MOTION SEGMENTATION FROM PERSPECTIVE IMAGES VIA RANK MINIMIZATION

A Thesis Presented

by

Mustafa Ayazoglu

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

Northeastern University
Boston, Massachusetts

July, 2008
Abstract

In this thesis, we consider the problem of segmenting data points coming from rigid bodies under perspective projection. As we all know the perspective projection camera model is the most general camera model of computer imaging, but unfortunately it is also known to be non-linear. Thus making the problem at the hand harder.

To solve this most general form of segmentation problem, we assume that we are given point correspondence data in image sequences. The data set consist of the projected coordinates of the points in 3D. The main idea of the method is to group the points according to the complexity of their motion in 3D. This idea intuitively formalizes, the fact that points from the same object would share more modes of motion and hence leading to less complex models than points from different objects.

We approached to the problem at the hand from a systems theory perspective. Estimating model order complexity and projective depth of the points is reduced to minimizing the rank of a Hankel matrix, which is constructed using the correspondence data and the unknown depths. This leads to a simple non-iterative segmentation algorithm that optimizes the use of spatial and temporal information. Since the presented algorithm exploits both spatial and temporal constraints, it does not require the estimation of the fundamental matrix and it is less sensitive to the effect of noise and outliers than approaches that rely solely on factorizations of the measurements matrix. In addition, the method can also naturally handle degenerate cases, e.g. cases
where the objects partially share motion modes.

We presented and demonstrated this novel motion segmentation algorithm, tested its performance on different kinds of cases as well as comparing it with some existing algorithms.
Acknowledgements

First of all I would like to thank my advisor Dr. Mario Sznaier, for tolerating a lazy student such as me, without his bright ideas and ever lasting patience I couldn’t be writing these lines. I would also like to thank Dr. Octavia Camps, for her support and guidance during the thesis.

I would specially like to thank Dr. Gilead Tadmor, for helping me on various subjects and believing in me since I first came to the United States. Without his help I couldn’t find my current advisor and couldn’t be as successful as I am now. He just told me not to fear of learning new things, his advice will be with me forever.

It has always been an honor to be a Fulbright Scholar. For giving me the opportunity of studying in the US. I am thankful to Fulbright officers in Ankara, Turkey. They were always very helpful. Especially Mrs. Gulesen Odabasioglu.

I would like to send my heartfelt thanks to my undergraduate professor Dr. Feza Arikan, she has always approached me as a son not as a regular university student, without her encouragements I wouldn’t be here now.

I also would like to thank my biology teacher, Mr. Kenan Karbek, from my high school years. He loved me as a son, he always believed me and expected more and more from me, without his support maybe even I wouldn’t be an Electronics Engineer. And of course I don’t want to forget my primary school teacher, Mrs. Zubeyde Erol,
she has regarded me as her "second" Mustafa. I have always been lucky to have started my education with such a motherly teacher.

And the Oscar goes... ups... sorry and my biggest thanks goes to my family, My father, Hadi, my mother Hulya and my sister, Muberra, I owe the everything that I have ever succeeded to them. You are my only reason being that I am. And my loving thanks and appreciations are only for "my" Behiye without you I couldn’t find the real Mustafa inside of me I will always love you each and every word in this thesis is dedicated to you.

P.S. I should admit that writing acknowledgments is the most joyful part of everything.
## Contents

Abstract

Acknowledgements

List of Figures

### 1 Introduction

1.1 Background .................................................. 1

1.2 Thesis Overview ............................................. 3

### 2 Rank Minimization Problem Overview

2.1 Rank Minimization Problem ................................. 6

2.2 Rank Minimization Relaxations ............................. 7

2.2.1 Trace Heuristic .......................................... 7

2.2.2 Log-Det Heuristic ........................................ 8

### 3 The Algorithm


3.1 Preprocessing Steps for the Algorithm ........................................ 9
  3.1.1 Recovery of the Missing Data ............................................. 9
  3.1.2 Initialization .............................................................. 13
3.2 Main Algorithm ....................................................................... 14
  3.2.1 The Role of Dynamics in Motion Based Segmentation ............ 14
  3.2.2 3D Motion Segmentation under Perspective Projection from Multiple Views ......................................................... 17
3.3 Implementation .................................................................... 27

4 Examples .................................................................................. 29
  4.1 Examples of Recovery ............................................................ 29
  4.2 Examples of the Algorithm ...................................................... 33

5 Results and Experiments .......................................................... 50
  5.1 Experimental Setup .............................................................. 50
  5.2 Experiments ........................................................................ 50

6 Conclusion ................................................................................ 56

Appendix .................................................................................... 57

A Source Code ............................................................................. 58
  A.0.1 Main ................................................................................. 58
  A.0.2 Block Hankel Create ......................................................... 66
A.0.3 Velocity .................................................. 67
A.0.4 Rep2 .................................................. 67

Bibliography 67
List of Figures

1.1 Physical effect of row or column permutation of measurement matrix 4

3.1 Two different objects and coordinate systems with respect to the objects 18

4.1 3D cubes data set frame 1 30
4.2 3D cubes data set frame 10 30
4.3 3D cubes data set frame 20 31
4.4 Randomly missing points on the trajectory 31
4.5 Randomly missing points on the trajectory after recovery method 32
4.6 Missing points are in the middle frames of the trajectory 33
4.7 Missing points are in the middle frames of the trajectory after recovery method 34
4.8 Depth estimation for point 1, 1st iteration 36
4.9 Depth estimation for point 2, 1st iteration 36
4.10 Depth estimation for point 3, 1st iteration 37
4.11 Depth estimation for point 4, 1st iteration 37
4.12 Depth estimation for point 4, 5th iteration ........................................... 39
4.13 Depth estimation for point 4, 5th iteration ........................................... 39
4.14 Depth estimation for point 4, 5th iteration ........................................... 40
4.15 Depth estimation for point 4, 5th iteration ........................................... 40
4.16 Depth estimation for point 1, 1st iteration ............................................ 42
4.17 Depth estimation for point 2, 1st iteration ............................................ 42
4.18 Depth estimation for point 3, 1st iteration ............................................ 43
4.19 Depth estimation for point 4, 1st iteration ............................................ 43
4.20 Depth estimation for point 1, 5th iteration ............................................ 44
4.21 Depth estimation for point 2, 5th iteration ............................................ 45
4.22 Depth estimation for point 3, 5th iteration ............................................ 45
4.23 Depth estimation for point 4, 5th iteration ............................................ 46
4.24 Depth estimation for point 5, 1st iteration ............................................ 47
4.25 Depth estimation for point 5, 5th iteration ............................................ 48

5.1 ArtificialBoxes data set frames 1, 11, 20 respectively .......................... 51
5.2 SubsampledBoxes data set frames 1, 5, 10 respectively .......................... 53
Chapter 1

Introduction

1.1 Background

The problem of motion segmentation, i.e. determining the number of moving objects from a sequence of 2D images and assigning points to each object has been the subject of substantial research in the last decade [13, 6]. A vast portion of the literature concentrates on 2D motion segmentation to estimate and segment the 2D motion field, or so called optical flow, on the image plane induced by the relative motion of the camera with respect to the objects in the scene, perspective effects, depth discontinuities, occlusions, etc. Approaches to 2D motion segmentation include temporal differences [14], background subtraction [22], and optical flow [1, 2, 3, 4, 9, 21, 30, 31].

A significantly smaller portion of the literature addresses the problem of 3D motion segmentation defined as the estimation of the 3D relative motion (rotation and translation) of multiple objects with respect to the camera. Feng and Perona [7] used K-means to cluster features. In [25] the problem is solved by successively finding the
most dominant motion through the use of RANSAC [15] to detect outliers. However, clustering techniques suffer from the problem that they can converge to suboptimal solutions depending on their initial condition.

