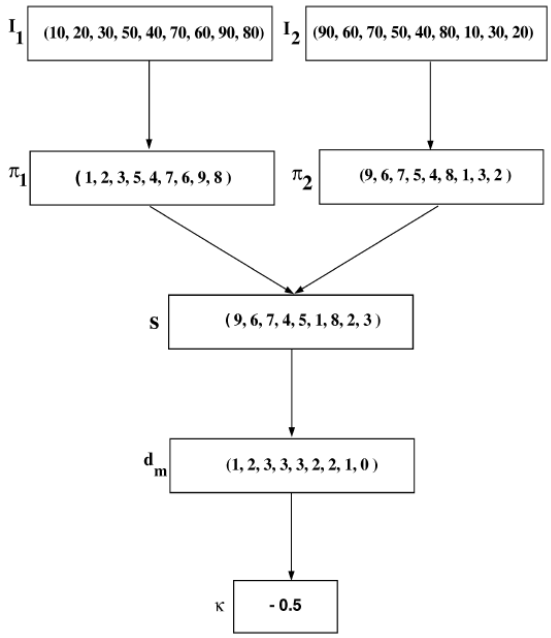


Figure 3: Example process of computing κ from [10][Fig. 1]

The final output of κ is similar to that of the CCC in that if both template and image are perfectly correlated then $\kappa = 1$, meanwhile $\kappa = -1$ if the template and candidate window are perfectly

negatively correlated. The OMκ coefficient is shown to have great robustness against impulse noise, however it suffers from poor discriminatory power based on window size. With small template sizes e.g. 3×3, only five values of κ would be possible ($\lfloor \frac{n}{2} \rfloor + 1$ where n is the number of pixels in the template). This downside is reduced as the template window size is increased [10]. The performance of κ has been tested for cloud tracking applications in [11], which showed that good performance when used within a probabilistic relaxation framework to overcome its poor discriminatory power for small window sizes, by offering robustness against impulse noise at the cost of computational complexity when compared to the CCC.

2.1.4 BBS

The Best Buddies Similarity measure is the algorithm of focus of the project. This was method proposed in proposed in a conference paper in 2015 [3] and later published in a journal in 2018 [4]. The BBS algorithm follows the traditional template matching approach of comparing template window to a candidate window. However opposed to the measures previously mentioned the BBS method is a bi-directional similarity measure which consists in counting the amount of Best-Buddies Pairs(BBPs) between a candidate window and the template window. As defined in [4]:

BBS definition. *Given two sets of points $P = \{p_i\}_{i=1}^{N}$ (Template window) and $Q = \{q_i\}_{i=1}^{N}$ (Candidate window) the BBS is the fraction of Best Buddies Pairs (BBPs) between the two sets of points. A pair of points is a BBP if p_i is a nearest neighbour of q_j in the set Q and vice-versa:*

$$bb(p_i, q_j, P, Q) = \begin{cases} 1, & \text{if } NN(p_i, Q) = q_j \wedge NN(q_j, P) = p_i \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where $NN(p_i, Q) = argmin_{q \in Q} d(p_i, q_j)$, and $d(p_i, q_j)$ is the distance measure:

$$d(p_i, q_j) = \|p_i^{(A)} - q_j^{(A)}\|_2^2 + \lambda \|p_i^{(L)} - q_j^{(L)}\|_2^2 \tag{7}$$

where (A) and (L) determine the appearance (colour/intensity) of the points and the location respectively and λ is an chosen weight parameter. Finally BBS between point set P and Q is given by

$$BBS(P, Q) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} bb(p_i, q_j, P, Q) \tag{8}$$

The BBS method has three main properties that make it advantageous to other methods:

1. It only relies on the pairs of points that are BBPs and therefore can reject the rest of the points as outliers making it robust to noise.

2. It can find the bi-directional inliers in the data without needing prior knowledge of the data or its deformation.

3. Uses rank as it counts the number of BBPs instead of using the distance metric directly.

Because of these three properties, the paper [4] shows that the BBS method is robust to high levels of outliers in the data and that that the BBS method is more likely to match points from the same distribution meaning that it matches important foreground points (points taken from the same object eg: a persons face in the template) while not forcing background points to match and

therefore the BBS method does not need to have to model the background and foreground.

The core computational operation of the BBS algorithm is the calculation of the BBPs. To do this, the BBS algorithm implementation proposed in [4] splits the template and candidate window into a patch coordinate system of $k \times k$ non-overlapping patches of pixels that represent each 'point' thats considered for BBPs. As shown in Equation 7, to calculate the distances between two points both their location in the window and the difference in colour/intensity are added together, using a weight factor for the location parameter. To compute the distance between each pair of points a distance matrix D is constructed. Looking at Equation 6, $NN(p_i, Q)$ is the minimal element in the ith row of D and $NN(q_j, P)$ is the minimal element in the jth column of D [4]. The computational complexity of exhaustively computing the BBS for every candidate window in an image given $L =$ image width $\times$ height, $l =$ template width $\times$ height, k is the patch size and g is the number of dimensions of the feature space (eg: 3 for an RGB image) has been calculated to be the following in [4] :

$$O\Big(\frac{Ll^4g}{k^4}\Big) \tag{9}$$

This computational load is can be higher than previous measured such as SSD which would be $O(Llg)$. The BBS paper [4] proposes an optimisation technique that involves saving and reusing distance matrices calculated which can reduce the computational load to:

$$O\Big(\frac{Ll^2g}{k^4}\Big) \tag{10}$$

The discriminatory power of BBS is directly correlated with the chosen template size and patch size. As the output of BBS is given by counting the number of BBPs within a template and a candidate window, the best possible output is that at which every point is a BBPs. Because of this, for a square template of m pixels wide using non overlapping patches of k pixels wide, the number of different outputs of BBS is limited to the total number of points given by $\lfloor \frac{m}{k} \rfloor^2$

2.2 Full-field motion estimation & probabilistic relaxation

In remote sensing applications it is often required to track the motion of large non-rigid objects for scientific or other purposes. A key application of this is cloud tracking from satellite footage in which the motion vectors from the cloud movement can be used as input for weather forecasting. To generate motion vectors fields of large non-rigid objects such as clouds and glaciers a commonly used technique is template matching in conjunction with probabilistic relaxation [1], [2], [12], [13]. This technique is split into two main sections: Template motion estimation and generation of candidate matches, and probabilistic relaxation labelling.

2.2.1 Template motion estimation and generation of candidate matches

The first part of determining the overall motion flow of large non-rigid objects requires aligned digital images from a motion sequence. These images are then divided into equal sized templates. Each template has a correlation surface generated based on the results of template matching for that template from one image to another. Many different similarity measures can be used to generate the correlation surface. Some metrics have been proven to be especifically effective for this first stage such as the CCC in [14] and OMκ in [11]. In many remote sensing applications including cloud tracking it is the case that the images used are often low contrast and experience non-rigid

motion and deformation [2]. Because of this the highest peak in the correlation surface generated for a template may not be the correct match and many other points in the correlation surface could be of similar value. Therefore the top n matches from the correlation surface that meet certain criteria such as being above a set threshold are selected for each template in the image and are then converted to motion vectors based on the location of the matched candidate window in the second image. Depending on the similarity metric used or by selecting peaks from different modes in the correlation surface [13], a better subset of n vectors can be selected as the input of the relaxation method. Once these calculations are carried out for all templates, the relaxation labelling technique can be applied.

2.2.2 Probabilistic relaxation labelling

Probabilistic relaxation labelling is a technique that aims to assign specific labels to objects. It does this by iteratively refining the probability that each label is assigned to an object. In this application, each template is considered an object and each of the n best candidate vectors are considered the possible labels for that object. By looking at the probabilities of each vector for each template and the probabilities and vectors of the templates surrounding it, the best velocity vectors can be selected that match the overall movement and therefore smoothing the motion field. To begin the relaxation labelling, initial probabilities must be assigned for each template J to have vector template j. This is given by:

$$P^{(0)}(J \to j) = \frac{\rho(J \to j)}{\sum\limits_{\lambda \in C_J} \rho(J \to \lambda)} \tag{11}$$

where $\rho(J \to j)$ is the value of the CCC (or other measure used normalized from 0 to 1) for template J and vector j, and C_J is the set of candidate vectors for the template. Once the probabilities for each template and vector is set, the probabilities for each velocity vector are updated iteratively using the non-linear relaxation formula from [15]:

$$P^{(n+1)}(J \to j) = \frac{P^{(n)}(J \to j)Q(J \to j)}{\sum\limits_{\lambda \in C_J} P^{(n)}(J \to \lambda)Q(J \to j)} \tag{12}$$

where $Q(J \to j)$ is the support function which determines how compatible each candidate vector $J \to j$ is compared to those in the local neighbourhood $\mathcal{N}_{\mathcal{J}}$. The support function is calculated by using the formula below, which calculates the product of the sum of the probabilities of each velocity vector in the neighbourhood multiplied by a mutual information measure.

$$Q(J \to j) = \prod_{I \in \mathcal{N}_J} \sum_{i \in \mathcal{C}_I} P^{(n)}(I \to i)R(J \to j, I \to i) \tag{13}$$

The mutual information measure $R(J \to j, I \to i)$ is a compatibility coefficient which measured the likeness between two vectors. The calculation for the mutual information measure can be carried out in different forms. [16] uses a formula which takes into account the vectors magnitudes $(S_{J,j}, S_{I,i})$, orientation $(\theta_{J,j}, \theta_{I,i})$ and distance $(D_{I,J})$ between them which can be seen below in Equation 14. Another version of the mutual information measure is used in [12] is shown in Equation 15. This formula only takes into account the distance between the vectors $(D_{I,J})$ and the vectors horizontal $(\Delta x_{J,j}, \Delta x_{I,i})$ and vertical $(\Delta y_{J,j}, \Delta x_{I,i})$ displacement instead providing a more computationally efficient calculation of the mutual information measure. The output of both implementations is a value between 0 and 1 with 1 being a perfect match between both vectors.

$$R(J \to j, I \to i) = \exp\left(-\frac{|\theta_{J,j} - \theta_{I,i}|}{\sigma_\theta}\right) \times \exp\left(-\frac{|S_{J,j} - S_{I,i}|}{\sigma_S}\right) \times D(I,J) \tag{14}$$

$$R(J \to j, I \to i) = \exp\left(-\frac{|\Delta x_{J,j} - \Delta x_{I,i}|}{\sigma}\right) \times \exp\left(-\frac{|\Delta y_{J,j} - \Delta x_{I,i}|}{\sigma}\right) \times D(I,J) \tag{15}$$

σ_θ and σ_S in Equation 14 and σ in Equation 15 are parameters to be tuned for each specific application of relaxation labelling that are used to change the rate of convergence of the vectors.

There is a main limitation of relaxation labelling with template matching and that is that relaxation labelling will smooth out the motion field, selecting the best possible vector out of the best n available. However if all vectors for a template are erroneous, the relaxation procedure will aim to select the least worse one of them. To fix these remaining vectors two approaches are commonly used. The first is the use of a no-match label as seen in [1]. This label shown below in Equation 16 can be added alongside the rest of the vectors for each template.

$$\rho(J \to \phi) = 1 - max\{\rho(J \to j)\}, \qquad j \in C_J \tag{16}$$

The second alternative to fix bad vectors that remain after relaxation labelling without using a no-match label is to use a post filter. This technique has been used in [12] and [17]. The use of a post filter to improve the motion field allows remaining bad vectors to be removed and replaced with a vector that is consistent with the nearby neighbourhood of vectors and it can also be used to set a vector for a template that had no match in the initial generation of vectors, but still has consistent neighbourhood vectors.

To apply the filter as proposed in [12], for each vector the median horizontal and vertical components of the eight nearest neighbouring vectors are calculated to be Δx_{med} and Δy_{med}. If more than half the templates in the neighbourhood have no match the filter is not applied. If there are enough neighbourhood vectors then the horizontal and vertical components of the vector in question Δx_J andΔy_J are then compared with the median components and if

$$abs(\Delta x_J - \Delta x_{med}) + abs(\Delta y_J - \Delta y_{med}) > k(abs(\Delta x_{med}) + abs(\Delta y_{med})) \tag{17}$$

then the vector is replaced the median vector. k is a smoothness constraint that is used how strict the filter is when replacing vectors.

3 Project development tools and software

This project is primarily computational and investigative and thus all of the engineering tool decisions come down to the choice of programming language, development environment and software used.

3.1 Programming language and development environment

For the choice of programming language, the main decision was to choose between C, MATLAB and Python. Each of these have several advantages and disadvantages

Table 1: Programming language comparisons

Language	Pros	Cons
MATLAB	• Easy to build quick prototypes • Libraries and functions for image processing and graphs are built in • Workspace window allows for variables to be saved for tests • IDE and programming language are a single package • Great debugging tools	• Code runtime may be slower due to IDE overhead (altough can be improved with MEX files) • Not open source and needs license to use
Python	• Jupyter notebooks allow for variables to be saved for testing • Large amount of libraries exist for tasks such as image processing and graph plotting • Language and most custom libraries are free to use	• No built in IDE • Harder to set up and get prototypes working compared to MATLAB
C	• More control over memory used • Optimized C code can be faster than MATLAB or Python	• Hard to quickly prototype in • Harder to code in than Python and MATLAB • External image processing libraries are harder to use

In the end MATLAB was chosen as the programming language and IDE in which to carry out the project. The main reasons for this choice were that the MATLAB workspace window, console and debugging environment are extremely intuitive to use and useful when compared to that of most IDEs used with C and Python (such as visual studio for C or Jupyter notebooks for python)and thus will allow for better and faster coding and debugging of the projects code. MATLAB also contains several in-built functions as well as an Image Processing Toolbox that provide in-built functions to carry out different tasks such as quickly reading and displaying images or different types of noise to images which will allow more time to be focused on the core part of the project as opposed to coding these functions in other languages that may not have them such as C or Python without importing other libraries. Finally although C would be the better programming language for optimal computational speed, since the key part of the project is to compare different template matching similarity metrics to BBS, as long as all of them are implemented in the same environment(MATLAB) the potential runtime differences between optimized code in C and optimized code in MATLAB are irrelevant.

3.2 Image editing software

Part of this project required cloud image to be closely analysed to create a set of manually labelled ground truth templates for tests of the template matching methods. To do this software capable of

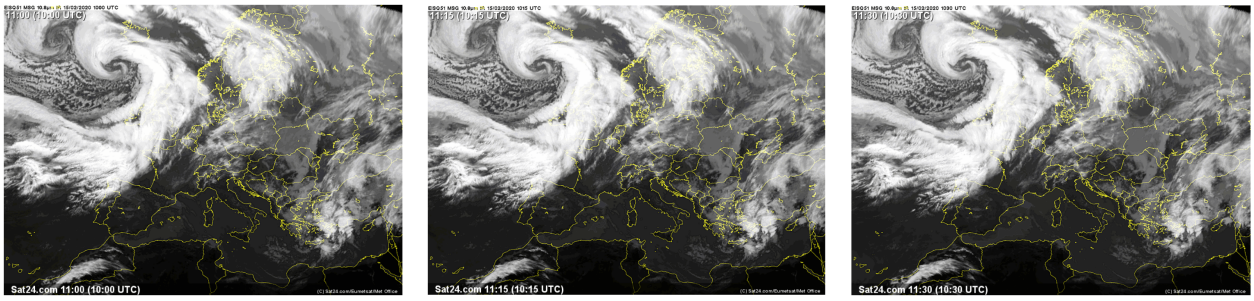
opening two image files simultaneously while allowing close zoom and indexing of the pixel locations within the image was needed. Although this could poentially be done within MATLAB, this task was able to be performed much faster using image editing software. Gimp was chosen for this due being free and open source and due to prior personal experience with the software.

4 Test data collection and creation

Since the core objective of the project is to test the viability and performance of BBS for use in cloud tracking, the first step of the project was to obtain suitable test image sequences and process them into a suitable format.

4.1 Obtaining test image sequences

The test images of clouds were all taken from Meteosat Second Generation (MSG) satellites. The SEVIRI instrument onboard the MSG satelittes takes images every 15 minutes with a resolution of 3 km/pixel for the infrared and other visible channels [18]. Three different sets of data were collected. Set 1 was obtained from Sat24.com, an open source website that provides images of clouds over Europe and other areas of the world from MSG satellites and archives then. The Set 1 images seen below in Figure 4 were taken during Storm Dennis on 15th February 2020 and provide a large range of different of cloud movement, speed and formations that will be useful for determining the effectiveness of different template matching methods

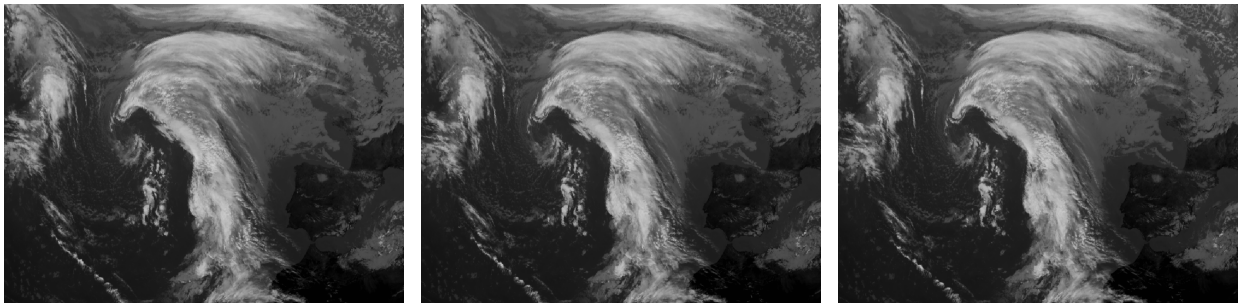


(a) 11:00 (b) 11:15 (c) 11:30

Figure 4: Set 1 test images. High velocity clouds during Storm Dennis over Europe. Each image is 15 minutes apart.

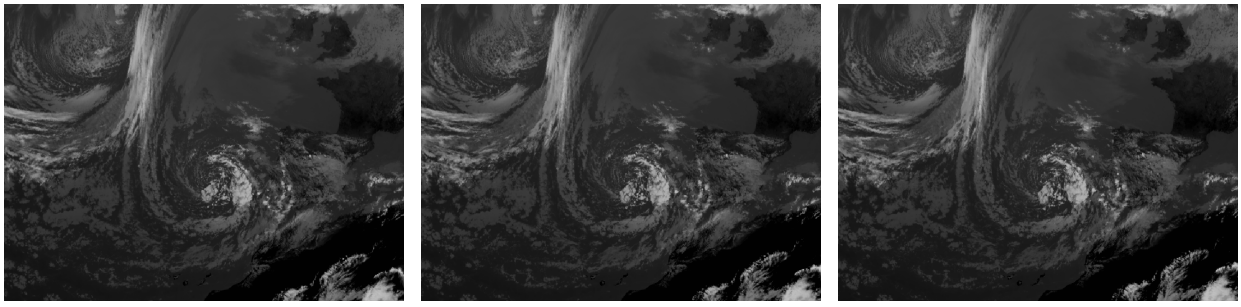
The images for Sets 2 and 3 were obtained at later dates across March and early April from EUMETSAT mapviewer[1] 0 degree MSG, which archives the images taken from the satellites for two weeks time (because of this time limit the images from Set 1 were not able to be obtained from this same source). These present more common type of cloud movements as opposed to a large storm with very fast moving winds in Set 1. For these three sets of data, images from the infrared channels from the MSG satellite were taken. The images were taken at both 15 minutes and 30 minute interval. Although most test will be carried out with the 15 minute time difference as this is the standard interval used by MSG satellites, the 30 minute interval image was also included to be able to test the effects of greater distortion from the cloud movement due to larger time gaps between images.

[1] https://eumetview.eumetsat.int/mapviewer/



(a) 11:00 (b) 11:15 (c) 11:30

Figure 5: Set 2 images. Large cloud formation over the Atlantic Ocean slightly west of Europe (Iberian Peninsula is visible on the right side of the images)



(a) 11:00 (b) 11:15 (c) 11:30

Figure 6: Set 3 images. Different cloud formation in a similar location to that of Set 2

4.2 Image pre-processing

4.2.1 Greyscale convertion and overlay removal

Once all the image sets were acquired, they had to be processed in order to be able to be properly used for the template matching tests. The first step was to convert all images to greyscale in order to reduce the computational load of the template matching algorithms. All the images obtained from the MSG satellites are greyscale images, however when downloaded from the internet, the .png files contain separate colour channels that add up to greyscale values. For the images from Sets 2 and 3, through the use of the `rgb2gray()` MATLAB function [19], each image was easily converted to a single channel image. The images from Set 1 however had an issue of having yellow coloured overlay indicating country borders. Figure 8a shows that when converted to greyscale these borders showed up as a bright white intensity that could potentially skew results during template matching. In order to remove the country borders, each channel of the image was displayed on a different MATLAB subplot. From this it was found that the third channel of the image contained no traces of the country border overlay, due to the borders being a bright yellow colour. Figure 7 shows the three image channels plotted and the mask used to remove the borders created from the difference between the image channels.

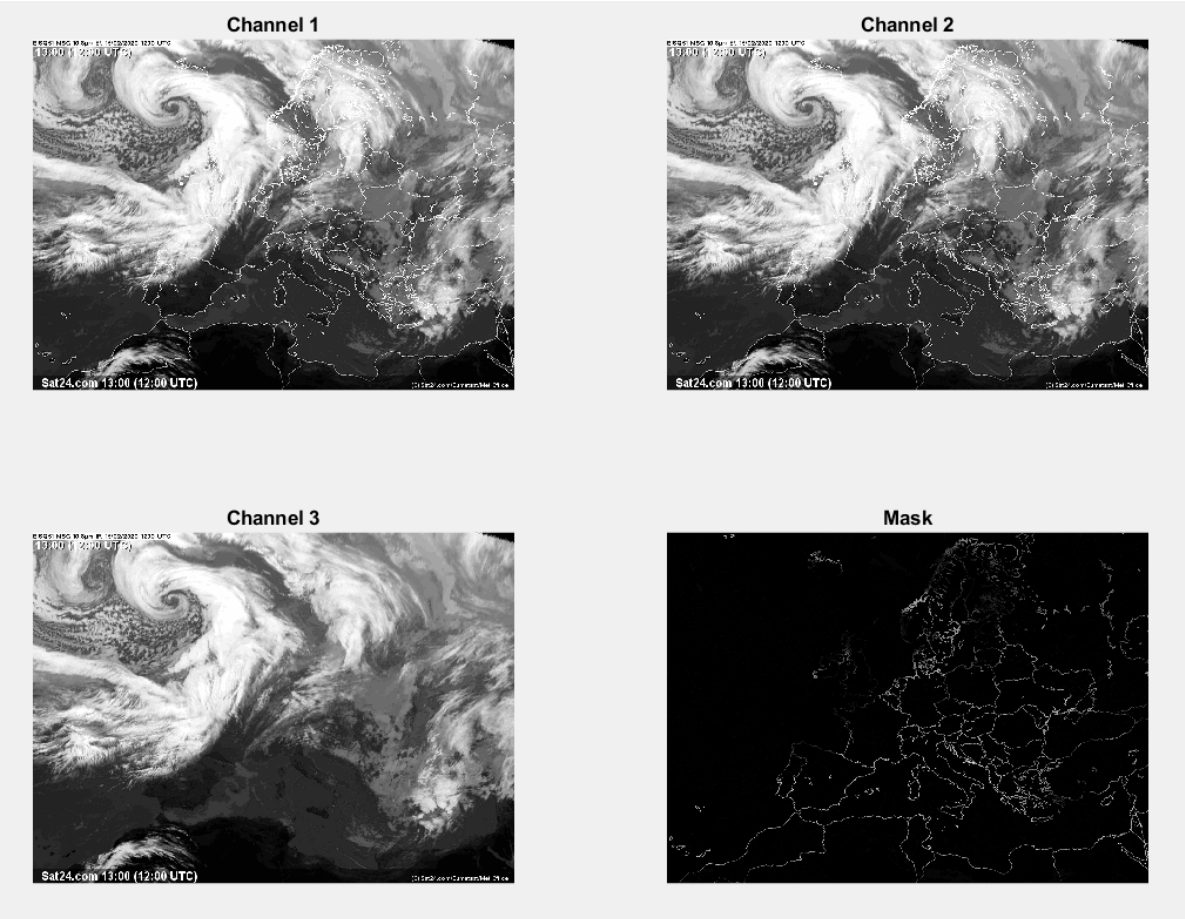
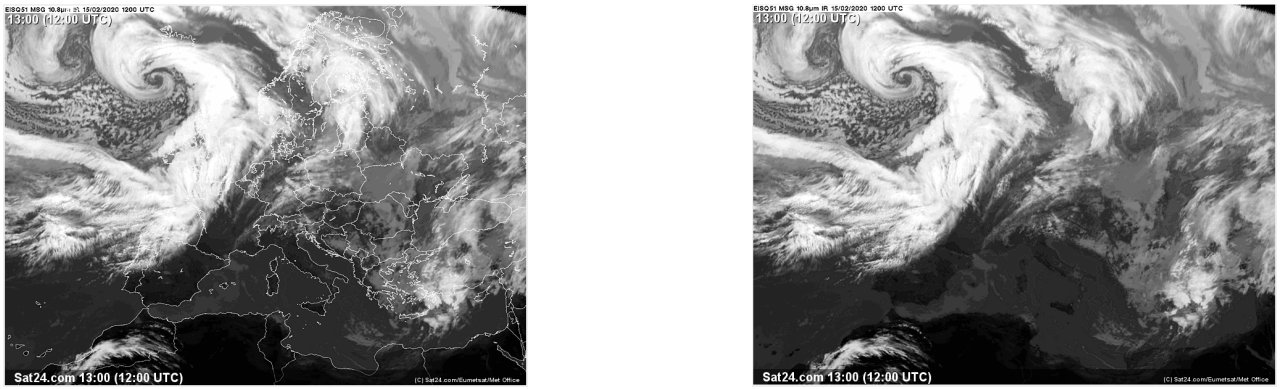


Figure 7: Set 1 image displayed by each different colour channel. The first two channels contain the image borders while the third channel does not. On the lower right the mask created by taking the differences between the channels is shown.

