# Motion Estimation in Remote Sensed Multi-Channel Images

M. P. Foster
MEng Computers, Electronics & Communications,
University of Bath

May 12, 2005

“I certify that I have read and understood the entry in the Student Handbook for the Department of Electrical and Electronic Engineering on Cheating and Plagiarism and that all material in the assignment is my own work, except where I have indicated with appropriate references.”

**Abstract**

This report begins by introducting the concepts and background of, remote sensed data, motion estimation (ME) and non-linear relaxation labelling (RL). It then introduces the idea of multi-channel data, and discusses how a ME RL system might be extended to use it.

Next, the tools and languages used to implement a ME RL system are described, and an Octave/Matlab toolkit implementing such a system is introduced, along with various scripts and functions which help simplify certain operations like plotting, and accessing certain infomation. The usage and style of the toolkit is descried and explained, allowing future use of the code by others. In the results chapter, several sets of results are shown. The distribution of active blocks across two channels for various block sizes and variances are analysed, and results for both single and multi-channel ME and RL are given. Finally, the ME and RL results are discussed, and conclusions drawn about the distribution, consistency and quality of the output vector fields presents in the results section.

Finally, practical and future uses for such systems are discussed.

# Contents

# List of Figures

# Symbols and Abbreviations

| Symbol | Description |
|---|---|
| $D_{I,J}$ | The distance between two blocks, relative to the size of a block. |
| $\Delta y_m$ or $\Delta x_m$ | The median displacement of a set of vectors. |
| $G$ | The set of all blocks considered neighbours of a given block. |
| $i$ or $j$ | Vectors |
| $I$ or $J$ | A specific template/block. |
| $J \rightarrow j$ | A vector $j$ within the template $J$. |
| $n$ | Any of the natural numbers ($n \in \mathcal{Z}$). |
| $\Omega_{2J}$ | Set of vectors for the template/block ($J$). |
| $P^{(n)}(J \rightarrow j)$ | The probability attached to vector $j$, in template $J$ ater $n$ iterations. |
| $\rho$ | The Cross Correlation Coefficient |

Table 1.1: Symbols

| Abbreviation | Meaning |
|---|---|
| CCC | Cross Correlation Coefficient |
| CLI | Command Line Interface |
| CVS | Concurrent Version System |
| DSRS | Dundee Satellite Receiving Station |
| GUI | Graphical User Interface |
| IR | Infra Red |
| ME | Motion Estimation |
| MSE | Mean Square Error |
| RL | Relaxation Labelling |
| SAVD | Sum of Absolute Value of Differences |
| SSD | Sum of Squared Differences |
| UM | Unified Model |
| WV | Water Vapour |

Table 1.2: Abbreviations

# Introduction

Techniques that estimate motion in single channel images have existed for some time, and are now integrated into everyday technology like satellite set top boxes. Less widespread, is the use of more advanced motion estimation (ME) techniques, like relaxation labelling, which improve the output of traditional ME schemes, and are used in fields like object tracking, meteorology and could (eventually) have uses in defense systems and warfare. Even more obscure are systems which use multi-channel data to estimate motion; a field which is virtually untouched[1].

The aim of this project is to develop a ME system which is capable of using multi-channel data[2] in order to asses the possible advantages that this could give. In order to carry out this task, the following questions need to be answered:

1. What are the characteristics of remote sensed images?
2. How do traditional ME systems work?
3. When is multi-channel data useful? How can this usefulness be guaged?
4. How can traditional ME systems be adapted to make use of multi channel images?
5. How well do the adapted techniques work on multi channel remote sensed data?
6. How can the above findings be used in practice, and how could they be used in the future?

In the following sections, the above questions will be answered in order, starting, of course, with a discussion of remote sensed images, and then moving onto an examination of traditional motion estimation systems, how they can be further improved, and then how they can be extended to support multi-channel imagery.

## 2.1 Remote Sensed Images

Remote sensed images are generally images which have been gathered electronically using remote instruments like satellites, probes, or from measurements derived from some other *remote* platform. Images need not represent EM information (which is, however, the norm) and could instead show information such as attenuation due to water vapour, the distribution of certain gasses, or any data that could be plotted spatially.

Remote sensed images are normally characterised by the following:

- **Overall low contrast**. Most satellite images are greyscale, and are inherently low contrast.
- **Low spatial resolution**: although this is becoming less and less true, remote links are often low bandwidth because of path loss, weight limits or other reason, and therefore resolution is sacrificed. This is not the case with most modern satelite imagery, where high resolution cameras are often used.
- **Non rigid objects:** especially in satellite imagery, where there may be clouds which are highly non-rigid, occlude one another, and have the annoying property of disappearing!
- **Low or very low temporal resolution**. Unlike video sequences where there could be as many as 25 frames per second, remote sensed sequences tend to have a time between frames of minutes, or hours. This means that using motion estimation systems like optical flow is totally out of the question.

[1] See [Phi04]
[2] See section 3.4

# Motion Estimation and Relaxation Labelling

## 3.1 Motion Estimation

This section aims to answer the question: *How do traditional motion estimation systems work?*.

Motion estimation is a technique which aims to produce some kind of an estimate of the motion between areas in two or more images. [1]

Generally, there are three basic assumptions made by motion estimation systems, these are listed below in order of importance:

- The **maximum velocity** that any given object in the scene can travel at is a known quantity. This allows a search bound to be imposed, based on the fact that the maximum distance the object can have travelled is the product of the maximum velocity, and the time delay between frames.
- The change in velocity is bounded by a small constant: i.e there is a **small acceleration**.
- Each point in a frame corresponds to exactly one other point in subsequent frames. I.e points are **mutually correspondent**. This is listed last because it is not always the case with weather images, and is not too important if suitable processing occurs after block matching. Generally, the property of mututal correspondance only holds true for motion of rigid bodies, where there is no rotation.

The idea of motion estimation using blocks matching is similar to *template matching*, a technique where a known pattern, or template is compared with an image, by shifting and rotating the template over the image, and looking at the similarities between the template and the image (see [MSB99], chapter 15, and [GW01], chapter 4, which deals with template matching in the frequency domain). When the similarity metric is high enough, the template has been found in the image. Motion estimation expands on this by making the template a section of an image (the first frame), and searching in exactly the same way.

Typically, motion estimation (across two images) is carried out by diving the images into blocks and then comparing the blocks in one image with translated versions of the same blocks in the next image. This is known as block matching. Comparisons between the image blocks can be carried out in a variety of ways, but as with most things, there is a trade off involved. In this case, the trade off is between how well the comparisons work (i.e how good the vectors are), and how computationally intensive they are (i.e how many instructions must be carried out). Some of the commonly used comparison metrics include:

- The Sum of Absolute Value of Differences (SAVD). A reasonably simple calculation which, unmodified, takes no account of the mean intensity of the block being examined. This SAVD is a *dissimilarity* measure, meaning that identical blocks return a zero result.

$$SAVD = \sum_{x=1}^{N}\sum_{y=1}^{N} |f(x,y) - g(x+u, y+v)| \tag{3.1}$$

  In order to improve the results from the SAVD, it is a good idea to make it *mean invariant* (i.e to subtract the mean and make it invariant to intensity changes across blocks).

$$SAVD = \sum_{x=1}^{N}\sum_{y=1}^{N} \left| \Big(f(x,y) - \overline{f(x,y)}\Big) - \Big(g(x+u, y+v) - \overline{(g(x+u, y+v)}\Big) \right| \tag{3.2}$$

[1] The output from a motion estimation system could be field of vectors, a set of affine matrices, a system of equations describing the relationships between the images, or any other suitable way of describing how things have changed. For the purposes of this project, the output will always be a field of vectors, called a 'motion field'.

- The Sum of Squared Differences (SSD). This is similar to the SAVD, however the terms are first squared, and then square rooted in order to make the equation independent of the sign of the differeces.
- The Cross Correlation Coefficient (CCC). The CCC technique is more computationally intensive than the methods listed above. CCC is a *similarity* measure (giving results between $\pm 1$, where a value of 1 means that the blocks are identical:

$$\rho = \frac{\sum_{x,y}\left(f(x,y) - \overline{f(x,y)}\right) \times \left((g(x+u,y+v) - \overline{g(x+u,y+v)}\right)}{\left(\sum_{x,y}\left(f(x,y) - \overline{f(x,y)}\right)^2\right)^{\frac{1}{2}} \times \left(\sum_{x,y}\left(g(x+u,y+v) - \overline{g(x+u,y+v)}\right)^2\right)^{\frac{1}{2}}} \tag{3.3}$$

  The CCC is mean invarient, and is based on the cross-covariance of the two images, divided by the product of their standard deviations. See sections $2.7$ and $8.1$ in [Cha04].

In [QXWP97] Wu *et al.* investigated the efficiency of the various techniques, concluding that the CCC provides results *at least* as good as the SAVD whilst generally providing better quality results than both the SSD and SAVD, thereby reducing the number of iterations needed in the next stages of the motion estimation. Most modern motion estimation systems therefore use the CCC as the metric of choice in motion estimation.

Another important consideration is whether or not it is actually worth considering every block. Generally, and especially with remote sensed images, there will be some blocks who's mean or variance is so low that the information content in that block is next to zero. In these cases, it makes sense to simply disregard such blocks, giving an overall decrease in processing time. This modification of block matching is not used in video systems, but is very useful when dealing with remote sensed images, and will be considered in slightly more detail later.

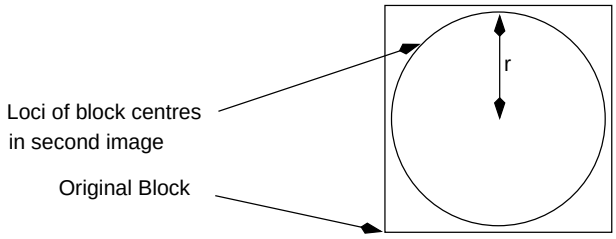


Figure 3.1: Showing how the two images are searched by translating the block in the second image. The circle shows an example loci of block centres. Every position that the block is shifted to in the second image leads to an output 'candidate vector'.

After the comparisons have been carried out each block will have a set of vectors and metric values associated with it. These must be evaluated for 'goodness', and the best matches found and incorporated into the output motion field. Various methods for determining the best vectors exist, and could include:

- Picking the vector attached to the highest valued comparison metric.
- Ranking the vectors according to the goodness of their individual comparison metrics and taking the median, or some other quartile value.
- Picking the vectors attached to the top $n$ comparison metric values, and sending them for further processng. E.g. relaxation labelling.

Sending the output for further processing is particularly useful when the data being examined are not *simple*. Images which aren't considered simple could have all sorts of problems which motion estimation can't cope with on it's own - in particular, objects may not be rigid, and could rotate, or occulde one another. Also, block matching is not very good at dealing with low contrast data.

The following section introduces an excellent method of 'further processing' - relxation labelling. It starts with it's origins in scene labelling, and then moves on to how it can be developed into a tool for use in motion estimation.

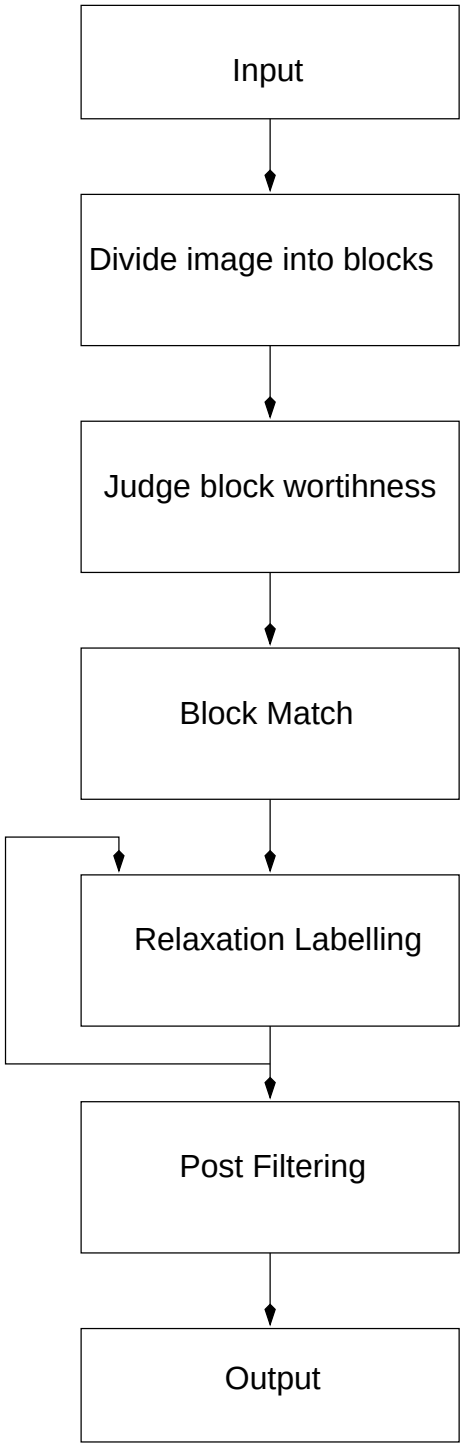


Figure 3.2: A motion estimation and relaxation labelling system.

## 3.2 Relaxation Labelling

### 3.2.1 Traditional Scene Labelling

Relaxation labelling is a technique which was designed for the labelling of objects in a scene. Imagine a digital photograph of a scene of, say a landscape (with hedgerows, trees, and a few fences, like figure 3.3). The aim of scene labelling on such a photograph might be to assign names to each of the objects in the photo, whilst not requiring any human intervention.

Figure 3.3: An English Countryside Scene (view South from near Newton St. Loe, Bath). Scene labelling could be used in this example to label the hedges, trees and buildings - and, of course, the sky.

'Normal' methods would start by assigning arbitrary labels to each object, and then running the resulting lables through a set of constraints, modifing labels so that they comply. For simple scenes this can be very effective. However, when a totally consistent labelling can't be found, the technique totally fails to give a result. It is also only really suited to segmentation applications, and it is not immediately obvious how one would go about adapting it to deal with numerical vector data.

### 3.2.2 Enter, Relaxation Labelling

Relaxation labelling is an iterative type of scene labelling which uses probabilistic methods and is much more robust than traditional labelling techniques. Whilst not guaranteeing a 100% consistent output, it does always give some result, and so for situations where there probably isn't a 'correct output', it is ideal.

Relaxation labelling based techniques work particularly well where other techniques fail, such as in images where objects are non-rigid (I.e. clouds), and may occlude or block one another.