Tomasi and Kanade [24] proposed the factorization method to recover simultaneously 3D motion and geometric structure of a single moving object from 2D images under the assumption of orthographic projection. This method uses an algebraic approach where a matrix, which is called measurement matrix $W$, with the coordinates of the tracked points over a number of frames is decomposed into the product of a motion matrix and a structure matrix, by using the rank deficiency of $W$. The measurement matrix $W$ of a data set consisting of $N_p$ number of feature points over $N_f$ frames can be written as:

$$W = \begin{bmatrix}
    p_{1,1} & p_{1,2} & \cdots & p_{1,N_p} \\
    p_{2,1} & p_{2,2} & \cdots & p_{2,N_p} \\
    \vdots & \vdots & \ddots & \vdots \\
    p_{N_f,1} & p_{N_f,2} & \cdots & p_{N_f,N_p}
\end{bmatrix} \quad (1.1)$$

where $p_{i,j} = (u^j_i, v^j_i)'$ here $u^j_i$ and $v^j_i$ are $j$th feature’s horizontal and vertical coordinates on $i$th frame respectively. Tomasi and Kanade’s method has been extended to paraperspective [17, 18] and perspective [23, 27] projection. Several extensions of these methods have been proposed to deal with non rigid objects [26, 36, 16] and to segment multiple moving objects [5, 8, 11, 15, 25, 35, 34, 29, 12, 33, 20].

An important limitation of these approaches is that they rely entirely on geometrical constraints between views. These constraints are derived from either the epipolar constraint between pairs of views or by imposing a rigidity constraint: the positions of a rigid moving object at the times when two views were captured are related by a rigid
transformation (a rotation and a translation). In fact, these approaches only use the ordering of the frames while tracking the features and establishing correspondences across frames. Once these correspondences are collected, the ordering of the frames is not longer used and the solutions obtained are invariant to frame permutations. This is due fact that most of these approaches uses some kind of factorization over a matrix, but as we all know permutation of rows or columns of a matrix does not change the rank of it. On the other hand, physically, the sequence resulting from permuting frames corresponds to a different motion. Figure: 1.1 shows this issue on a simple example; if we swap the first and the last frames of the image sequence, the direction of the car motion changes. Due to this simple but key issue, algorithms that rely only on the geometrical structures of the objects, have difficulties disambiguating objects that partially share motion modes – i.e. degenerate cases.

Recently, Lublinerman et al [16] addressed this issue by incorporating temporal information as well. This was done by modeling the trajectories of the features as the outputs of linear time invariant (LTI) systems and grouping features according to the complexity of the corresponding motion models.

Furthermore, until now, purely algebraic solutions only exist for the cases of two views [29, 32], multiple views under affine projection [5, 28, 16], and perspective projection of linearly moving points [10].

1.2 Thesis Overview

In this thesis, we present an algebraic non-iterative solution to the general problem of multi body 3D motion segmentation from an image sequence. The proposed method does not require knowing the number of objects, nor pairwise or triple wise processing of views nor the estimation of epipolar geometry nor the fundamental matrix and can
handle degenerated cases and missing data, due to occlusion. Instead, this work is inspired by the method in [16]. It exploits the temporal information encoded in the ordering of the given image sequence to segment 3D motion and simultaneously finds the projective depths of the scene points – i.e. the projective scale factors that represent the depth information lost during projection and from which it is possible to recover the scene structure up to a projective transformation. This work differs from Lublinerman’s work by the projection model that it assumes. While Lublinerman’s algorithm works under orthogonal projection, the algorithm in this thesis works under perspective projection.
The overview of this thesis is as following, in Chapter 2 we will discuss the rank minimization problem, which is at the heart of this master thesis, and some convexifying relaxations of the rank minimization problem as it was discusses by Fazel in her PhD thesis, [39]

In Chapter 3 we will describe the algorithm in detail in two sections. In Section 3.1 we will summarize the preprocessing steps of the algorithm. These steps are basically recovery of missing data information and initialization step for optimization. These steps are the essential preprocessing blocks of the initial data, before it is fed to the main algorithm block, which is designed to work with complete data. In Section 3.2 the algorithm itself will be demonstrated. We will explain the algorithm and we will give the flow chart of it, to make it more understandable. Even the idea itself is simple, to make it work robustly on most of the cases, careful implementation is needed and hence a flow chart is helpful. In Section 3.3 we will address implementation issues and implementation environment.

In Chapter 5 we will show the experimental settings with the results and experiments.

In Chapter 6, conclusions and possible future work will be presented. The code for the algorithm will be given in the thesis as an appendix section as well.
Chapter 2

Rank Minimization Problem

Overview

2.1 Rank Minimization Problem

As it is known Rank Minimization Problem (RMP) arises diverse in areas of science, from statistics to control theory. Unfortunately rank minimization problems are NP-hard so there is no known efficient solution. One of the best efforts to solve the RMP’s was made by Fazel in her PhD thesis, [39]. We will discuss it in Section 2.2 but before going in that direction, let us give the formal problem definition of RMP:

\[
\begin{align*}
\text{minimize} & \quad \text{Rank}(X) \\
\text{subject to} & \quad X \in C
\end{align*}
\]  

(2.1)

where \( X \in R^{m \times n} \) and \( C \) is a convex set.
2.2 Rank Minimization Relaxations

As it was mentioned in Section 2.1, since RMP is NP-hard we need some relaxations before attempting to solve the problem Fazel described two of them in [39]. Those are named trace and log-det heuristics. We will be using these heuristics in this thesis, whenever we need rank minimization.

2.2.1 Trace Heuristic

This is the first one of the relaxations. Fazel proved in her work that, when we have a RMP as defined in equation 2.1, we can relax it in the following way. Any RMP in equation 2.1 can be written as follows:

\[
\begin{align*}
\text{minimize} & \quad \text{Rank}(\text{diag}(Y, R)) \\
\text{subject to} & \quad \begin{bmatrix} Y & X \\ X & R \end{bmatrix} \geq 0 \\
& \quad X \in C
\end{align*}
\] (2.2)

where \(Y\) and \(R\) are both symmetric matrices and \(Y \in \mathbb{R}^{m \times m}\) and \(R \in \mathbb{R}^{n \times n}\) are additional variables, which converts the rank minimization problem into a positive semidefinite rank minimization problem. The necessity of the matrix to be rank minimized being positive semidefinite is, if the matrix is positive semidefinite in equation 2.1 then we can relax problem by changing \text{Rank} function with \text{Trace} function. Minimization of trace instead of rank was proven to minimize the convex envelope of rank in [39].

And hence equation 2.1 turns to:
\[
\begin{align*}
\text{minimize} & \quad \text{Trace}(\text{diag}(Y, R)) \\
\text{subject to} & \quad \begin{bmatrix} Y & X \\ X & R \end{bmatrix} \succeq 0 \quad (2.3)
\end{align*}
\]

which is an Linear Matrix Inequality (LMI) can be solved by using the existing tools.

### 2.2.2 Log-Det Heuristic

The second algorithm is an extension of the first one. Instead of replacing \textbf{Rank} function with \textbf{Trace} function, it uses \textbf{Log-Det} function. This leads to a very similar algorithm to the one with trace but with some updated weights at every iteration:

\[
\text{diag}(Y_{k+1}, R_{k+1}) = \arg\min \text{Trace}((\text{diag}(Y_k, R_k) + \delta I)^{-1}\text{diag}(Y, R)) \quad (2.4)
\]

\[
\text{subject to} \quad \begin{bmatrix} Y & X \\ X & R \end{bmatrix} \succeq 0
\]

\[
X \in C
\]

where \(k\) is the iteration number. Fazel said in her work that for initial points, they used identity matrices for \(Y_0\) and \(R_0\) but our extensive research showed that identity matrices are not the best selection as initial points for our case. Initialization issues will be discussed in Section 3.1.2.
Chapter 3

The Algorithm

3.1 Preprocessing Steps for the Algorithm

In this section we will describe two preprocessing steps required for the algorithm. The first step is for recovery of the missing data, and the second one is for initialization of the algorithm.