A greyscale image without the country borders was able to be obtained for the images of Set 1 seen below in Figure 8b next to the initial greyscale image with the greyscale converted country border overlay.



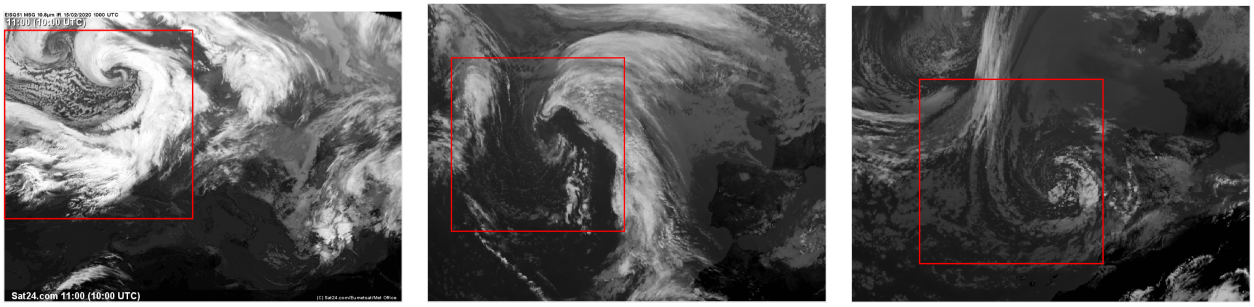
(a) Greyscale image with country borders

(b) Greyscale image with removed borders

Figure 8

4.2.2 Image area selection

Once all the image sets were converted to greyscale, a 400 × 400 pixel area so that all test images across sets would be the same size. This size was chosen to have enough area on the image to produce large vector fields of over 500 vectors even when using medium size templates (15×15) to properly perform the relaxation labelling tests while still having short enough runtimes that allow multiple tests to be run in order to test and optimize different parameters. In Figure 9 below, the areas selected for each set of images is marked with a red area.



(a) Set 1 area

(b) Set 2 area

(c) Set 3 area

Figure 9: 400×400 pixels area selected for each image set is shown in red. The area selected in each image set has the most features of interest for performing template matching tests

Since the different sets of images have different total size and the index of the 400x400 area within the image is not the same for all, the indexes for the area of interest was written in a text file for each image set. For the area selection, greyscale conversion and country border removal, a function called `load_image()` was created to load an image into the MATLAB environment, given the data folder directory, image set number and image time (these last two arguments were used in the naming of each image file, eg: "1100_set1.png" for the image at 11:00 from set 1). The flowchart in Figure 10 below shows how this function works.

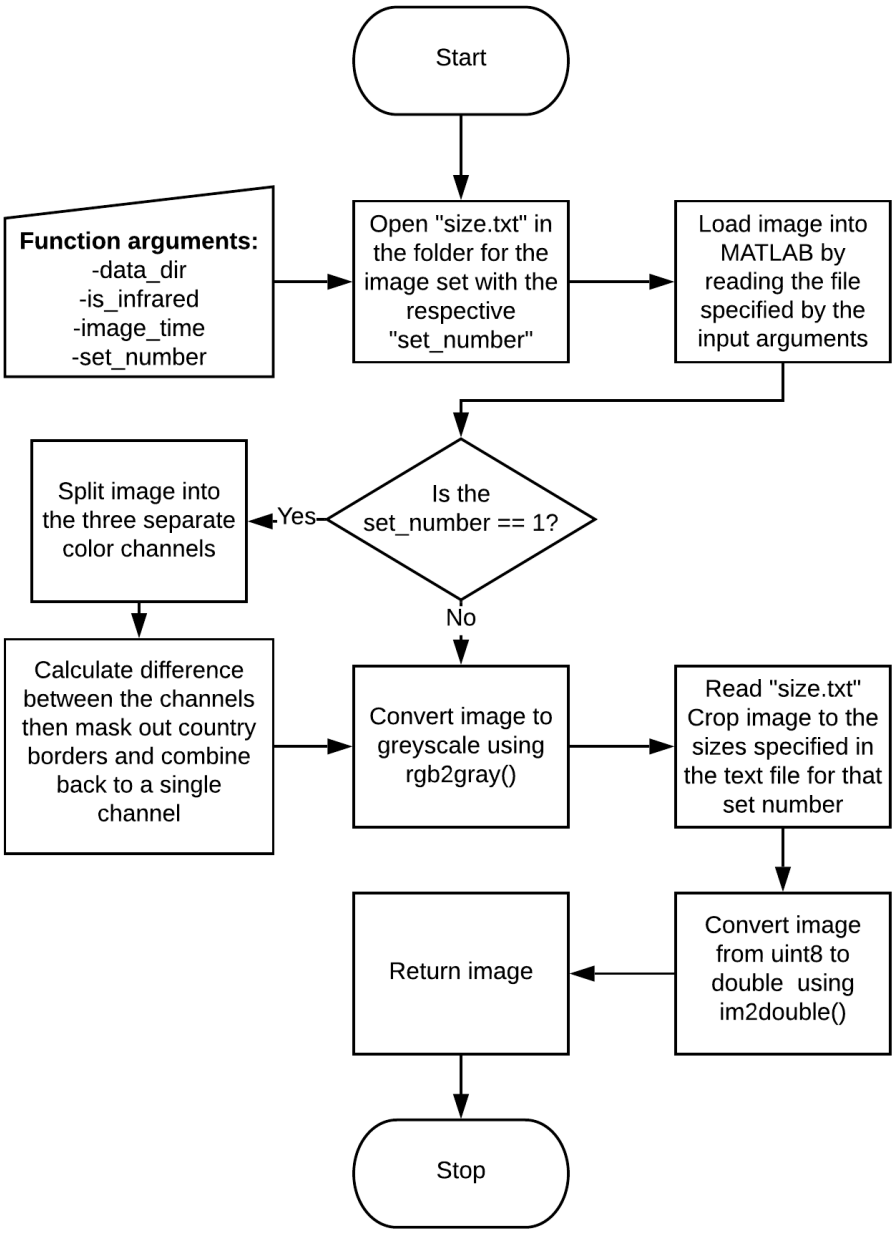
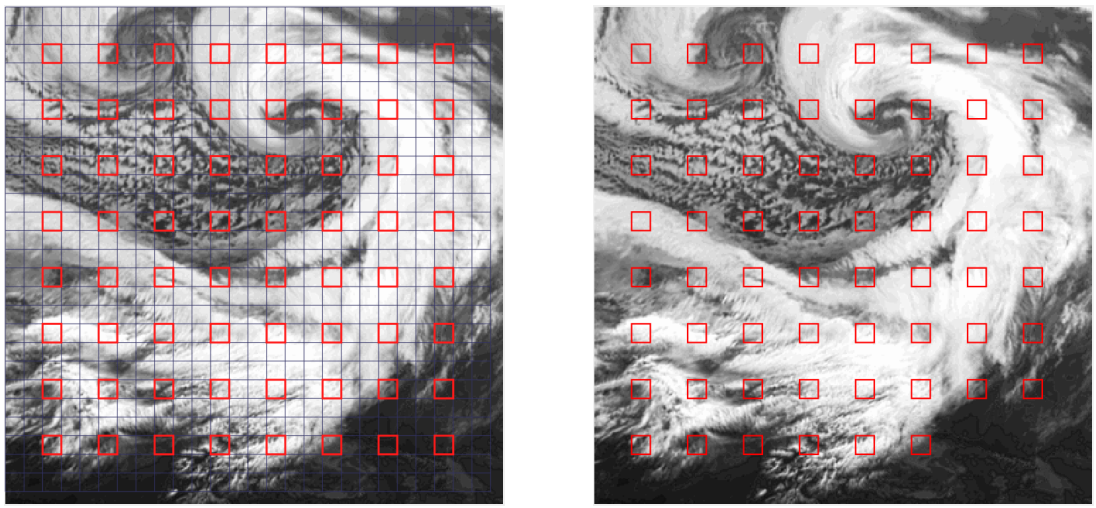


Figure 10: `load_image()` function operation flowchart

4.3 Bounding box test data

In order to test the performance of template matching algorithms, large pre-labelled data sets such as the Visual Tracking Benchmark introduced in [20] are often used as seen in [4]. However these large data sets are commonly made up of images of well defined known objects such as vehicles, animals or human faces. This presents a problem when trying to evaluate the accuracy of different methods when tested on cloud motion tracking applications as data sets containing templates and their respective ground truth estimates for cloud movement are not readily available. Previous work has focused mainly on qualitative analysis to analyze performance of matching methods for cloud tracking as a large survey of quantitative test data for cloud tracking has not been carried out [21]. Therefore in order to measure the effects of different BBS parameters when applied to cloud tracking a custom testing data set was created using the images from Sets 1, 2 and 3 to be able to perform quantitative testing.

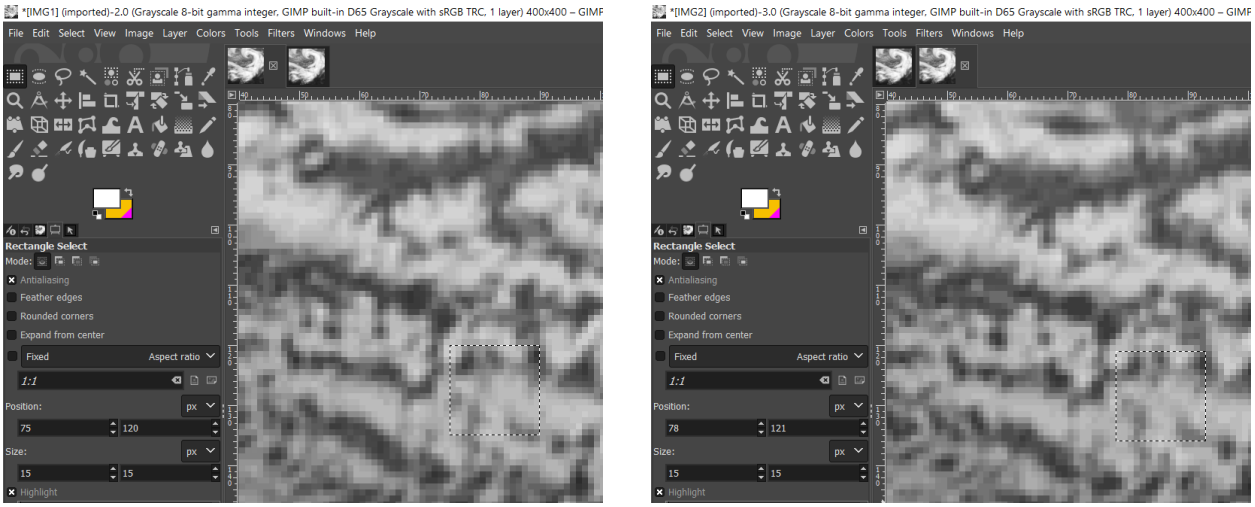


(a) (b)

Figure 11: (a) Set 1 image split into 15×15 pixel templates with one every three templates being outlined in red. (b) The test templates that will be used for testing outlined in red, note two templates have been removed at the bottom right area of the image due to having no cloud features within them.

To create the testing data, the image was first split into 15×15 pixel templates. Small template sizes are usually favoured for cloud tracking due to the non rigid motion of clouds causing large deformations for large templates. Previous work on template matching for cloud tracking typically uses small template sizes such as 8×8 in [17]. However due to the low discriminatory power of OMκ and BBS for very small size templates, a medium 15×15 size template was chosen which can give OMκ 113 different possible coefficient values and BBS a total of 25 different values when using a patch size of 3×3. The test templates were then selected at intervals of one test template every 3 templates, giving a total maximum of 64 test templates for each image set. The test templates were selected at equal intervals in order to avoid biasing towards specific areas of each image that may give better results and to guarantee a variety of different templates from different areas of the images. Some templates from each image set had to be removed due to there being no distinguishable cloud features in the the selected area. A total of two templates were removed from Set 1, four from Set 2 and none from Set 3 giving a total of 186 test templates across the three image sets. The selected test templates for Set 1 are shown in Figure 11. Once all the test templates for each image set were selected, the next step was to manually find the best matching

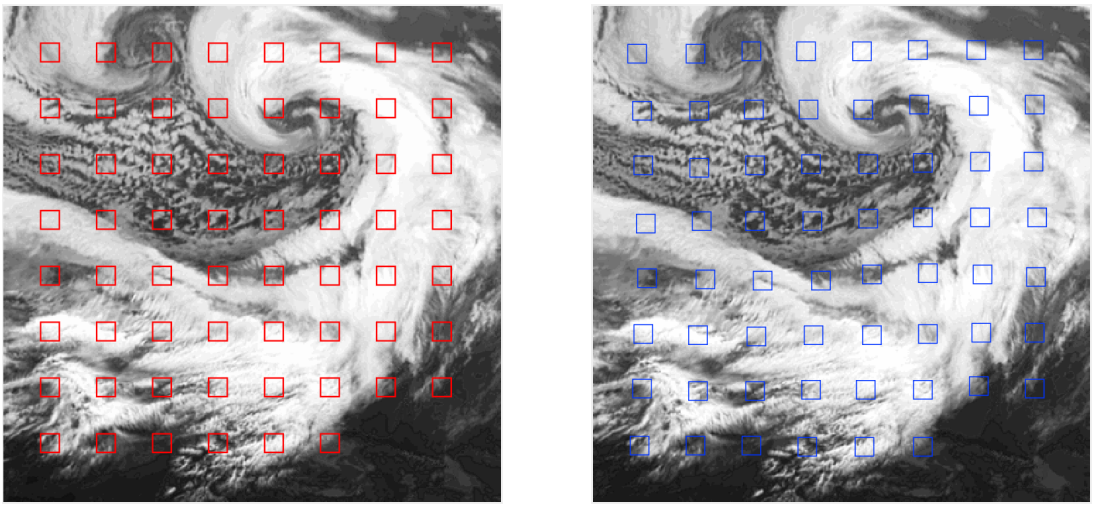
template in the next image of the sequence for each image set. Using Gimp to open both image files simultaneously then quickly shifting between the two images and using the rectangle tool to mark the original template on the first image, the ground truth template was found on the second image. An example of this for a template in Set 1 is seen in Figure 12 below:



(a) (b)

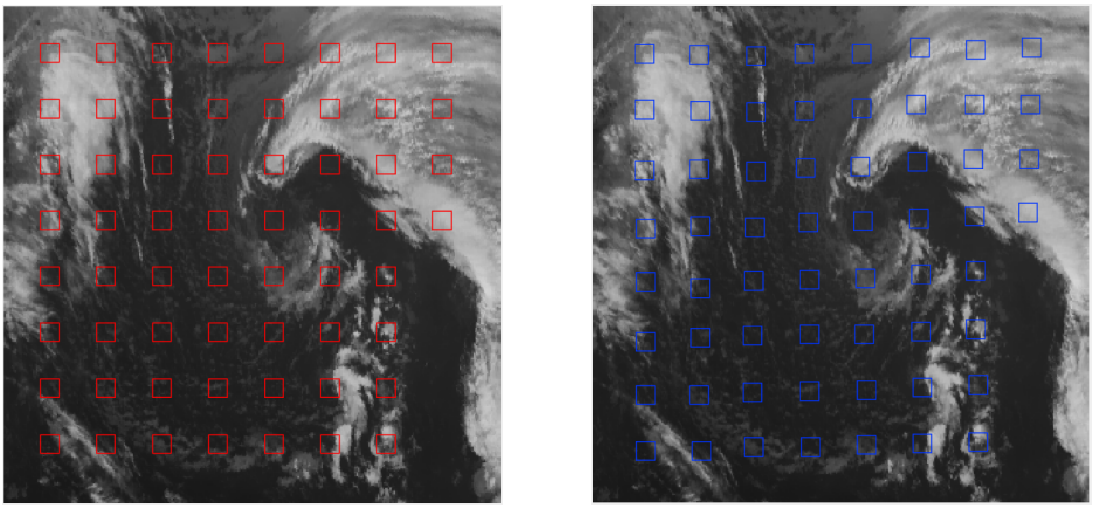
Figure 12: (a) Test template at coordinates (75, 120) on the first image of Set 1. (b) Ground truth template found at coordinates (78, 121) on the second image of Set 1

The coordinates for each ground truth template was then written down in a .txt file in which the first row of text contained the X coordinates each template and the second row of the text file contained the Y coordinates of each template. The removed template coordinates were written down as -1 for both X and Y to act as a flag indicating to ignore those templates. Figures 13, 14 and 15 below show the test templates and ground truth templates for image sets 1, 2 and 3 respectively.



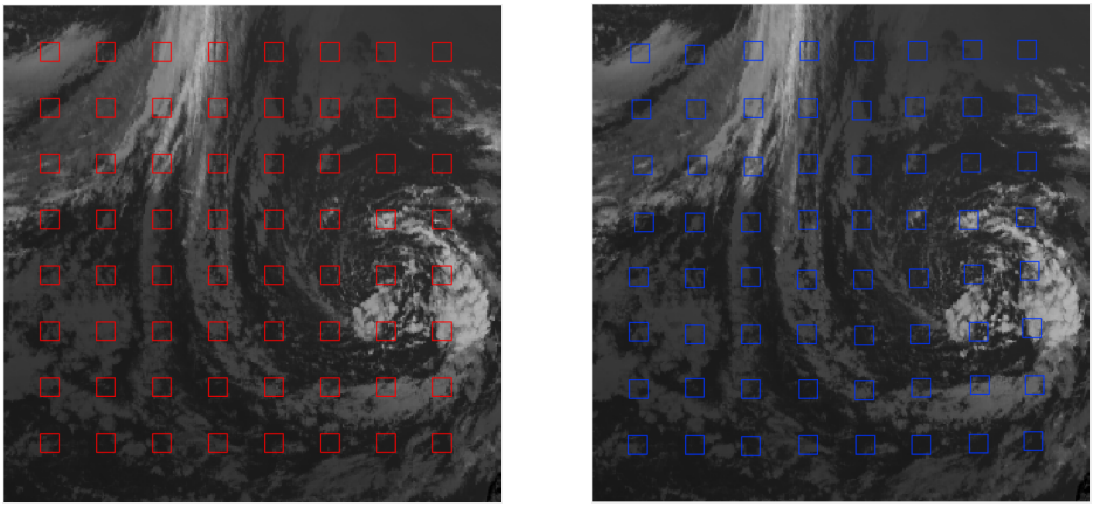
(a) (b)

Figure 13: (a) First image test templates for Set 1. (b) Ground truth templates on second image for Set 1



(a) (b)

Figure 14: (a) First image test templates for Set 2. (b) Ground truth templates on second image for Set 2

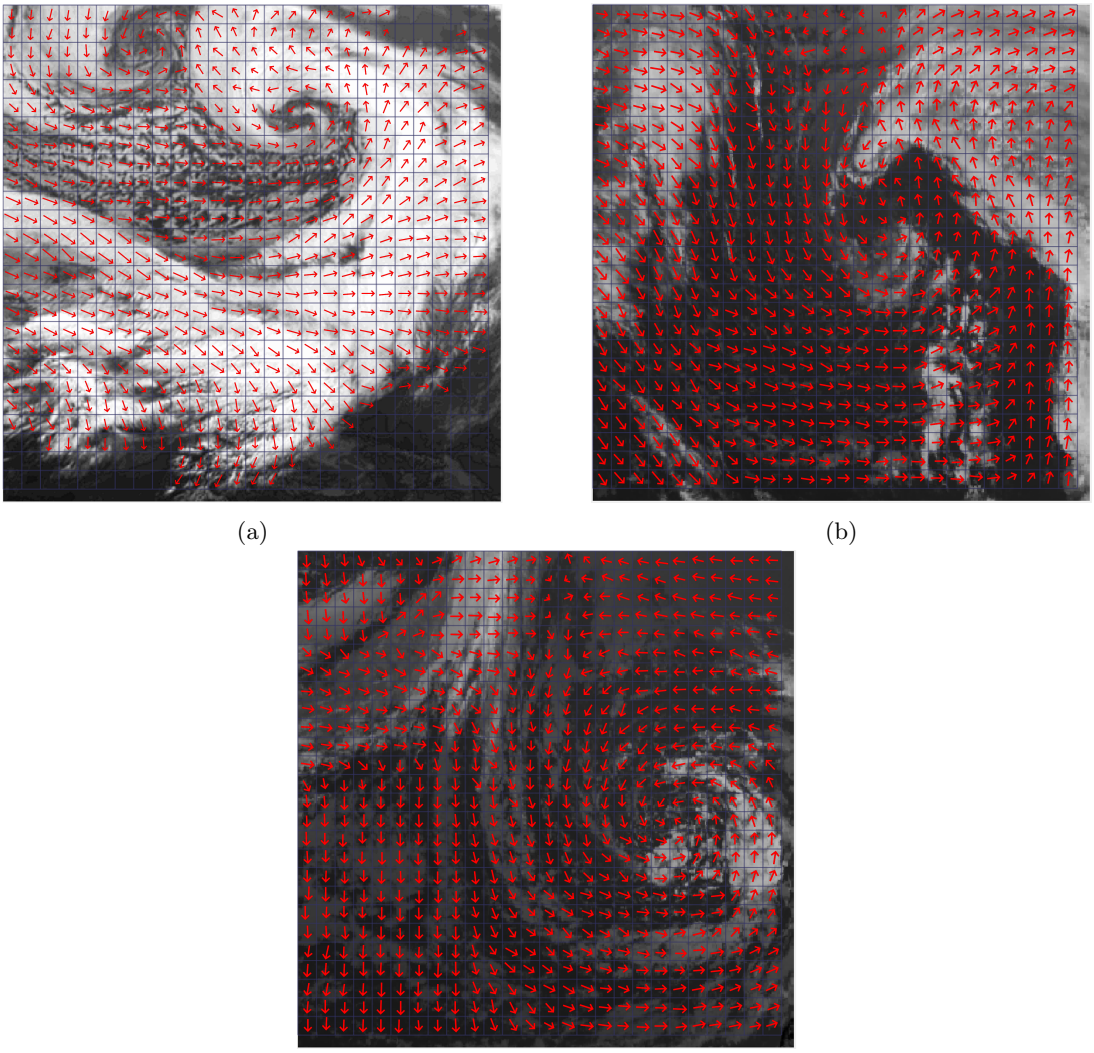


(a) (b)

Figure 15: (a) First image test templates for Set 3. (b) Ground truth templates on second image for Set 3

4.4 Motion field manual estimation

When testing the results of full field motion estimation using relaxation labelling, a ground truth vector field can be used to check weather the overall direction of the generated vectors is correct as well as to check if any individual vectors are greatly misdirected. Previous work such as [17] has used Numerical Weather Prediction (NWP) models to generate the estimated motion field. For this project, the approximated motion field were generated by manually using a similar approach to that used above in Section 4.3, by looking at the approximate motion of each 15×15 template and drawing an arrow in the direction of the template. Doing this creates an estimated motion field that while it cannot be used for quantitative testing, it is useful as a baseline for comparing the generated motion fields. The final motion fields are shown in Figure 16 below.



(a) (b)

(c)

Figure 16: (a), (b) and (c) show the manual motion field estimation for the Sets 1, 2 and 3 respectively. Each arrow shows the manual approximatted direction of movement for that template between the first image and the next image in the image set

5 Template matching implementations

5.1 Non-BBS method implementations

The first step to implement the different template matching metrics was to code a model function that would carry out the basic template matching algorithm of iterating and comparing each candidate window in the selected search size. A flowchart for this is outlined in Figure 17 which is a more implementation focused flowchart to that of Figure 2. This function would serve as the starting point to implement the SAD/SSD, CCC and OMκ methods and has a specific labelled process in the flowchart in Figure 17 in which the main difference in calculation between methods occurs. The function uses the following input arguments:

- `img`: Second image to be searched to find potential matches of template.
- `tmplt`: Template to be searched in the image.
- `t_coord`: Two element array holding the x coordinate(column) and y coordinate(row) in the first and second slots respectively of the top left pixel of the template in the original image. Used to determine where to search for candidates image.
- `search_sz`: How far to search in all directions for candidate windows in relation to the coordinates of the template.