Relaxation labelling works by assigning a set of labels to each object in a given set of objects, and adjusting the probability of each label being the correct one using a set of supporting equations[2]. Using the Fig 3.3 as an example, the operation of a relaxation labelling system can be described as follows:

[2]Known in this report as 'support functions'

1. Assume that the scene has been segmented (I.e. it is ready to be labelled).
2. Assign every label to every object in the scene.
3. Assign an initial probability to each label given to each object:
   - These probabilities could be be based on equation describing, say the shape of a tree, or hedge.
   - Normalise the probabilities given to each object's labels.
   - The output from this stage should be a set of probabilities describing the 'treeiness' and 'hedgeiness' (etc.) of each object in the scene.
4. Use the support functions to gauge the consistency of each label in each object with it's neighbours. Adjust the probabilities accordingly.
5. Loop to step 4, unless the condition to stop [3] has been reached.
6. The output should be a (hopefully) consistent labelling scheme, where the label with the highest probability belonging to each item is taken as the output label.

A typical (contrived) example might be a desk, where the objects could include a phone, a monitor and a mouse. A relaxation labelling scheme would give each object on the desk a probability that it was a mouse, a phone or a monitor based on it's characteristics, and then adjust those probabilities according to the neighbouring objects labels and their probabilities.

Moving from labelling scenes to labelling vectors may seem like a large step, but if you think of the each block of an image as a one of the objects above, and each candidate vector as being one of the labels for the block, the idea is instantly portable - with a few small changes.

Initially, the probabilities must be set using a block compatibility metric. Generally the metric used is the cross correlation coefficient (CCC). Initial probabilities are set using normalised CCC values.

Second, the equations which judge compatibility with the local objects need to be rewritten to take account of the fact that that the labels are now vectors, and so the comparisons must essentially be made between the actual labels themselves. This means that instead of a single label (identifying an object as being say, a desk or chair) the label will have two components (in $x$ and $y$). It is therefore useful to keep the candidate vectors in some kind of data structure, and to think of the vector as both a 'label' and a standard vector.

The relaxation labelling method being used throughout this report will be the same method as employed by Evans ( [Eva00]), which will be described in detail below. It has been shown that using relaxation labelling following motion estimation using a CCC metric (on remote sensed data) produces reasonable results where other techniques fail. This is beacuse using CCC produces good matches to start with, and relaxation labelling is highly robust to noise, and non-rigidity.

### 3.2.3 Relaxation Labelling In Detail

As explained above, relaxation labelling relies on a system of equations to get results, the order of events is shown schematically in fig 3.4.

#### Initial Probabilities

To start with, the proabability attached to each label (vector) in every block must be initialised. This is done by normalising the comparison metric value obtained during block matching. When using the CCC, the formula is the following:

$$P^{(0)}(J \to j) = \frac{\rho(J \to j)}{\sum_{\lambda \in \Omega_{2,J}} \rho(J \to \lambda)} \tag{3.4}$$

[3] The stop condition could be anything. E.g. a fixed number of iterations

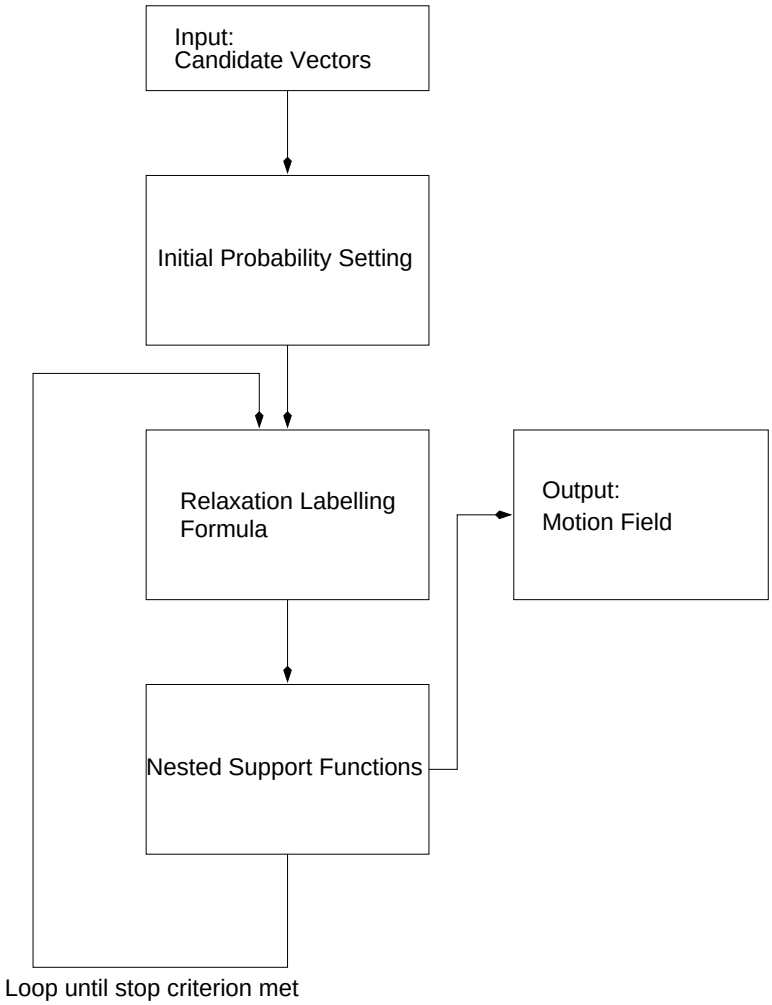


Figure 3.4: Relaxation labelling flow diagram.

Becuase of equation 3.4's reliance on the comparison metrics mentioned in section 3.1, it is very important that the metric used is as good as practically possible. Having a really good metric will reduce the number of relaxation iterations needed.

### Relaxation Labelling Formulae

After the initial probabilities have been set, the proababilities can be iteratively updated using the non-linear relaxation formula and it's various support functions.

The first equation updates the vector probability according to the current probability and a non-linear support function $\mathcal{Q}$ which judges neighbouring vector compatibilities. The probabilities are also normalised across each template, so the the sum of all probabilities in a given template is one, as equation 3.5 shows, this is fairly simple.

$$P^{(n+1)}(J \to j) = \frac{P^{(n)}(J \to j)Q(J \to j)}{\sum_{\lambda \in \Omega_{2J}} P^{(n)}(J \to \lambda)\mathcal{Q}(J \to \lambda)} \tag{3.5}$$

$\mathcal{Q}$ (equation 3.6), the first support function computes the product of sums of each candidate vector in the local neighbourhood [4].

$$\mathcal{Q}(J \to j) = \prod_{I \in G_j} \sum_{i \in \Omega_{2j}} P^{(n)}(I \to i)\mathcal{R}(I, J, i, j) \tag{3.6}$$

$\mathcal{R}$ (equation 3.7) judges the compatibility between the vectors $J \to j$ and $I \to i$. For this reason, it is known as the *compatibility function*.

$$\mathcal{R}(I, J, i, j) = \exp\left[\frac{|\Delta x_{I,i} - \Delta x_{J,j}|}{\sigma}\right] . \exp\left[\frac{|\Delta y_{I,i} - \Delta y_{J,j}|}{\sigma}\right] . \mathcal{D}(I, J) \tag{3.7}$$

The compatibility is evaluated using the differenced $x$ and $y$ displacements of the vectors being examined. $\sigma$ is called the global 'displacement variance', and controls the rate at which the output converges on a final value.

By now, it should not come as a surprise that there is yet another nested support function: $\mathcal{D}$. This function depends on the relative position of the current blocks, as well as a constant, $g$. Put simply, $\mathcal{D}$ decides if blocks $J$ and $I$ are neighbours, returning zero if not, or a measure of their proximity if they are.

The variable $D_{I,J}$, is the a measure of the distance between the blocks, relative to the block spacing, rather than the pixel spacing.[5]

$$\mathcal{D}(I, J) = max\left\{0, D_0 - D_{I,J}\right\} . g \tag{3.8}$$

The final descision to be made is when to stop iterating. There are several ways of deciding this, including:

- Reaching a certain number of iterations.
- Reaching a certain critical probability in each block.
- Setting a maximum bound on the time available for iterating.

Once the relaxation labelling is done, the output field can be made by simply taking the vector with the highest probability from each block.

The next section discusses post filtering, specifically using the vector median filter.

[4] The local neighbourhood is the set of blocks considered 'local' to the block being updated

[5] I.e adjacent blocks would be a distance of $1$ apart, not a $n$ pixels.

## 3.3 Post Filtering

After relaxation labelling, the output field may not be as smooth, or consistent as desired. In these cases, it is a good idea to use some kind of a post filter to help smooth the data, and add the desired consistency.

One particularly good post filter is the adaptive vector median filter.[6] This filter works using a fixed window size (of say, $3 \times 3$), and computing the median vector of the window members. It then calculates the *central difference* relative to the median value, and if they differ by more than the absolute vlaue of the median vector, multiplied by a factor, $k$ (known as the smoothness constaint) the centre vector is set to the median vector. The constraint can be summarised by the following equation:

$$|\Delta y_c - \Delta y_m| + |\Delta x_c - \Delta x_m| > k\,|\Delta y_m| + |\Delta x_m| \tag{3.9}$$

## 3.4 Extending Motion Estimation

This section discusses the second and third questions mentioned in the introduction: *When is multi channel data useful? How can this usefulness be gauged?* and *How can traditional ME systems be adapted to make use of multi channel images?*, after a brief discussion of *why* it might be useful.

In order to improve the quality of the motion field obtained from block matching and relaxation labelling, it has been suggested that where available, multipl channels could be used. This should have the effect of increasing the overall quality of the motion estimation output (i.e. the motion field), because more information will have been used to obtain it, and the availability of good quality candidate vectors will be higher.

In order to develop this idea further, a few concepts must first be developed. This first of which is what, exactly is meant by multiple channels.

### 3.4.1 Multi Channel Data

Typically, we think of multi channel data as being things like colour images, where there are three, very well defined data channels. In fact, lots of data are multi channel, and colour images have very little different information across the channels[7].

For the purposes of this report, the definition of a multi channel image will be:

> *Any data set which can be displayed spatially, in overlapping layers, with separate layers being comprised of data from a different frequency band, set of calculations, or observations.*

This definintion allows all sorts of (not necessarily obvious) data to be considered a multi channel image, including atmospheric data gathered using integration models[8], and satellite data showing cloud tops in infrared, water vapour, and normal visible wavelengths[9].

### 3.4.2 Judging Channel Information Content

Deciding whether to bother combining the channel data is one grey area. Intuitively, for the best possible results, the diversity between channels should be high enough that the same objects are being represtented, but low enough that the channels do not simply duplicate one another. If the channels are too similar, the candidate vectors will be 'wasted' by duplication, thereby contstraining the output field more that is desirable. Channel diversity can be judged by using the cross correlation coefficient, discussed in section 3.1. The use of such a metric to judge the diversity can be justified by the fact that the mean intensity level is ignored, and the fact the the cross corrrelation coefficient results in a similarity measure. The fact that it is already widely used for comparisons in signal processing add The fact that it is already widely used for comparisons in signal

[6] The vector median has been shown to be the optimal noise reduction filter for certain types of impulsive and non-impulsive noise.
[7] When described by the typical RGB model.
[8] Such as the data provided by the TSAR group at the University of Bath
[9] Such as the data available from the Dundee University Earth Station

processing adds further weight to this argument. This issue of exactly how diverse the channels need to be will need further investigation.

### 3.4.3 Combining Channel Data

Most of a ME system which uses multiple channel data will work in exactly the same way as a single channel system. Block matching will still need to be carried out on each channel individually, and relaxation labelling doesn't *care* about the origins of the vectors it operates on, as long as there are probabilities attached. It is only the steps between block matching and relaxation labelling that need to be modified, or indeed added.

The combination of the channel data is more an issue of implementation than anything else, possible methods could include:

- Choosing the top $n$ candidates from each channel, and using all of them. - An equal number of the candidates used come from each channel. This method could be called *non-competitive combination*.
- Putting all of the candidates together, and picking the top $n$, such that all of the candidates are in direct competition, and there is no guarantee that equal numbers of candidates from each channel will be used. This could be called *competitive combination*.
- Weighting the number of candidates somehow, such that a certain fraction comes from each channel. It is not immediatly obvious how such as scheme could work. Such a method could be called *weighted combination*.

Because of the way in which relaxation labelling works, any candidate vectors could be passed in, inside of any given template, and be operated on as if they started out there.

The following chapter deals with the implementation of a motion estimation with relaxation labelling system toolkit, after a brief introduction to the tools used to design such a system.

# Implementation

## 4.1 Introduction

From the previous chapters and sections, it can be seen that the following items need to be implemented in order to create a working relaxation labelling system, and answer the last two questions from the introduction.

1. Efficient code implementing block matching.
2. Efficient code implementing relaxation labelling.
3. Efficient code supporting the above two deliverables where necessary.
4. Modifications of the above code in order to analyse different techniques and ideas in relaxation labelling.

This chapter works through the implemetation that was carried out, describing the tools and languages, and functions were been written in order to fulfill the above criteria.

## 4.2 Tools

The section briefly describes the tools used throughout the project, including the programming language, editor and supporting software.

### 4.2.1 GNU Octave

GNU Octave is a free, open source Matlab clone, designed for doing numeric computations. Octave has several advantages over Matlab for this project, including:

- Being free, meaning no expensive license is needed.
- Being open source, meaning that if necessary, bug can be fixed, and Octave recompiled by end users.
- Having a published manual [Eat97], excellent support mailing lists, and a friendly and helpful online community.
- Not being encumbered by a large GUI, whilst some people like a large programming environment, others prefer a text editor and terminal; Octave caters for the latter.

### 4.2.2 Revision Control

In [HT99], Hunt and Thomas devote a section to discussing the use of revision control systems, coming to the conclusion that you should 'Always Use Source Code Control'. This project will use the subversion[1] revision control system, so that bugs can be traced, and changes tracked. One of the main advantages of version control is the fact that they allow 'rolling back' a file to a previous version - perfect if something breaks.

Subversion is an open source revision control system which was written to address some of the problems and shortcomings of CVS (concurrent version system). In particular, subversion adds the ability to move, delete and rename files. It also tracks the version number of the whole directory together, so there is no need to worry about specific versions of each file. All commits are also totally atomic in Subversion, meaning that problems with the repository are easier to fix, and tend not to break things.

[1] Subversion homepage: `http://subversion.tigris.org/`

### 4.2.3 Gvim

`Gvim` is an excellent programmer's editor. It builds on the unix tradition of *vi*, adding syntax highighting, automatic indentation, online help and a plugin architecture. For this project, the existing Matlab syntax highlighting has been extended to work with Octave's extra keywords `endfor`, `endfunction` etc. Also, the `taglist` plugin has been extended so that it can use 'm' file tags (see below).