The first step is required because the algorithm itself deals with complete data to make it work with missing data, a preprocessing step of the input data is required. The second preprocessing step is not a mandatory but a helpful preprocessing step, it helps the optimization side, which will be discussed later on, of the algorithm to converge faster.

3.1.1 Recovery of the Missing Data

Ideally we want the measurement matrix, $W$, to be complete – i.e having all of the feature trajectories over all frames. Unfortunately this is not always the case for real
world applications. The data can be missing due to the following facts;

- Some of the features that we are tracking might have been occluded over some period of time.
- Some of the features that we are tracking might enter/leave the camera’s view.
- Measurement imperfections.

When there are missing data, some columns of $W$ in equation 1.1 would be partially filled. That is $p_{ij} = (u^i_j, v^i_j)'$ for some $i$ and $j$ is undefined. The approach that we will be using to recover the missing pieces is somewhat similar to the approach that is mentioned by Ding in [37]. Ding et al used a system theoretic approach to find the missing pixels in image inpainting problem. We use the same approach here but instead we will not try to find the missing pixels but the missing coordinates of trajectories.

To estimate the missing coordinates of a feature’s trajectory, think about the following vector as it is defined in equation 3.1 for $i$’th feature:

$$t(k) = \begin{bmatrix} u^i_k \\ v^i_k \end{bmatrix}$$ (3.1)

We will assume that $t(k)$ is generated by a stationary Gaussian-Markov process as Ding did in his work. See equation 3.2.

$$t(k) = \sum_{j=1}^{n} g_j t(k - j) + h_j e(k - j).$$ (3.2)
where, \( g_j \) and \( h_j \) are fixed coefficients and \( e(.) \) denotes a stochastic input. Note that this can always be assumed without loss of generality, since if we are given \( N_f \) measurements of \( t(.) \) and \( e(.) \), there exist an operator of the form in equation 3.2. (See: Chapter 10 of [19]), and furthermore we can always assume \( e(.) \) as an impulse since the necessary spectral info can always be absorbed in the coefficients \( g_j \) and \( h_j \).

Recall that \( t(k) \) is defined for some values of \( k \) and it is undefined for the rest.

As Ding stated in his work if \( g_j \) and \( h_j \) coefficients are known then missing values of \( t(k) \) can be found, but explicit identification of the model is unnecessary in this case. Missing values of \( t(k) \) can be found using directly from known values of \( t(k) \) by solving a rank minimization problem. The idea behind of it is that, minimum rank solutions tends to create less complex models and hence more smooth trajectories.

Informally we can say that we will try to find the missing values of the features such that the system that might generate this sequence has the lowest system order as possible.

Based on the work of Moonen et al in [38]; Moonen’s work was also used in [16]

**Fact 1.** Consider a system with minimal state system representation of the following form:

\[
\begin{align*}
\xi(k+1) &= A\xi(k) + B e(k) \\
t(k) &= C\xi(k)
\end{align*}
\]

(3.3)

where \( A \in R^{n \times n} \) and \( B \in R^{n \times m} \) and \( t(k) \) is the coordinates of a feature at time \( k \) as it appeared in equation 3.2

Construct a concatenated infinite matrix \( H = (H_t^T \ H_e^T)^T \) where \( H_t \) and \( H_e \) are defined as follows:
\[ \mathbf{H}_t = \begin{bmatrix} t(1) & t(2) & \cdots & t(m) & \cdots \\ t(2) & t(3) & \cdots & t(m+1) & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots \\ t(n) & t(n+1) & \cdots & t(m+n) & \cdots \\ \vdots & \vdots & \cdots & \vdots & \ddots \end{bmatrix} \]  

Then if \( \varepsilon(\xi(k+1)e(k)) = 0 \), where \( \varepsilon \) stands for expectation, \( \text{rank}(\mathbf{H}) = n + m \)

And note, as we aforementioned we can always assume \( e(k) \) as an impulse and \( \text{rank}(\mathbf{H}_t) = n \).

So if \( N_f \gg n \), which is usually the case in practice, we can try to solve the following rank minimization problem to find the missing coordinate information.

Let

\[ \mathbf{H}_t = \begin{bmatrix} t(1) & t(2) & \cdots & t(N_f) \\ t(2) & t(3) & \cdots & t(N_f/2 + 1) \\ \vdots & \vdots & \ddots & \vdots \\ t(N_f/2) & t(N_f/2 + 1) & \cdots & t(N_f) \end{bmatrix} \]
Recall that $t(k)$ is defined some for values of $k$ and it is undefined for the rest. So defined values are fixed constants and undefined values are our optimization variables. In this case, the formal rank minimization problem is as follows:

$$
\text{minimize } \text{Rank}(H_t) \\
\text{w.r.t. missing } t(k)\text{'s}
$$

(3.7)

And after applying trace heuristic as it was described in Section 2.2 we get the LMI problem that we will try to solve:

$$
\text{minimize } \text{Trace}(\text{diag}(Y, R)) \\
\text{subject to } \begin{bmatrix} Y & H_t \\ H_t & R \end{bmatrix} \succeq 0 \\
\text{w.r.t. missing } t(k)\text{'s}
$$

(3.8)

And hence the algorithm that we will be using to recover missing data is as follows:

<table>
<thead>
<tr>
<th>Algorithm 1 Data Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. for all $t$ such that at least one of its elements is missing do</td>
</tr>
<tr>
<td>2. Construct $H_t$ as in equation 3.6</td>
</tr>
<tr>
<td>3. Solve the relaxed rank minimization problem defined in equation 3.8</td>
</tr>
<tr>
<td>4. Replace the missing variables in $t$ with the ones found in step 3</td>
</tr>
<tr>
<td>5. end for</td>
</tr>
</tbody>
</table>

3.1.2 Initialization

As we mentioned in Section 2.2.2 log-det heuristic requires some initial matrices for $Y_0$ and $R_0$ to work. Fazel used identity matrices of the correct size for $Y_0$ and
R₀ in [39] but our extensive amount of research showed that identity matrices are not the best selection as initial points for our case. This is because initial identity matrix selection of these matrices forces the depths to be determined to be initialized to some functions (trajectories) which makes perfect sense mathematically but not always physically, and if we start the algorithm somewhere wrong either it will take long number of iterations to converge or worse it will converge to some local minima. Hence we supply a methods to initialize these matrices to overcome these issues.

Basically, starting from constant trajectories for the depths is helpful, but since rank minimization problem needs Y₀ and R₀ we need to find the corresponding matrices. Assume for now that the problem that we will be solving is of the form of equation 2.3. We will refine it according to our needs in Section 3.2. The important line for now is X ∈ C which defines the constraints of the problem. To be able to solve for Y₀ and R₀ we need to change C such that our intention, fixing depth trajectories, is imposed to the problem. If we solve this slightly modified LMI, we will get Y₀ and R₀ which can later be used as initial matrices for the rank minimization problem for segmentation which will be defined in Section 3.2.

3.2 Main Algorithm

3.2.1 The Role of Dynamics in Motion Based Segmentation

The importance of dynamics in motion segmentation was originally discussed by Lublinerman et al in [16]. There, they pointed out that the temporal evolution of the motion of the objects – i.e. their dynamics – (which is encoded in the time ordering of the features) encapsulates important information that can be exploited to segment moving objects, even in the cases when the objects share partial motion.
modes. Intuitively, the motion of a single rigid object can be completely described by the motion of a coordinate system attached to the object. On the other hand, a joint description of the motion of two rigid moving objects is more complex: it must describe the motion of one of the objects as above, and additionally it must describe the relative independent motion of a coordinate system attached to the second object with respect to the first one.