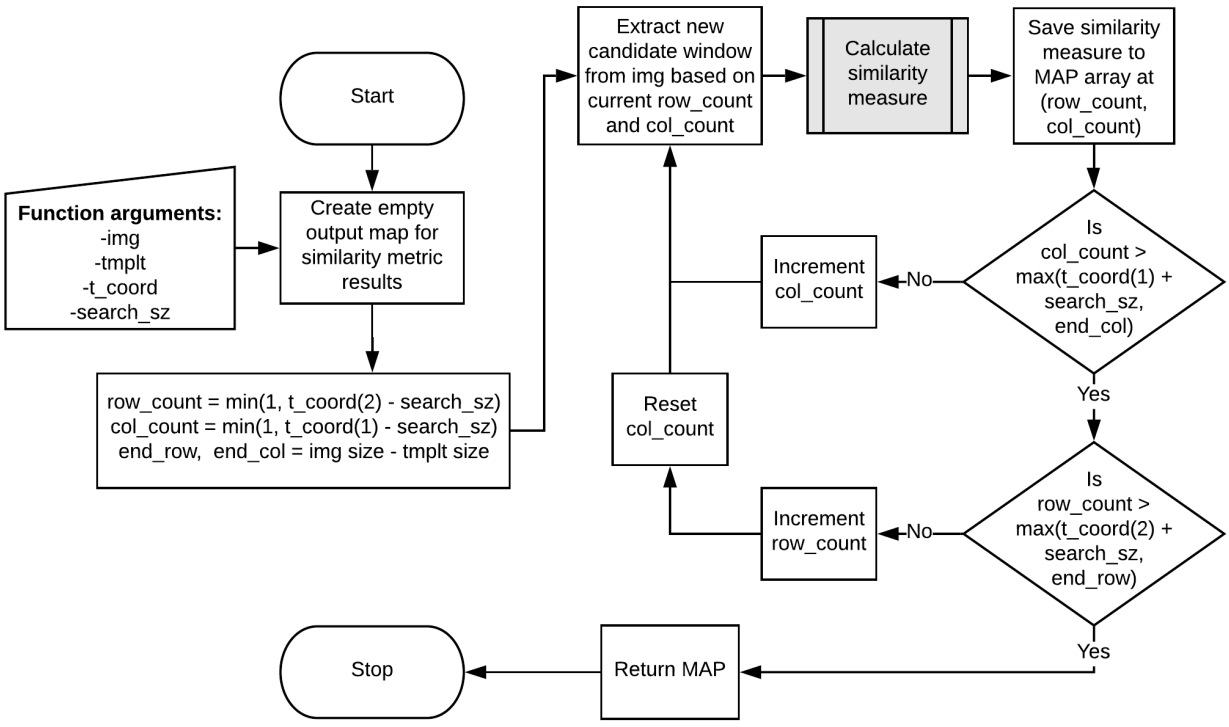


Figure 17: Baseline template matching function flowchart

The output of the function is a map of the similarity measure output for each possible candidate window in the search area. A followup function called `get_bestn_vectors()` was coded to search and obtain the best n results from the map and return the map values alongside their respective location within the image. The input arguments for this function are:

- `vector_map`: Map containing the values of the output coefficient of each candidate window in the section of the image that the template matching function searched.
- `threshold`: Any numbers within the map that are below this value will be set to -infinity (-Inf in MATLAB) to allow values below a certain threshold to be removed from the results.
- `n_best`: Number of best matches to return. E.g. `n_best` value of 5 will return the best 5 matches in the map.

This function makes use of MATLAB's `maxk()` function [22], which returns the largest k elements from an array. In order to make use of this function, the map was converted into a 1 dimensional array, then the indexes of the best matches from the `maxk()` function were converted back into a 2 dimension format. The code snippet in Listing 1 shows the section of the `get_bestn_vectors()` function that performs this task. An example output of this function is shown in Figure 18 using a 4×4 size array of randomly generated numbers, to find the largest 5 numbers within it and their respective location, without using a threshold.

```
1  %Find n_best values in vector map.
2  %The values get placed in max_n while the indexes in the converted 1D array get ...
       placed in indx
3  [max_n, indx] =  maxk(vector_map(:), n_best);
4
5  %Get size of map
6  map_size = size(M);
7
8  %Convert 1D indexes to row, column indexes
9  [row, column] = ind2sub(map_size, indx);
10
11 %Place data into t_data to be returned
12 t_data = [max_n, column, row];
```

Listing 1: Section of code from `get_bestn_vectors` that gets best matches from the map and places them into an array with the value its coordinates

4x4 double

	1	2	3	4
1	0.7431	0.7060	0.0971	0.9502
2	0.3922	0.0318	0.8235	0.0344
3	0.6555	0.2769	0.6948	0.4387
4	0.1712	0.0462	0.3171	0.3816

(a)

5x3 double

	1	2	3
1	0.9502	4	1
2	0.8235	3	2
3	0.7431	1	1
4	0.7060	2	1
5	0.6948	3	3

(b)

Figure 18: (a) Randomly generated array of numbers to be used as input of the `get_bestn_vectors` function. (b) Top 5 results of the array. The first column shows the ordered values. The second and third column shows the column and row index respectively

5.1.1 SAD implementation

The first similarity metric implemented was the SAD. As defined in Section 2.1.1 the SAD/SSD similarity measure focuses on calculating the L_p norm distance between the pixel values in the template and candidate window. For the implementation only SAD was used as both metrics had very similar results to each other when compared to the BBS method in [4] and SAD has a faster computation time. A flowchart for the implementation of this similarity measure is shown in Figure 19.

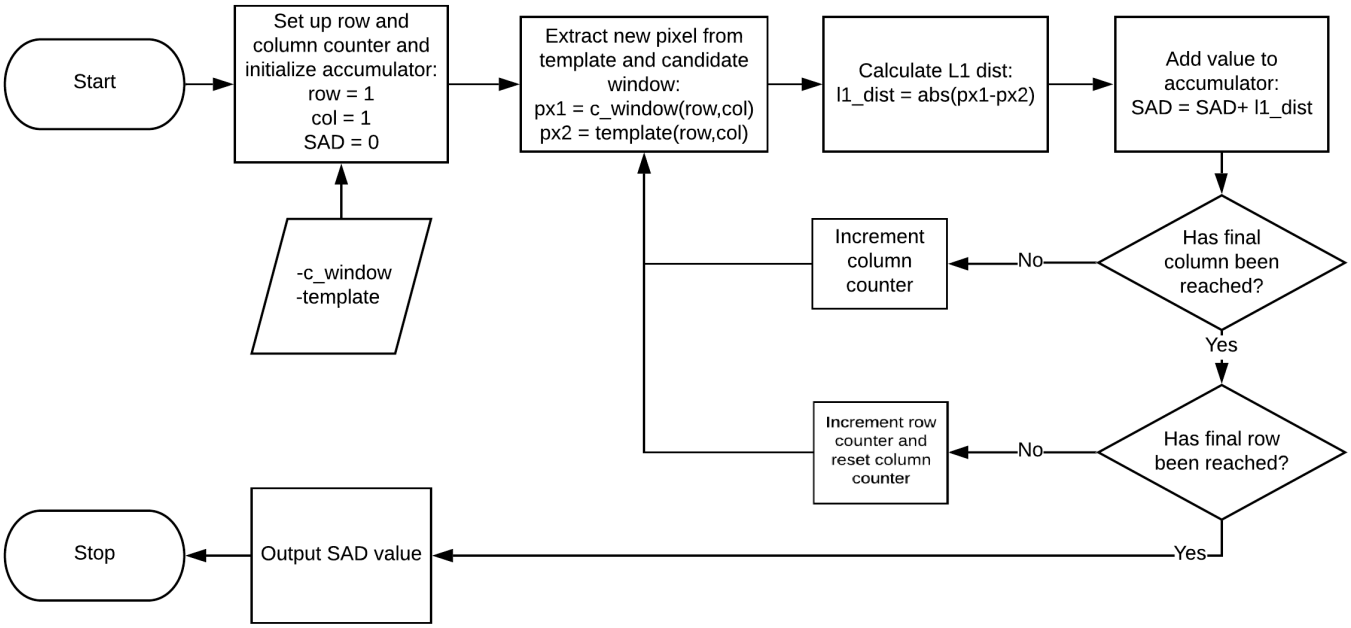
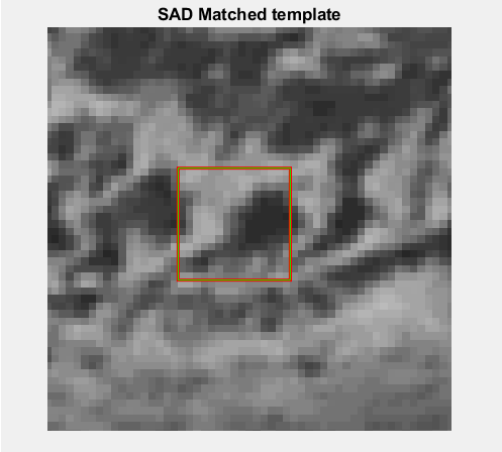
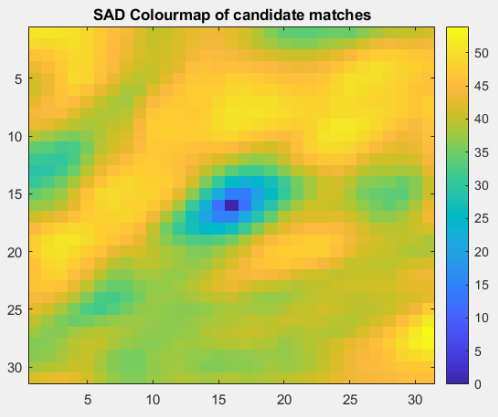


Figure 19: SAD implementation flowchart

To find the best matches for the SAD a variation of the `get_bestn_vectors()` function was created to find the smallest values since a perfect match for SAD will be a value of 0. This was done by using MATLABs `mink()` function [23] instead of `maxk()`. In order to test that the implementation worked correctly the SAD implementation was tested under perfect correlation assumption. This means that the test template was taken from the same image to be matched against in order to test the implementation under perfect correlation conditions. For this, the test template at coordinates (166,166) from Set 1 was used. A template size of 15×15 was used with a search size of 15 to have a large enough search size to fit an additional template sized space in all directions. Figure 20 shows the perfectly matched template as well as the colourmap of the output of the SAD function.



(a) Original template location in red. Matched template in green



(b) Colormap of SAD map. Smaller value is better

Figure 20: Results of perfect correlation test for SAD.

5.1.2 CCC implementation

The next similarity measure implemented was the CCC. For this implementation, the mean and deviation of the template and each candidate window needs to be calculated. In order to reduce computation time, a change was made to the basic template matching function to calculate the mean of the pixel values of the template and $\left(\sum_{x=1}^{N}\sum_{y=1}^{N}\left(f(x,y)-\bar{f}\right)^2\right)^{\frac{1}{2}}$ from the denominator in Equation 2 before entering the loop that checks all candidate windows to avoid recomputing the same value repeatedly. A flowchart for this calculation is shown in Figure 21. Figure 22 shows the flowchart for the rest of the CCC calculation for one candidate window.
As with the SAD implementation, the CCC implementation was tested under perfect correlation conditions using the same values for the SAD test (15 size template and search size and testing it with the same template from Set 1). The output for these test shown in Figure 23 show that the CCC implementation correctly found the perfect match at the center of the search size. The colourmap has a key difference to that of one for the SAD since better matches correspond to a higher coefficient value.

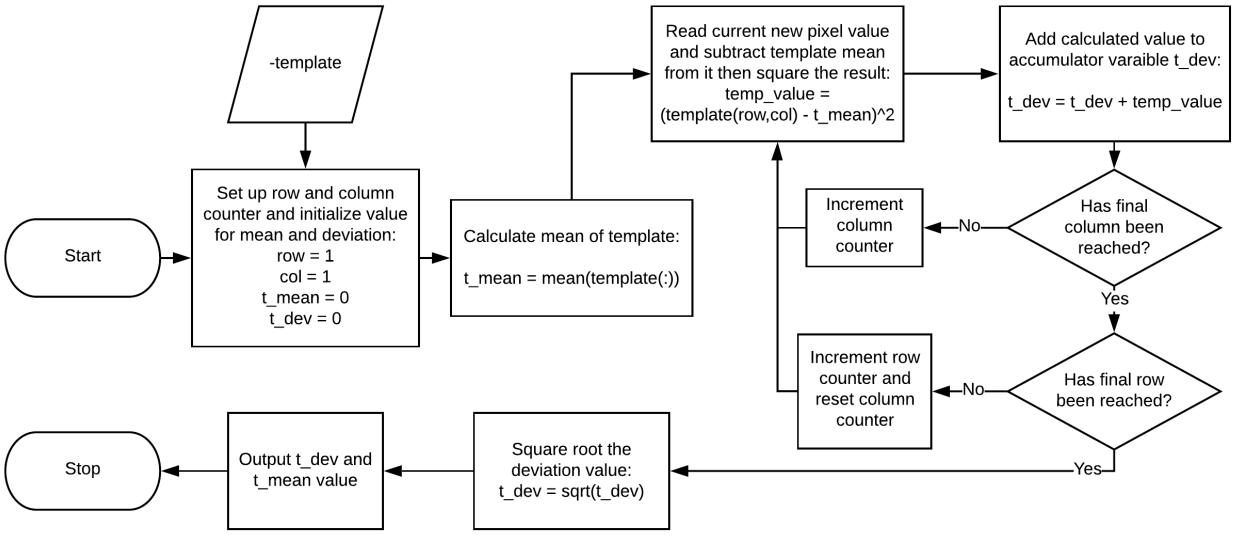


Figure 21: Mean and deviation calculation for template flowchart for the CCC implementation

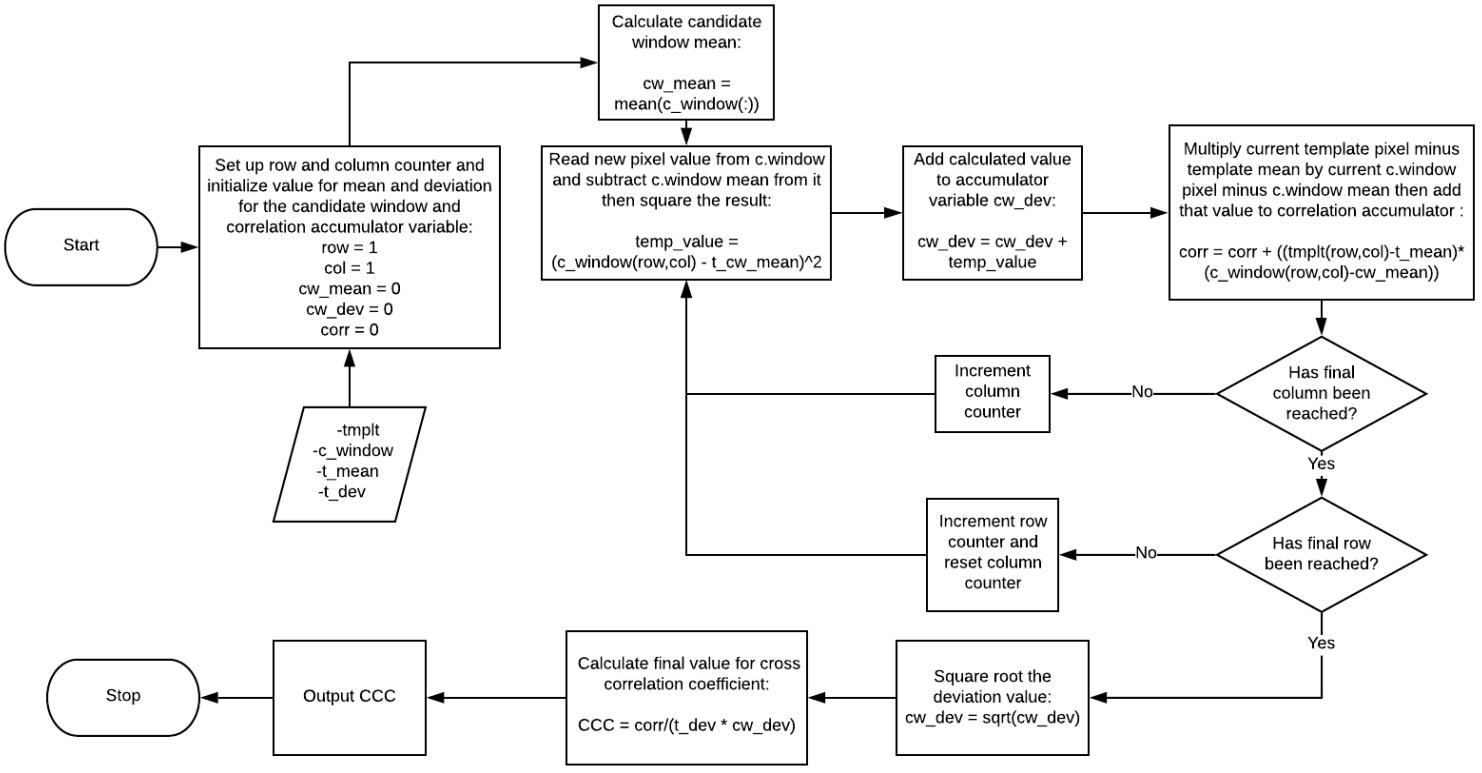
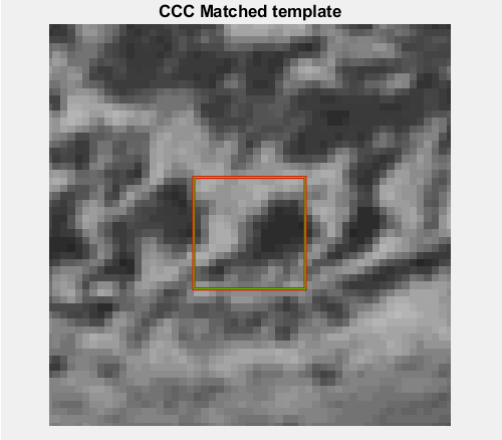
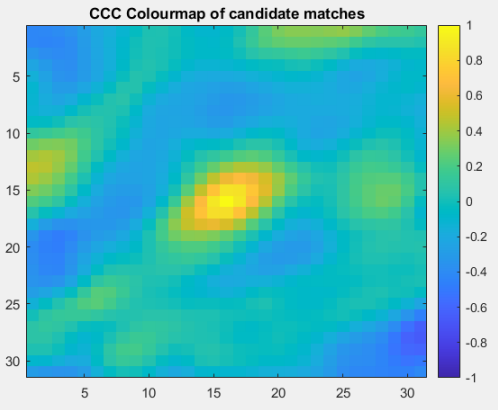


Figure 22: CCC implementation flowchart



(a) Original template location in red. Matched template in green



(b) Colormap of CCC map. Larger value is better

Figure 23: Results of perfect correlation test for CCC.

5.1.3 OMκ implementation

The OMκ implementation again used the functions proposed in Section 5.1 as a starting point. Since π_1 and $(\pi_1)^{-1}$ correspond to the rank intensities of the template window they can be calculated prior of entering the main loop of checking candidate windows. To calculate $(\pi_1)^{-1}$, the ordered indexes of ascending ranks in π_1 are set as the respective rank values in $(\pi_1)^{-1}$. This process is shown in Figure 24. Once this calculation is done, within the main template matching loop, for each candidate window π_2 is calculated by ranking the intensities within the window and following the definition in Equation 5, the value for κ is calculated. A flowchart for this is shown in Figure 25

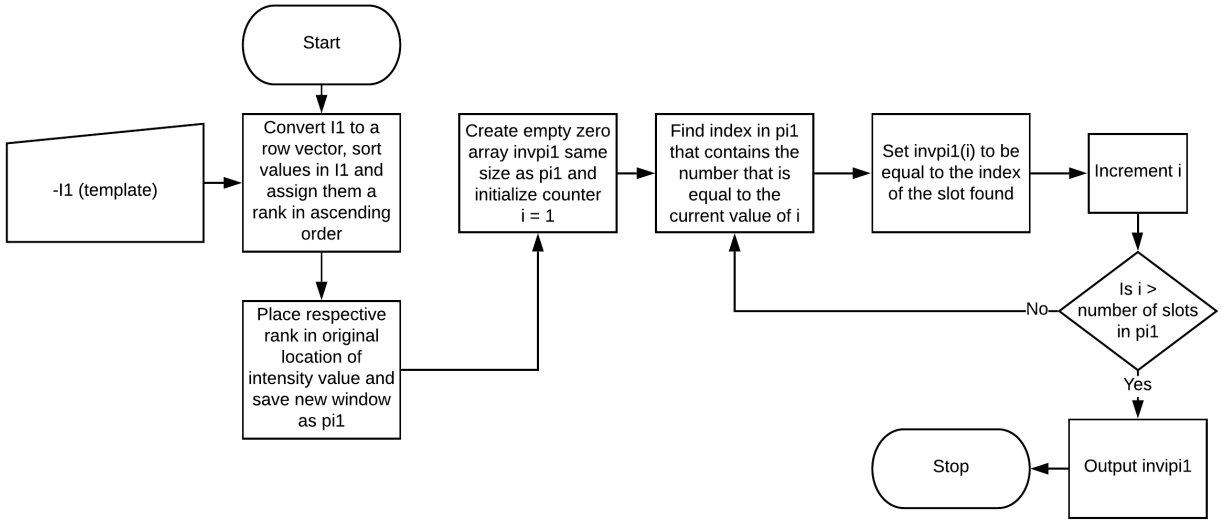


Figure 24: Calculating inverse $(\pi_1)^{-1}$ flowchart

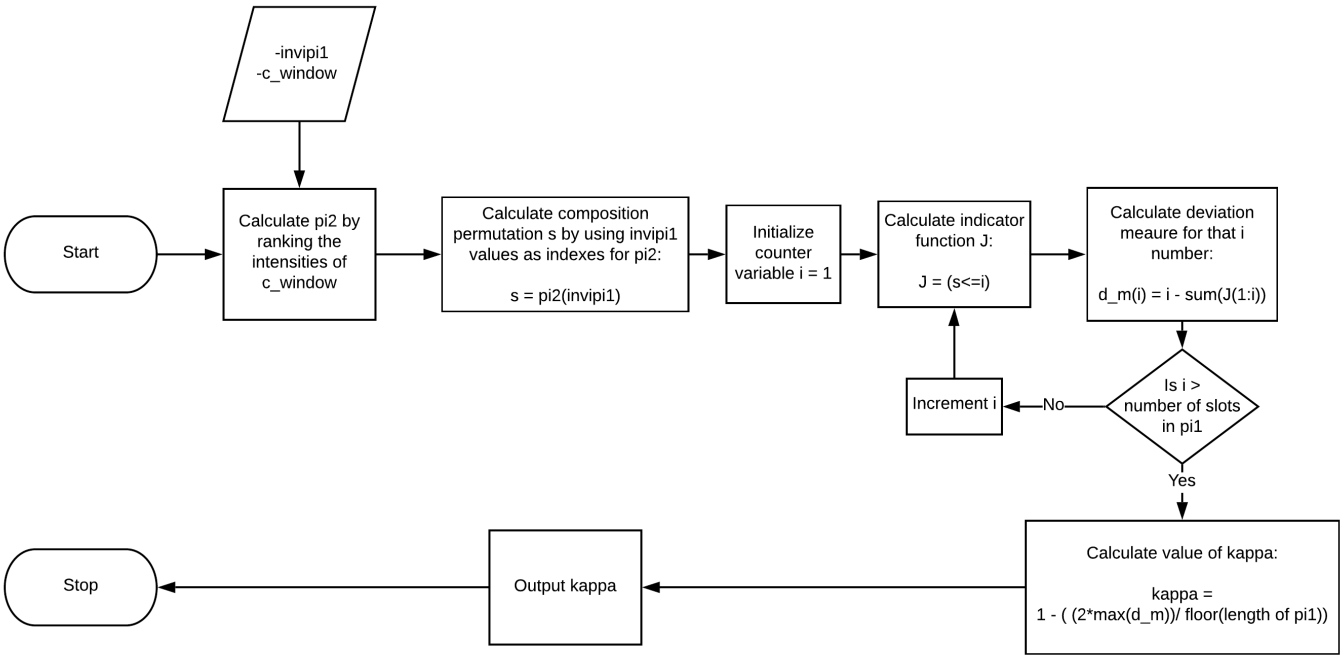
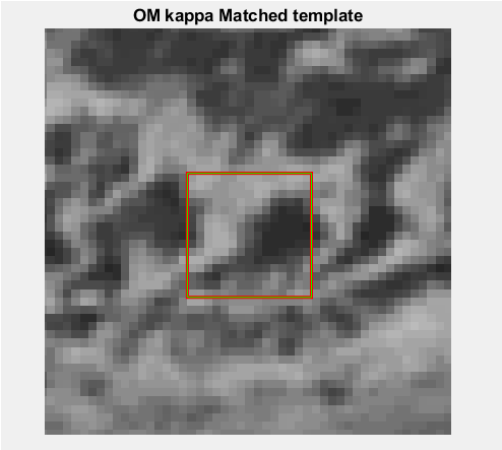
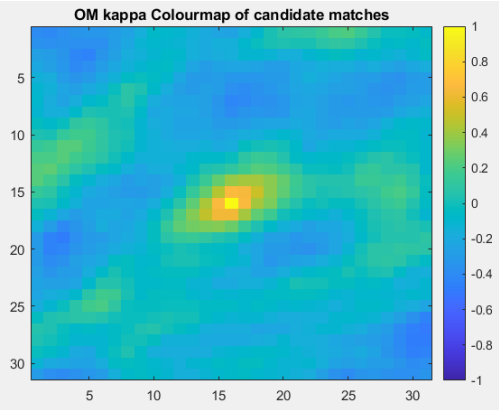


Figure 25: OMκ implementation flowchart

A perfect correlation test was then carried out for the OMκ implementation using the same test values as for the other methods. As expected the perfect match was found at the center of the template. The colourmap for the OMκ in Figure 26 is similar to that of the CCC since both metrics have a range of -1 to 1 for their respective output coeficients. Under perfect correlation the colourmap for OMκ shows a sharper peak near the correct match than that of the CCC colourmap in Figure 23. This slight difference can be better seen in Figure 27, by using MATLAB's `surf()` function [24] to plot a surface plot of the output maps for comparison.