### 4.2.4 Ctags

`Ctags` is a program which indexes function names, along with the file that the function resides in, the idea being that the function name can be *followed*, from within a text editor to allow quick editing. The index contains entries called 'tags'. For this project, `ctags` has been extended to read 'm' files, so that the gvim editor can use the tags generated.

## 4.3 Code Style

Like most programming languages, Octave code can be written in many different styles. This section briefly describes the style used throughout this project, such that a third party would be able to read and understand the code if necessary. It should be noted, however that there is no right or wrong code style, so if needs be, extensions could be written however the programmer wants. Code has generally been written such that it should be compatible with both Octave and Matlab, although it has not been tested in the latter.

#### Function & Variable Naming Conventions

Although variable names like $i$ and $j$ have been used for counters, and indexing, most variables have been given longer, more descriptive names, such as `candidateMatches`, where spaces have been removed, and words after the first have been capitalised to make the names easier to read. In general, the first part of the variable name describes the specifics of the variable, and the second part describes what type of data the variable holds. `combinedCandidateMatches` for example refers to a structure array of candidate matches created by combining two or more sets of matches. Function naming follows the same conventions.

#### Structures

At the top level, structures are simply variables, and so follow the same conventions as described above.

#### Function Definitions

All functions used at the top level, i.e. for use by an end user, take arguments of the same format (shown in figure 4.3).

```
functionName( candidateMatches, blocks, [info], [other arguments])
```

Figure 4.1: Function definition format.

Where items in square brackets are dependent on the requirements of the function. Help is available for all functions that have been written by issuing the help command, as is the case with most Octave and Matlab functions.

```
help functionName
```

Figure 4.2: Getting help in Octave

## 4.4 Data Structures

Data structures are a very important part of any complex program, or system because they represent the **relationships** between data. Using good data structures can have a huge impact on the readability and use of a program.

Structures in Octave are similar to those in C, except that they are somewhat less strict. They do not need to be declared, or inititalised, and the same field in two different elements of an array of structs need not share the same type.

```
a(1).s = 1
a(2).s = 'string'
```

Figure 4.3: Array field types are in no way constrained.

There are three main data structures used in the motion estimation and relaxation labelling code, these are the `info` struct, which hold general, global inforation. The `block` struct, which hold information on the blocks used for block matching, and then relaxation labelling, and finally, the `candidate match` structure, which hold information on the candidate vectors created by block matching, as well as the probability information used by relaxation labelling.

#### Struct Access

Unfortunately, there are some quirks to accessing struct information in both Matlab and Octave[2]. While the accessing only one element of a nested struct works fine in exactly the way one would expect:

```
candMatches(1).cands(1).p
```

Figure 4.4: Accessing an individual field in a doubly nested struct.

Occasionally it is necessary to extra information from more than one of the nested structs. In this case, the expected access method doesn't work as expected, and a comma-separated list is returned. In order to get around this, the following syntax should be used:

```
matches = candMatches(1:10);
% Access members as structs, instead of cs-lists.
u = {matches.cands}{1}.u;
```

Figure 4.5: Accessing more than one field in a doubly nested struct.

The above example shows how to access the `u` field of several members of candidate match struct, and works by casting the output cs-list into a *cell* array.

#### General Information Structure

- Size: Single element.
- Members:
  - `blockSize`: The size of the blocks the image has been split into, in pixels.
  - `searchSize`: The radius of the search used in block matching.
  - `frameRows`: The number of rows in the current frame.
  - `frameCols`: The number of cols in the current frame.

[2]Since Octave attempts (for compatibility) to emulate Matlab's behaviour.

  - `blocksPerRows`
  - `blocksPerCol`: The number of blocks per rows or columns for the current frame.
  - `totalBlocks`: The total number of blocks in the current frame.
  - `g`: Relaxation labelling constant.
  - `sigma`: Relaxation labelling constant.
  - `d0`: Relaxation labelling constant. Size of local neighbourhood.

**Block Structure**

- Size: variable - info.totalBlocks.
- Members:
  - `x1`
  - `y1`
  - `x2`
  - `y2`: Block corner coordinates.
  - `var`: The variance of a given block.
  - `mean`: The mean of a given block.
  - `ok`: Whether a given block will be used in precessing.
  - `cands`: A structure containing block candidate matches.

**Candidate Match Structure**

- Size: variable - depends on number on candidates and therefore search size and block position.
- Members:
  - `u`
  - `v`: Candidate vector coordinates.
  - `c`: Cross correlation coefficient for candidate vector.
  - `p`: Probability for candidate vector. Sum of probs for a give blocks candidate matches should always be either 1, or 0 (if block is ignore).
  - `localDists`: Information used in relaxation labelling support functions.

## 4.5 Block Matching

Before block matching occurs, the blocks must be created. That is, the co-ordinates of each block must be set. This is done by the function `getBlocks`, which examines a frame, and produces a struct containing the right number of blocks, with the necessary information attached to those blocks.

After the blocks have been created, the function `checkVariance` should be run. This function also examines the content of each block, checking that the variance is above a certain threshold, and setting the block struct's `ok` field if it is. By setting this field, only blocks which are worthy of being examined are used in the block matching process. `checkVariance` can also test that the block's mean is above a given threshold value, although this is somewhat less useful.

The function `activeBlockPlot` generates a checkerboard plot showing which blocks are active, and which are inactive. It takes a block struct, an info struct and an output filename as argument, and works by saving

the `block.ok` field, and then calling a *Ruby*[3] script which loads the field from the file, and then creates a checkerboard pattern showing which of the blocks are active, and which aren't. Since there is no obvious way of making Gnuplot draw grids of filled squares, using Ruby seemed like a reasonable way of getting the desired output with minimal extra work.

The code for block matching is very simple. It loops over a block structure, comparing sections of two given frames using a two deminsional CCC. Various checks are carried out to ensure that it is worth taking the CCC, such as checking the mean, and checking that the block is actually inside the image. The output CCC, and the $x$ and $y$ components of the displacement vector are written into a `cands` struct within the block structure which was passed in - this new, modified block structure is then returned.

The CCC function is vectorised for speed, and simple test showed it to be as many as 15 times faster on test data when compared to the loop based version.

## 4.6 Relaxation Labelling

The code implementing relaxation labelling goes through each of the steps discussed in section in turn, with a few small changes to the funtionality in order to improve the overall speed.

First, the function `initialRelaxProb` initialises the probabilities, and writes them into the candidate match section of a block structure. This structure is updated throughout the relaxation labelling steps, and is not the same structure as the one describing the blocks positions etc.

Then, before the relaxation labelling takes place, the function `getNeighbourVectorDists` calculates the difference between each vector displacements and it's neighbours displacements, and computes the value of the support function $\mathcal{R}$ (equation 3.7). These values are then written into the block structure, which does not change again for the duration of the relaxation labelling. This step is done before the relaxation labelling occurs, because it only needs to be done once, thereby decreasing the overall complexity of the computations.

Next, the structure returned by the initial probabilitiy setting function is passed to the function `relaxationLabelling`. This function is a loop, which iterates over the blocks it has been passed, calling the function `relaxationLabellingSingleBlock`. In this function, each candidate match in the given block is processed in turn, and then the probabilities are normalised, as in equation 3.5.

Figure 4.6 shows the function called within the main relaxation labelling function.

The algorithmic complexity of the above steps is very large, and computation time grows exponentially with the number or canidate vectors. For this reason, it is a good idea to keep the number of canidates as low as possible - from experiementation, 40 seems to be a good number, although iterations still take a *long* time, and it is a good idea to let the system run overnight.

## 4.7 Post filtering

The implementation of post filtering makes use of two main functions, `postFilter` and `vectorMedian`.

`postFilter` takes the standard argument list (i.e a candidate matches struct, a block struct, and an `info` struct), with the addition of a variable which sets the convergence rate $k$.

`postFilter` starts by extracting the $x$ and $y$ vector displacements from each *active* block in a three-by-three window, and then calculating their vector median. The vector median [4] function constructs a table of vector distances[5] sums the rows, finds the row with the minumum total and returns the vector assiciated with that row as the median. This method is both intuitively easy to follow, and computationally efficient.

[3] Ruby is an OO scripting language, which is very easy to both read and write, and has bindings for many handy libraries, such as `imblib2` which the script makes use of in oder to construct the board. [TH00] is an excellent introduction and reference for anyone interested in learning the Ruby language.

[4] As described by: [JA90]

[5] Where a vector distance can be defined by $|\overline{a} - \overline{b}|^n$, where $\overline{a}$ and $\overline{b}$ are vectors, and $n$ is a real integer. The Pythagorean distance is obtained when $n = 2$

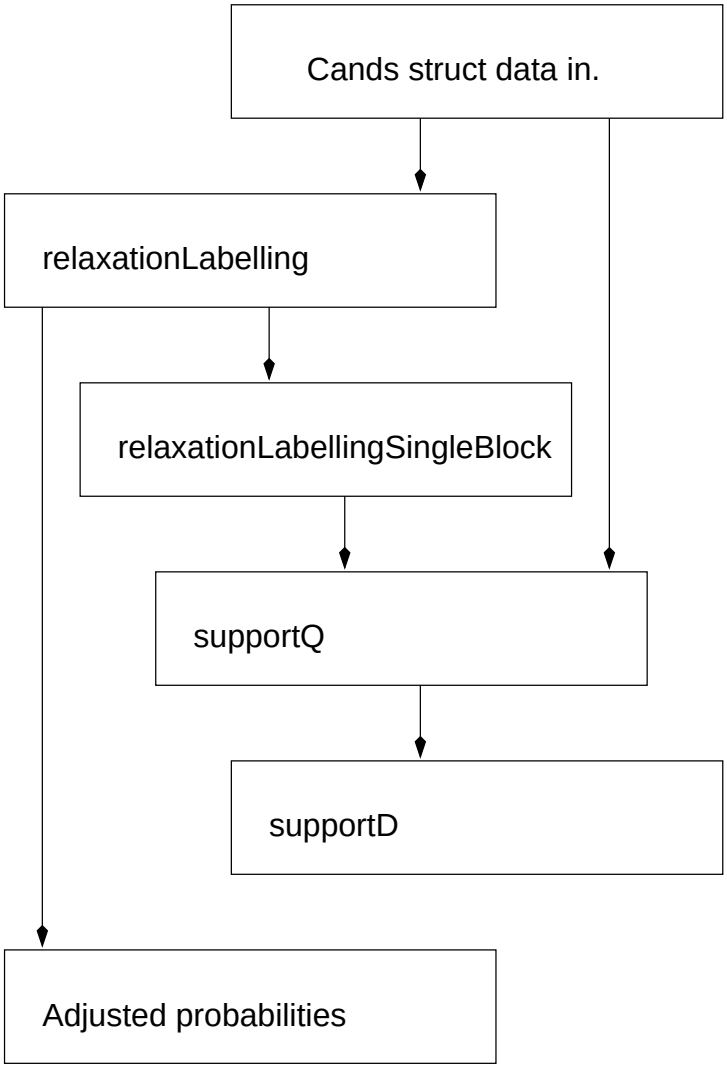


Figure 4.6: Relaxation Labelling Functions as implemented.

After calculating the median values, the central difference is taken (relative to the median), and if the values differ more the a certain thresholded amount, the median is taken as the value of the central block. The process then repeats for all other active blocks in the image.

## 4.8 Support Functions

Various functions were written to support other the other core functions in the suite. A few of the more important ones will be discussed here.subsectionGeneral Functions

`checkArgs`: Octave does not enforce function prototypes, and does not require the 'type' of arguments to be defined. Whilst this is quite useful in some situations, sometimes more rigidity is useful in debugging, and ensure the function runs as expected. For this reason, a function was writted which takes the number of expected arguments, and then a list of the argument names with strings describing what type they should be. Placing lines of the following form at the beginning of a function raises an error if anything is wrong with any of the arguments:

Where arguments could have more that one possible 'type', the string argument can take the form `'type1,type2'`, allowing greater flexibility.

`getMinPos` and `getMaxPos` expand on the `min` and `max` build in functions which only return one coordinate

```
% Check arguments.
if ~checkArgs(3, nargin, blocks, 'struct', info, 'struct',
    outputName, 'string')
    error('Argument Error');
end
```

Figure 4.7: Using the `checkArgs` function.

describing where the maximal or minimal element lies. These two function return both the row and column of the max. or min. element, and find use in the vector median filter among others.

`var2` and `mean2` are simply *syntactic sugar*, that is, they just wrap the standard 1D variance amd means functions, and perform them twice, giving the 2D variance and mean. This makes the code more concise and easy to follow. These functions are used in check for active blocks, and block matching.

`vectorDist` is a function uesd by the vector median filtering function which calculates the 'distances' between vectors.

### 4.8.1 Block Manipulation and Information Functions

`blockNoToxy` and `xyToBlockNo` convert from block numbers to block coordinates and visa versa. This is very useful, since the block structure is addressed by block number rather and by coordinates, and converting between the two is a common requirement.

`getLocalBlocks` return the blocks which are the neighbours of a given block, out to a given distance (usually defined by the `d0` field in the info struct).

`getRanges` converts the `x1,2` and `y1,2` fields from a block struct into *ranges*, which are then used in block matching.

### 4.8.2 Saving and Plotting Functions

As well as the functions for plotting active blocks (described in full in 4.5, there are several functions which assist in plotting and saving data. These are described below:

`saveData` does exactly that: it saves the data passed to it after transforming the data to make them easier to plot.

The first operation that occurs is thresholding, by choosing only the higest valued vectors. The value which is examined depends on whether the relaxation labelling probabilities exist. If they do, the match with the highest probability is used (and the function `getHighestProbs` is called), and if they don't, the vector with the highest CCC is used (the function called in this case is `getHighestProbs`).

The second operation converts the threshold candidates into a form ready for plotting using Gnuplot's *vectors* style, and involves squeezing the candidate vectors into a matrix, where the column used is determined by the channel from which the vector originates. The function which carries out this conversion is `candsToMatrix`. The vector displacements are also normalised.

Finally, the data are saved.

`plotData` takes a filenameand whether the data are single or multi-channel as arguments, and plots them. The plotting is not done directly, however, since there are no *vector* plotting functions in Octave. Instead, a shell script builds a gnuplot script which is then executed. This complicated sounding scheme allows the plot's title to be set, and the correct type to plot to be generated with minial effort, since the plotting scripts effectivly 'wirte themselves' - a technique called metaprogramming.