The dynamics based segmentation algorithm proposed in [16], also known as the **Hankel-based motion segmentation** algorithm does not make prior assumptions about either the structure or motion of the objects. Instead, based in the ideas described above, it clusters points in *orthographic 2D images* according to the complexity of the model required to explain their joint motion in the image plane. The core of the method is a procedure to estimate model complexity from the rank of a **Hankel** matrix formed from the measurements of the features which retains the temporal ordering of the frames as described below.

The algorithm is based on the fact that given a finite sequence of $N$ measurements of a vector $\mathbf{f}$, it is always possible to model its dynamics by a sufficiently high order linear time invariant (LTI) dynamic model of the form:

$$
\mathbf{f}_k = \sum_{i=1}^{n} g_i \mathbf{f}_{k-i} + h_i \mathbf{e}_{k-i}.
$$

(3.9)

where $n \leq N$ is the order of the system, and $\mathbf{e}(\cdot)$ is an exogenous stochastic input with appropriate statistics (Chapter 10 of [19]). In particular, in [16], the algorithm considers feature vectors given by the difference of the first two coordinates of two scene points:
\[ f_{k}^{i,j} = \begin{bmatrix} X_{ik} - X_{jk} \\ Y_{ik} - Y_{jk} \end{bmatrix} \] (3.10)

where \( Q_i(k) = (X_i(k), Y_i(k), Z_i(k))' \) and \( Q_j(k) = (X_j(k), Y_j(k), Z_j(k))' \) represent the locations at time \( k \) of the 3D points \( Q_i \) and \( Q_j \), measured with respect to a coordinate system attached to an orthographic camera. Since the camera is assumed orthographic, the image of point \( Q_i \) at time \( k \), \( q_i(k) \) is given by:

\[ q_i(k) = \begin{bmatrix} u_{ik} \\ v_{ik} \end{bmatrix} = \begin{bmatrix} X_i(k) \\ Y_i(k) \end{bmatrix} \] (3.11)

and the feature vector \( f_{k}^{i,j} \) can be obtained directly from the tracked image features:

\[ f_{k}^{i,j} = q_i(k) - q_j(k) \] (3.12)

Next, the lowest order \( n \) of an LTI system compatible with the measurements \( f_{k}^{i,j}, k = 1, \ldots, N_f \) and an impulse input, is estimated by computing the rank of the Hankel matrix formed as:

\[ H_{f^{i,j}}(k, l) = \begin{bmatrix} f_{1}^{i,j} & f_{2}^{i,j} & \cdots & f_{l}^{i,j} \\ f_{2}^{i,j} & f_{3}^{i,j} & \cdots & f_{l+1}^{i,j} \\ \vdots & \vdots & \ddots & \vdots \\ f_{l}^{i,j} & f_{l+1}^{i,j} & \cdots & f_{k+l-1}^{i,j} \end{bmatrix} \] (3.13)

where \( l \geq k \gg n \), since under mild conditions [38]
Once the order of the motion for each pair of image points is found, the algorithm proceeds to cluster the image points to obtain clusters of points according to their motion complexity.

### 3.2.2 3D Motion Segmentation under Perspective Projection from Multiple Views

The main limitation of the Hankel-based segmentation algorithm summarized in Section 3.2.1 is that it assumes an orthographic projection model so it can work directly with image coordinates to build the Hankel matrix in equation 3.13. In the more general case, when the camera is modeled as a perspective projection, the Hankel matrix depends on the unknown point depths $Z_i(k)$. In this section, we present a formulation that allow us to generalize the Hankel-based segmentation algorithm to remove this limitation in order to segment and find the unknown depths simultaneously from a sequence of perspective images.

Suppose that we have an unknown number of 3D objects moving relatively to a perspective camera, which are captured in $N_f$ frames. Assume for now (this assumption will be removed later), that a set of 3D points from these objects are visible in all the frames and that their image projections have been successfully tracked. Our goal is to recover the 3D structure (point locations) and the motion of these objects with respect to the camera frame, from the image measurements. We will assume that the camera calibration (or any additional geometric information) is not available.

Consider the standard camera coordinate system, with the origin at the center of
projection, and its $X, Y$ and $Z$-axes along the row and column directions of the image array and its optical axis, respectively.

As before, let $Q_i(k)$ be the unknown 3D coordinates of the observed point $i$ in frame $k$, measured with respect to the camera coordinate system, in this case we can always write any point in an object as a linear combination of 4 points from the same object.

By examining the Figure: 3.1 we may prove our claim. Let us think of four points ($Q_1, Q_2, Q_3$ and $Q_4$ in the figure), if we see the things from camera coordinate system nothing can be fixed and things change from frame to frame, but if we carry our coordinate system to one of these points say $Q_1$ things will be easier. In that
case think about a point $Q_5$ we can write the following equation about $Q_5$ (Note we dropped the frame index but this equation is true for all frames):

$$Q_5 - Q_1 = \sum_{j=2}^{4} a_j (Q_j - Q_1)$$  \hspace{1cm} (3.15)

here $a_j$'s are suitable constants, and if $Q_5$ is from the same object equation 3.15 holds for all frames (same $a_j$'s) (Note that this is also true when we select $Q_5=Q_1$--i.e $a_j$'s will be constants but not necessarily zero since $(Q_j-Q_1)$'s are not orthogonal to each other ), informally we can say the position of a point in an object does not change with respect to any other point in the same object, but if $Q_5$ is from another point then even equation 3.15 holds the coefficients would be changing frame by frame, leading more complex models.

If we take derivative of both sides, the equation 3.15 will keep holding with resulting velocity vectors say $V_j$'s. Then equation 3.15 becomes:

$$V_5 - V_1 = \sum_{j=2}^{4} a_j (V_j - V_1)$$  \hspace{1cm} (3.16)

Note that taking derivative of both sides may seem redundant but, as we keep explaining things will be clearer, and taking the derivatives is important for normalization purposes.

Normally we define derivative as follows:

$$V_j(k) = \frac{d}{dk} Q_j(k)$$  \hspace{1cm} (3.17)

Remember $Q_j(k)$ is a function of time. We propose first finite difference as an approximation of the derivative and $V_j(k)$'s turn out to be:
\[ V_j(k) = Q_j(k) - Q_j(k - 1) \quad (3.18) \]

If we define the following feature vector:

\[
f_k = \begin{bmatrix}
V_5(k) - V_1(k) \\
V_4(k) - V_1(k) \\
V_3(k) - V_1(k) \\
V_2(k) - V_1(k)
\end{bmatrix}
\quad (3.19)
\]

*equation* 3.9 will still hold. If we knew the 3D coordinates of the points then we could simply use these features as described in [16] to get the segmentation but, in reality we know 2D projections (*equation* 3.20) of these coordinates onto image plane but not 3D points itself. What we know is if these points are from the same object they will lead to less complex models than the points coming from different objects. So we may try to estimate the 3D points in the following manner while segmenting the data at the same time.

\[
q_i(k) = \begin{bmatrix}
u_{ik} \\
v_{ik}
\end{bmatrix} = \begin{bmatrix}
f \frac{X_i(k)}{Z_i(k)} \\
f \frac{Y_i(k)}{Z_i(k)}
\end{bmatrix}
\quad (3.20)
\]

As it is clear from *equation* 3.20 we need depth information to recover 3D coordinate data. By absorbing the focal length \( f \) in the depth we can also write:

\[
X_i(k) = Z_i(k) u_{ik} \\
Y_i(k) = Z_i(k) v_{ik}
\quad (3.21)
\]
And then since we know the system order would be low for the points from the same object, by constructing suitable LMI systems and solving for variable depths \((Z_i(k)'s)\) such that the system order is minimized leads to the following algorithm. As it was mentioned before to find the system order, we do not need to identify the system by the same reasoning as described in Fact 1 and in [38].