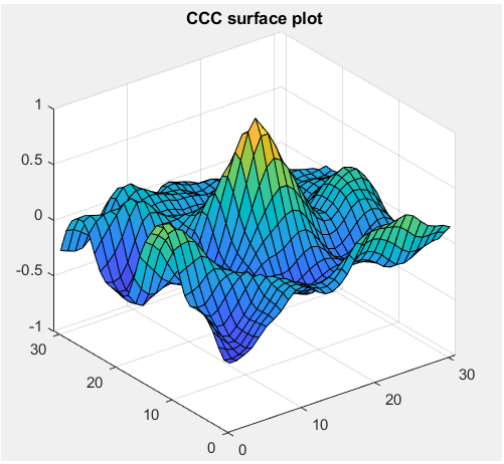


(a) Original template location in red. Matched template in green

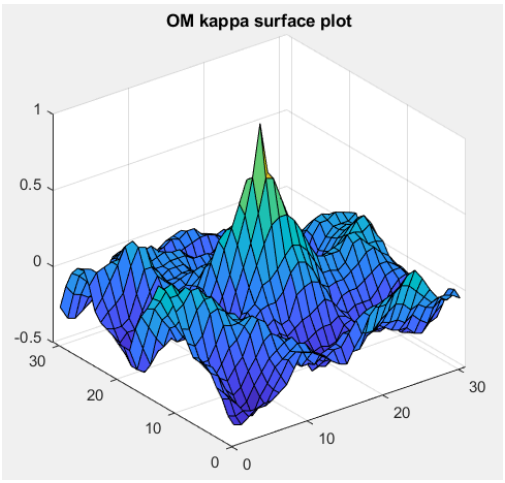


(b) Colormap of OMκ map. Larger value is better

Figure 26: Results of perfect correlation test for OMκ.



(a) Surface plot of CCC output map.



(b) Surface plot of OMκ output map.

Figure 27: Comparison of surface plots of perfect correlation test of CCC and OMκ.

5.2 BBS implementations

In the paper that proposed BBS [4], two different methods of calculating the distances between points for the BBPs were presented. The first was outlined in Section 2.1.4 involves splitting the template and candidate window into $k \times k$ patches to calculate distance between colour features for each patch. The second method presented on the paper focused on applying BBS to deep features using VGG-Deep-Net [25]. This project will use two variations of colour feature descriptor (first method). As the test images obtained from the MSG satellites are greyscale in nature, the method was adapted to use only one channel to calculate the BBPs.

5.2.1 Distinct patch BBS implementation

The distinct patch BBS implementation closely follows the proposed plan in [4]. Given a chosen patch size of k the steps followed for the naive implementation function of BBS are as follows:

1. Adjust template and image to be searched to be evenly divisible by k in all directions. This is needed so that all candidate windows have the same number of patches as the template.

2. Divide image and template into two matrices containing the $k \times k$ blocks in each column of the matrix by using MATLAB's `im2col()` function [26]

```
%Patch size k = p_sz, tm = template, im = image
tm_mat = im2col(tm, [p_sz,p_sz], 'distinct');
im_mat = im2col(im, [p_sz,p_sz], 'distinct');
```

3. Calculate distance matrix for the location descriptor of the BBS formula in Equation 7. This was done using MATLAB's `squareform()` and `pdist()` functions [27] [28]. The distance matrix was normalised to have a maximum distance value of 1.

```
%Normalize to have the max distance be 1. tm_sz is the size of ...
    template(Implementation assumes that only square templates will be used
n_factor = 0.707/(tm_sz-p_sz);

%tm_mat_sz is the size of tm_mat
%Create empty distance matrix
dmat_loc = zeros(tm_mat_sz);

%Compute meshgrid for the distances
[x_dist,y_dist] = ...
    meshgrid([0:p_sz:tm_sz-1]*n_factor,[0:p_sz:tm_sz-1]*n_factor);

%Transpose meshgrid vectors
d_vect = [x_dist(:)'; y_dist(:)'];

%Compute location distance matrix
dmat_loc = squareform(pdist(d_vect'));
```

4. Start main loop of iterating through each candidate window. For each candidate window calculate the appearance distance matrix (based on pixel intensity values). To do this first the corresponding candidate window is extracted from the image by obtaining the index of the patches for that candidate window. These indexes correspond to the column in

im_mat(calculated in Step 3) which contain the pixel values for each patch. Once the indexes are obtained, the sum of squared differences between each patch in the template to every patch in the candidate window is calculated to make up the appareance distance matrix.

```
    %Loop all candidate windows
    for col = 1:(im_sz(2)/p_sz)-(tm_sz/p_sz)+1
        for row = 1:(im_sz(1)/p_sz)-(tm_sz/p_sz)+1

            % Extract candidate window
            cw_p_indx = im_p_indx(row:row+(tm_sz/p_sz)-1, ...
                row:row+(tm_sz/p_sz)-1);
            cw = im_mat(:,cw_p_indx(:));

            % Compute distance matrix
            for i = 1:tm_mat_sz %Loop all patches patches in candidate  window
                for j = 1:tm_mat_sz %Loop all patches in template
                    %Calculate difference from candidate window patch at i ...
                        to all patches in template
                    dist_val(:,j) = (cw(:, i) - tm_mat(:, j)).^2;
                end
                %Save distance matrix column
                dmat_int(:,i) =  sqrt(sum(dist_val));
            end
            ...
```

5. Normalise the intensity distance matrix then add location and intensity distance matrices together by first multiplying the location distance matrix by the chosen λ value.

```
            %Normalise intensite distance matrix
            norm_value = max(max(dmat_int(:)));
            dmat_int_norm = dmat_int/norm_value;

            %Calculate final distance matrix
            dist_mat = dmat_int_norm + lambda*dmat_loc;
```

6. Find the indexes of the minimum values along each axis of the final distance matrix. Then create two "best buddies" matrices to hold the indexes of the values found (one matrix for each axis). These new matrices are initialised to Inf(infinity) and are filled in with a value of 1 in the respective slot and will be used to calculate the BBPs:

```
            %Find indexes of min values along each axis of the distance matrix
            [¬, minval_indx1] = min(dist_mat,[],2) ;
            [¬, minval_indx2] = min(dist_mat) ;

            %Create best buddies(bb) matrices for candidate window to
            %template and template to candidate window
            bb_mat1 = Inf(tm_mat_sz,tm_mat_sz);
            bb_mat2 = Inf(tm_mat_sz,tm_mat_sz);

            %Fill in best buddies locations
            for i = 1:tm_mat_sz
                bb_mat1(i,minval_indx1(i)) = 1;
                bb_mat2(minval_indx2(i),i) = 1;
```

```
14              end
```

7. Calculate and count the number of BBPs, which correspond to when the two best buddies matrices have a value of 1 in the same slot of each array. The total number of best buddies pairs is then added to the BBS map as the BBS value for that candidate window. A perfect correlation match will have a value of 1 across the diagonal of both distance matrices

```
1               %Count number of best buddies pairs
2               BBPs = bb_mat1 == 1 == bb_mat2;
3               bbs_map(row,col) = sum(BBPs(:));
```

8. Once all candidate windows have been calculated, normalise all BBS values by the number of patches in a template to give a max value for the BBS of 1.

```
1       %Normalize BBS results (max possible number of BBPs is tm_mat_sz)
2       bbs_map = bbs_map/tm_mat_sz;
```

This naive implementation can be improved by saving and reusing computed values for the patches in the distance matrices based on the current location of the candidate window. The function iterates columns first (in order to be consistent with most MATLAB's functions such as `im2col` which also iterates matrices columns first). Step 4 in the list above can be extended into four sub-cases depending on the row and column index in the for loop. The following examples will use a 9×9 size template with a patch size k of 3, which ultimately results in 3×3 patches per template and candidate window:

Case 1: row index == 1 and column index == 1:
Under this conditions the operations to calculate the distance matrix is the same as those outlined in Step 4 above. However there is one key difference in that the calculated appearance distance matrix is saved in a buffer that holds the calculated distance matrices for new column in the image.

Case 2: row index > 1 and column index == 1:
This case is used when the distance matrix to be computed corresponds to a candidate window in the first column of the image, but on a new row. Because of this using the example template, only 3 new patches need to be calculated, while 6 previous patches from the last computed distance matrix can be reused. See Figure 28.

Case 3: row index == 1 and column index > 1:
This case is used when the distance matrix to be computed corresponds to the first candidate window in a new column. Using the example template, only 3 new patches need to be computed, while 6 previous patches from the distance matrix in the adjacent column can be reused. See Figure 29.

Case 4: row index > 1 and column index > 1:
This case is used for any candidate window that is not on the first row and not on the first column of the image. Using the example template any window in this case, only one patch needs to be recomputed, while the other 8 patch calculations can be reused from previous distance matrices in the buffer. See Figure 30.

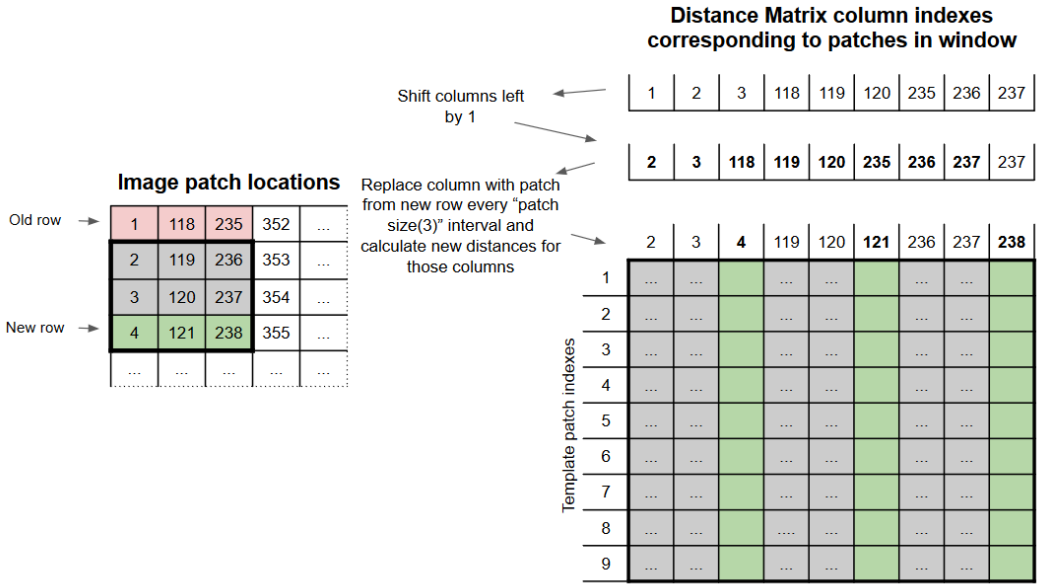


Figure 28: Optimized BBS case 2 distance matrix operations

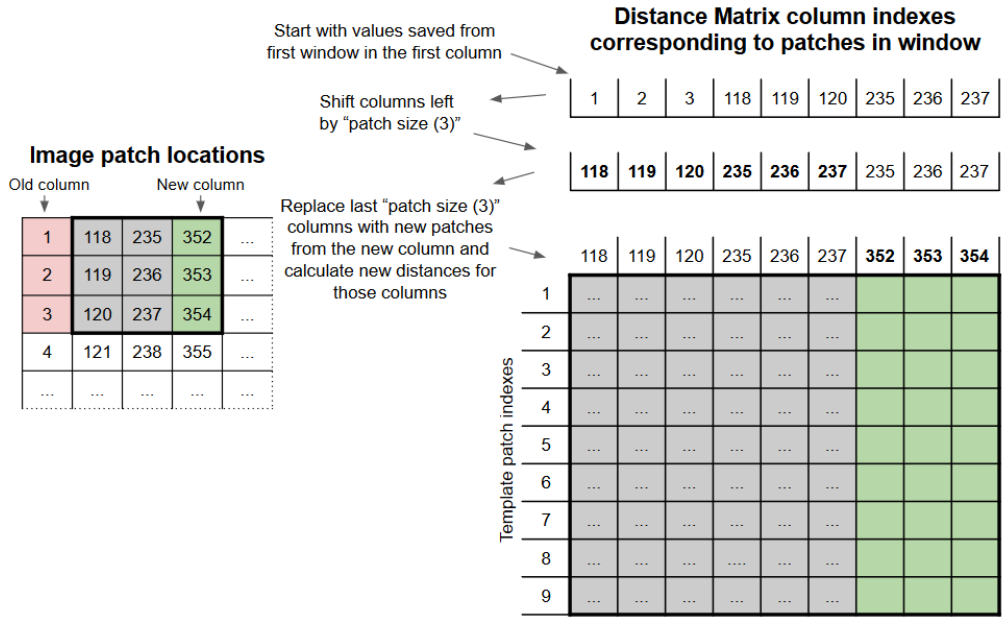


Figure 29: Optimized BBS case 3 distance matrix operations

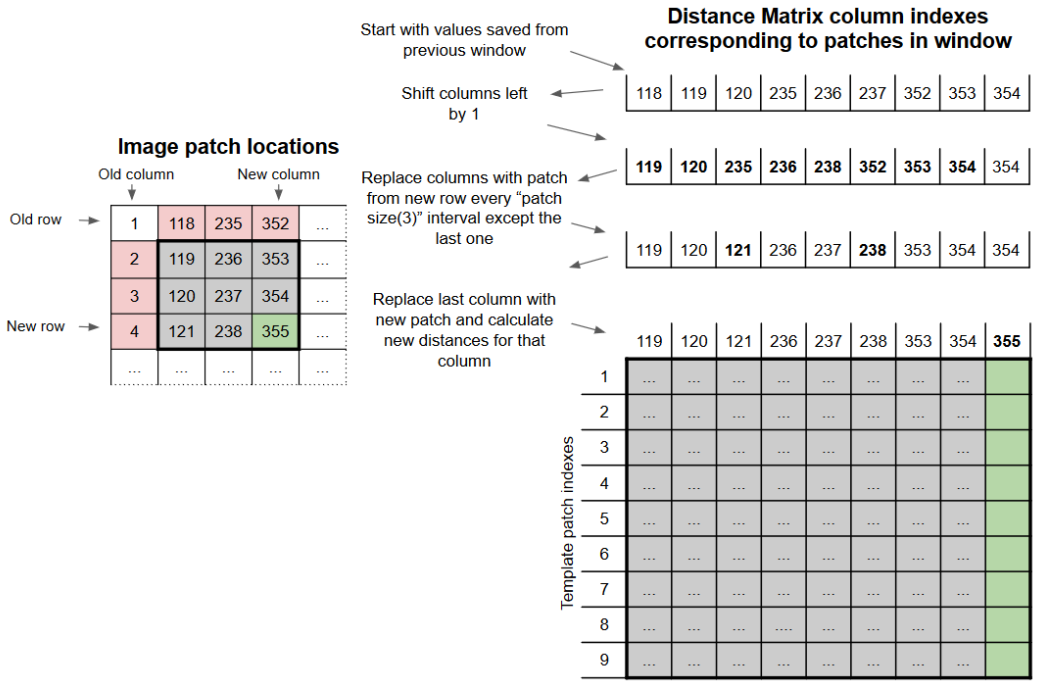


Figure 30: Optimized BBS case 4 distance matrix operations

This optimisation compromises memory cost since computed distance matrices need to be saved to obtain a much faster computation runtime. In order to further optimize the BBS implementation, a variable search size was added as an input. Using the optimal values for BBS in [4] of a patch size of 3 and λ value of 0.2, the naive and optimized implementations were tested for a template size 15×15 with an increasing search size from 10 to 50 in steps of 5. To ensure that any time differences are due to the changes in the algorithm and not any hardware related factors an average of 5 seperate tests was taken for the results. The results of the this test in Figure 31 show that while both version have an exponential complexity, the optimized version greatly reduces the computational time.

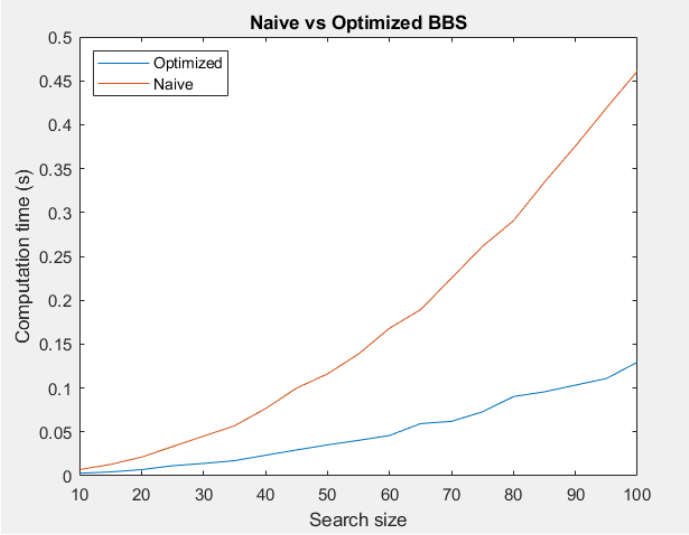
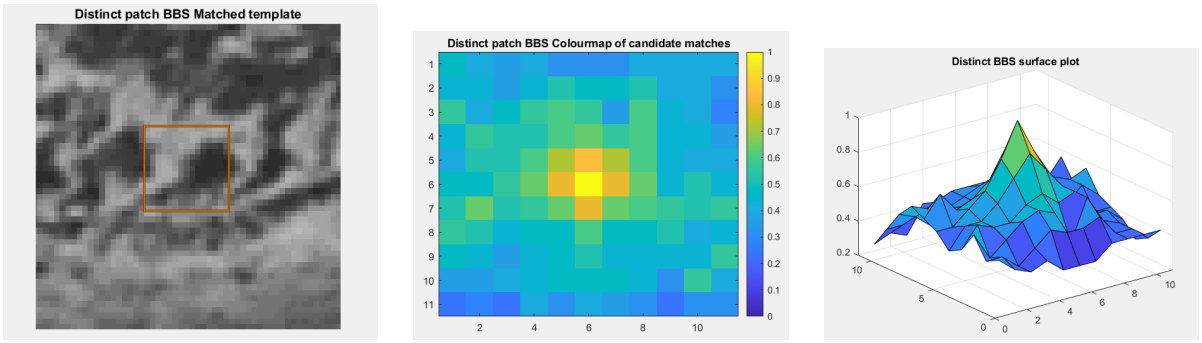


Figure 31: Average runtime of 5 test of naive BBS vs optimized BBS of increaseing search size

Next a test under perfect correlation conditions using the same parameters as the tests done for the previous template matching methods was carried out for the distinct patch BBS implementation. The results of the test and surface plot of the output can be seen in Figure 32. While these test show that the BBS implementation correctly matches a cloud template under perfect correlation conditions, they also show that the amount of candidate windows tested is lower than the tests for similarity metrics even though the search size used was the same as the other tests. This presents key weakness with the proposed implementation for BBS when applied to cloud tracking applications. The BBS method works by calculating BBPs between a set of points, which are implemented as $k \times k$ pixel patches. This is part of what makes BBS robust to impulse noise and also greatly reduces the computational complexity, with larger size patches providing a greater reduction. However due to the patches being non-overlapping, this results in the BBS method having a maximum resolution equal to that of the patch size, i.e. the minimum distance that a template can be matched away from its original location is that of the patch size. In [4], BBS is used to match faces, animals and other objects, which are all applications where having this reduced resolution in exchange for a much faster computation time is a good tradeoff. However as the resolution of MSG satellites is 3 km/pixel, implementing the $k \times k$ patches as non-overlapping reduces the precision at which BBS can match templates for cloud tracking to $k \times 3$ km. For example, given a template in one of the test images whose ground truth match is 1 pixel away, using a patch size k of 3 will only allow BBS to match the template either at the same location of the original template, or 3 pixels away from the original template. Because of this an alternative implementation of BBS termed "sliding patch" BBS was devised to overcome this limitation for cloud tracking applications.



(a) Original template location in red. Matched template in green

(b) Colormap of distinct patch BBS map. Larger value is better

(c) Surface plot of distinct patch BBS output map for perfect correlation test

Figure 32: Results of perfect correlation test for distinct patch BBS.

5.2.2 Sliding patch BBS Implementation

The sliding patch BBS implementation was made in order to overcome the limitation of distinct patch BBS mentioned above. As this version of BBS was similar to the distinct BBS version and most of the code was able to be re-purposed with a few adjustments to the implementation. The key difference of this implementation is the way that the image and template are divided into patches. By changing the parameters of the `im2col()` function from `'distinct'` to `'sliding'`, the patches taken from the image will include every possible patch based on the patch size entered as opposed having pixel locations be exclusive to their patch. Listing 2 shows the change done to Step 2 of the BBS method in Section 5.2.1 to implement the sliding patches and the difference between these two options of dividing the image into blocks is shown in Figure 33.

```
1       t_mat = im2col(tm, [p_sz,p_sz], 'sliding');
2       im_mat = im2col(img, [p_sz,p_sz], 'sliding');
```

Listing 2: Changes to step 2 of the distinct BBS method for sliding BBS

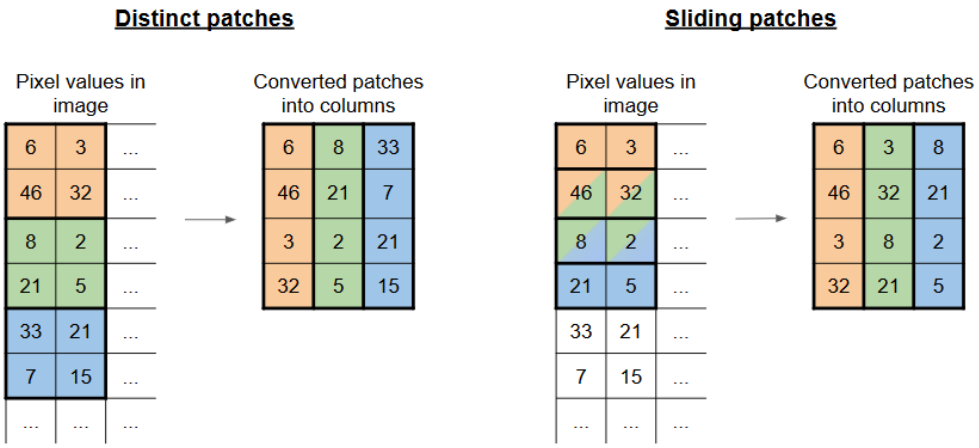
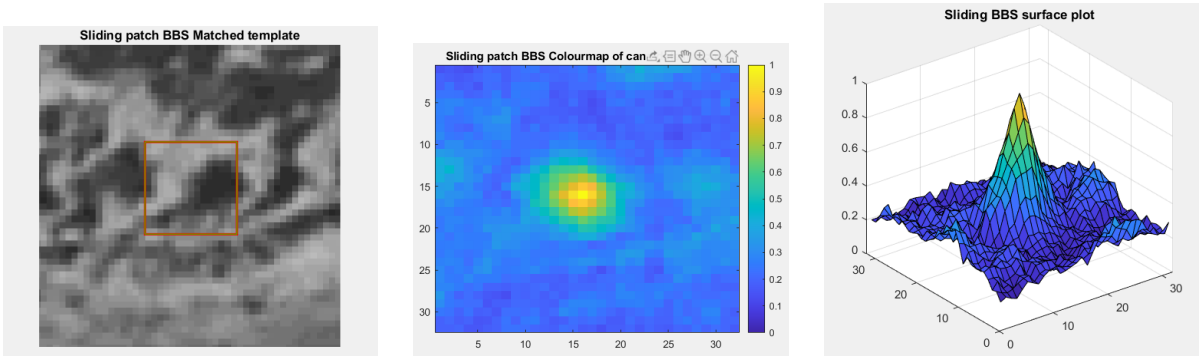


Figure 33: Distinct vs sliding patch division of images example for the first three patches in an image with a patch size of 2×2 pixels

In order to adapt the optimized version of distinct BBS to sliding BBS, the variables used to count the maximum amount of patches in a template of $\lfloor \frac{\text{template size}}{\text{patch size}}^2 \rfloor$ had to be changed to account for the increased number of patches to $(\text{template size} - \text{patch size} - 1)^2$. When testing the sliding BBS under perfect correlation conditions as previously done for the other metrics, the improvement over distinct BBS can be clearly seen. Figure 34 shows how sliding BBS successfully finds the perfect match with a correct output of 1 at the peak while also having a better resolution than distinct BBS.