## 4.9 Using the Toolkit

Using the motion estimation and relaxation labelling toolkit is simple. The following figure shows how to load, and process two images using ME and RL. Obviously, extending the example to include more iterations would be simple, as would changing the position of the thresholding to change how the candidates are chosen.

```
% Load up various images.
irFrame1 = imread('frame1-frame1.jpg');
irFrame2 = imread('image1-frame2.jpg');
wvFrame1 = imread('image2-frame1.jpg');
wvFrame2 = imread('image2-frame2.jpg');

% Set the important variables in info.
info.blockSize = 17; info.searchSize = 25;
[info.frameRows, info.frameCols] = size(irFrame1);
info.blocksPerRow = ceil(info.frameRows/info.blockSize);
info.blocksPerCol = ceil(info.frameCols/info.blockSize);
info.totalBlocks = info.blocksPerRow*info.blocksPerCol;
info.g = 1.5; info.sigma = 2;
info.d0 = info.searchSize;

% Get the block ranges for the first channel.
irBlocks = getBlocks(irFrame1, info.blockSize);
% Mark blocks according to variance:
irBlocks = checkVariance(irBlocks, irFrame1, info, 1e4, 0);
% Run block matching on the first channel.
irCandidateMatches = blockMatch(irFrame1, irFrame2, irBlocks, info);

% Repeat for the second channel.
wvBlocks = getBlocks(wvFrame1, info.blockSize);
wvBlocks = checkVariance(wvBlocks, wvFrame1, info, 1e4, 0);
wvCandidateMatches = blockMatch(wvFrame1, wvFrame2, wvBlocks, info);

% Combined the channels.
combinedCandMatchesPreThreshold = catCands(irCandidateMatches, \
        wvCandidateMatches);
combinedBlocks = catBlocks(irBlocks, wvBlocks);
% Threshold the combined channel data.
combinedCandMatches = thresholdMatches(combinedCandMatchesPreThreshold, \
        combinedBlocks, 40, 0);
combinedCandMatches1 = getNeighbourVectorDists(combinedCandMatches, \
        combinedBlocks, info);
% Initialise the probabilities.
combinedCandMatches2 = initialRelaxProb(combinedCandMatches1, combinedBlocks);
% Run RL once.
combinedCandMatches3 = relaxationLabelling(combinedCandMatches2, \
        combinedBlocks, info);
```

Figure 4.8: Using the toolkit functions.

For information about the other available functions, see section 4.8.

# Results

## 5.1 Introduction

This gives a brief introduction to the data sources used throughout the project, before presenting results for experiments on active blocks, and single and multi-channel motion extimation and relaxation labelling. General observations may accompany the figures, by conclusions will not be drawn until the final chapter. All motion estimation was carried out using a search size of 25, which was determind by examining the input data. Also, all block sizes except those which are varied in section 5.3 are fixed at 17.

## 5.2 Data Sources

There are several sources of multi channel remote sensed data that could be used with the motion estimation and relaxation labelling software. The two main sources of data used in this project are images provided by the Dundee Satellite Receiving Station (DSRS), and data provided by the TSAR group at the University of Bath.

The DSRS provide a wide variety of images, gathered by the Meteosat stationary satellites[1]. Meteosat-8 - a body spun platform [TD01], which sits in the geostationary orbit at 3.4°W - is of particular interest because it is capable of gathering data in 12 spectral channels.

The TSAR data is derived from resampled Nimrod and UM weather data, and can provide a variety of different channels at various spatial and temporal resolutions. Generally, data takes the form of 'attenuation of $x$ due to $y$' - an example of data which can be shown as an image, although it isn't actually optical in nature. Whilst extrmemely useful, the fact that the results are 'generated' rather that actual measurements means that they are not necessarily as linked to one-another as one might hope. I.e rain attenuation and gaseous oxygen attenuation might be representative of different heights in the atmosphere, or represent data about the *entire* atmosphere and so may not be the ideal data to use. For this reason, although initial experiments were done using the TSAR data, most of the actual relaxation labelling results have been derived using DSRS data. Figures 5.1 and 5.2 show the two frames of each channel used to test the toolkit and to evaluate the effect of relaxation labelling.

## 5.3 Active Block Results

As mentioned in section 5.3, the issue of which blocks are active in an image is a fairly important and useful way of reducing the complexity of the system. The main problem however, is what variance should be used for the threshold function. This section shows how the number of active blocks varies with the variance threshold, for the IR and WV channels of the DSRS data. Figures 5.3 and 5.4 shows how the percentage of blocks which are active changes with block size and variance threshold. There is a clear decrease in the number of active blocks when the block size is smaller - this is almost certainly due the fact that a small block will contain fewer features (and therefore a lower range of values) than a larger block. Because of the reasonably high channel correlation, the two graphs are fairly similar.

The distribution of the active blocks is also fairly interesting, and, as mentioned previously, can be plotted fairly easily using the Ruby script for that purpose by using it's octave wrapper function. The following figures show how the active blocks are distributed using a block size of 9-by-9 pixels, for both the IR and WV channels. As the variance threshold increases, more and more blocks are removed, corresponding to areas with progressivly higher thresholds. Figures 5.5, and 5.6 demonstrate how the changes occur.

[1]There are four of them, see `http://www.eumetsat.int` for more information.

## 5.4 Single Channel Results

This section shows the results gathered after running ME and RL on single channel DSRS data. The data is described in section 5.2, and was downsampled by a factor of two, before block matching, and then 7 iterations of RL. s mentioned previouslty, in all results, the block size is 17, and the motion estimation search size is 25, which is more than enought to enclose any movement. See Figures 5.7 through to 5.12.

## 5.5 Multi Channel Results

### 5.5.1 Non-Competitive Candidate Match Combination

This section shows the results gathered using downsampled DSRS IR and WV data[2] , spaced 6 hours apart, and run through 7 iterations of relaxation labelling, before finally post filtering with a vector median filter. The candidate matches were combined using a non-competitive method, whereby the top 20 candidates from each channel were take, and combined by straight concatenation. The only competition that between channels occured in the relaxation labelling functions. The results can be seen plotted in figure 5.13 and 5.14, where the different colours represent data from separate channels.

### 5.5.2 Competitive Candidate Match Combination

The data here is exactly the same as used in section 5.5.1, except that instead of combining the candidate matches without any competition, all of the candidates were ranked, and only the top 40 of the entire set was taken. The meant that there was direct competition between channels at the very beginning. The results are plotted in figure 5.15, and 5.16, which show where the different output vectors come from.

[2]See figure 5.1 and 5.2.

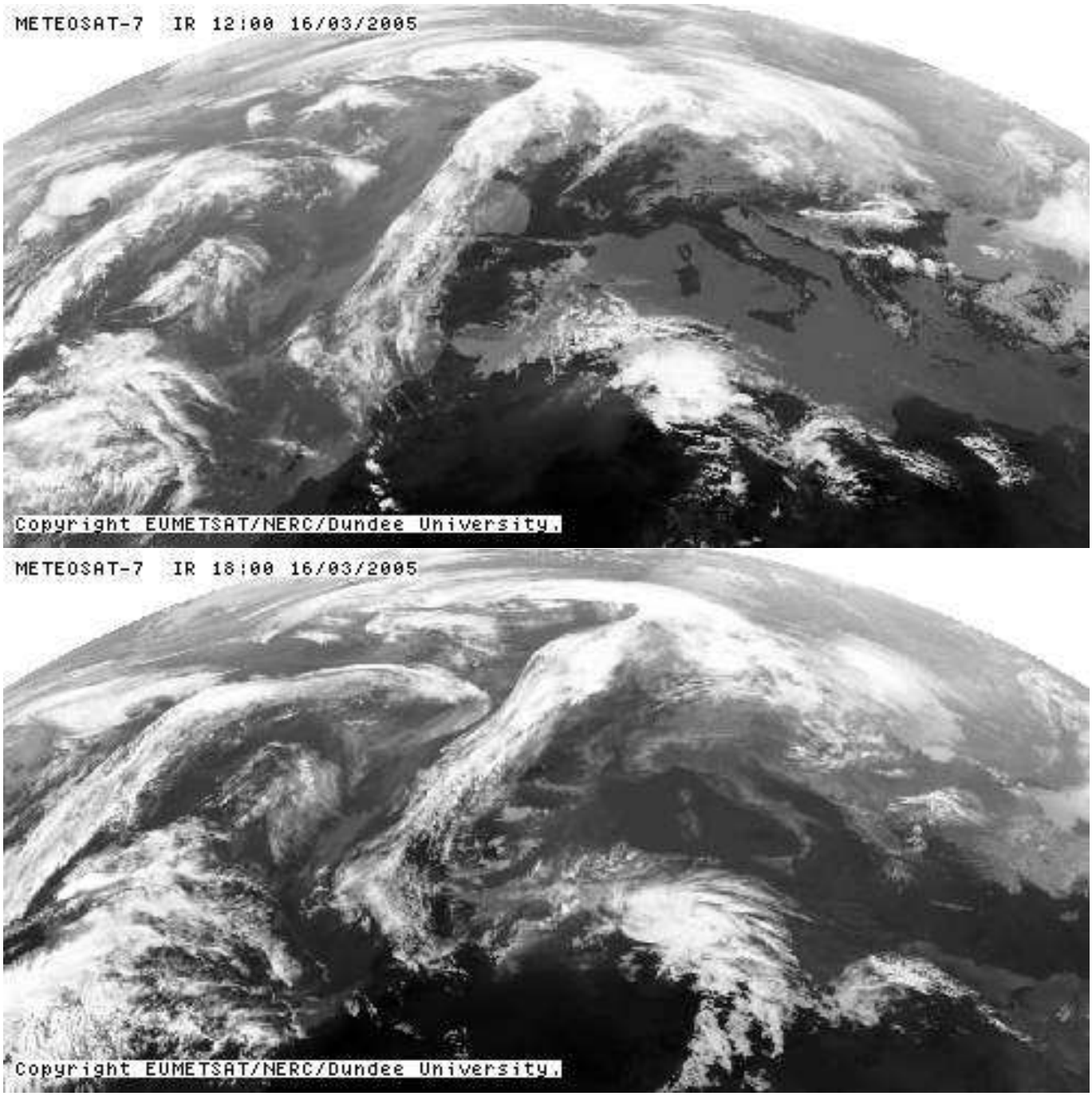


Figure 5.1: The IR Channel of data for March 3 2005 at 1200 and 1800.

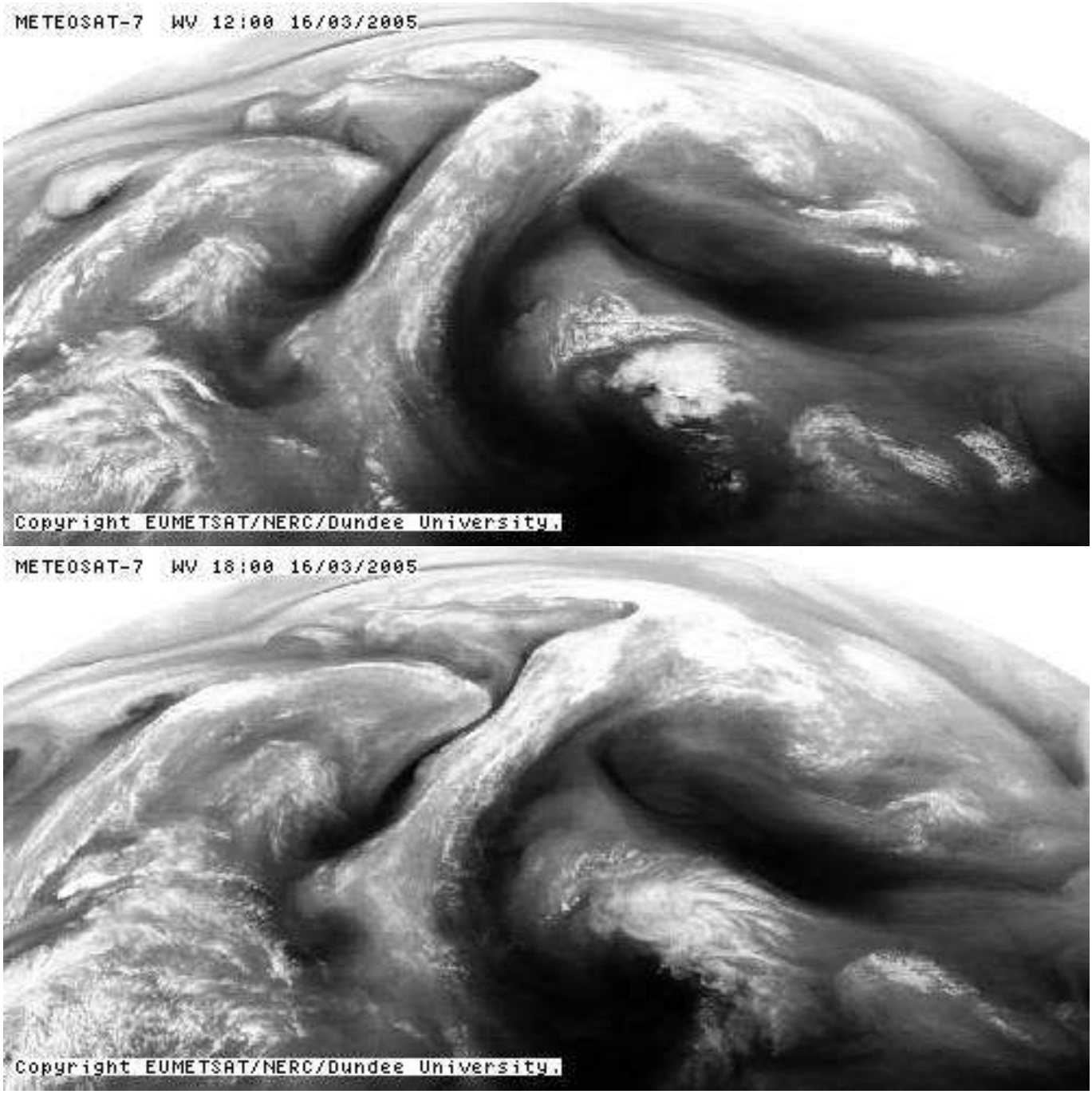


Figure 5.2: The WV Channel of data for March 3 2005 at 1200 and 1800.