Let us assume all points are in an "abstract bag". What we need do is to select a point, \(Q_1\), label it and keep it somewhere fixed. Then we need to select three more points from the bag randomly and name them as \(Q_2, Q_3\) and \(Q_4\) (as for now let us assume all four points are coming from the same object), we can form truncated versions of 3D points as follows:

\[
Q_i(k) = \begin{bmatrix}
Z_i(k)u_{ik} \\
Z_i(k)v_{ik}
\end{bmatrix}
\]

(3.22)

where \(Z_i(k)'s\) are optimization variables and hence \(Q_i(k)'s\) are variables as well.

Note that truncation not necessarily detrimental for segmentation since most type of motions first two components are enough and besides this truncation will reduce the computational complexity.

By using equation 3.18 we then obtain corresponding velocity vectors as variables, and form the following matrices as variables:

\[
H_{Vj} = \begin{bmatrix}
V_j(1) & V_j(2) & \cdots & V_j(l) \\
V_j(2) & V_j(3) & \cdots & V_j(l+1) \\
\vdots & \vdots & \ddots & \vdots \\
V_j(k) & \mathbf{f}_{k+1}^{ij} & \cdots & V_j(k+l-1)
\end{bmatrix}
\]

(3.23)
where $j = 1, 2, 3, 4$,

$$H_V = \begin{bmatrix} H_{V2} - H_{V1} \\ H_{V3} - H_{V1} \\ H_{V4} - H_{V1} \end{bmatrix}$$

(3.24)

$H_V$ is a variable matrix which depends on selection of $Z_i(k)$'s, our optimization variables.

Note that equation 3.24 can be thought as, it is coming from a multi-output system, whose output can be defined as in equation 3.25 and impulse being as its input, refer to [38] for details.

$$y_k = \begin{bmatrix} V_2(k) - V_1(k) \\ V_3(k) - V_1(k) \\ V_4(k) - V_1(k) \end{bmatrix}$$

(3.25)

Note that equation 3.25 can be though as the truncated version of equation 3.19 which is the feature vector that we defined for 3D motion segmentation. Where $Q_5$ is assumed equal to $Q_1$ and redundant zero vector is dropped.

Once we construct the variable $H_V$, we may try to solve the following rank-minimization problem since the rank of $H_V$ is the order of the multi-output system, under mild conditions, according to [38].

$$\text{minimize} \quad \text{Rank}(H_V)$$

w.r.t. missing $Z_i(k)$'s

such that $Z_i(k) \geq c$
Where \( c \) is simply a positive constant to prevent a trivial solution which is all \( Z_i(k)'s \) being zero.

We stated that rank minimization problem is an NP-hard problem in Chapter 2, so we will use relaxation described in Section 2.2 after the log-det relaxation the problem turns to:

\[
\text{diag}(Y_{k+1}, R_{k+1}) = \arg\min \text{ Trace}((\text{diag}(Y_k, R_k) + \delta I)^{-1}\text{diag}(Y, R))
\]

subject to

\[
\begin{bmatrix}
Y & H_V \\
H_V & R
\end{bmatrix} \succeq 0
\]

\( Z_i(k) \geq c \)

w.r.t. missing \( Z_i(k)'s \)

To initialize the matrices, \( Y_0 \) and \( R_0 \), we will use the approach demonstrated in Section 3.1.2 and before solving equation 3.27 we need to solve the following LMI system.

\[
\text{diag}(Y_0, R_0) = \arg\min \text{ Trace}((Y, R))
\]

subject to

\[
\begin{bmatrix}
Y & H_V \\
H_V & R
\end{bmatrix} \succeq 0
\]

mean(\( Z_i(.) \)) = \( K \)

where \( K > c \).

By solving equation 3.27 we can find the estimates of depths - i.e. \( Z_i(k)'s \). Let Rank(\( H_V \)) be \( r^* \) and remember we assumed all four points are coming from the
same object so theoretically $r^*$ is the lowest possible system order. Then we need to proceed, and select another point say $Q_5$. After finding the corresponding velocity vector $V_5$ as it was defined before, we may construct the Hankel matrix corresponding to that point as in the form of equation 3.23 and we can construct new $H_V$ as:

$$H_V = \begin{bmatrix} H_{V2} - H_{V1} \\ H_{V3} - H_{V1} \\ H_{V4} - H_{V1} \\ H_{V5} - H_{V1} \end{bmatrix}$$ (3.29)

where the only variable is $H_{V5}$ and corresponding depth, the rest of the elements are fixed from the previous step.

Note this time that equation 3.29 can be thought as, it is coming from a multi-output system, whose output can be defined as in equation 3.30 and impulse being as its input.

$$y_k = \begin{bmatrix} V_2(k) - V_1(k) \\ V_3(k) - V_1(k) \\ V_4(k) - V_1(k) \\ V_5(k) - V_1(k) \end{bmatrix}$$ (3.30)

Note $y_k$ is the feature vector that we defined for 3D motion segmentation.

Once we construct the variable $H_V$, we may try to solve the following rank-minimization problem:
\[
\text{diag}(Y_{k+1}, R_{k+1}) = \arg\min \text{ Trace}\left((\text{diag}(Y_k, R_k) + \delta I)^{-1}\text{diag}(Y, R)\right)
\]

subject to
\[
\begin{bmatrix}
Y & H_V \\
H_V & R
\end{bmatrix} \succeq 0
\]

(3.31)

\[Z_i(k) \geq c \quad \text{w.r.t. missing } Z_i(k)'s\]

Which is the same with equation 3.27 but the only difference is this time the only variable is \(Z_5(k)'s\). Let \(\text{Rank}\) of updated and optimized \(H_V\) be \(r\); in this case we have two possibilities:

- If the point \(Q_5\) is from the same object as the other four points \(r\) would be ideally be equal to \(r^*\), in reality we expect \(r\) to be close to \(r^*\)

- If \(Q_5\) is from a different object than the other four points \(r\) would be greater than \(r^*\),

Thus leading the following algorithm:

**Algorithm 2** Motion Segmentation

1. \(Nr=50\)
2. Put all points to a set say, \(R_{set}\)
3. Set class to 0
4. while t such that at least one of its elements is missing do
5. Increase class by 1
6. Select a random point from set \(R_{set}\) say \(Q_t\) to this point, label it with current class number and, exclude this point from \(R_{set}\)
7: Set $r^*$ to a big number \{Ransac type random selection\}
8: \textbf{for} $i=1$ to $Nr$ \textbf{do}
9: \quad Select three different random points from set $R_{set}$ say $Q_2$, $Q_3$, and $Q_4$ to these points
10: \quad Select a value for $\delta$
11: \quad Solve \textit{equation} 3.28 to initialize $Y_0$ and $R_0$
12: \quad \textbf{for} $t=1$ to maxiteration \textbf{do}
13: \quad \quad Iterate using \textit{equation} 3.27
14: \quad \textbf{end for}
15: \quad Determine the rank of $H_V$ using singular value thresholding say $r$ to this rank
16: \quad \textbf{if} $r \leq r^*$ \textbf{then}
17: \quad \quad Set $r^*$ as $r$
18: \quad \quad Keep this quadruple point combination and found depths somewhere say these points as $Q^*_2$, $Q^*_3$, and $Q^*_4$ to these points
19: \quad \textbf{end if}
20: \quad \textbf{end for}\{End of Ransac type random selection\}
21: \quad Construct $H_{Vj}'s$ using \textit{equation} 3.23 and estimated depths for points; $Q_1$, $Q^*_2$, $Q^*_3$, and $Q^*_4$ \{Note that now $H_{Vj}'s$ are constant matrices\}
22: \quad Exclude $Q^*_2$, $Q^*_3$, $Q^*_4$ from $R_{set}$, label them with current class number
23: \quad Initialize $Y_0$ and $R_0$ using a similar approach to the one at step 11
24: \quad \textbf{for all Points remaining in $R_{set}$ do}
25: \quad \quad Call currently selected point as $Q_5$
26: \quad \quad \textbf{for} $t=1$ to maxiteration \textbf{do}
27: \quad \quad \quad Iterate using \textit{equation} 3.31
28: \quad \quad \textbf{end for}
29: \quad \quad Determine the rank of $H_V$ using singular value thresholding say $r$ to this
```plaintext
30: if $r \approx r^*$ then
31:     Label the current point with current class number
32: end if
33: end for
34: Exclude newly labeled points from $R_{set}$
35: end while
```

### 3.3 Implementation

In this section we will briefly discuss implementation issues of the algorithm.