(a) Original template location in red. Matched template in green

(b) Colormap of sliding patch BBS map. Larger value is better

(c) Surface plot of sliding patch BBS output map for perfect correlation test

Figure 34: Results of perfect correlation test for sliding patch BBS.

Although initial correlation tests for sliding patch BBS indicate that the main presented weakness of distinct patch BBS was overcome, this comes at a great cost of computational complexity. Using distinct patches reduces the computational complexity by a factor of k^4, however using a sliding window to calculate the patches results in an increase of computational complexity by a factor of $(k \times l)$. Given $L =$ image width$\times$height, $l =$ template width$\times$height and each patch is of size $k \times k$, the estimated big O notation of sliding patch BBS is:

$$O\Big(Ll^3k^4\Big) \tag{18}$$

The increased number of patches per template also shows that the discriminatory power of sliding BBS(number of possible different output values) is greatly increased when compared to distinct BBS and is not as dependant on patch size, as the maximum number of output values of BBS is equal to the total number of points(patches) used; for which sliding BBS is much greater.

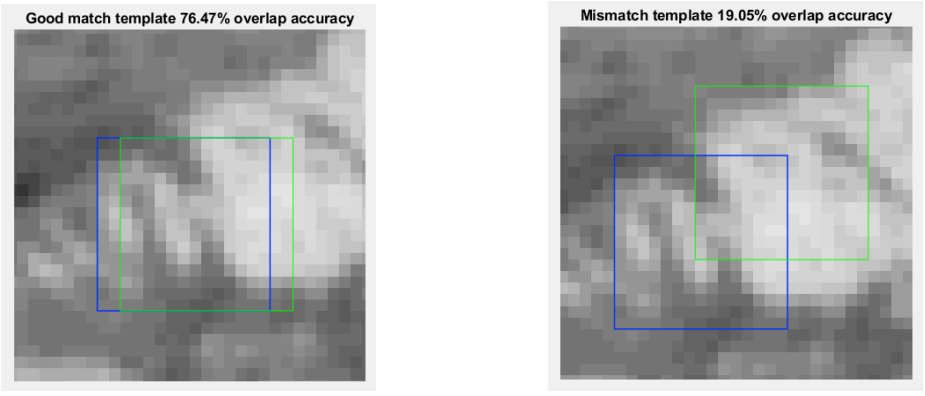
6 BBS parameter optimisation & tests

Both implementations of BBS differ from other template matching methods in that they have two parameters that may be tuned to adjust the performance based on the application. The main parameters is λ, which is a weight that determines the importance of the location descriptor when calculating the nearest neighbour points to find the BBPs in Equation 7. A second parameter that may be changed to tune performance is the patch size k. While in [4] the BBS tests only focused on the patch size as a method of reducing computation time, the effects of matching performance of this parameter when applied to cloud tracking were also be investigated. In order to tune and test these parameters, quantitative testing was be carried out using the 186 hand labelled test/ground truth templates from Section 4.3 by measuring the accuracy of each match using the Intersection over Union which measures the overlap of template bounding box B_t and the ground truth bounding box B_g:

$$\text{Overlap score} = \frac{B_t \cap B_g}{B_t \cup B_g} \tag{19}$$

BBS was originally tested for object tracking purposes using a large range of different test scenarios for different sized objects where the estimated motion for each scenario and is unknown. Because of this, a full search of the entire image was done to find each match and results were based on whether any overlap was found and if there was overlap by then looking at the intersection over union of bounding boxes [4]. For the BBS tests for cloud tracking, each image is known to be 15 minutes apart, and since the nadir resolution of the MSG satellites is known, a search size of realistic possible movement of the clouds can be used to greatly increase the probability that there will be overlap between the ground truth and predicted bounding boxes for the templates tested. Given the 3 km/pixel resolution, a search size of ± 15 pixels was selected for test in order to account for possible wind-speeds of up to 180 km/h.

For each of the BBS parameters tested, the cumulative overlap accuracy score of the 186 test templates was assessed while also taking note of the the number of total mismatches. A template test was deemed a mismatch if the overlap accuracy score was under 20%. Figure 35 shows an example mismatched and high accuracy match template.

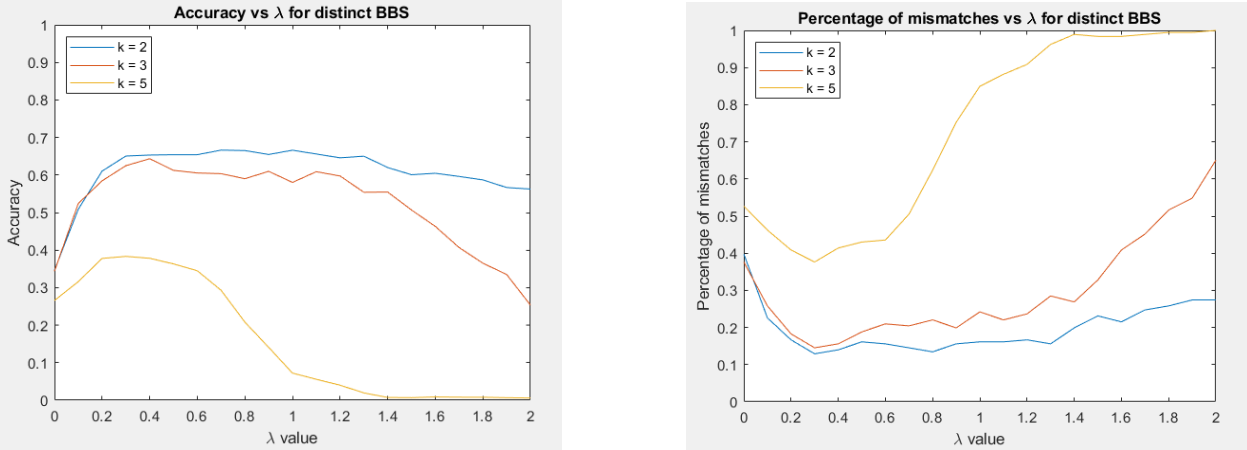


(a) (b)

Figure 35: Simulated examples of a good match template vs a mismatch. In blue is the ground truth for the template and green is the calculated location

6.1 Effects of λ and k on distinct patch BBS performance

The first test carried out for distinct patch BBS was a sweep of λ from values 0 to 2 in steps of 0.1 to find the values for lambda at which distinct patch BBS best performs. These sweeps were performed for both a patch size k of 2, 3 and 5.



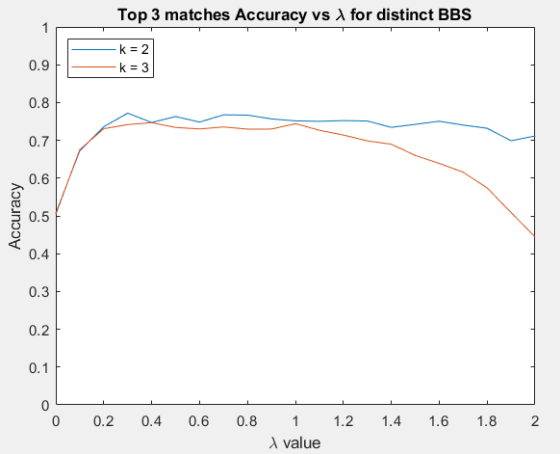
(a) Best match accuracy

(b) Best match percentage mismatches

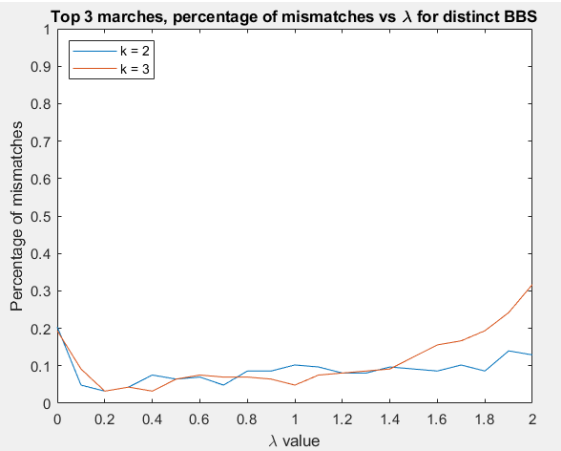
Figure 36: Distinct BBS accuracy and mismatch percentages when using the best match for varying values of λ and k.

The test results in Figure 36 show the average accuracy of distinct BBS for the best match for each template and the percentage of mismatches based on λ and k. Overall the smallest patch size for distinct BBS gave better accuracy than the other two patch sizes tested. A λ value of 0.3 gave the least number of mismatches for all patch sizes tested. These test initially show that distinct BBS is not very sensitive to choice of λ in the range $0.2 < \lambda < 1.4$. While using a value lower than 0.2 has similar effects when using patch size of $k = 2$ or 3, values higher than 1.4 only significantly decrease performance for a patch size $k = 3$. The patch size $k = 5$ was included for this test as an extreme value to show that using large patch sizes in cloud tracking applications heavily deteriorates accuracy of distinct BBS due to limiting the coordinates where potential matches for a template can be found. Due to this, in remaining tests, only a patch size of 2 and 3 were compared.

Next the same test was performed again, but instead of using the best match found, for each template tested, the top 3 results were compared to the ground truth bounding box, and the best result of those three was selected. This aims to reduce possible mismatches due to a bad best match and use the second or third best matches found in the search area instead. While in real applications there would be no way of knowing which match is best as the ground truth templates are not available, these bad matches can be removed through the use of relaxation labelling. Figure 37 shows the results of the top 3 matches test. When compared to the results in Figure 36 for the best match, the total accuracy has been increased by 0.1 (+10%) for all values of λ. Using one of the best top 3 results in the search area is also shown to greatly reduce the number of mismatches which is a direct result of the accuracy score increasing.



(a) Top 3 matches accuracy



(b) Top 3 matches percentage mismatches

Figure 37: Distinct BBS accuracy and mismatch percentages when using the top 3 matches for varying values of λ and k.

6.1.1 Impulse noise performance

Next the same test for λ and k were performed but by first corrupting the image with 5% impulse noise. This was done by using MATLAB's `imnoise()` function [29]. In order to ensure consistency in the noise between tests, MATLABS random number generator was set to a fixed seed. An image from Set 1 with the added impulse noise is seen in Figure 38.

```
%Keep rng seed the same between tests
rng('default');

%Add 5% salt and pepper noise to image
img1 = imnoise(img1,'salt & pepper',0.05);
img2 = imnoise(img2,'salt & pepper',0.05);
```

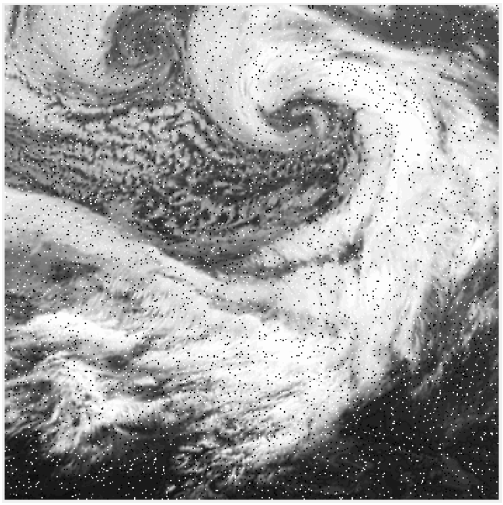
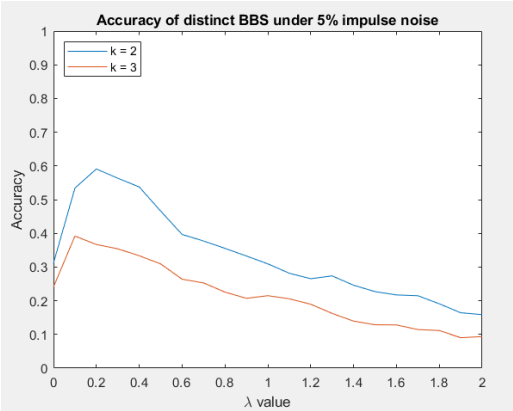


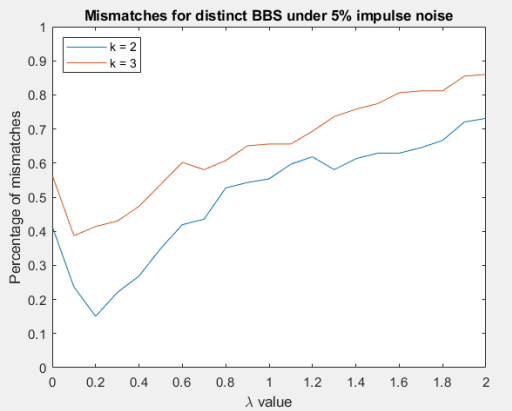
Figure 38: Set 1 image corrupted by salt & pepper impulse noise

The total accuracy score and percentage of mismatches were plotted again from a range of $0 \leq \lambda \leq 2$ for both $k = 2$ and 3. The results when looking at the best match in the search area in Figure 39 show that a patch size of $k = 3$ is greatly affected when impulse noise is introduced. The smaller patch size of $k = 2$ show a reduction of best total accuracy from 0.66 to 0.59, however the biggest change when compared to the normal image tests is that the performance of distinct BBS is much more sensitive to the choice of λ when impulse noise is added, with a range of $0.1 < \lambda < 0.3$ giving the best accuracy and least number of mismatches when looking at the best match results.

When looking at the test results for the top 3 matches in the search area with added impulse noise seen in Figure 40, an improvement in accuracy can be seen when using a patch size of $k = 3$ when compared to the best match test, however the best accuracy for this patch size remains lower compared to the test with no added noise. For the patch size of $k = 2$ there was again an improvement in accuracy and a decrease of mismatches, especially for the range of $0.1 < \lambda < 0.3$. This shows that distinct BBS using a patch size of $k = 2$ has good robustness to impulse noise, but matching performance becomes more sensitive to changes in λ.

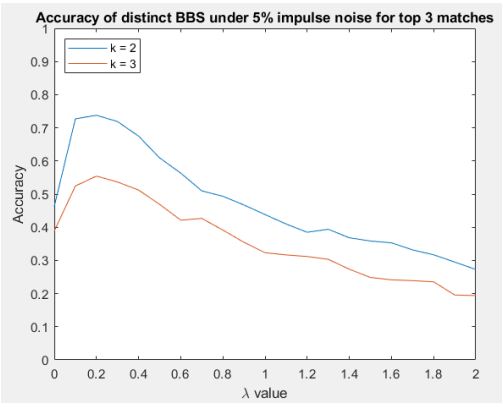


(a) Best match accuracy

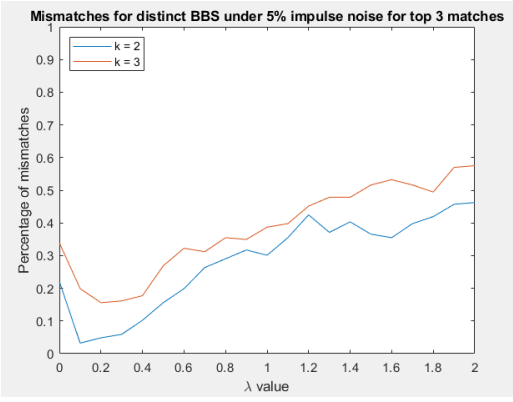


(b) Best match percentage mismatches

Figure 39: Distinct BBS accuracy and mismatch percentages with 5% impulse noise, using the best match found for varying values of λ and k.



(a) Top 3 matches accuracy



(b) Top 3 matches percentage mismatches

Figure 40: Distinct BBS accuracy and mismatch percentages with 5% impulse noise, using the best of the top 3 matches found for varying values of λ and k.

6.1.2 Gaussian noise performance

The final tests done for distinct BBS were to analyse performance based on the parameters when adding gaussian noise instead of impulse noise. For these test, the images were corrupted using gaussian white noise with 0 mean and a variance of 0.005 as shown below. An image from Set 1 with the added gaussian noise is seen in Figure 41.

```
%Keep rng seed the same between tests
rng('default');

%Add gaussian noise to image
img1 = imnoise(img1,'gaussian',0,0.005);
img2 = imnoise(img2,'gaussian',0,0.005);
```

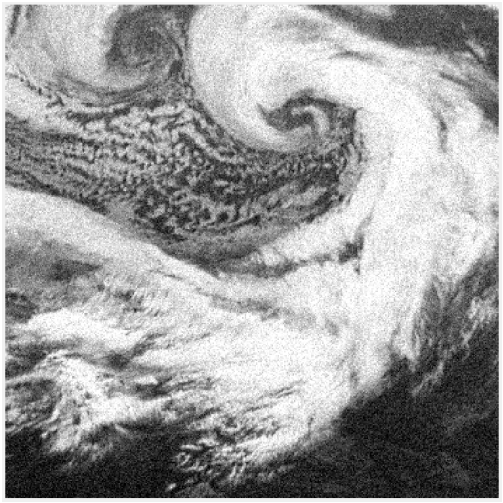
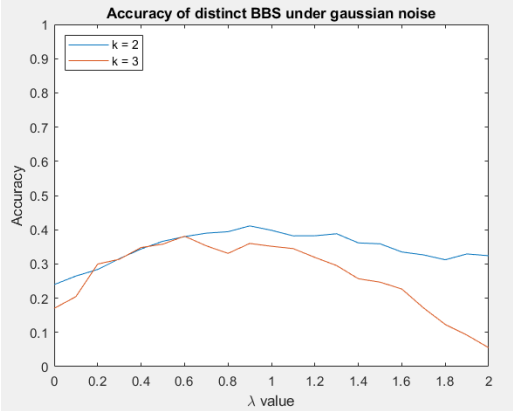


Figure 41: Set 1 image corrupted by gaussian noise

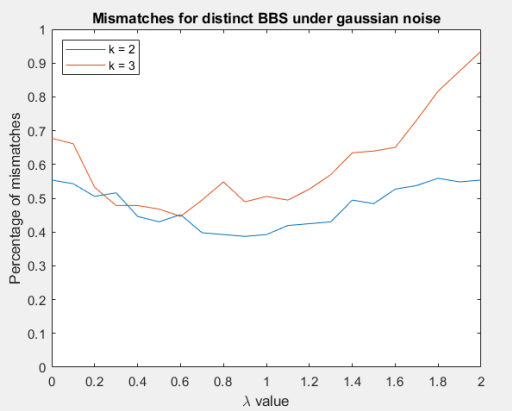
For the best match test, sweeping λ from 0 to 2 showed very poor results compared to the same test for impulse noise. For both patch sizes, the accuracy using the best match was under 50%, with λ having only small effects on the total accuracy for patch size $k = 2$. For the larger patch size, values of λ over 1.2 and under 0.2 showed an even greater sharper decrease in accuracy. Over 40% of the templates tested where considered mismatches for the best values of lambda. Out of both patch sizes tested, the smaller patch size of $k = 2$ yet again outperformed the larger patch size.

When looking at the top 3 best match tests, while the sensitivity and best values for lambda remain mostly similar, the total accuracy has heavily improved. For the patch size $k = 2$, the top accuracy when using the best of the top 3 matches was 0.59 compared to 0.42 when using only the top match for the same patch size. The percentage of mismatches was also much lower for the top 3 best matches test, although for both tests the percentage of mismatches started to rise when using a λ value of 1.2 or more. The low amount of accuracy for the gaussian noise tests coupled with the fact that the smaller patch size had overall better results for both tests indicates that the reason for the smaller patch size of $k = 2$ outperforming the larger patch sizer is due to the larger non-overlapping distinct patches creating a greater limitation of the locations at which the templates can be matched and the fact that using a smaller patch size directly increases the discriminatory power of distinct BBS e.g. for a 15×15 template using $k = 3$ gives a total of 25

different values that BBS can output ($\lfloor \frac{15}{3} \rfloor^2 = 25$), while using a smaller patch size of $k = 2$ gives a total of 49 possible outputs for BBS ($\lfloor \frac{15}{2} \rfloor^2 = 49$).

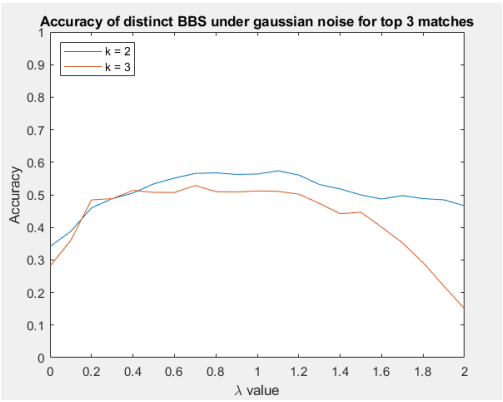


(a) Best match accuracy

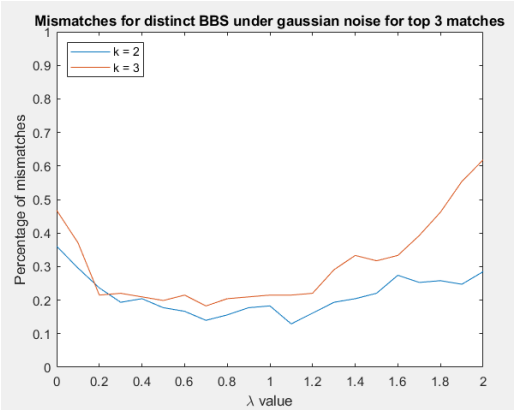


(b) Best match percentage mismatches

Figure 42: Distinct BBS accuracy and mismatch percentages with gaussian noise, using the best match found for varying values of λ and k.



(a) Top 3 matches accuracy

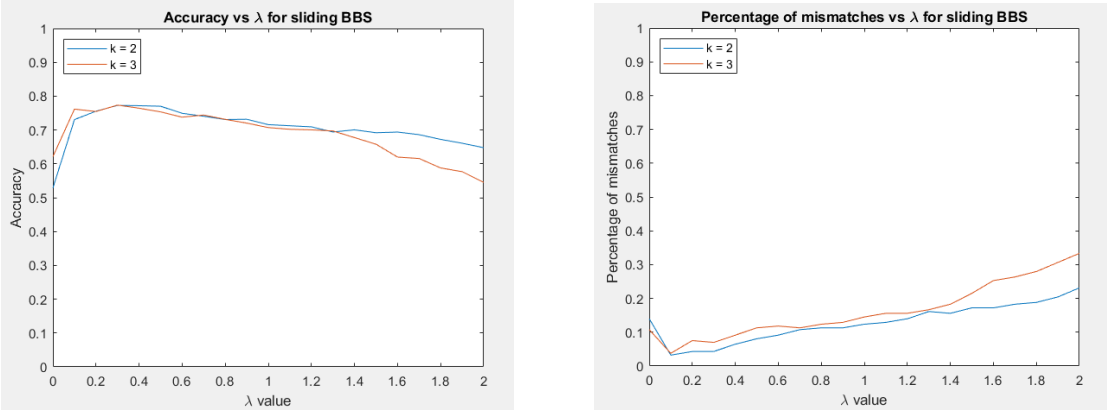


(b) Top 3 matches percentage mismatches

Figure 43: Distinct BBS accuracy and mismatch percentages with gaussian noise, using the best of the top 3 matches found for varying values of λ and k.

6.2 Effects of λ and k on sliding patch BBS performance

The same tests as for the distinct BBS version were carried out for sliding patch BBS to analyze the effects of the different parameters on this variant of the method. Again a sweep of λ from values 0 to 2 in steps of 0.1. These sweeps were performed for both a patch size k of 2, 3. A patch size of 5 was deemed not needed to test due to larger patch sizes being too computationally taxing.



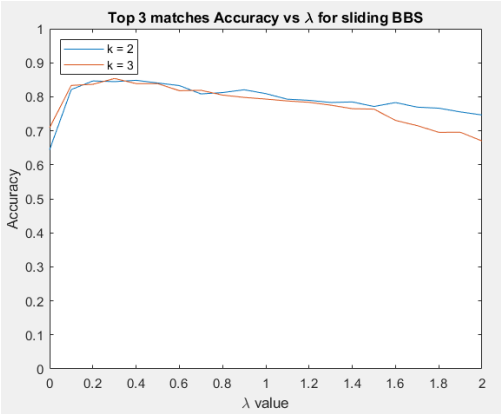
(a) Best match accuracy

(b) Best match percentage mismatches

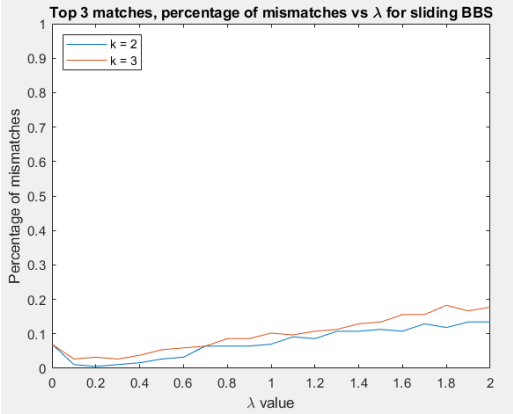
Figure 44: Sliding BBS accuracy and mismatch percentages when using the best match found for varying values of λ and k.

The results in Figure 44 show that sliding BBS has overall better accuracy totals than its counterpart. For both patch sizes used, values of λ from 0.1 to 1.3 had nearly identical accuracy results showing an even smaller sensitivity to λ than distinct BBS. The larger patch size $k = 3$ suffered a decline of accuracy when $\lambda > 1.3$ when compared to the smaller patch size.