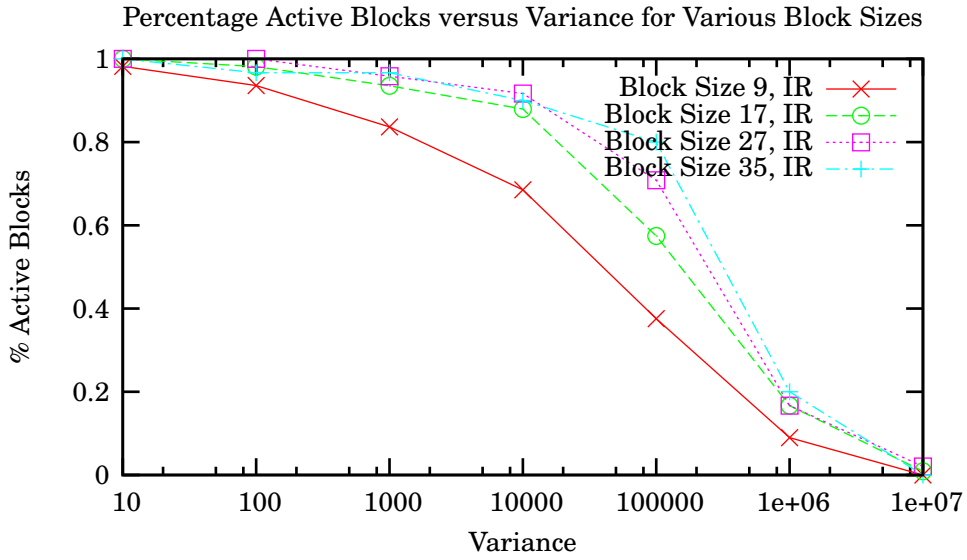


Figure 5.3: Active blocks in the IR channel for various variances.

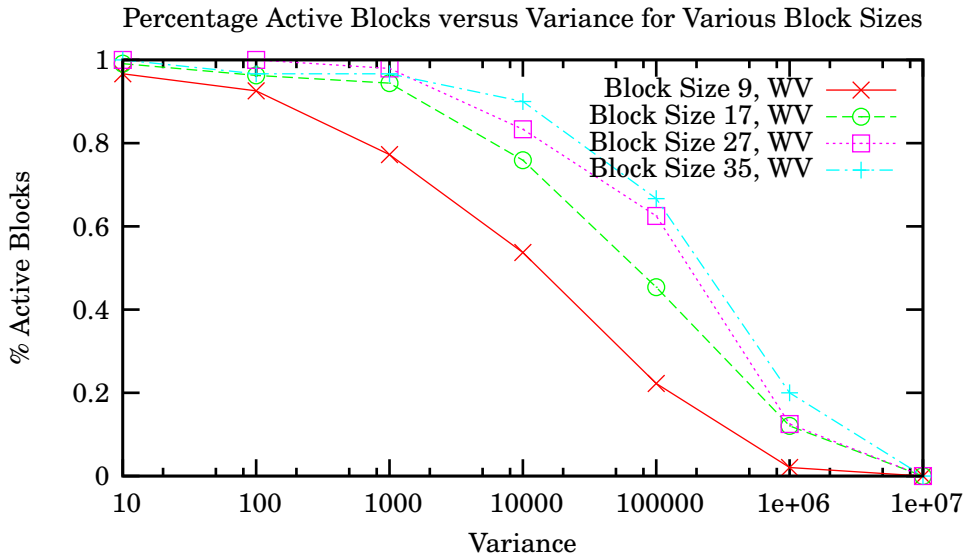


Figure 5.4: Active blocks in the WV channel for various variances.

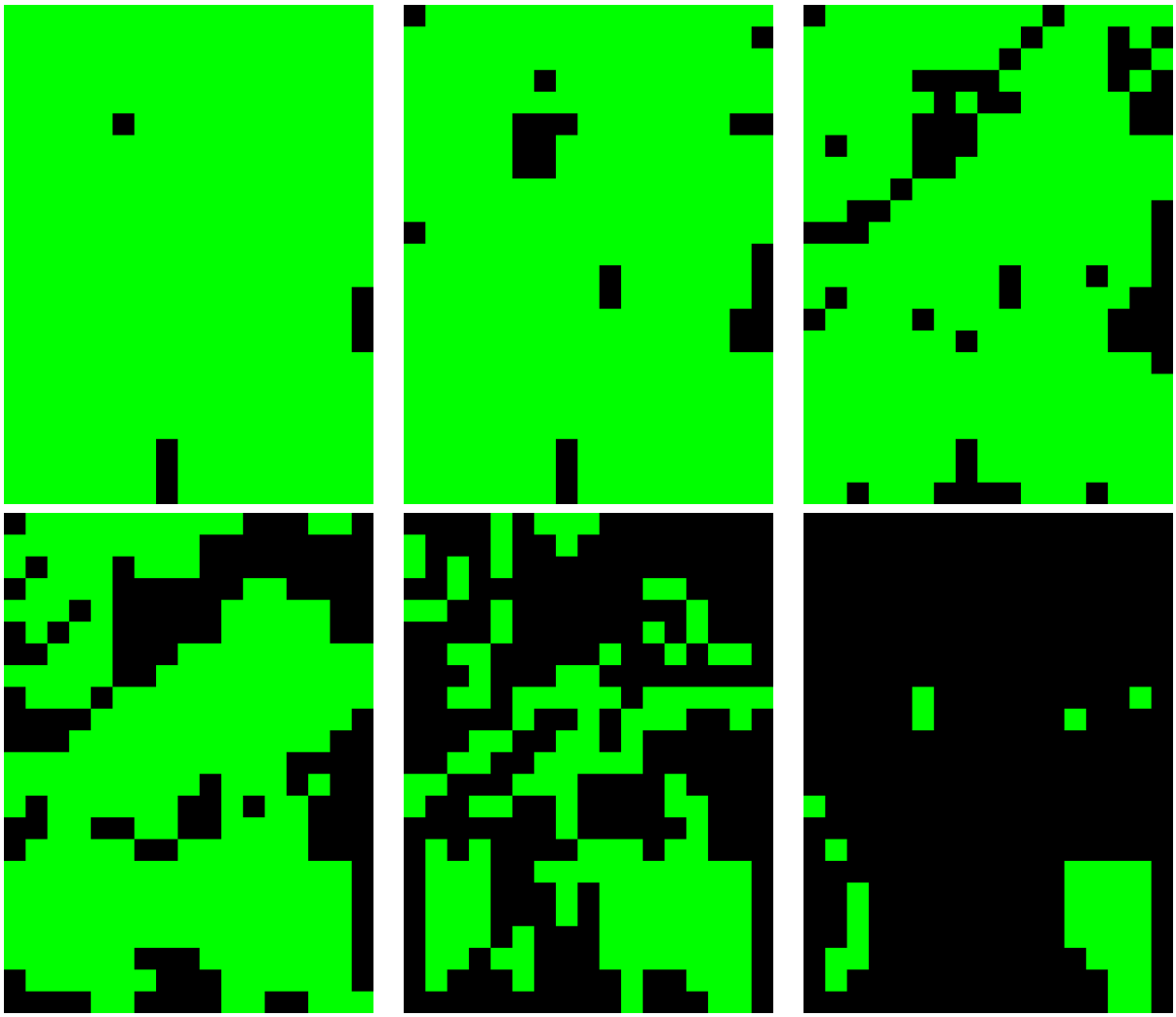

Figure 5.5: Active blocks in the WV channel for various variances. Starting at the top left, the variances are $10^1$, $10^2$, $10^3$, $10^4$, $10^5$, $10^6$.

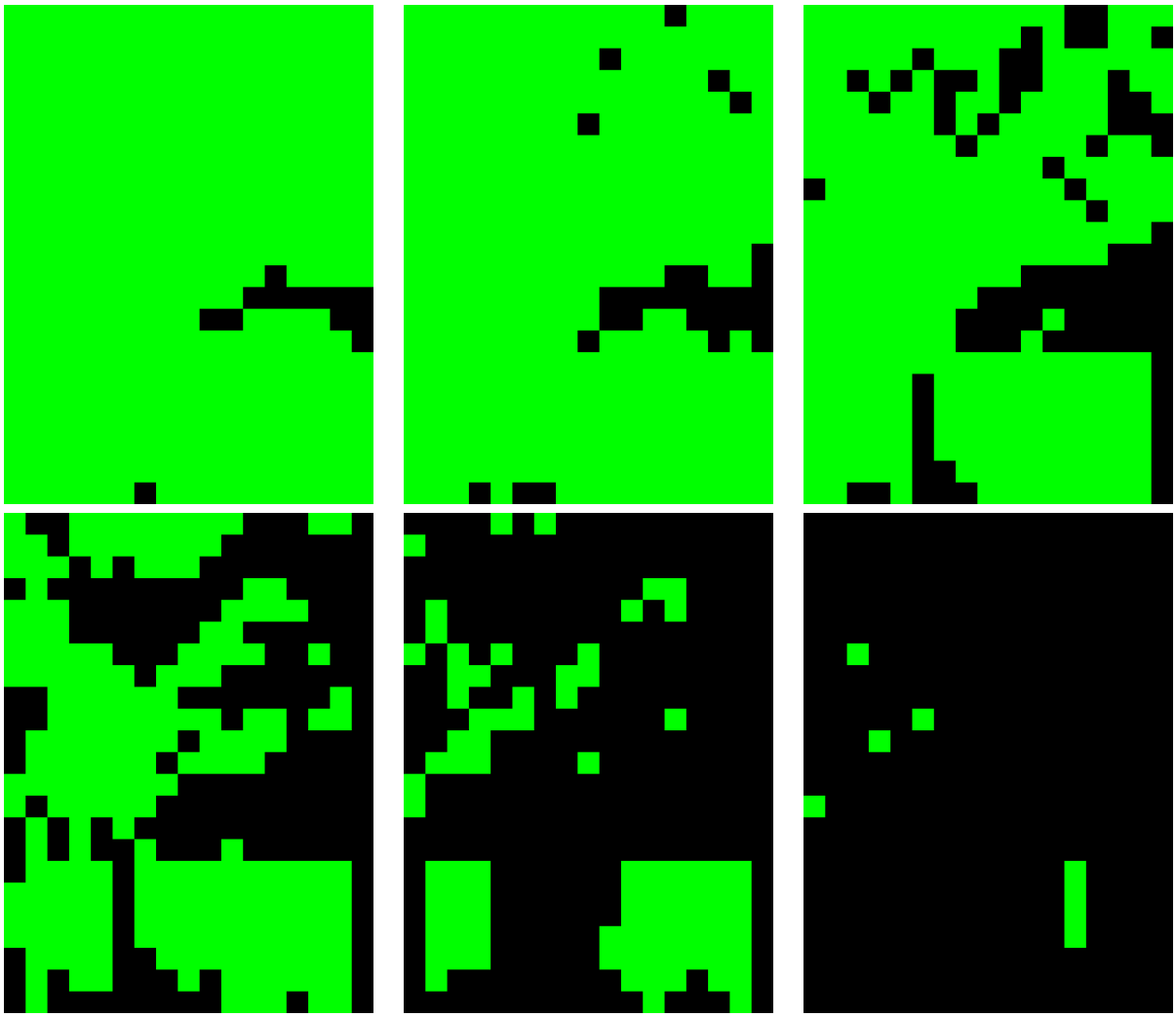

Figure 5.6: Active blocks in the WV channel for various variances. Starting at the top left, the variances are $10^1$, $10^2$, $10^3$, $10^4$, $10^5$, $10^6$.

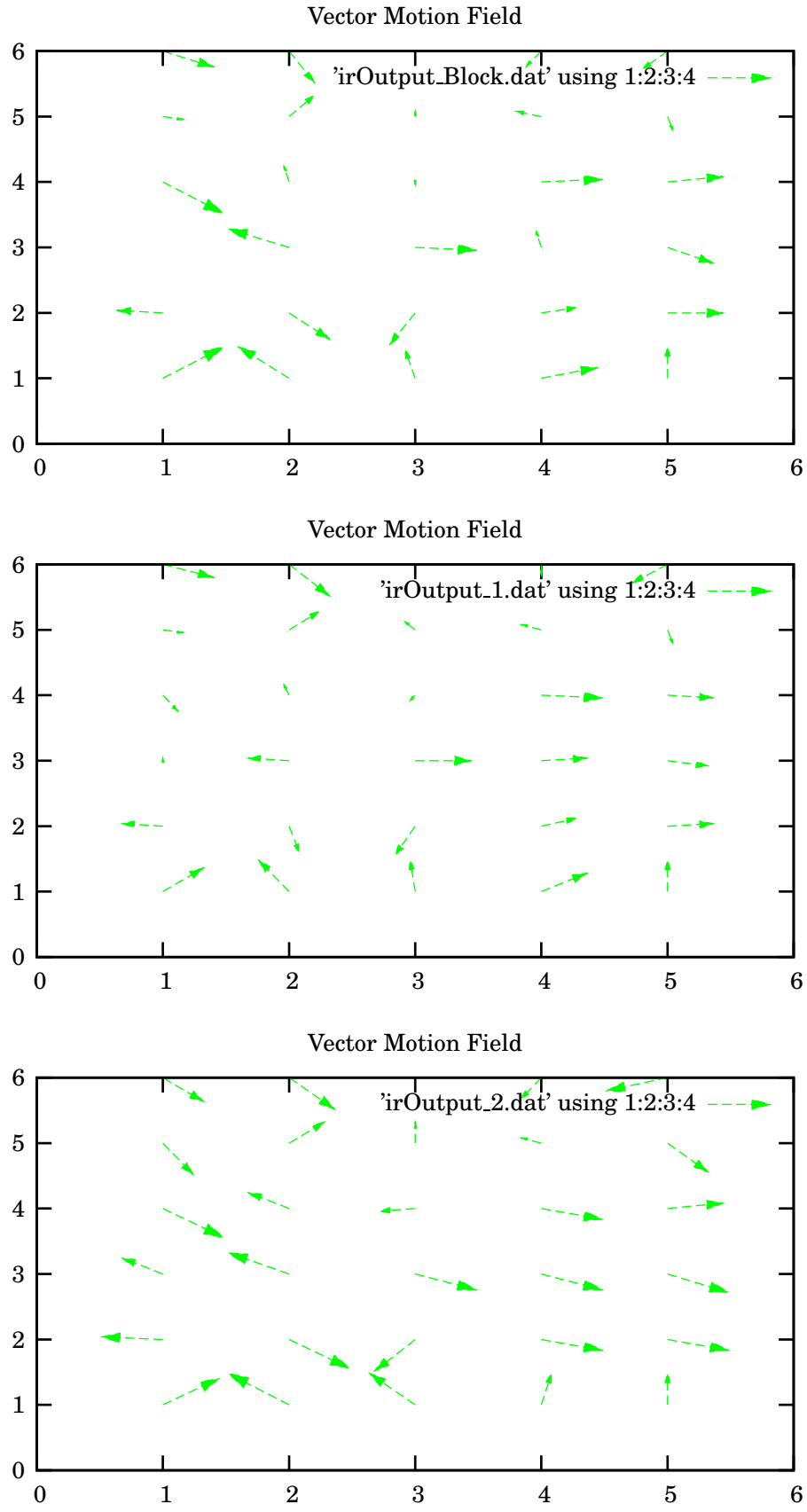


Figure 5.7: The result of motion estimation followed by four iterations of RL on a downsampled IR channel.