We have developed the algorithm under MATLAB environment. We used CVX tool (See [40]), which is a free disciplined convex optimization package for MATLAB. CVX tool greatly simplifies the syntax, so it is strongly recommended.

One of the most important implementation issue worth considering is, even theoretically the background of this thesis is strong there can be problems while implementing it, One of them is as we have covered, we need to iterate to find the depth which is our rank minimizing $Z$ if we leave all points in $Z$ as variables after some iterations the solution may diverge, just to keep the solution around our best guess, remember if we do not have knowledge about the problem then our best guess is constant $Z$, what someone can consider, randomly selecting a point in $Z$ and making it equal to its previous value while the rest are variables. So with such a strategy the phenomenon of one point diverging to infinity while others remain close to zero is prevented. This implementation strategy proves itself as a robust way of identification. Let us assume after some iterations we exactly found the depth so the rank is already minimum if we
iterate further with previous strategy we fix one point which becomes the true value of $Z$ for this frame and rest changes since it has already been found the algorithm would not update the other z’s and it will stuck at the correct $Z$ as expected.
Chapter 4

Examples

In this chapter we will give several examples of the missing data recovery method and the algorithm, without comparing it with existing methods.

4.1 Examples of Recovery

As we have mentioned in Section 3.1.1, the main algorithm requires complete data to process, this section gives several examples of the estimates of the missing data.

For this issue we have created an artificial data set coming from the following frames. This data set consists of movement of two different boxes (green and red) freely in the space, to emphasize the algorithm’s most powerful side it has a strong motion component in the direction of Z-axis. The data points are the corners of each box. So we have 16 data points in this scenario. Let’s randomly delete some data from the data set and let’s see how our data recovery algorithm performs.

Figure 4.4 show trajectory of a point’s one of the coordinate. Here circles are real data points crosses at zero are the missing points. So if we have a circle on top of a cross
Figure 4.1: 3D cubes data set frame 1

Figure 4.2: 3D cubes data set frame 10
Figure 4.3: 3D cubes data set frame 20

Figure 4.4: Randomly missing points on the trajectory
we know that point if not that point is missing. The missing points on Figure 4.4 are randomly selected and only consist of one frame run of missing points. In this case as seen from Figure 4.5 all missing points are recovered perfectly.

![Figure 4.5: Randomly missing points on the trajectory after recovery method](image)

The missing points on Figure 4.6 are the middle points of the trajectory consisting of a seven-frame run of missing points. In this case as seen in Figure 4.7 the points that are closer to known points are recovered nicely and estimation degraded as expected when we move away from known points as it can be seen from Figure 4.7. Note that we have only 20 frames so 7 frames missing means only 65 percent of the data is fully known.
4.2 Examples of the Algorithm

In this section we will discuss how difficult the 3d segmentation problem is by using some examples obtained with the algorithm. In this section not only do we care about the algorithm’s results but also the internal dynamics of the algorithm. As a test data set, we will use our artificially generated boxes, since we know everything about it –i.e. Z components of each point over all frames (See Figures 4.1 4.2 4.3). Note in this data set points from 1 to 8 are from one box and points from 9 to 16 are from the other box.

Let’s assume point 4 as our fixed $Q_1$ we proceed and select three random points, let they be 10,13, and 14 for $Q_2,Q_3$ and, $Q_4$ respectively. Clearly these four points are not from the same box even the last three are from one box.

We convert these four points to corresponding velocity vectors namely $V_j$’s and con-
struct $H_{V_j}$ matrices and $H_{V}$ matrix to solve for initial values of $Y_0$ and $R_0$ by solving corresponding rank minimization problem \textit{equation} 3.28.

Since we have the depths before proceeding, we can check the singular values of $H_{V}$ matrix by using real depths (Note: Singular values normalized such that the total is 100)
Figures 4.8 4.9 4.10 4.11 show the estimated depths after the first iteration of rank minimization process where blue circles are the current estimates, red circles are the previous estimates (in the first iteration initialized values) and filled blue circles are the actual depths, corresponding singular values are shown below:
Figure 4.8: Depth estimation for point 1, 1st iteration

Figure 4.9: Depth estimation for point 2, 1st iteration
Figure 4.10: Depth estimation for point 3, 1st iteration

Figure 4.11: Depth estimation for point 4, 1st iteration
If we keep proceeding till last iteration of rank minimization process the depths are found as in *Figures* 4.12 4.13 4.14 4.15

And singular values are as:

\[
\begin{bmatrix}
54.6126 \\
42.9494 \\
2.3448 \\
0.0776 \\
0.0100 \\
0.0036 \\
0.0014 \\
0.0004 \\
0.0001 \\
0.0000
\end{bmatrix}
\]

(Note: Singular values normalized such that the total is 100)

(4.3)

Note that these singular values correspond to more rank deficient matrix than the ones at first iteration and with a suitable selection of threshold, rank can be estimated.

Now if we proceed further the algorithm returns back and selects three random points again. Let them be 1, 5, and 7 for $Q_2, Q_3$ and, $Q_4$ respectively the same procedure applies to these points but this time note that all of the four points are from the same object.

We have the depths so before proceeding, we can check the singular values of $H_V$ matrix by using real depths (Note: Singular values normalized such that the total is 100)
Figure 4.12: Depth estimation for point 4, 5th iteration

Figure 4.13: Depth estimation for point 4, 5th iteration
Figure 4.14: Depth estimation for point 4, 5th iteration

Figure 4.15: Depth estimation for point 4, 5th iteration
Clearly this is a system of order 2, and as expected the rank is much more less than the case where points are from different objects.

Figures 4.16 4.17 4.18 4.19 show the estimated depths after the first iteration of rank minimization process for the case where all of the points are from the same object.

And the singular values after the first iteration:
Figure 4.16: Depth estimation for point 1, 1st iteration

Figure 4.17: Depth estimation for point 2, 1st iteration
Figure 4.18: Depth estimation for point 3, 1st iteration

Figure 4.19: Depth estimation for point 4, 1st iteration
If we keep proceeding till last iteration of rank minimization process the depths are found as in Figures 4.20 4.21 4.22 4.23.
Figure 4.21: Depth estimation for point 2, 5th iteration

Figure 4.22: Depth estimation for point 3, 5th iteration
Figure 4.23: Depth estimation for point 4, 5th iteration

And singular values are as: (Note that 3rd singular value of equation 4.6 is about 5 times less than its correspondent in equation 4.3)

\[
\begin{pmatrix}
56.8613 \\
42.6017 \\
0.4932 \\
0.0345 \\
0.0074 \\
0.0016 \\
0.0002 \\
0.0000 \\
0.0000
\end{pmatrix}
\]  

(4.6)
See how the 3rd singular value diminished after 5 iterations. If we keep studying further after preset iteration number is completed the algorithm would have found the "best" quadruple combination. And then algorithm proceeds to the second step of iterations, by using these four points namely, \( Q_1, Q_2, Q_3 \) and, \( Q_4 \) and selecting each not processed points as \( Q_5 \) the algorithm tries to find their class.