When testing the top 3 matches for sliding BBS, the accuracy and percentage of mismatch graphs in Figure 45 showed the same shape graph as the best match test, except with greater values of accuracy and less mismatches. A top accuracy of 0.85 was found when using a λ value between 0.2 and 0.5 for both patch sizes which was higher than the top accuracy results for the same test for distinct BBS. The results of these test favour using the smallest patch size for both accuracy and runtime performance.



(a) Top 3 matches accuracy



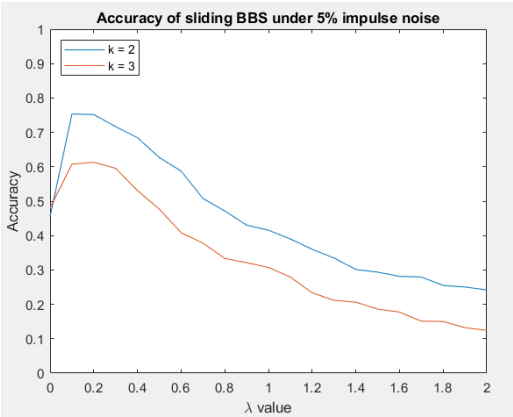
(b) Top 3 matches percentage mismatches

Figure 45: Top 3 matches, sliding BBS accuracy and mismatch percentages for varying values of λ and k.

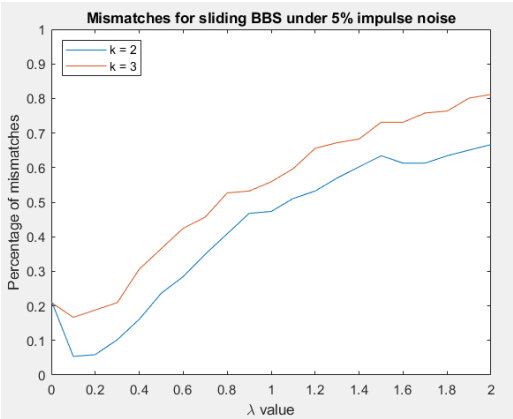
6.2.1 Impulse noise performance

The same test parameters were used for sliding BBS as the ones indicated above for distinct BBS. The test images were corrupted with 5% impulse noise. The results for the best match test with impulse noise in Figure 46 show that for certain parameters, sliding BBS has great robustness to impulse noise. When using λ between 0.1 and 0.3 with a patch size of $k = 2$ the accuracy score is almost the same as the test without noise. When compared to the smaller patch size, the larger $k = 3$ has a constant decrease in accuracy score across all λ values tested.

The top 3 best match test in Figure 47 showed the same shape of graph for both patch sizes as the best match test, but with higher accuracy values. This indicates that the majority of accurate best matches were close to the ground truth, and the top 3 best match test adjusted to more closely match the ground truth template for increased accuracy score.

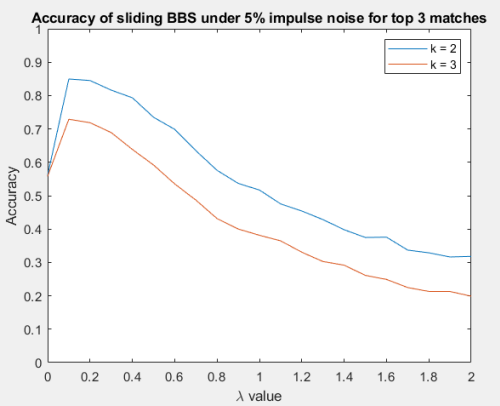


(a) Best match accuracy

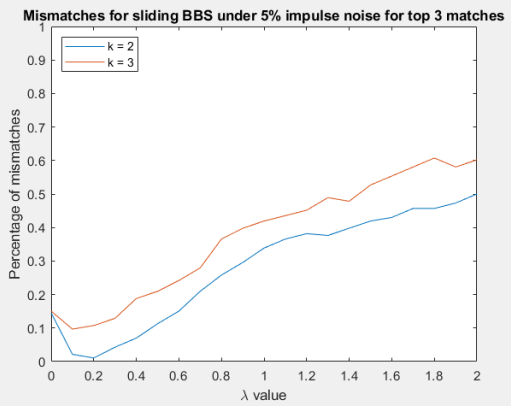


(b) Best match percentage mismatches

Figure 46: Sliding BBS accuracy and mismatch percentages with 5% impulse noise, using the best match found for varying values of λ and k.



(a) Top 3 matches accuracy



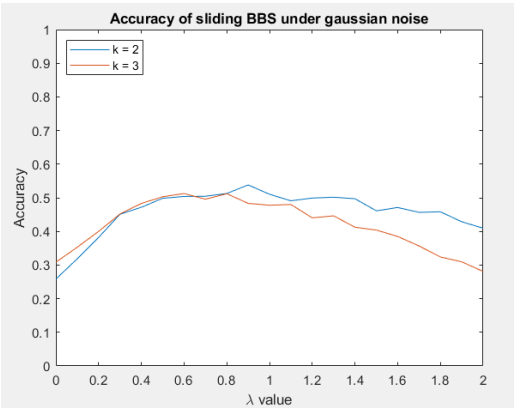
(b) Top 3 matches percentage mismatches

Figure 47: Sliding BBS accuracy and mismatch percentages with 5% impulse noise, using the best of the top 3 matches found for varying values of λ and k.

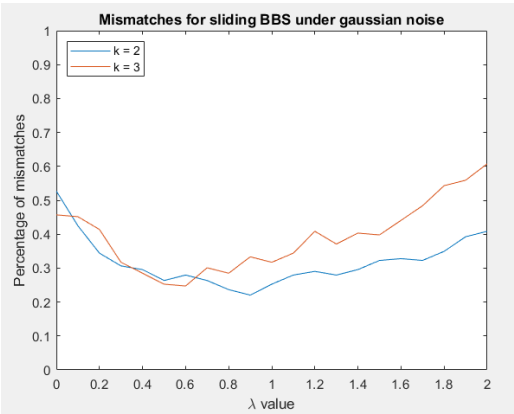
6.2.2 Gaussian noise performance

The performance of sliding BBS with gaussian noise was then tested with the same conditions as the test for distinct BBS. Figure 48 shows that sliding BBS does have better accuracy than distinct BBS for the same test(0.54 best accuracy for sliding BBS compared to 0.42 best accuracy for distinct BBS), however the gaussian noise performance is still worse than that for impulse noise.

The top 3 match test results in Figure 49 shows that similar previous tests, the accuracy for the top 3 test improved over the best match test for both patch sizes tested. The accuracy results were not very sensitive to changes in λ for values over 0.5 for the small patch size. The larger patch size had a clearer decline in accuracy score for values of λ of 1.5 or more.

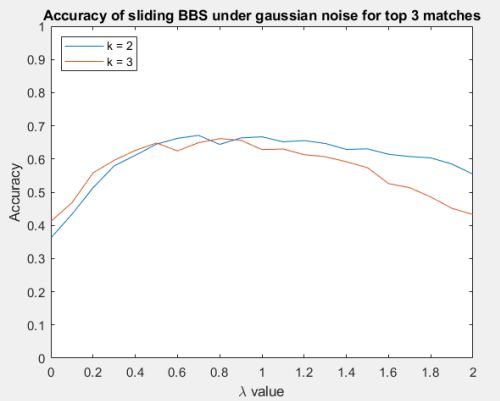


(a) Best match accuracy

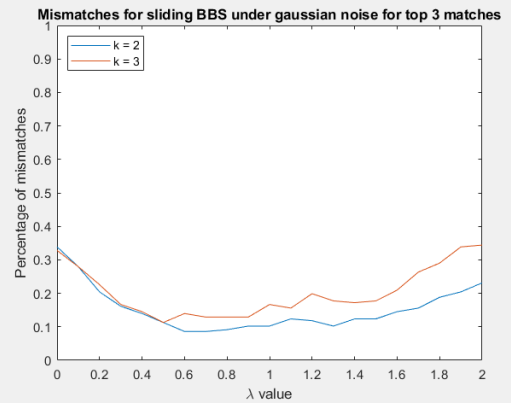


(b) Best match percentage mismatches

Figure 48: Sliding BBS accuracy and mismatch percentages with gaussian noise, using the best match found for varying values of λ and k.



(a) Top 3 matches accuracy



(b) Top 3 matches percentage mismatches

Figure 49: Sliding BBS accuracy and mismatch percentages with gaussian noise, using the best of the top 3 matches found for varying values of λ and k.

7 BBS vs other methods

Once the parameters and differences between the BBS implementations were tested, the best parameters for BBS for cloud tracking found were compared to other methods. For both the distinct and sliding BBS implementations, a patch size of $k = 2$ was used, as this patch size consistently outperformed the larger patch size in the parameter tests. A smaller patch size has opposite effects on the computational complexity of each BBS implementation, greatly decreasing the sliding BBS runtime while slightly increasing the distinct BBS runtime. The value for λ chosen was 0.3 based on the quantitative test results.

7.1 Quantitative accuracy comparisons

The tests comparing bounding box overlap done in Section 6 were run for the SAD, CCC and OMκ metrics. Table 2 shows the percentage score for each metric for the normal, impulse and gaussian noise image tests. These results indicate that initially the CCC and SAD have slightly better accuracy than both BBS methods and OMκ when no additional noise is added. Distinct BBS has the least accuracy score mainly due to the non-overlapping patches limiting the position of templates. When impulse noise is added, both the CCC and SAD accuracies fall considerable compared to the other robust methods. OMκ and sliding BBS hold similar results for impulse noise while distinct BBS again has slightly lower accuracy than its counterpart. Finally for gaussian noise, although all methods perform considerably worse, the methods that performed worse for impulse noise(CCC and SAD) performed slightly better for gaussian noise than those that were robust to impulse noise(BBS and OMκ), yet none of the methods can be considered robust to gaussian noise.

Table 2: Accuracy percentages for each similarity metric for noiseless, impulse noise and gaussian noise.

Method	Noiseless accuracy	Impulse noise accuracy	Gaussian noise accuracy
Distinct BBS	0.6520	0.5988	0.3254
Sliding BBS	0.7556	0.7520	0.3823
OMκ	0.7654	0.7620	0.4516
CCC	0.8107	0.3529	0.5053
SAD	0.8185	0.3868	0.5260

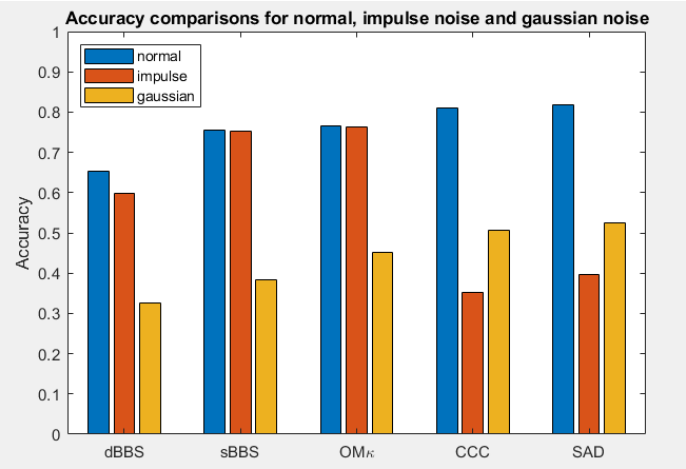


Figure 50: Results from Table 2 in a bar chart

7.2 Runtime comparison

Table 3 shows results of comparing the runtime of each method to perform template matching. To test this, each method was run with a 15×15 template in a search size of ± 15 for a set of 10 templates. This test was repeated 5 times for each method to to create the average runtime. The SAD and CCC are the two fastest methods. Distinct BBS is slower than these two methods, but much faster than both sliding BBS and OMκ. While OMκ and sliding BBS are the two slowest methods, sliding BBS is still considerable slower than OMκ.

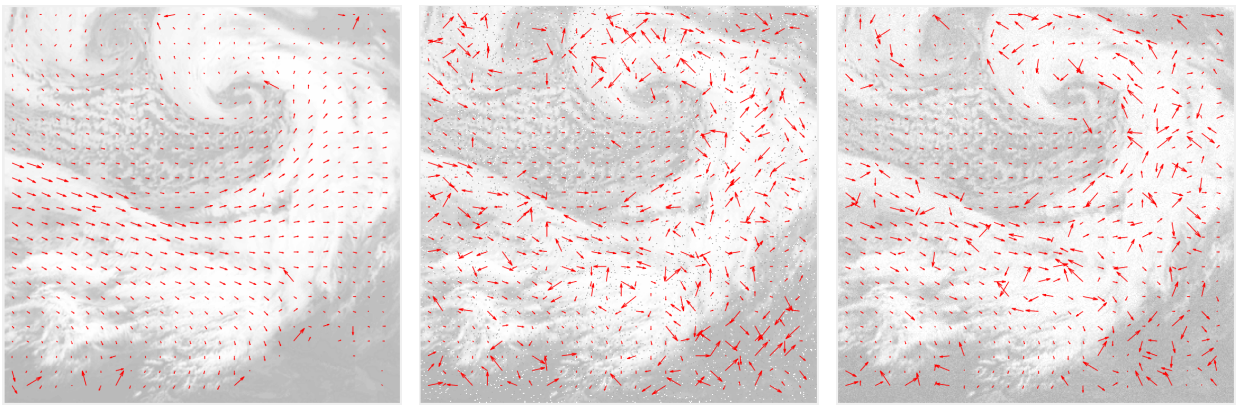
Table 3: Average runtimes for different similarity metrics

Method	**Average runtime (s)**
Distinct BBS	0.186
Sliding BBS	5.12
OMκ	1.41
CCC	0.068
SAD	0.048

8 Full-field motion estimation of clouds

8.1 Simple full field motion estimation

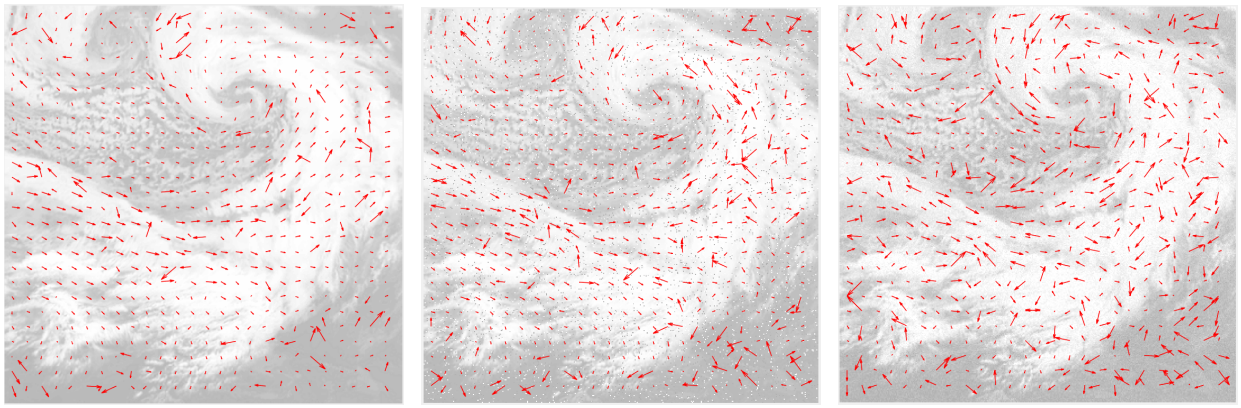
The estimation and creation of motion fields of clouds is done by dividing images taking from a time sequence into evenly sized templates, performing template matching on each template then taking the best match for each template and plotting the motion vector. However the best match in the correlation surface is often not the best possible motion vector; adding noise may reduce this accuracy even further depending on the metric used. Figure 51 shows the motion field generated by the the CCC using 15×15 pixel templates for the Set 1 images with no additional noise and the same image corrupted by 5% impulse noise and gaussian noise. Incorrect motion vectors that go against the expected flow in Figure 16 are present in all three versions shown, however the noise corrupted images have a large amount of incorrect vectors.



(a) (b) (c)

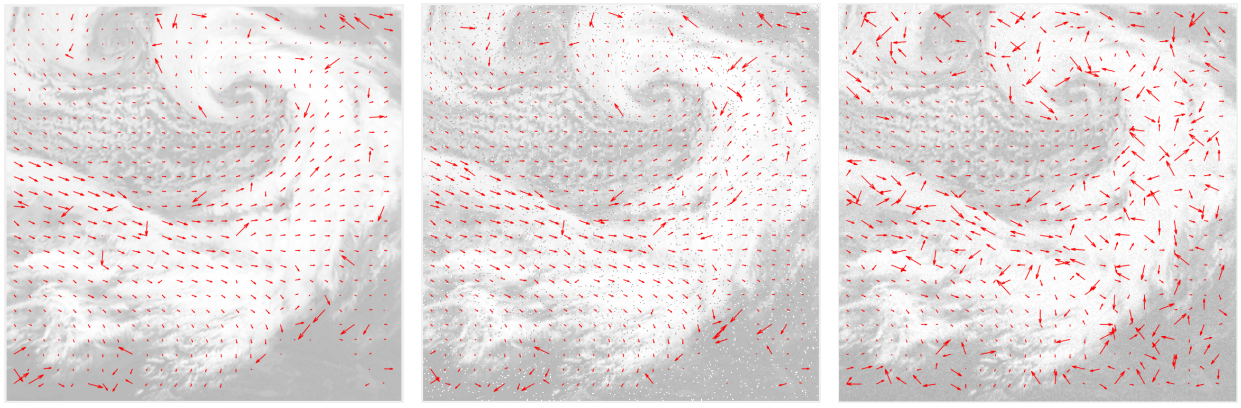
Figure 51: Motion field generated using the best match found for CCC. Image has been displayed with transparency to better show motion vectors. (a) Normal Set 1 image, (b) Impulse noise corrupted Set 1 image, (c) Gaussian noise corrupted Set 1 image

Next, Figures 52 and 53 show the motion field generated by using distinct BBS and OMκ respectively. For the noiseless image, the CCC motion field had the least wrong vectors, followed by OMκ and distinct BBS having the most. For the impulse noise images, OMκ had the least wrong vectors, with the motion field being comparable to the noiseless version. BBS also had a better motion field with less erroneous vectors than the CCC. Finally for the gaussian noise motion fields, the CCC and OMκ had a similar amount of bad vectors, with the CCC having a slight edge. BBS had the most bad vectors out of the metrics used for gaussian noise. Overall the CCC, distinct BBS and OMκ best match motion fields show results that match the comparative results of the quantitative tests. Sliding BBS was not used since the quantitative tests showed that it had similar accuracy performance to OMκ yet the computation time was over 4 times larger, which made the processing time to generate vectors for the 676 templates extremely long and not viable to use compared to the rest of the methods.



(a) (b) (c)

Figure 52: Motion field generated using the best match found for distinct BBS. Image has been displayed with transparency to better show motion vectors. (a) Normal Set 1 image, (b) Impulse noise corrupted Set 1 image, (c) Gaussian noise corrupted Set 1 image



(a) (b) (c)

Figure 53: Motion field generated using the best match found for OMκ. Image has been displayed with transparency to better show motion vectors. (a) Normal Set 1 image, (b) Impulse noise corrupted Set 1 image, (c) Gaussian noise corrupted Set 1 image

8.2 Probabilistic relaxation implementation

The vectors in the tests done above are selected due to being the top match for that template. However depending on the similarity metric used, the correlation surface may contain other maxima not initially chosen that are better matches. To select these better matches an implementation of probabilistic relaxation was coded based on the steps outlined in Section 2.2.2. The implementation was split into two main parts.

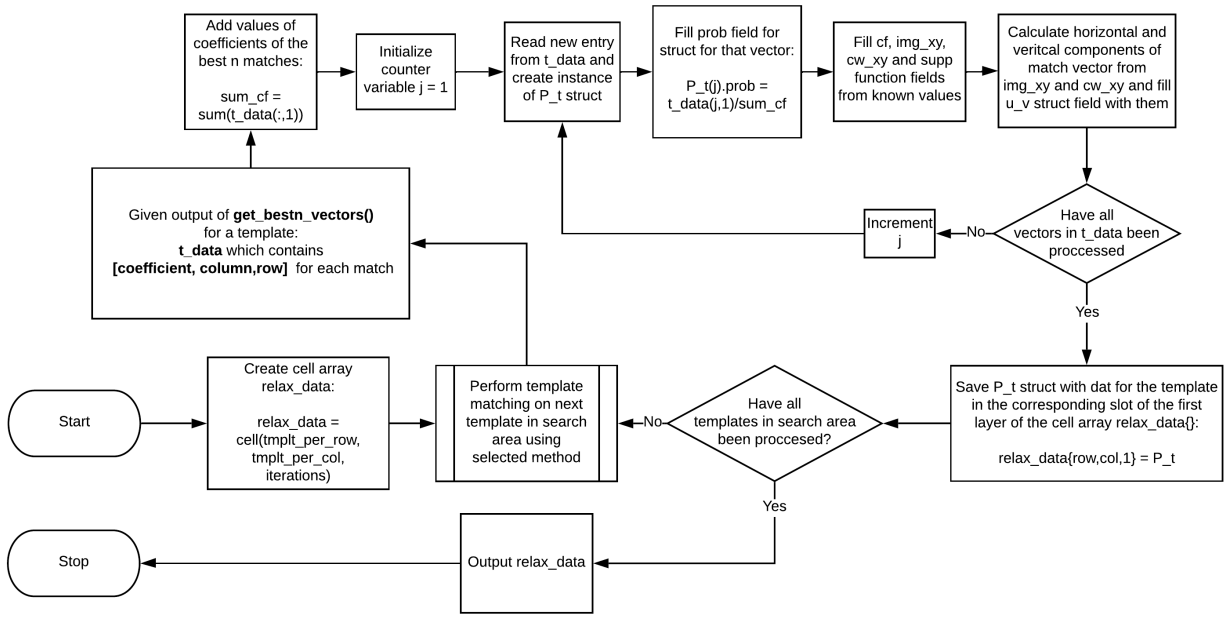


Figure 54: Relaxation labelling pre-processing flowchart

The first part focused on carrying out template matching, extracting the best n matches and placing them into a data structure alongside the needed starting values for the relaxation labelling. A 3-D MATLAB cell array called `T_mat` was created to hold template data corresponding to each template in the image in the x and y axis of the array. Each layer in the "z" axis of the array was reserved to hold the data of each template for each relaxation labelling iteration. For every template, an instance of a custom MATLAB `struct` called `P_t` was made that contained the following fields for each of the top n matches found for each template:

- `prob`: Contains the initial probability for that vector calculated as shown in Equation 11.
- `cf`: Contains the output value (coefficient) of the similarity metric for that vector (range from 0 to 1).
- `img_xy`: Contains the x and y coordinates of the template.
- `cw_xy`: Contains the x and y coordinates of the candidate window.
- `u_v`: Contains the horizontal and vertical components of the motion vector.
- `supp_funct`: Contains the value of the support function in Equation 13. Initialised to 1.

For each of the different matching methods, the implementations shown in Section 5 were used, since these were coded to be able to output the best n matches for each method based on input

arguments. The process of filling this data structure for each template is shown in Figure 54.

The next stage of the relaxation labelling implementation focused on the core section of iteratively adjusting the probabilities of each vector by adjusting the support function based on a mutual information measure between nearby vectors. For the mutual information measure, the horizontal and vertical components of the vectors were used for comparison instead of the angle and magnitude as shown in Equation 15. To calculate the support function and mutual information measure for each template and vector two functions were coded and these were used to perform the relaxation procedure alongside the main iteration loop as shown in the flowcharts in Figure 55 and 56.