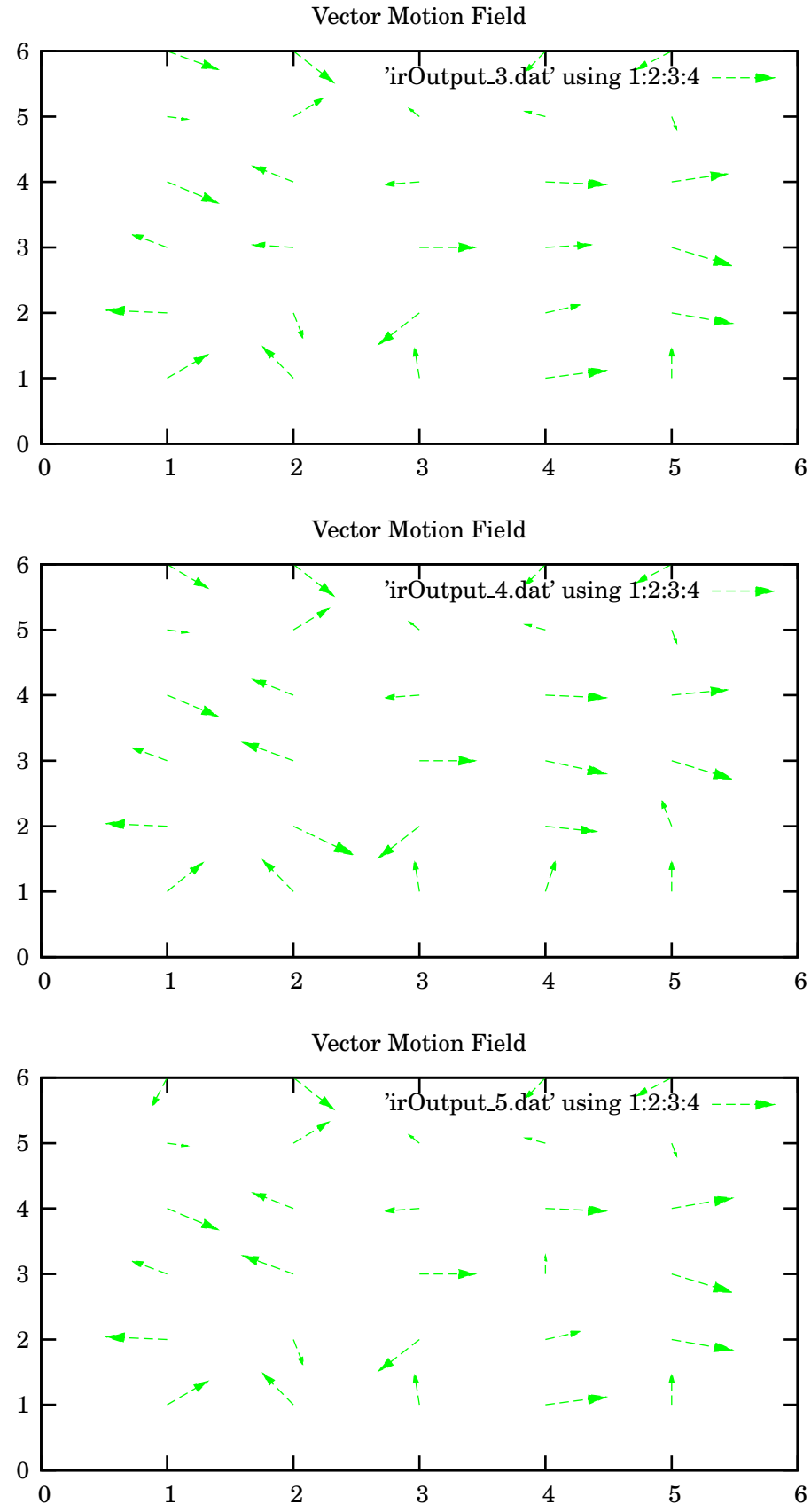


Figure 5.8: The 3-5 iterations of relaxation labelling on a single IR channel.

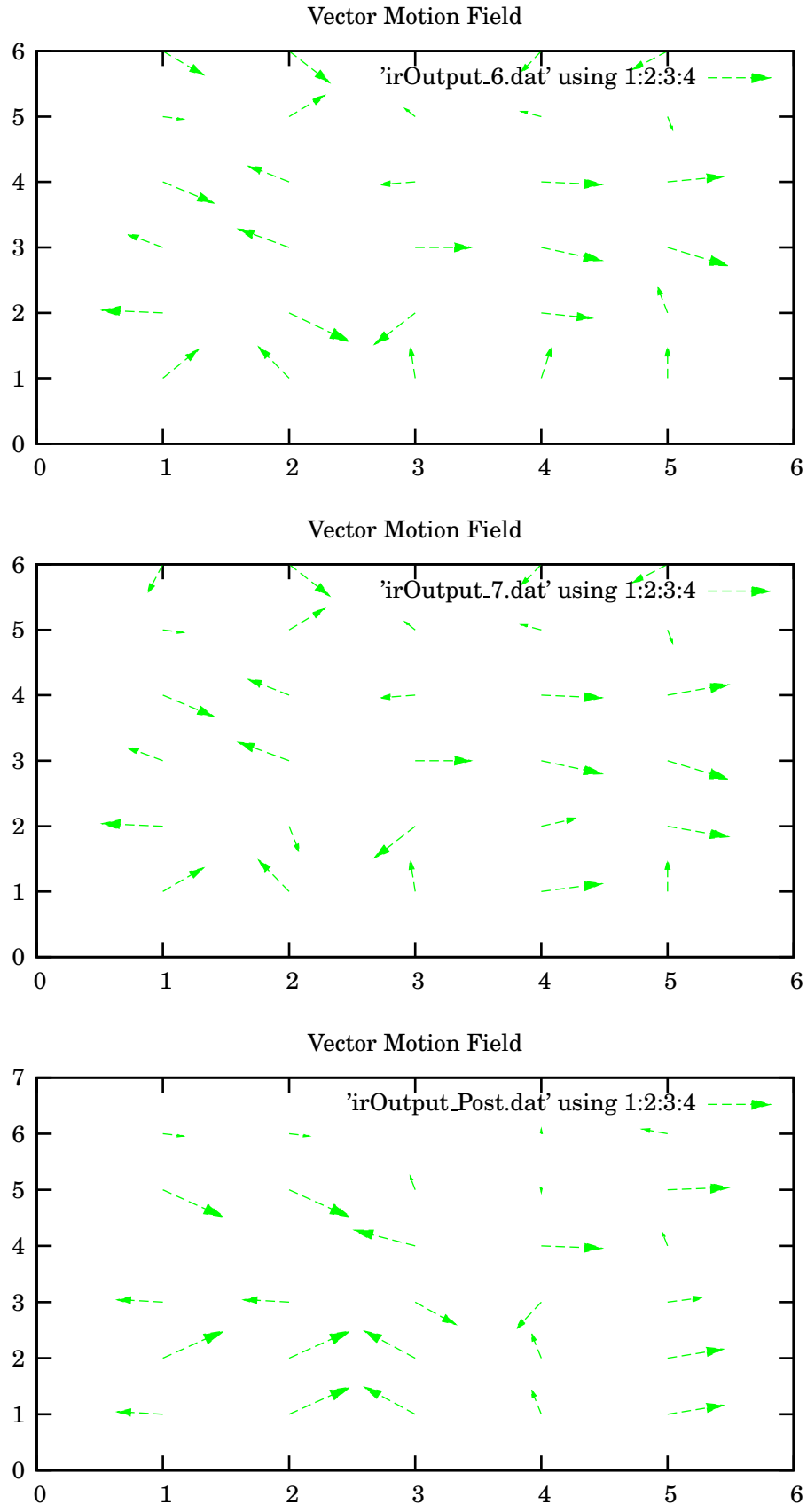


Figure 5.9: The 6th and 7th iterations of RL, followed by the post filtered output for downsamples IR data.

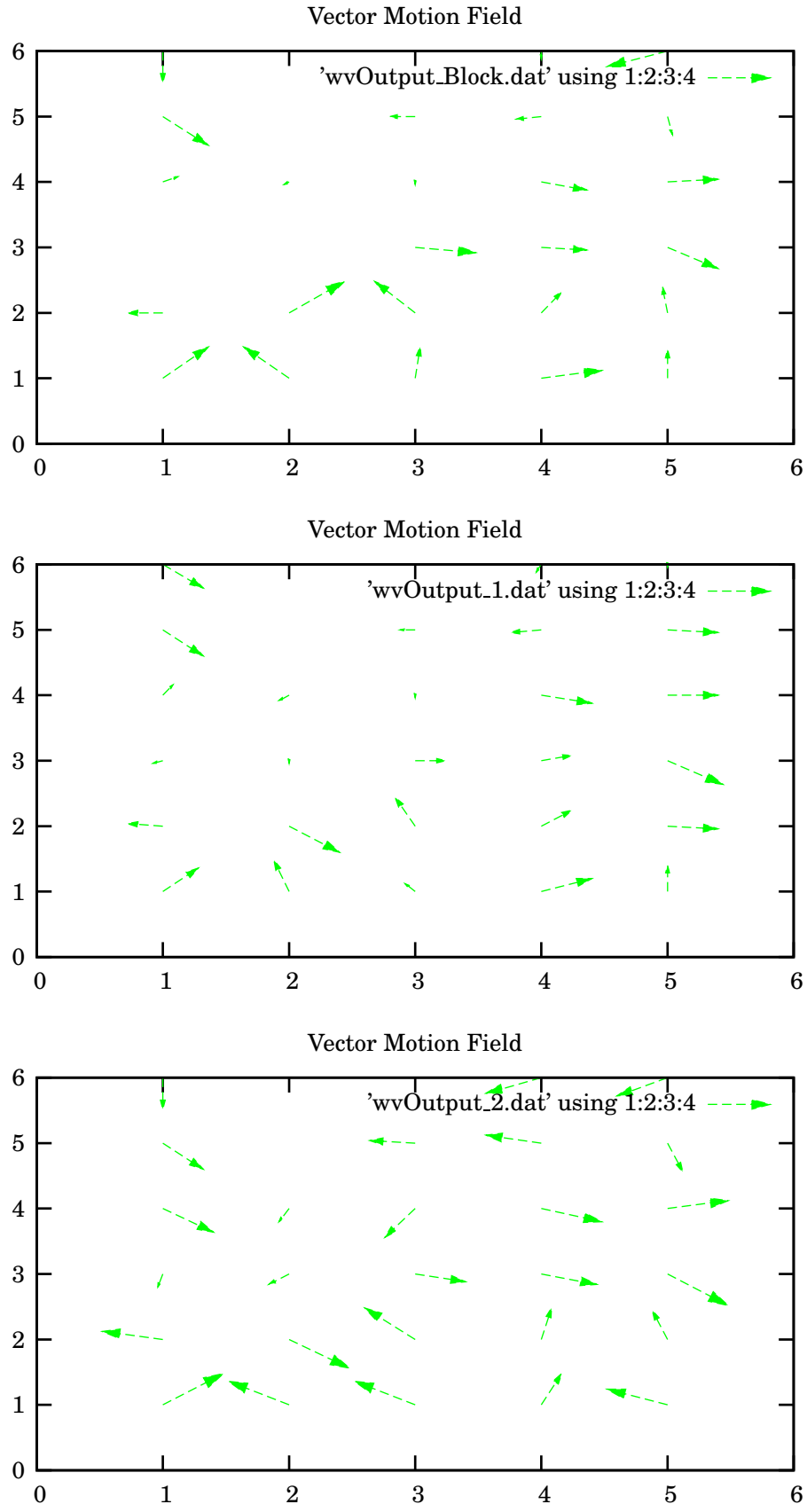


Figure 5.10: The result of motion estimation followed by four iterations of RL on a downsampled WV channel.

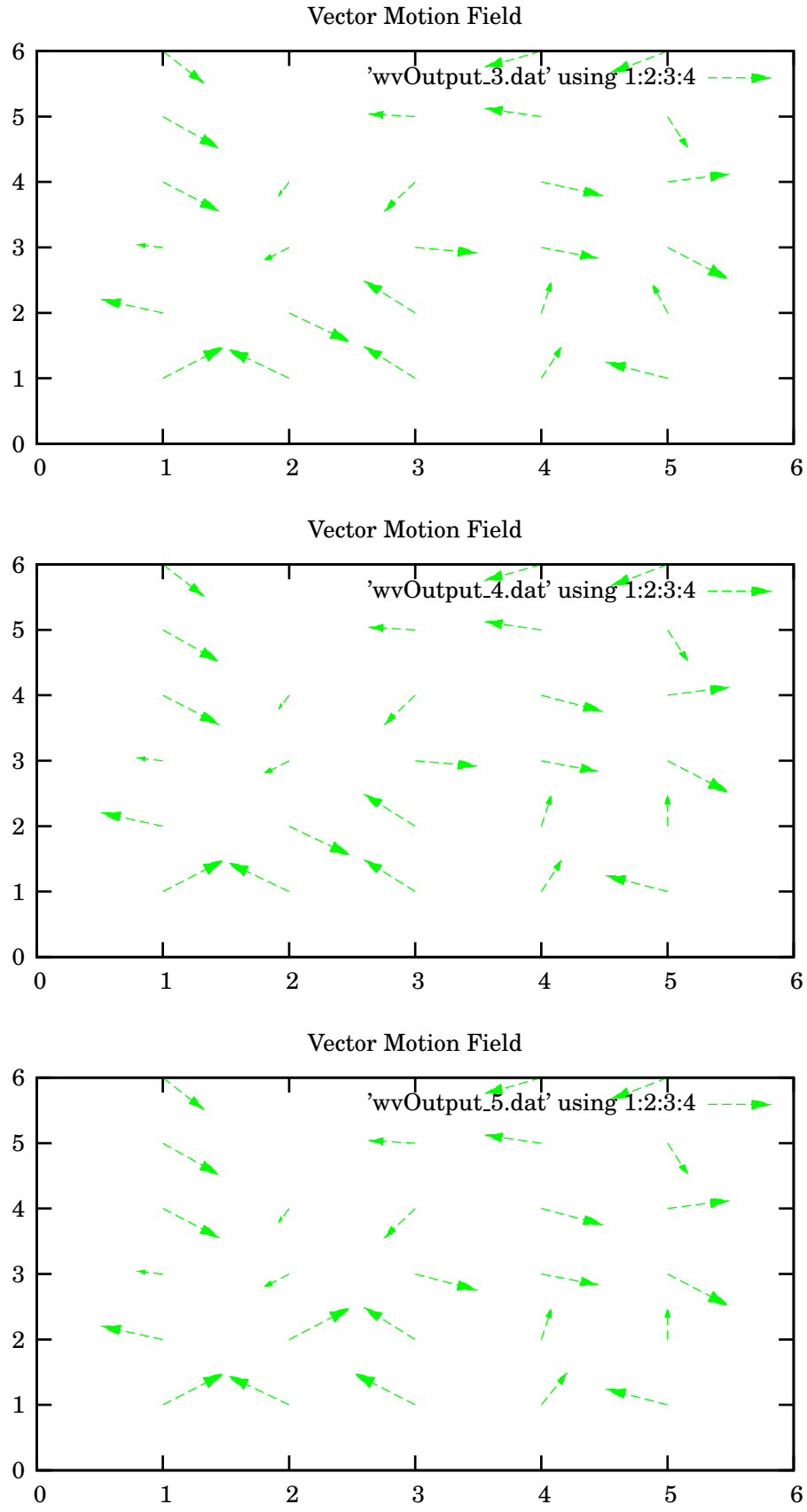


Figure 5.11: The 3-5 iterations of relaxation labelling on a single WV channel.