Let’s for now assume the points 1,4,5,7 is the best combination. Then the algorithm labels point 4,5,7 with the label of 1 and exclude these points from the set.

Algorithm takes \( Q_5 \) and finds its corresponding initial \( Y_0 \) and \( R_0 \) matrices with the same approach –i.e. assuming constant depth and solving for the matrices.

Then Algorithm starts solving for rank minimization problem, as it is depicted in equation 3.31, After the first iteration, when \( Q_5 \) selected as point 2, the estimated depth is as in Figure 4.24

![Figure 4.24: Depth estimation for point 5, 1st iteration](image-url)
and the corresponding singular values as in equation 4.7

\[
\begin{bmatrix}
57.1022 \\
38.1234 \\
4.1503 \\
0.5377 \\
0.0806 \\
0.0049 \\
0.0007 \\
0.0001 \\
0.0000 \\
0.0000
\end{bmatrix}
\] (4.7)

After 5th iteration depth estimation is as follows in Figure 4.25

Figure 4.25: Depth estimation for point 5, 5th iteration
and the corresponding singular values as in equation 4.8

\[
\begin{bmatrix}
57.0656 \\
39.1069 \\
3.2491 \\
0.4659 \\
0.1012 \\
0.0094 \\
0.0018 \\
0.0002 \\
0.0000 \\
0.0000
\end{bmatrix}
\] (4.8)

Since the rank is approximately same as quadruples’ then \( Q_5 \) is labeled as the labels of quadruple, then algorithm selects the next point and proceeds. If the rank with in some tolerance level close to quadruple rank then the point is marked as the same label with quadruple and excluded from the set if not it is not marked and kept in the set for further processing.
Chapter 5

Results and Experiments

5.1 Experimental Setup

To conduct the experiments we have used a laptop with Intel Core 2 Duo Processor at 1.83 GHz with 1GB of memory. We used MATLAB to implement the algorithm and and CVX package [40].

We created our artificial data set as well as using some real videos.

5.2 Experiments

We used two different data sets to evaluate the performance of our newly developed algorithm. One of the data sets is an artificially generated data set and the other one is a real data set. We call these data sets as, ArtificialBoxes, SubsampledBoxes.

ArtificialBoxes data set consist of two boxes rotating and moving freely in the space. Their motion is adjusted such that the effects of perspective projection can be seen
easily. Data points are the corners of each box and it consists of 20 frames. Sample frames of this data set can be seen from *Figure: 5.1*

![ArtificialBoxes data set frames 1, 11, 20 respectively](image)

*Figure 5.1: ArtificialBoxes data set frames 1, 11, 20 respectively*

Our experiments with ArtificialBoxes data set showed that, when the complexity of the motion and the number of frames are consistent the algorithm can successfully segment the data set, and since we have the ground truth for the depths we can claim
that it can estimate the projective depths quite reasonable. Even if we used truncated version of the hankel matrices, the algorithm showed a robust performance.

As it can be seen from the data set this experiment is a really hard problem because most of the significant motion is on z direction, and the boxes are so close to each other hence making the problem at the hand harder.

SubsampledBoxes data set is actually a data set, used in [20] and can be found in the author’s web page. The original data set in [20] is subsampled such that it includes less number of data points to make conducting the experiments faster. This data set consist of two objects pulled on a table. This data set also have significant motion in Z direction hence perspective projection effects are visible. 5 points are from one object and the other is from the other object, making total number of tracked features 10. This data set consist of 10 frames, but as it was mentioned earlier since we need at least 2 times number of frames than the complexity of motion, we used the following strategies to increase the number of frames and hence to move the operation of the algorithm to a safer side. The first strategy is to Repeat the data set itself so that motion is repeated 2 times this makes the constructed hankel matrices circulant and increases the discrimination of singular values so the rank estimation becomes easier. The second method one can try is flipping the data set and adding it to the end of the original data set, hence when the original motion is completed the frames plays backward, this kind of strategy is also have similar effects as the previous one namely, increasing the number of frames and singular value discrimination. Sample frames of this data set can be seen from Figure: 5.2

Note that points 1 to 5 belong to the objects at the left hand side of the frames and points 6 to 10 belong to the objects at the right.

Our experiments with SubsampledBoxes data set showed that the original number of frames, which is 10, is not enough for the algorithm. When we use the data set as
Figure 5.2: SubsampledBoxes data set frames 1, 5, 10 respectively
it is, the algorithm fails most of the time to differentiate the objects, this is due to fact that complexity of motion is high and we cannot capture the system order from these frames, but when we use either flipped and added or just repeated version of the data set, the algorithm can show its potential by successfully segmenting points. Even the repetition is a kind of artificial way of increasing the number of frames the results are satisfactory, so whenever someone decided the data set is not enough by itself, might use such an approach.

Again from our extensive number of experiments, selection of the constraints to solve related optimization problem is important in segmentation, the selection both affects the segmentation performance and the time to complete the segmentation. Also this flexible way of solving the motion segmentation problem in terms of system theoretical approach gives us the possibility of using our best knowledge about the problem at the hand, by some clever thinking the most of the physical constraints can be added to the algorithm to make its performance better and better.

For this data set we used the following constraint since we can always use such a constraint because our depth estimation is up to a constant (This is because of unknown focal length of the camera comes as a multiplier to our constructed hankel matrices):

$$\text{mean}(Z_1 + Z_2 + Z_3 + Z_4) = 4$$ \hspace{1cm} (5.1)

and we selected $c = 0.25$

and to estimate the rank we first normalized w.r.t the sum of singular values and count the number of singular values higher than 0.9 (9 percent)

Here are some results associated with SubsampledBoxes data set;
\[ R = [1, 1, 1, 1, 2, 2, 2, 2] \] (5.2)

here R is an array showing the segmentation of the points. 1 represents class 1 and 2 represents class 2. As it is clear from the array the algorithm classified the points perfectly, under the aforementioned parameters.

The algorithm resulted rank 5 for the points from the same object and rank 7 for the points from the different objects.

Since there is some randomness associated with the algorithm it is not always guaranteed that it will correctly classify the same problem in the same manner, but the probability of correct classification is high. The algorithm sometimes fails this is due to fact that randomly selected points might be in error or does not contain enough information about the system for example quasi-static points can be the example of such points.
Chapter 6

Conclusion

In this work we developed a new motion segmentation algorithm working under perspective projection case. Our algorithm depends on the idea that the complexity of the model required to explain the objects’ joint motion. To do this we used tools from systems and optimization theory, and since the algorithm does not depend on some intuitive idea, it has some nice mathematical properties. Again because of its mathematical basis, it is really flexible and by just changing or adding some constraints to the optimization problems, we can project our best knowledge about the problem to the algorithm.

Constraint selection for the algorithm is really important since it affects both the time to complete and the performance of the algorithm. Unfortunately, the computational complexity of the algorithm is really high and the algorithm is slow, this slowness is due to the repeated optimization problem solving during the process. Since there is some randomness embedded in the algorithm because of RANSAC type sampling, we cannot guarantee the exact segmentation results for the problem that the algorithm worked on multiple times. This randomness can be avoided by searching all of the quadruples instead of RANSAC type sampling but this is a combinatorial problem.
and it increases the complexity tremendously.

Dedicated implementation might solve some of the aforementioned problems, since the optimization problems in the algorithm have some specific structures. While implementing the algorithm, these can be exploited to quicken the process.