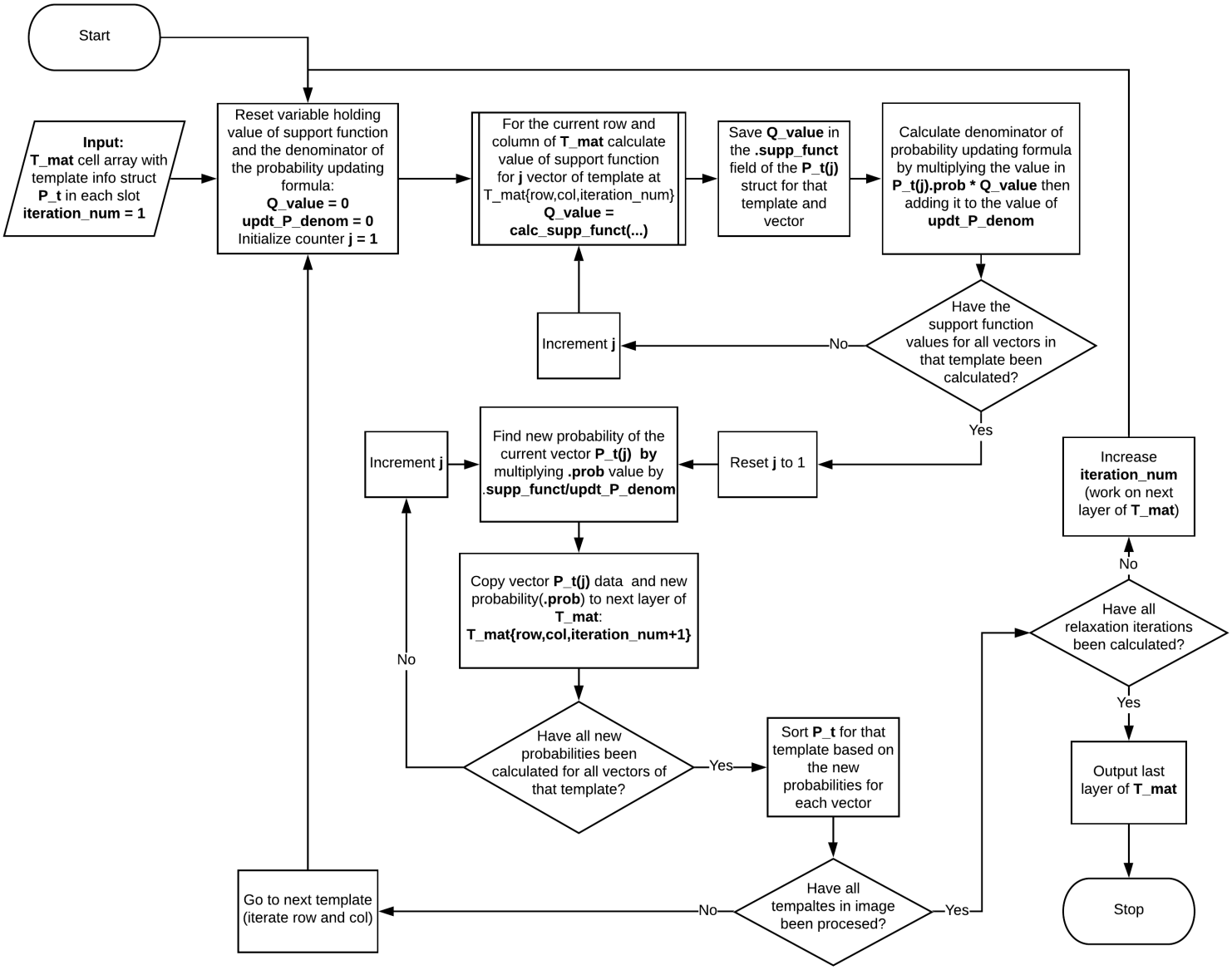


Figure 55: Relaxation labelling main loop flowchart

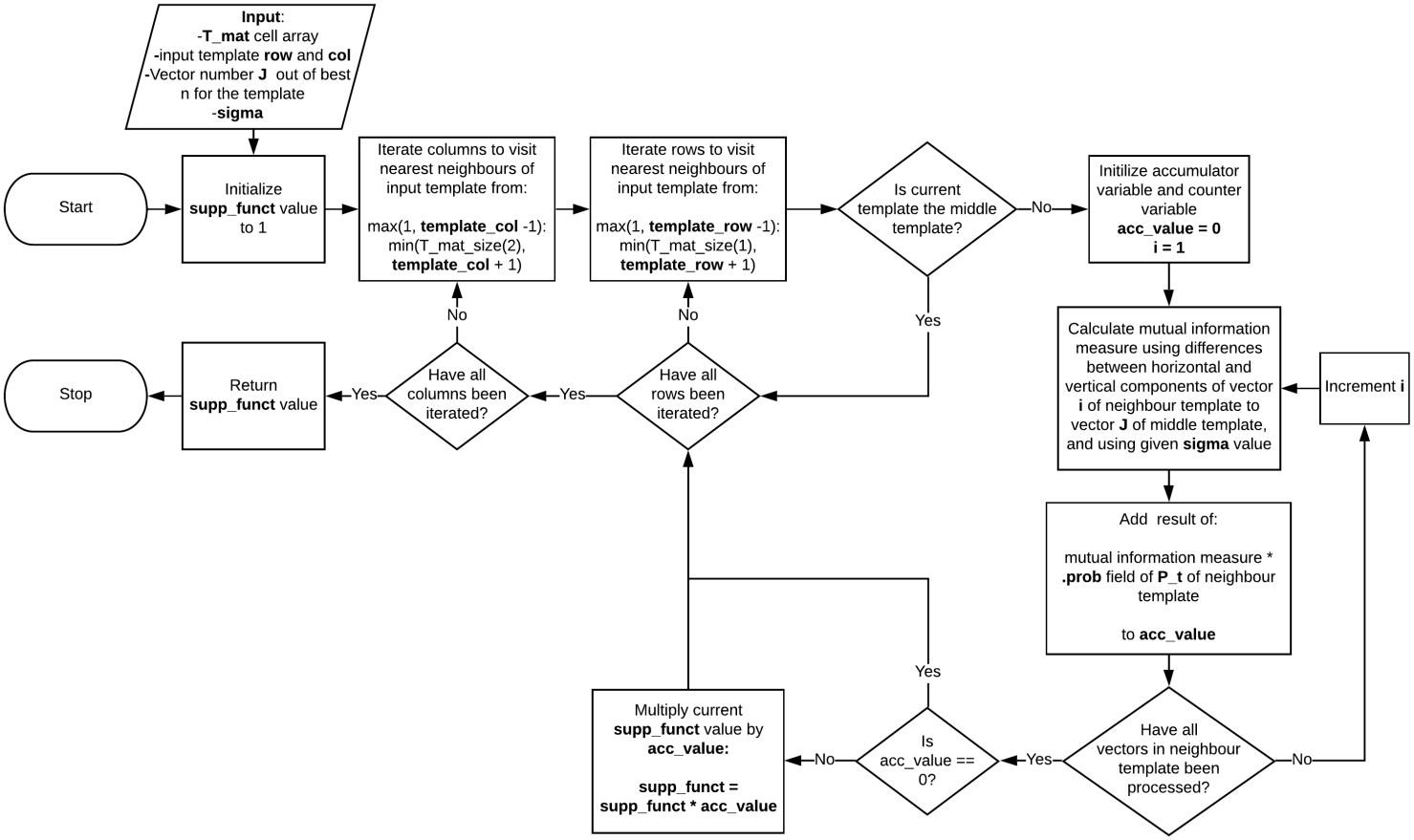
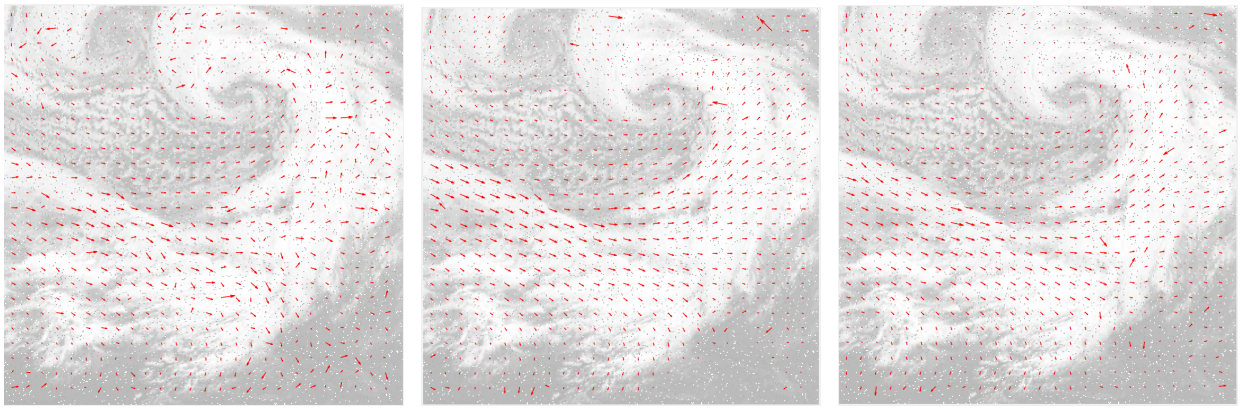


Figure 56: Support function calculation for given template and vector in T_mat cell array, function flowchart

Previous work [17] has used methods such as a vector median post filter to replace any remaining inconsistent vectors, however in order to compare performance between the different metrics, neither a post filter or a no match label was added to be able to analyse the amount of remaining inconsistent vectors for the motion fields created by each different metric after the relaxation process. Instead the conditional used by the post filter proposed in [12] shown in Equation 17 was used with a smoothness constraint of 0.7 to count the number of vectors that the post filter would have deemed inconsistent and therefore replaced.

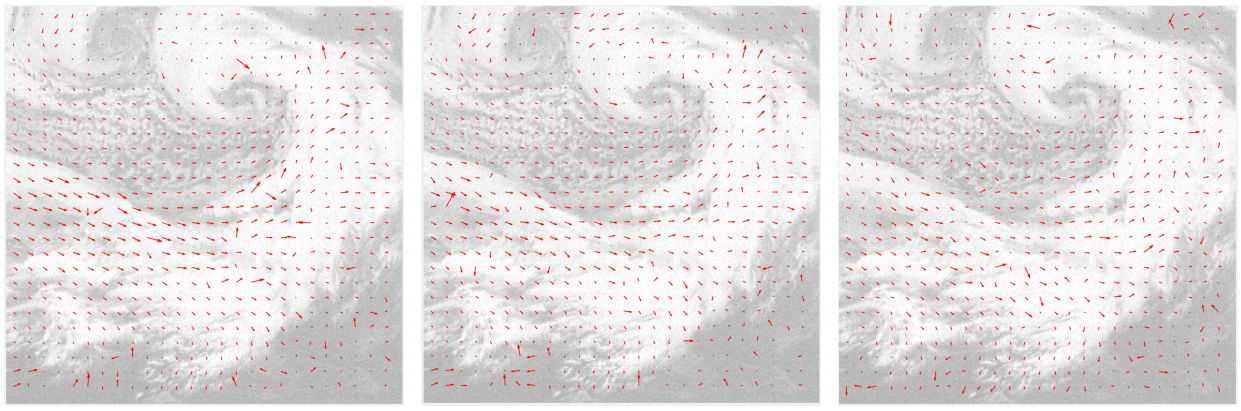
8.3 Relaxation labelling tests and results

For each of the relaxation labelling tests, the CCC, OMκ and distinct BBS metrics were chosen. The CCC was chosen as a baseline since it is an optimal method for relaxation labelling applications [14]. OMκ was also tested since it is method that is known to be a suitable metric for relaxation labelling that surpasses the CCC in noise corrupted images [11]. For each method, the input of the relaxation process was done by taking the top 15 vectors of each 15×15 template for a search size of $\pm$15pixels. The relaxation process then was run for 8 iterations using a σ value of 20 for Equation 15, which was tested to be enough iterations for all three methods to converge into not changing any further vectors in the motion field. The relaxation framework was tested on all three image tests for both a 15 minute time difference and a 30 minute time difference. The last iteration results were analysed both qualitatively by comparing the optical flow to the estimated flow in Figure 16 and quantitatively by counting the number of vectors rejected by the post filter.



(a) (b) (c)

Figure 57: Motion field after 8 iterations of relaxation process for Set 1 sequence 15 minutes apart corrupted by impulse noise using different methods as input. (a) CCC, (b) OMκ, (c) BBS



(a) (b) (c)

Figure 58: Motion field after 8 iterations of relaxation process for Set 1 sequence 15 minutes apart corrupted by gaussian noise using different methods as input. (a) CCC, (b) OMκ, (c) BBS

When looking at the output of the relaxation process, BBS and OMκ performed very similarly

in creating consistent smooth motion fields for both impulse and gaussian corrupted sequences. An example of the final output for the relaxation procedure for CCC, OMκ and BBS for impulse and gaussian noise corrupted sequences is seen in Figure 57 and 58 respectively. The rest of the motion fields outputs for each metric tested with no noise, impulse noise and gaussian noise for image sets 1, 2 and 3 can be seen in Appendices A, B and C respectively. Overall, for sequences with impulse noise BBS was able to produce acceptable optical flows with less bad vectors when compared those produced by CCC. When compared to OMκ for both noiseless and impulse noise corrupted sequences, BBS produced similar motion fields. For the sequences with gaussian noise, the optical flow for all three methods had larger amounts of bad vectors. The CCC output had better smoothness out of the three methods for the gaussian noise test, with BBS and OMκ producing fields with an overall correct direction, but jagged appearance in the optical flow.

Table 4: Number of inconsistent vectors determined by post filter with and without relaxation labelling for the average of all three image sets images for 15 and 30 minute time differences

Input method	**Noise type**	**Percentage of inconsistent vectors (15 minutes delay)**		**Percentage of inconsistent vectors (30 minutes delay)**	
		No relaxation	Last iteration	No relaxation	Last iteration
CCC	None	15.53	4.29	27.66	11.09
	Impulse	68.78	33.73	71.45	30.76
	Gaussian	54.58	24.41	59.02	21.89
OMκ	None	22.04	6.06	39.35	14.20
	Impulse	25.00	6.06	39.20	14.94
	Gaussian	65.82	32.25	65.24	25.89
BBS	None	38.17	6.80	48.96	11.09
	Impulse	48.67	12.72	55.33	13.61
	Gaussian	72.49	29.73	75.59	30.91

Next the post filter was run for each of the motion fields generated by each metric before and after the relaxation process. Table 4 shows the percentage of inconsistent vectors that the post filter rejected for all three similarity metrics, for noiseless, impulse noise and gaussian noise; averaged for the three different test image sequences. On average, these test show that CCC, OMκ and BBS have good performance as an input for relaxation labelling for noiseless images. For impulse noise corrupted images OMκ and BBS had similar results, both outperforming the CCC. For gaussian noise all three methods had poor results.

When testing a time difference of 30 minutes between the images instead of 15 minutes, all three methods had more bad vectors due to increased deformation in the templates caused the non-rigid motion of clouds. Even with a larger time difference, consistent smooth motion fields were still able to be generated for the sequences with no noise and impulse noise.

Overall due to the reduced time complexity of distinct BBS compared to OMκ, BBS provides a good alternative for use within a relaxation labelling framework for cloud image sequence corrupted by impulse noise. However the risk of the reduced resolution due to the patch size of distinct

BBS is still present if this method is chosen and as such the choice between $OM\kappa$ and BBS for impulse noise corrupted sequences compromises a slight decrease in match accuracy for improved computational runtime. While BBS had close results to the CCC for the noiseless and gaussian noise sequences, the CCC is still much faster computationally than BBS for these applications and thus remains the preferred choice.

9 Conclusion

The main aims for the project were all met. SAD, CCC and OMκ similarity metrics were all succesfully implemented. Several image sequences were obtained from MSG satelittes and a custom ground truth bounding box testing data set was created across three different image sequences to quantitatively assess performance of the different metrics for cloud tracking.

Two different implementations for the BBS method were coded. The first, distinct patch BBS was closely based on the originally proposed method in [4]. The second, sliding patch BBS was coded to solve the main issue found for the application of cloud tracking during the course of the project that caused the use of non-overlapping patches to limit the pixels at which a template could be matched at. Using quantitative analysis, optimal values of the input parameters of BBS λ and patch size were found for the application of cloud tracking to be $\lambda = 0.3$ using 2×2 pixel patches. Sliding patch BBS consistently outperformed distinct patch BBS in accuracy tests but was not deemed a suitable implementation, as this performance increase came at an extreme cost of computational complexity, yet it did not outperform other methods, only matching the accuracy performance of the OMκ metric and having worse results for noiseless images than both CCC and SAD.

A working implementation for relaxation labelling framework was coded and the different similarity metrics including BBS were succesfully added as the input layer to the relaxation framework. Several different parameters of relaxation labelling were tested, with 8 iterations with $\sigma = 20$ being used for the final performance analysis tests.

While the BBS method has been proven to be a suitable method for full field motion estimation of clouds as the input for a relaxation labelling framework for the specific case of impulse noise corrupted sequences, it is overall not a great template matching similarity metric for cloud tracking applications. The limited non-overlapping patches reduce the resolution of the matches to a minimum of a potential match every two pixels and the proposed method to overcome this limitation has an extreme computational cost. While OMκ had better smoothness and less wrong vectors over BBS for impulse noise corrupted sequences, the optical flow created for these two methods was similar, yet BBS had a lower computational cost. For all other situations, other methods performed similarly or better than BBS, while also having better runtime performances.

10 Further work

The main focus of this project was testing BBS for cloud tracking of greyscale satellite images from the infrared channel of MSG satellites of 15 minute intervals and its viability of use within a relaxation labelling framework. With more time, a larger variety of clouds image sequences could be analyzed and a larger set of quantitative test data would be able to be created for more precise tests.

The BBS method uses either a colour descriptor or deep feature descriptor calculate BBPs. For this project only an adaptation of the 3-channel colour descriptor to single channel intensity descriptor was used to test the performance for cloud tracking. A further area of work for BBS in cloud tracking would be to adapt the 3-channel colour descriptor of BBS to a 2-channel descriptor combining IR and visible light channels of the MSG SEVIRI instruments (similar work has been proven to be effective for the CCC metric in [17]).

Another area worth investigating given more time would be the to assess the effectiveness of BBS for use with artificially coloured images of clouds. EUMETSAT provides RGB composites and visualized products of the MSG satellite images of clouds for different features such as cloud top height. The additional properties of BBS not exploited for the single channel greyscale descriptor could be used in conjunction with these coloured images to provide better tracking results than other metrics.

Finally while this project focused predominantly on cloud tracking, other remote sensing and non rigid body deformation applications of BBS may be further investigated, although the limitations of resolution and discriminatory power for small sized templates in images where single pixel accuracy is important, might not make it a suitable metric for these kinds of applications.

References

[1] Q. X. Wu, "A correlation-relaxation-labeling framework for computing optical flow-template matching from a new perspective," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 843–853, Sep. 1995.

[2] A. N. Evans, "Full field motion estimation for large non rigid bodies using correlation/relaxation labelling," in *1997 Sixth International Conference on Image Processing and Its Applications*, vol. 2, pp. 473–477 vol.2, July 1997.

[3] T. Dekel, S. Oron, M. Rubinstein, S. Avidan, and W. T. Freeman, "Best-buddies similarity for robust template matching," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2021–2029, June 2015.

[4] S. Oron, T. Dekel, T. Xue, W. T. Freeman, and S. Avidan, "Best-buddies similarity—robust template matching using mutual nearest neighbors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 1799–1813, Aug 2018.

[5] D. Martinez Amigo, "Interim report - template matching using best buddies similarity and its application to cloud tracking," 2020. University of Bath.

[6] O. Shahine, H. Kelash, G. Attiya, and O. Faragallah, "Breast cancer detection based on dynamic template matching," vol. 20, pp. 193–205, 12 2013.

[7] W. Chantara, J.-H. Mun, D.-W. Shin, and Y.-S. Ho, "Object tracking using adaptive template matching," *IEIE Transactions on Smart Processing and Computing*, vol. 4, pp. 1–9, 02 2015.

[8] W. Ouyang, F. Tombari, S. Mattoccia, L. Di Stefano, and W. K. Cham, "Performance evaluation of full search equivalent pattern matching algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, pp. 127–143, Jan 2012.

[9] A. Rosenfeld and A. C. Kak, *Digital picture processing, volume 2.* Academic Press, 1982.

[10] D. N. Bhat and S. K. Nayar, "Ordinal measures for image correspondence," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, p. 415–423, Apr. 1998.

[11] A. Evans, "On the use of ordinal measures for cloud tracking," *International Journal of Remote Sensing*, vol. 21, pp. 1939–1944, 6 2000.

[12] A. Evans, "Glacier surface motion computation from digital image sequences," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 38, pp. 1064 – 1072, 04 2000.

[13] T. Horinouchi, S.-Y. Murakami, T. Kouyama, K. Ogohara, A. Yamazaki, M. Yamada, and S. Watanabe, "Image velocimetry for clouds with relaxation labeling based on deformation consistency," *Measurement Science and Technology*, vol. 28, no. 8, 2017.

[14] Q. X. Wu, S. J. Mcneill, and D. Pairman, "Correlation and relaxation labelling: An experimental investigation on fast algorithms," *International Journal of Remote Sensing*, vol. 18, no. 3, pp. 651–662, 1997.

[15] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxational operations," *IEEE Trans. Syst., Man, Cybern., vol. SMC–6*, pp. 420–433, 12 1976.

[16] Q. X. Wu and D. Pairman, "A relaxation labeling technique for computing sea surface velocities from sea surface temperature," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 1, pp. 216–220, 1995.

[17] A. N. Evans, "Cloud motion analysis using multichannel correlation-relaxation labeling," *IEEE Geoscience and Remote Sensing Letters*, vol. 3, no. 3, pp. 392–396, 2006.

[18] D. M. A. Aminou, B. Jacquet, and F. Pasternak, "Characteristics of the Meteosat Second Generation (MSG) radiometer/imager: SEVIRI," in *Sensors, Systems, and Next-Generation Satellites* (H. Fujisada, ed.), vol. 3221, pp. 19 – 31, International Society for Optics and Photonics, SPIE, 1997.

[19] MathWorks, "MATLAB documentation: rgb2gray." [Online] Availible at: https://www.mathworks.com/help/matlab/ref/rgb2gray.html.

[20] Y. Wu, J. Lim, and M. Yang, "Online object tracking: A benchmark," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2411–2418, 2013.

[21] J. Marcello, F. Eugenio, and F. Marques, "Cloud motion estimation in seviri image sequences," vol. 3, pp. III–642, 08 2009.

[22] MathWorks, "MATLAB documentation: maxk." [Online] Availible at: https://www.mathworks.com/help/matlab/ref/maxk.html.

[23] MathWorks, "MATLAB documentation: mink." [Online] Availible at: https://www.mathworks.com/help/matlab/ref/mink.html.

[24] MathWorks, "MATLAB documentation: surf." [Online] Availible at: https://www.mathworks.com/help/matlab/ref/surf.html.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[26] MathWorks, "MATLAB documentation: im2col." [Online] Availible at: https://www.mathworks.com/help/images/ref/im2col.html.

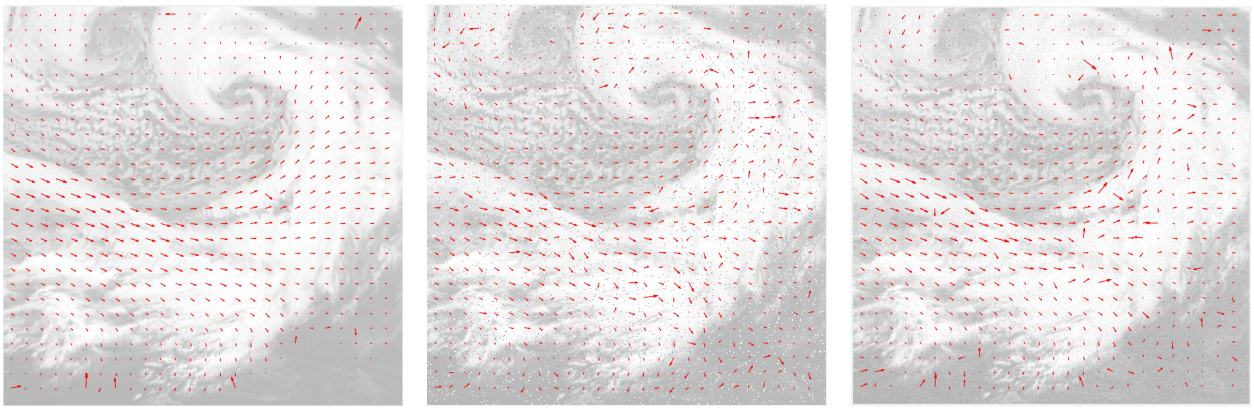
[27] MathWorks, "MATLAB documentation: squareform." [Online] Availible at: https://www.mathworks.com/help/stats/squareform.html.

[28] MathWorks, "MATLAB documentation: pdist." [Online] Availible at: https://www.mathworks.com/help/stats/pdist.html.

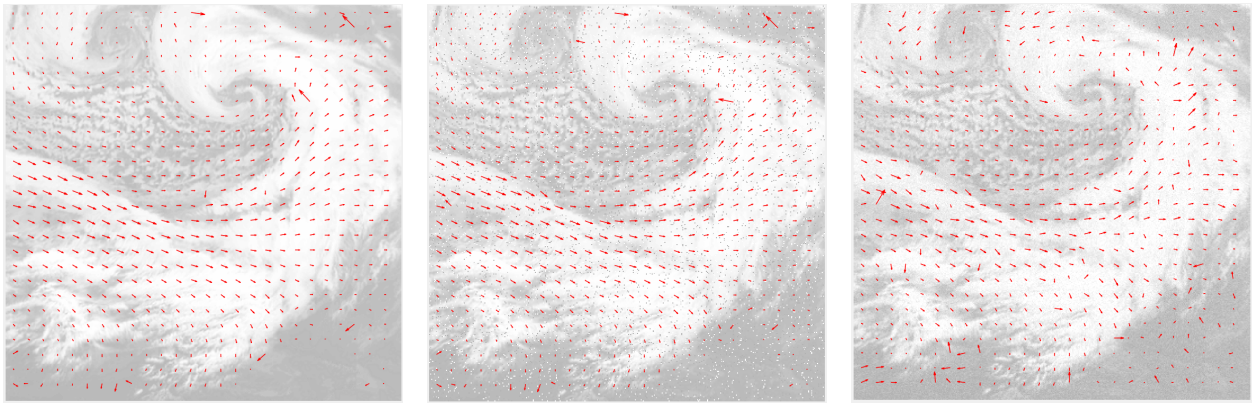
[29] MathWorks, "MATLAB documentation: imnoise." [Online] Availible at: https://www.mathworks.com/help/images/ref/imnoise.html.