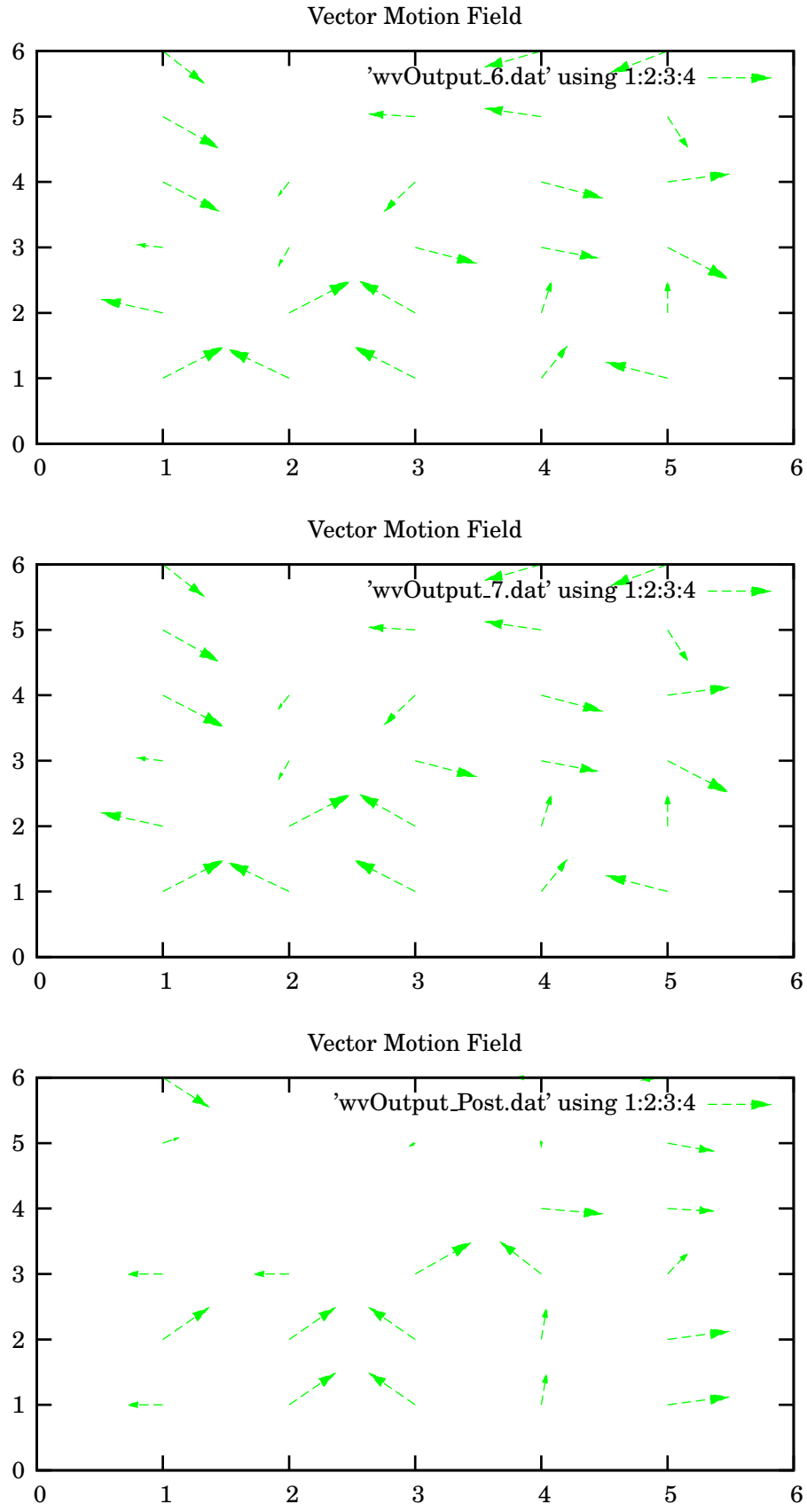


Figure 5.12: The 6th and 7th iterations of RL, followed by the post filtered output for downsamples WV data.

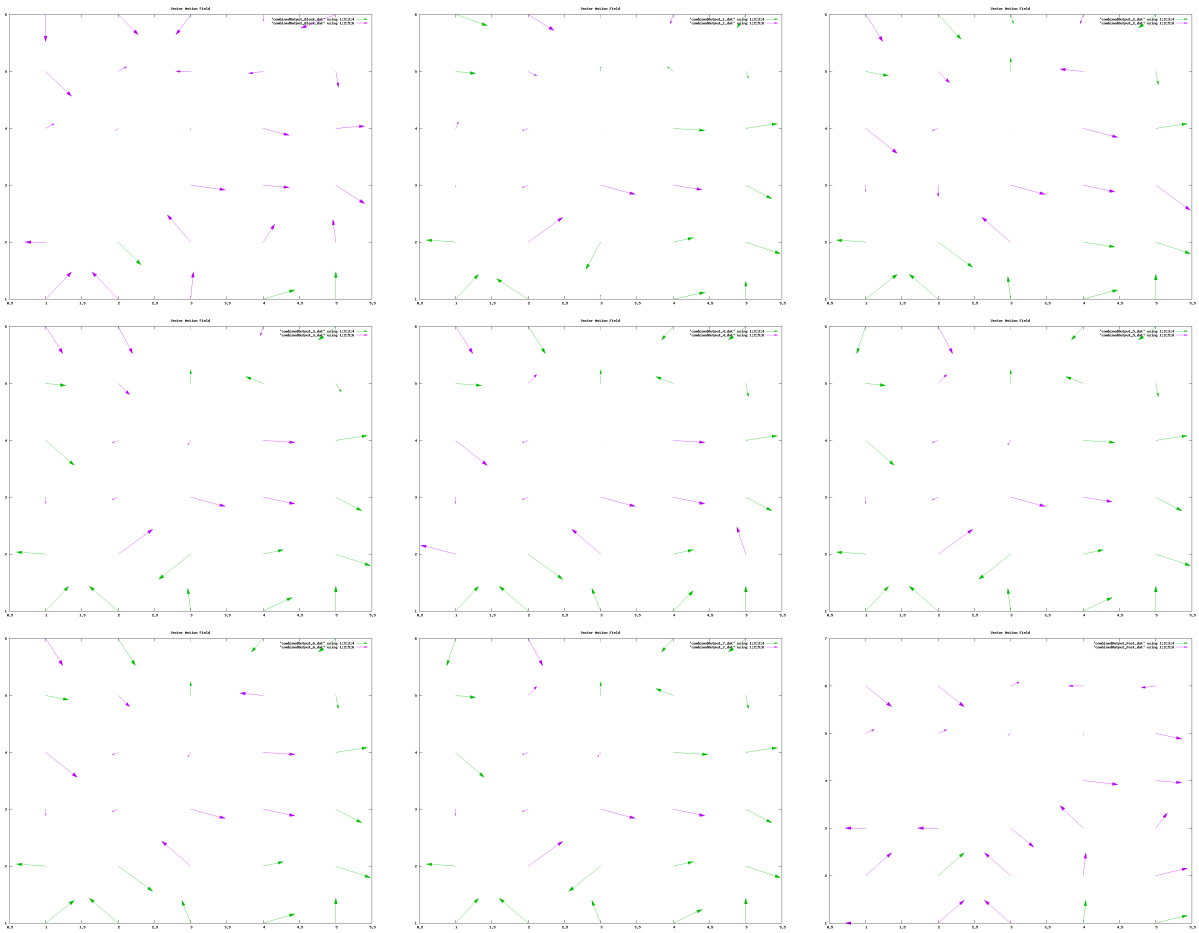

Figure 5.13: Dual Channel Motions Fields using non-competitive combination of candidaites. The images represent: The highest vector output after block matching, one iteration of relaxation labelling, two, three, four, five, six, seven iterations, and finally, the output after vector median post filtering.

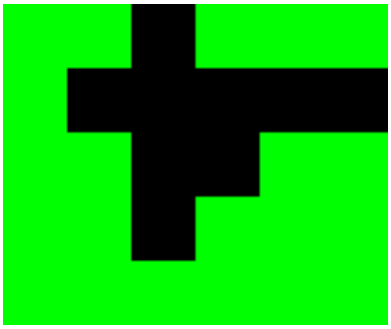

Figure 5.14: A diagram showing where the output vectors come from. The two different colours/shades represent the two channels, and the square blocks represent the blocks used in block matching to generate the candidate vectors.

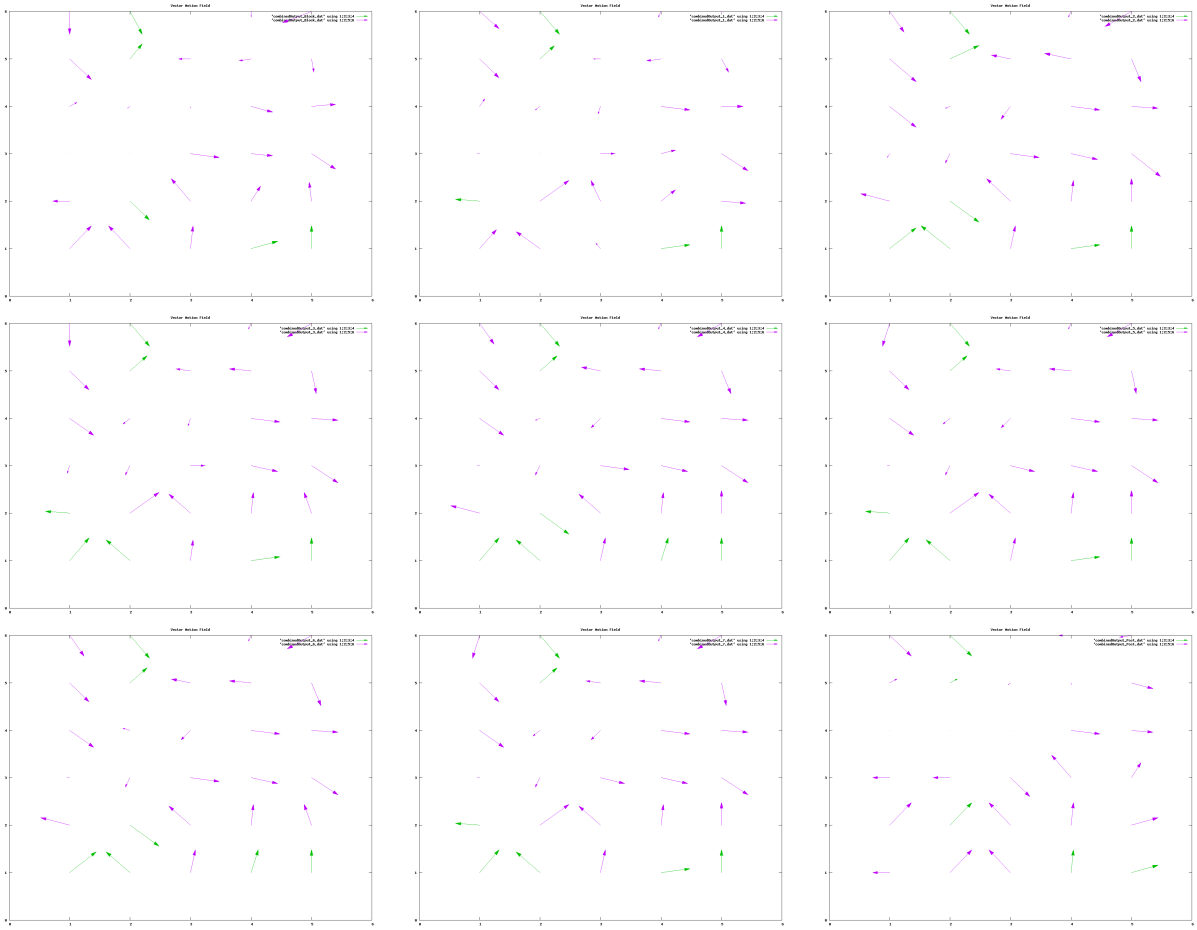

Figure 5.15: Dual Channel Motions Fields using competitive combination of candidaites. The images represent: The output after block matching, one iteration of relaxation labelling, two, three, four, five, six, seven iterations, and finally, the output after vector median post filtering.

Figure 5.16: A diagram showing where the output vectors come from. The two different colours/shades represents the two channels.

# Conclusions

## 6.1 Introduction

This aim of this chapter is to answer the final two questions from the introduction, namely:

- How well do the adapted techniques work on multi-channel remote sensed data?
- How can the above findings be used in practice, and how could they be used in the future?

First, the results gathered for single channel data will be analysed, and then the main issue of the multi-channel data will be tackled. Finally, some of the issued surrounding the last question will be examined.

## 6.2 Single Channel Data

As it stands, there isn't a lot that can be said about the single channel data. The post filtered outputs are clearly 'more consistent' than the input to the relaxation labelling stages, and *appear* to adhere more closely to the motion in the original frames. The RL iterations also appear to gradually improve the output quality, as expected, and the IR and WV output are also very similar in their general shape and flow[1]. The logical conclusions to draw from these findings is that the ME RL system actually works; which is *good*.

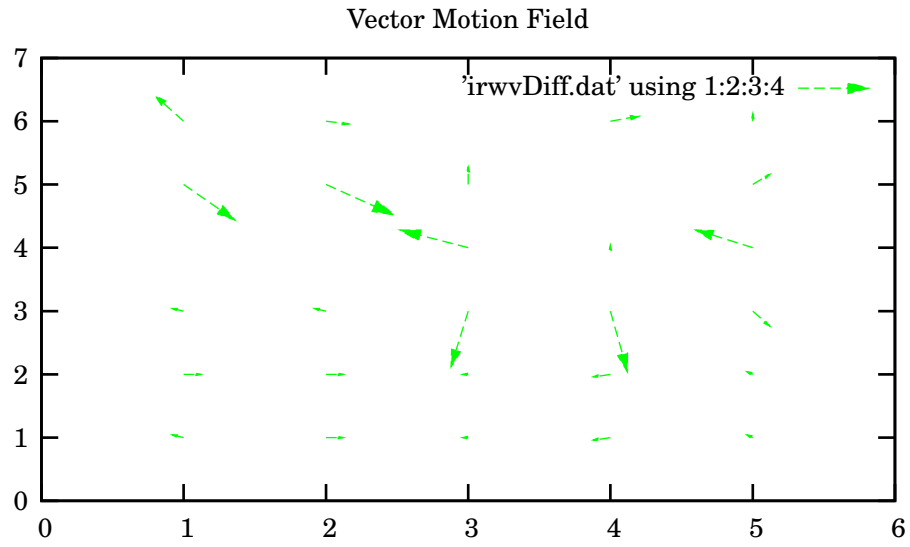


Figure 6.1: Vector differences between single IR channel and single WV channel outputs.

Figure 6.1 shows the differences between the motion fields from the IR and WV channels - the fact that they are different demonstrates the channel diversity.

[1]See figure 6.1

## 6.3 Multi-Channel Data

Unfortunately, it isn't possible to quantify the quality of any of the output data - there is simply no baseline (or correct) output field. Instead, observations must be done, and other metrics used to compare the data. Possible observations and metrics could include the vector differences between the output fields for different types of processing, examining the channels that the output data uses for each block, and judging the overall quality of the fields relative to one-another. Figures 6.2 through to 6.4 show each set of (normalised) results subtracted from each other in sequence. It is interesting to note the lack of differences between figures 6.3 and 6.4, only a very small number of the vectors show any major difference at all, indicating that despite the differences which are apparant in figure 6.2, overall, the two multi channel output fields are fairly similar. In fact, the main differences occur beacuse of the differences in which blocks are active.

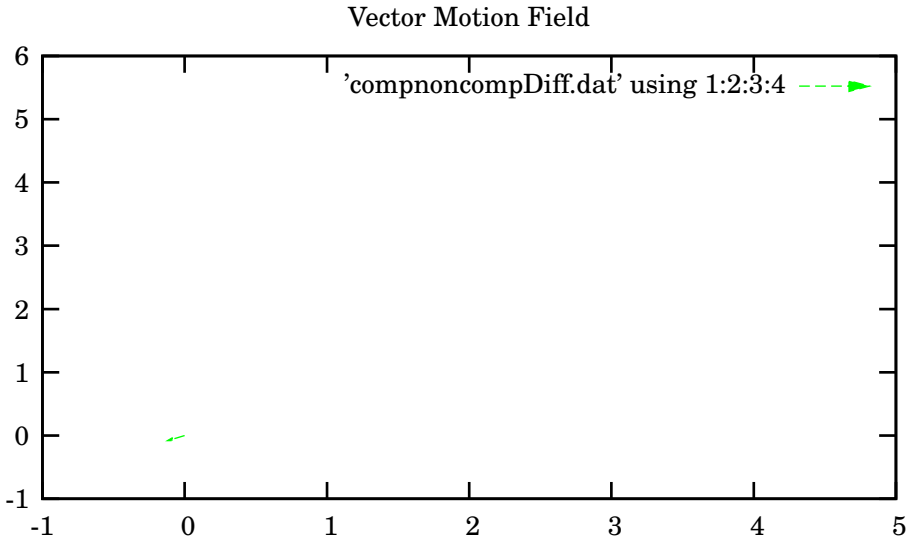


Figure 6.2: Vector differences between the competitive and non-competitive combination outputs.

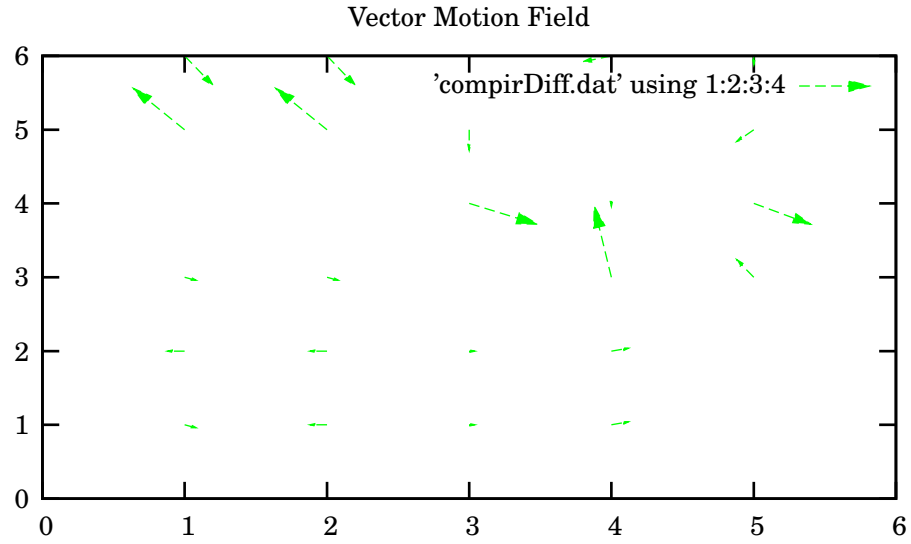


Figure 6.3: Vector differences between the dual channel competitive, and single IR channel outputs.

Some strong features exist throughtout all of the set of data, in particular, there is an interesting structure in the bottom centre of all of the final output fields, where the vector on the left and right face one another. This is

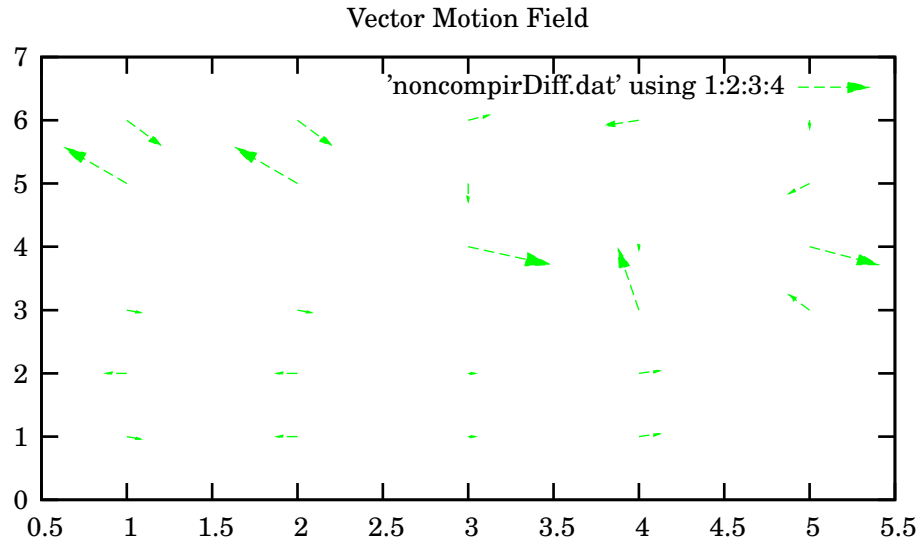


Figure 6.4: Vector differences between the dual channel non-competitive, and single IR channel outputs.

presumably present in all of the output motion fields because of the strength of the CCC values, and therefore the high probabilities assigned to those particular vectors.

Examination of the multi-channel motion fields also shows the origins of the different vectors in the outputs (different colours/shades indicate different channels). It is interesting to see how the origins change over the course of the RL iterations, with some vector's origins oscillating between two channels. Where output is strongly tied to one particular channel, the logical assumption is that the ME comparison metrics were higher for one particular channel in that area, indicating a very useful property of the multi-channel ME system:

*Areas who's outputs are weak in one channel may be strong in others.*

Interestingly, when the candidate combination is competitive, the number of vectors originating in each channel is far from equal, with only a few vectors coming from one of the channels, and the majority coming from the other. With the non-competitive combination, the distribution is more of less qual, with large blocks of vectors coming from each channel.

*Non-competitive Combination results in large areas where all of the vectors come from one channel.*

Also, both of the combination methods appear to result in vey similar, and fairly consistent vector motion fields.

To further reinforce the above conclusions, a far larger sample must be researched, however the results presented should give a general idea of the properties of multi-channel estimated fields.

## 6.4 Practical Uses: Present and Future

The most obvious practial use for such a system would be in meteorological prediction, where a RL generated field could be used to predict the movement of clouds, rain or other data that can be displayed as an image. The high computational cost of the scheme would also prove less of a problem, since it is not uncommon for weather prediction centres to have supercomputers.

Use in defence systems would also be a logical extension of the multi-channel ME-RL scheme, provided it could be made to run faster (see section 6.5). Possible uses might include RADAR systems, object tracking (or possibly even automated missile defense systems). A system could use visible images, SAR data and even IR data to improve locking on incoming targets.

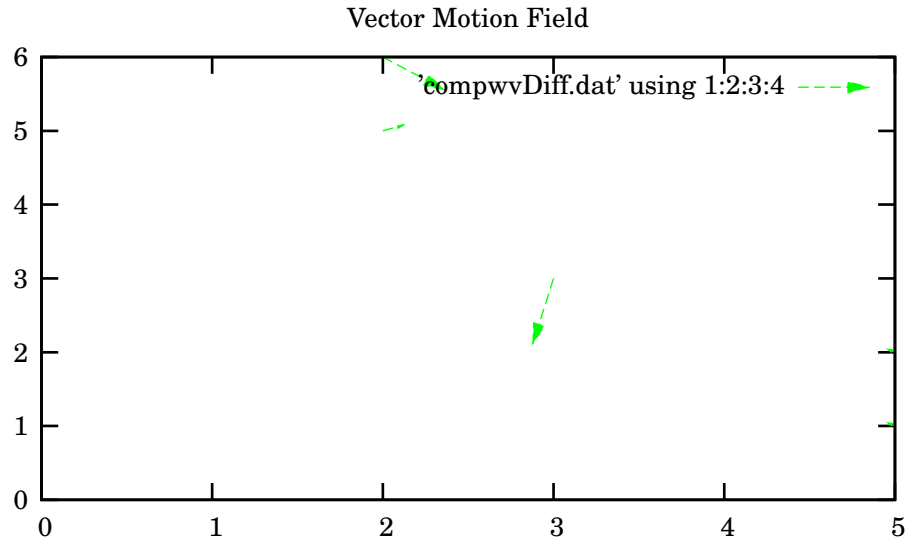


Figure 6.5: Vector differences between the dual channel competitive, and single WV channel outputs.

## 6.5 Future Developments

Possible developments that could be made to this project in the future include:

- The analysis of more sets of data. Data sets need not be 'remote sensed', and could include data from various other sources.
- Rewriting the toolkit in `C`, or `C++` would provide large speed gains, especially since a lot of the code makes use of `for` loops, which are much faster when optimised by a compiler. The design of the code could closely mirror the current Octave code, even down to the data structures used.
- Extending the results to more that two channels would also be interesting, and might provide more insight into the 'competition' between candidate vectors.
- Further research into available speed gain would be interesting, as a faster version of such a system could find use in defence systems, like RADAR object tracking for example.
- Eventually, there is no reason why a ME RL system could not work in three spatial dimensions. The computational cost would increase further still, because relaxation labelling would have to include many more neighbours, but there is no conceivable reason why it could not be done.

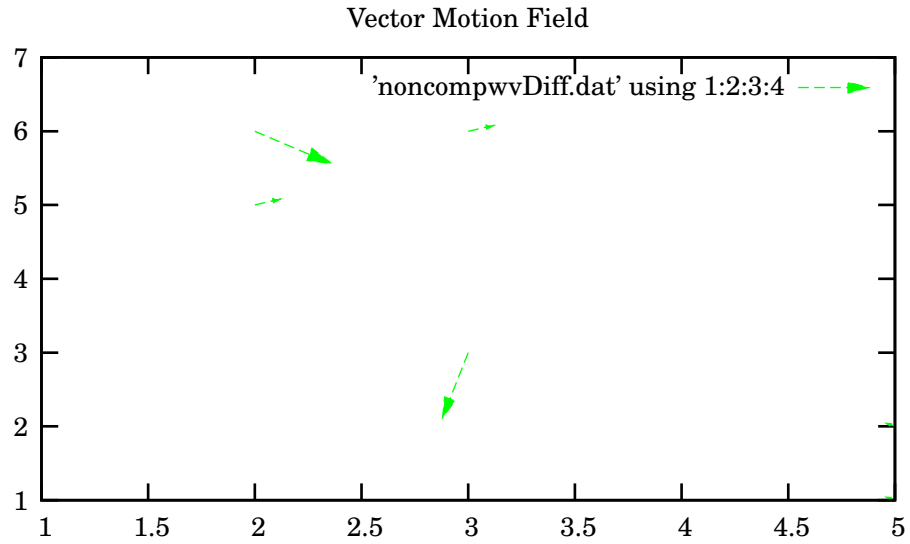


Figure 6.6: Vector differences between the dual channel non-competitive, and single WV channel outputs.

# Acknowledgments

I would like to thank Adrian Evans for his friendly help and support throughout this project.

# Bibliography


[Cha04] Chris Chatfield. *The Analysis of Time Series*. CRC Press LLC, 2004.

[Eat97] John W. Eaton. *GNU Octave Manual*. Network Theory Ltd, 1997.

[Eva00] Adrian N. Evans. Glacier surface motion computation from digital image sequences. *IEEE Transactions on Geoscience and Remote Sensing*, 38(2), 2000.

[GW01] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing 2/E*. Prentice Hall, 2001.

[HT99] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison Wesley, 1999.

[JA90] Y. Neuvo J. Astola, P. Haavisto. Vector median filters. *Proceedings of the IEEE*, 78(4), 1990.

[MSB99] Vaclav Hlavac Milan Sonka and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. PWS, 1999.

[Phi04] S Phisanbut. Feature tracking in remotely sensed multi-spectral imagery, 2004.

[QXWP97] S. J. McNeill Q. X. Wu and D. Pairman. Correlation and relaxation labelling: an experimental investigation on fast algorithms. *International Journal of Remote Sensing*, 18(3), 1997.

[TD01] EUM TD. Meterosat second generation: Msg system overview, 2001.

[TH00] David Thomas and Andrew Hunt. *Programming Ruby*. Addison Wesley, 2000.

# Source Code and Other Motion Fields

See the attached CD, which contains:

- Project code
- The full Subversion repository used to keep all revisions of the code. This will allow access to past version of all functions, as necessary.
- Compressed results. Results of ME RL using various block sizes and search sizes with and without post filtering. The results are compressed because uncompressed they amount to 4GB of data.
- Semester One report.
- This report.