Appendices

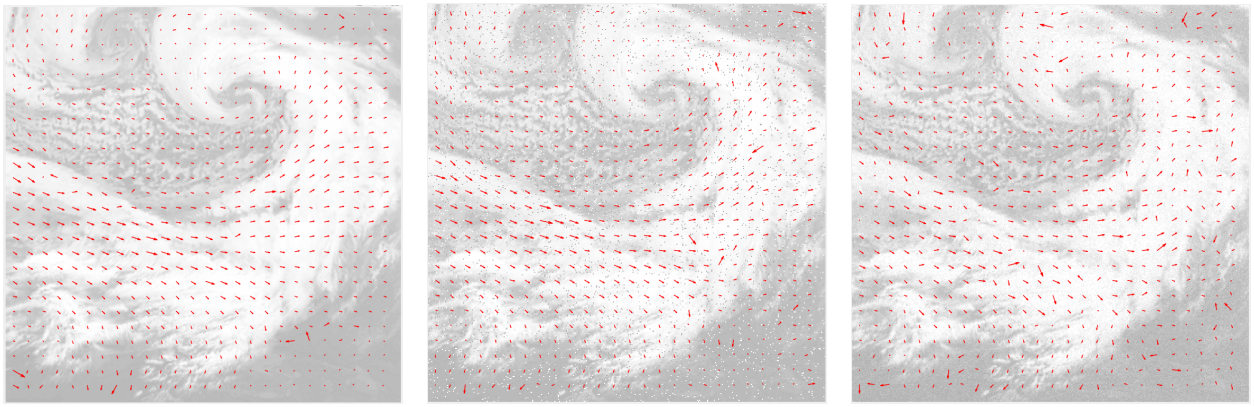
A Set 1 relaxation labelling output motion fields



(a) CCC normal (b) CCC impulse noise (c) CCC gaussian noise

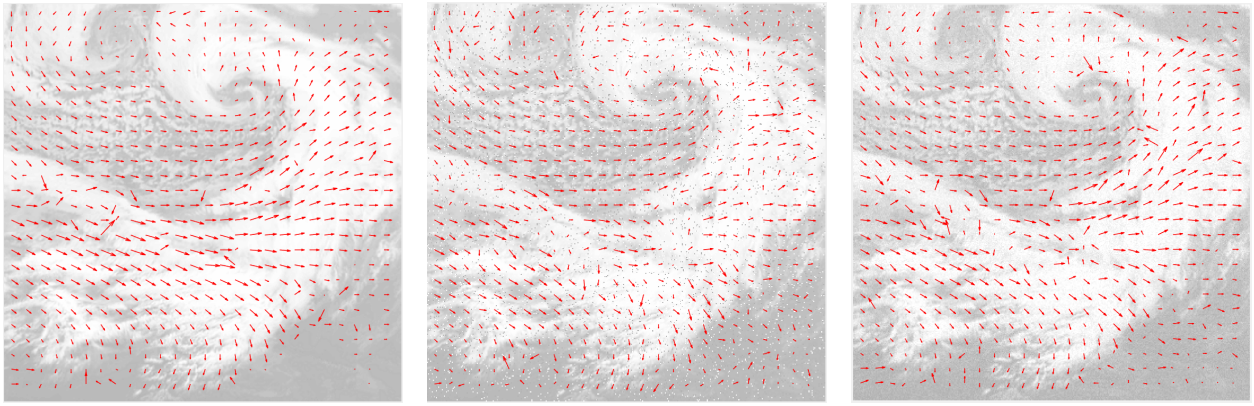


(d) OMκ normal (e) OMκ impulse noise (f) OMκ gaussian noise

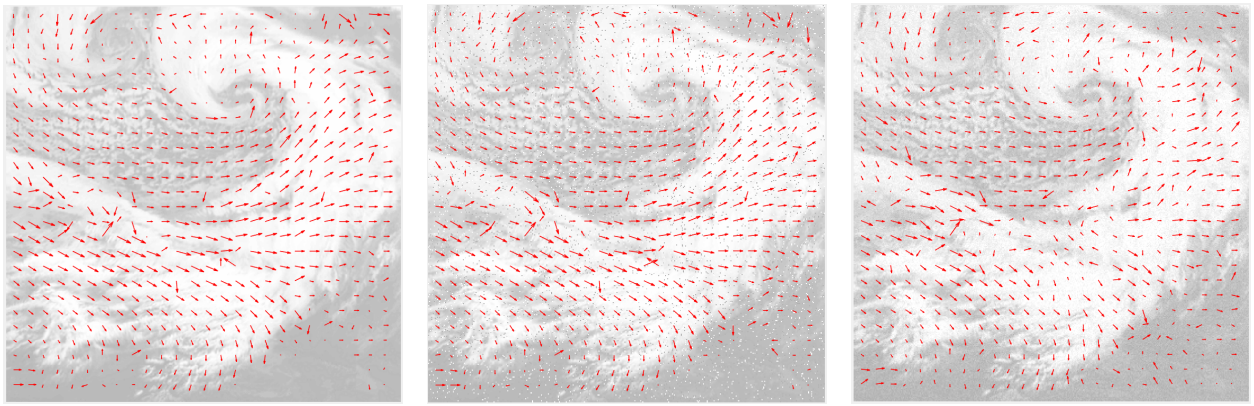


(g) BBS normal (h) BBS impulse noise (i) BBS gaussian noise

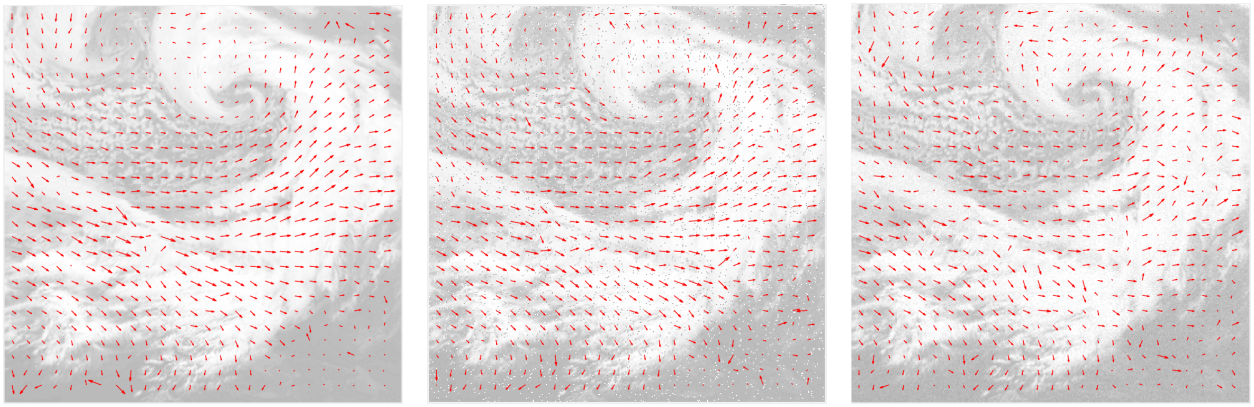
Figure 59: Relaxation labelling output after 8 iterations for Set 1 image sequence 15 minutes apart for different input methods and different noise corruptions



(a) CCC normal (b) CCC impulse noise (c) CCC gaussian noise



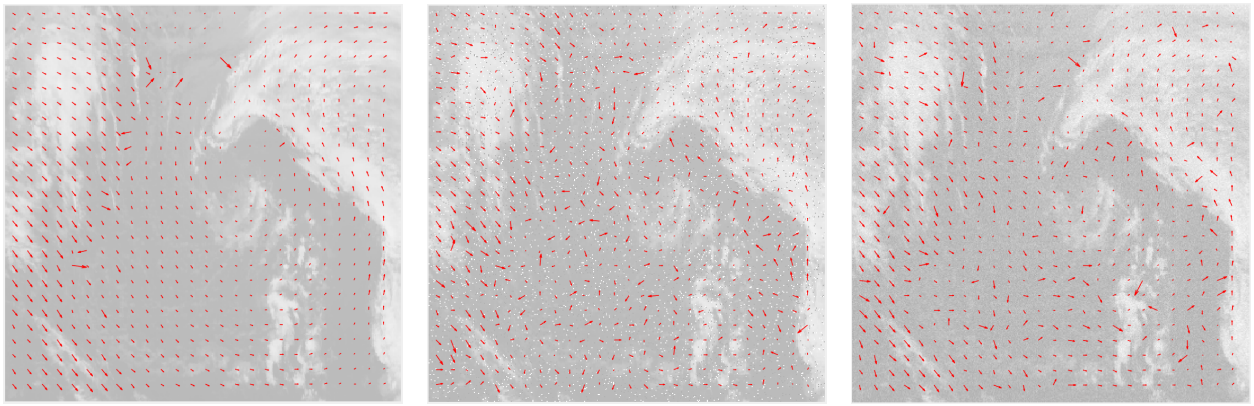
(d) OMκ normal (e) OMκ impulse noise (f) OMκ gaussian noise



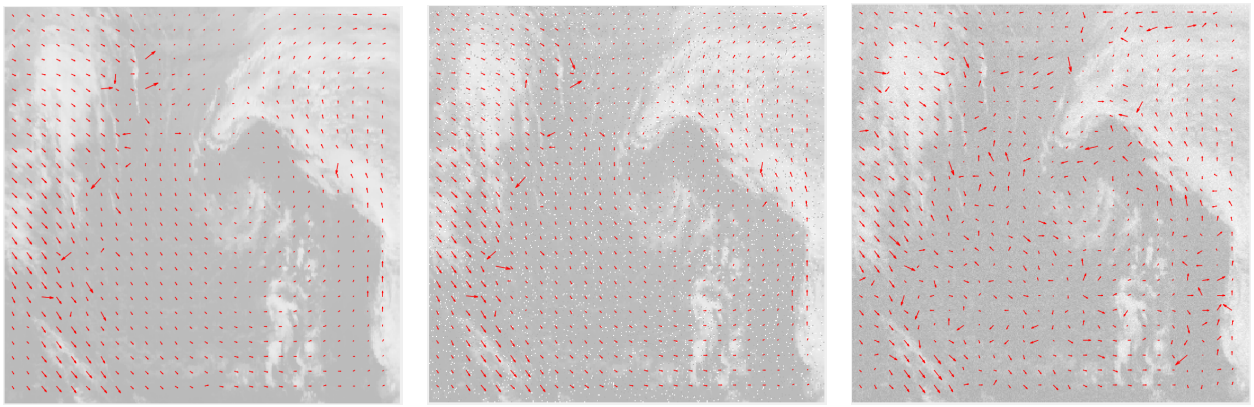
(g) BBS normal (h) BBS impulse noise (i) BBS gaussian noise

Figure 60: Relaxation labelling output after 8 iterations for Set 1 image sequence 30 minutes apart for different input methods and different noise corruptions

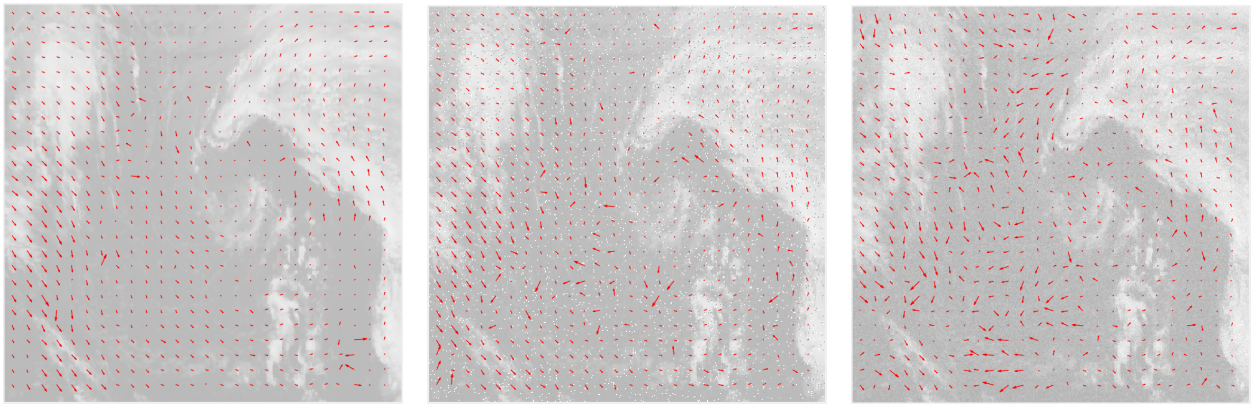
B Set 2 relaxation labelling output motion fields



(a) CCC normal (b) CCC impulse noise (c) CCC gaussian noise

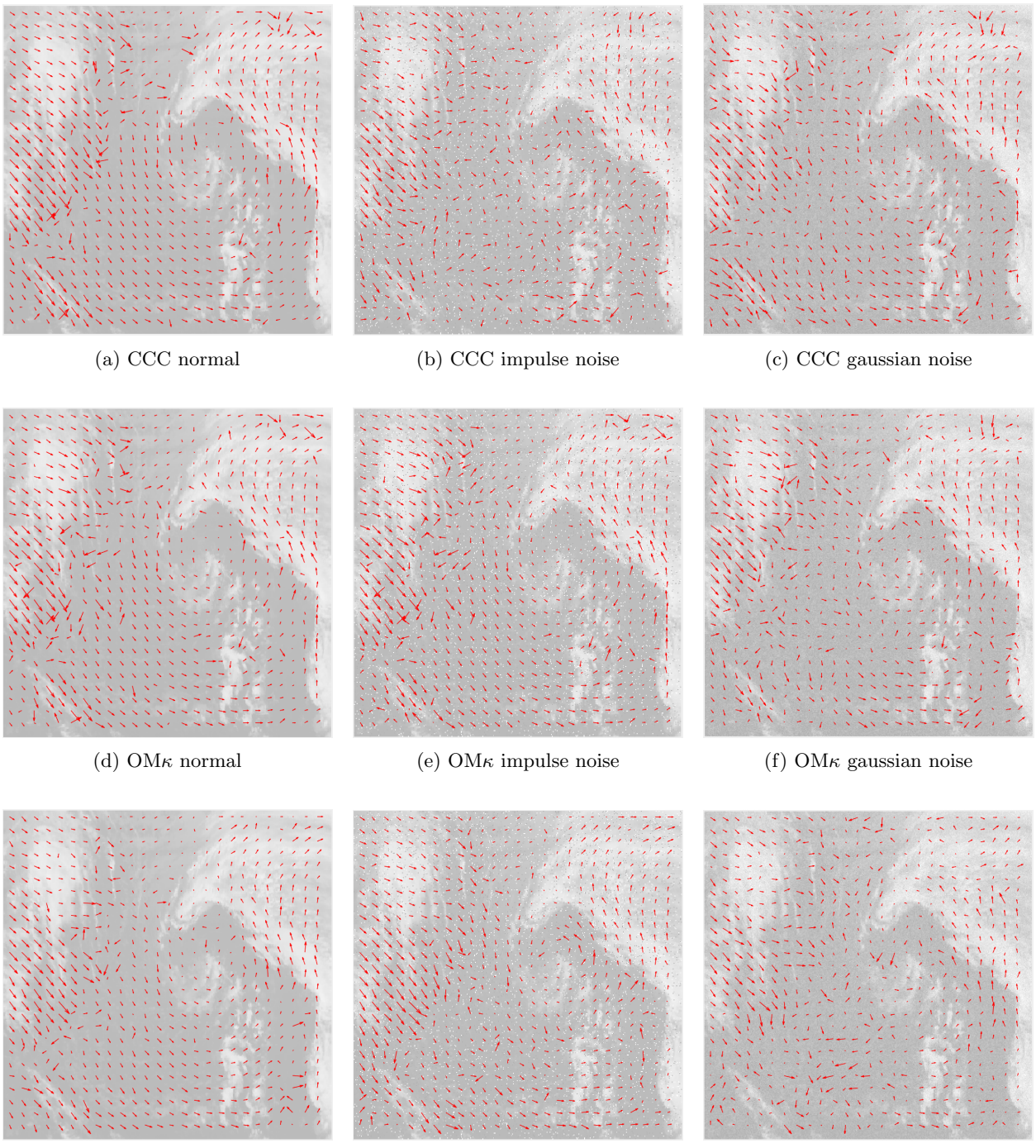


(d) OMκ normal (e) OMκ impulse noise (f) OMκ gaussian noise



(g) BBS normal (h) BBS impulse noise (i) BBS gaussian noise

Figure 61: Relaxation labelling output after 8 iterations for Set 2 image sequence 15 minutes apart for different input methods and different noise corruptions



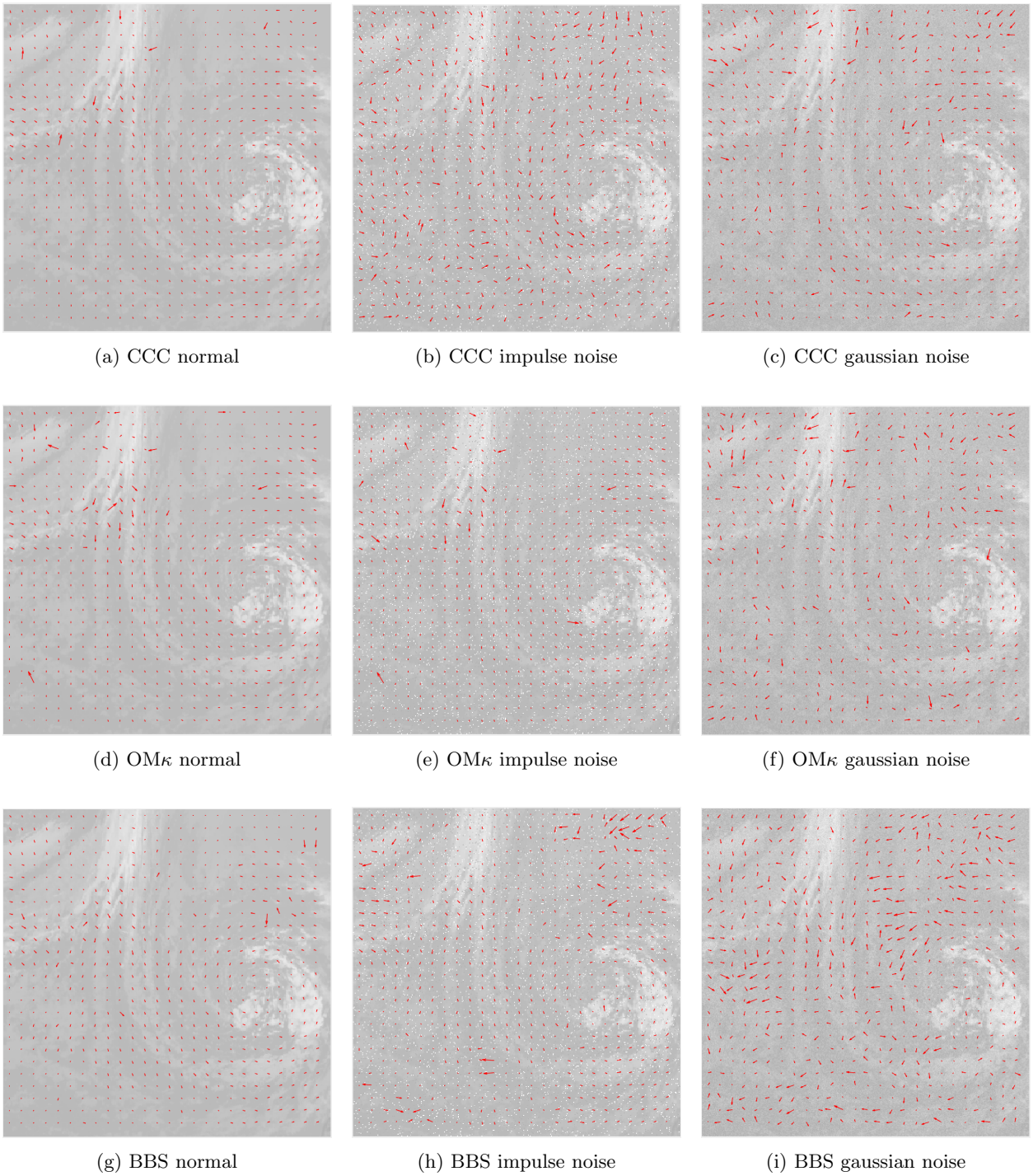
(a) CCC normal (b) CCC impulse noise (c) CCC gaussian noise

(d) OMκ normal (e) OMκ impulse noise (f) OMκ gaussian noise

(g) BBS normal (h) BBS impulse noise (i) BBS gaussian noise

Figure 62: Relaxation labelling output after 8 iterations for Set 2 image sequence 30 minutes apart for different input methods and different noise corruptions

C Set 3 relaxation labelling output motion fields

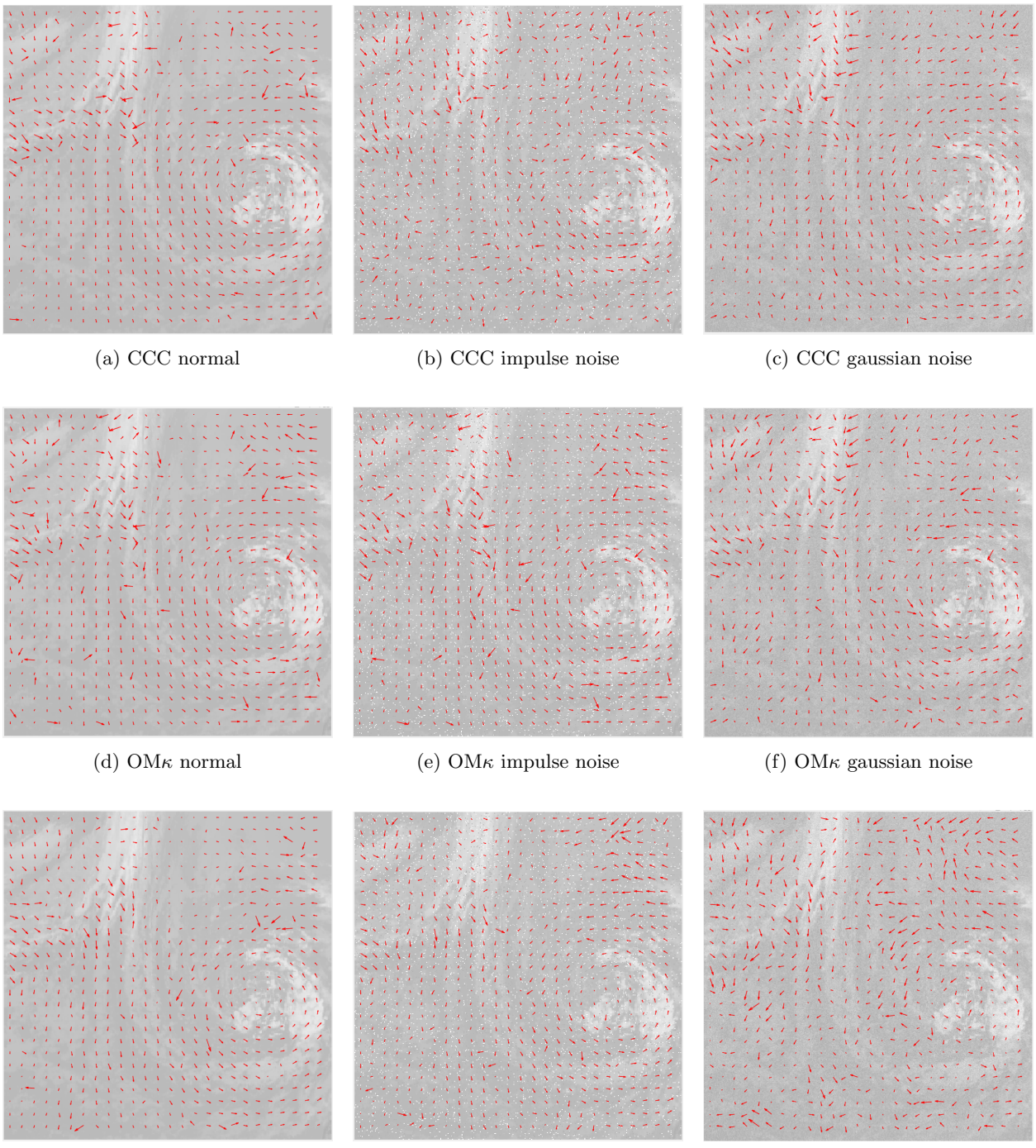


(a) CCC normal (b) CCC impulse noise (c) CCC gaussian noise

(d) OMκ normal (e) OMκ impulse noise (f) OMκ gaussian noise

(g) BBS normal (h) BBS impulse noise (i) BBS gaussian noise

Figure 63: Relaxation labelling output after 8 iterations for Set 3 image sequence 15 minutes apart for different input methods and different noise corruptions



(a) CCC normal (b) CCC impulse noise (c) CCC gaussian noise

(d) OMκ normal (e) OMκ impulse noise (f) OMκ gaussian noise

(g) BBS normal (h) BBS impulse noise (i) BBS gaussian noise

Figure 64: Relaxation labelling output after 8 iterations for Set 3 image sequence 30 minutes apart for different input methods and different noise corruptions