To sum up, there are not many algorithms working under perspective projection case in the literature. The problem at the hand is inherently hard, but our algorithm successfully solved the test cases, and we believe that our algorithm might help better algorithms to be developed in the future.
Appendix A

Source Code

A.0.1 Main

```matlab
function R=improved_seg_thesis(Wx,pct);

[Nf,Np]=size(Wx);
Nf=Nf/2;

W=zeros(2*Nf,Np);

sizy=768;  \%size of the image
sizx=1024;  \% size of the image
\%convert to x and y from pixels
W(1:2:end,:)=Wx(1:2:end,:)-sizx/2;
W(2:2:end,:)=Wx(2:2:end,:)-sizy/2;

r_set=[1:Np];
```

58
class = 0;
while ~isempty(r_set)
  class = class + 1;
  num = randint(1, 1, [1 length(r_set)]);
  P1 = W(:, r_set(num));  % fixed point
  R(r_set(num)) = class;
  r_set = setdiff(r_set, r_set(num));
  rank_s = 1000000000;
  for i = 1:50
    nums = randint(1, 3, [1 length(r_set)]);
    while nums(1) == nums(2) || nums(2) == nums(3) || nums(1) == nums(3)
      nums = randint(1, 3, [1 length(r_set)]);  % pick
    end
    PS = W(:, r_set(nums));
    P2 = PS(:, 1);
    P3 = PS(:, 2);
    P4 = PS(:, 3);

    regul = 0.15;
    Nf = length(velocity(P1, 2)) / 2;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% for initialization of zi
    cvx_begin;  % optimization
    clear z1 z2 z3 z4;

    variable z1t(Nf + 1, 1);
    variable z2t(Nf + 1, 1);
    variable z3t(Nf + 1, 1);
    variable z4t(Nf + 1, 1);
z1 = rep2(z1t);
z2 = rep2(z2t);
z3 = rep2(z3t);
z4 = rep2(z4t);

H1 = block_hankel_create(reshape(velocity(P1.*z1,2),2,Nf));
H2 = block_hankel_create(reshape(velocity(P2.*z2,2),2,Nf));
H3 = block_hankel_create(reshape(velocity(P3.*z3,2),2,Nf));
H4 = block_hankel_create(reshape(velocity(P4.*z4,2),2,Nf));

z1t ≥ 1*ones(Nf+1,1);
z2t ≥ 1*ones(Nf+1,1);
z3t ≥ 1*ones(Nf+1,1);
z4t ≥ 1*ones(Nf+1,1);

mean(z1t) == 1;
mean(z2t) == 1;
mean(z3t) == 1;
mean(z4t) == 1;

Hpq = [H2-H1; H3-H1; H4-H1];

variable Y(size(Hpq,1),size(Hpq,1)) symmetric;
variable Z(size(Hpq,2),size(Hpq,2)) symmetric;

[Y Hpq; Hpq' Z] == semidefinite(size(Hpq,1)+size(Hpq,2));
\[
\begin{align*}
F &= [Y \ zeros(size(Hpq,1), size(Hpq,2)); \ zeros(size(Hpq,1), size(Hpq,2))', Z]; \\
\text{minimize} & \left( \text{trace} \left( F \right) \right); \\
\end{align*}
\]

```matlab
% cvx_precision low;
% cvx_quiet(true);
% cvx_end; %end of optimization

%%% Y0=Y;
%%% Z0=Z;

z1t_last=z1t; 
z2t_last=z2t; 
z3t_last=z3t; 
z4t_last=z4t; 

for t=1:5 
    cvx_begin; %optimization-------------
clear z1 z2 z3 z4; 

    variable z1t(Nf+1,1);
    variable z2t(Nf+1,1);
    variable z3t(Nf+1,1);
    variable z4t(Nf+1,1);

    z1=rep2(z1t);
    z2=rep2(z2t);
    z3=rep2(z3t);
    z4=rep2(z4t);
```

```
H1=block_hankel_create(reshape(velocity(P1.*z1,2),2,Nf));
H2=block_hankel_create(reshape(velocity(P2.*z2,2),2,Nf));
H3=block_hankel_create(reshape(velocity(P3.*z3,2),2,Nf));
H4=block_hankel_create(reshape(velocity(P4.*z4,2),2,Nf));

z1t≥0.25*ones(Nf+1,1);
z2t≥0.25*ones(Nf+1,1);
z3t≥0.25*ones(Nf+1,1);
z4t≥0.25*ones(Nf+1,1);

mean(z1t+z2t+z3t+z4t)==4;

Hpq=[H2−H1;H3−H1;H4−H1];

variable Y(size(Hpq,1),size(Hpq,1)) symmetric;
variable Z(size(Hpq,2),size(Hpq,2)) symmetric;
[Y Hpq' Z]==semidefinite(size(Hpq,1)+size(Hpq,2));

M=[Y0 zeros(size(Hpq,1),size(Hpq,2)); zeros(size(Hpq,1),size(Hpq,2))';...
Z0]+regul*eye(size(Hpq,1)+size(Hpq,2));
M=full(inv(M));
F=([Y zeros(size(Hpq,1),size(Hpq,2)); zeros(size(Hpq,1),size(Hpq,2))';...
Z]);
minimize({trace(M* F })};

cvx_precision low;
cvx_quiet(true);
cvx_end; %end of optimization

Y0=full(Y);
Z0=full(Z);
[num r_set(nums)]'
	try
		svd(full(double(Hpq))/sum(svd(full(double(Hpq)))))
	catch
		Hpq=eye(size(Hpq));
		break;
	end
end

[num_im, [num r_set(nums)]']
current_rank=sum((svd(Hpq)/sum(svd(Hpq)))*100)>pct);
if current_rank<rank_s
    rank_s=current_rank;
    nums_s=nums;
    svd_s=(svd(Hpq)/norm(svd(Hpq)));z1_s=z1;
    z2_s=z2;
    z3_s=z3;
    z4_s=z4;
end
i
end %end ransac-ish for
PS=W(:,r_set(nums_s));P2=PS(:,1);P3=PS(:,2);P4=PS(:,3);
R(r_set(nums_s))=class.*ones(1,3);

H1=block_hankel_create(reshape(velocity(P1.*z1_s,2),2,Nf));
H2=block_hankel_create(reshape(velocity(P2.*z2_s,2),2,Nf));
H3=block_hankel_create(reshape(velocity(P3.*z3_s,2),2,Nf));
H4=block_hankel_create(reshape(velocity(P4.*z4_s,2),2,Nf));

r_set=setdiff(r_set,r_set(nums_s));

classified_points=[];
for point=r_set
    regul=0.15;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% for initialization of zi
    cvx_begin;
    clear zi;
    variable zit(Nf+1,1);
    %zi create
    zi=rep2(zit);
    Pi=W(:,point);
    Hi=block_hankel_create(reshape(velocity(Pi.*zi,2),2,Nf));
    Hpq=[H2−H1;H3−H1;H4−H1;Hi−H1];
    zit==1*ones(Nf+1,1);
    variable Y(size(Hpq,1),size(Hpq,1)) symmetric;
    variable Z(size(Hpq,2),size(Hpq,2)) symmetric;
    [Y Hpq;Hpq' Z]==semidefinite(size(Hpq,1)+size(Hpq,2));
    F=[Y zeros(size(Hpq,1),size(Hpq,2)); zeros(size(Hpq,1),size(Hpq,2))' Z];
    minimize({trace(F)});
    cvx_precision low;
cvx_quiet(true);
cvx_end;

Y0=Y;
Z0=Z;

for t=1:5
cik=0;
if cik==1
    break;
end

%---
cvx_begin;
clear zi;
variable zit(Nf+1,1);

%zi create
zi=rep2(zit);
Pi=W(:,point);

Hi=block_hankel_create(reshape(velocity(Pi.*zi,2),2,Nf));
Hpq=[H2-H1;H3-H1;H4-H1;Hi-H1];

zit>=0.25*ones(Nf+1,1);

variable Y(size(Hpq,1),size(Hpq,1)) symmetric;
variable Z(size(Hpq,2),size(Hpq,2)) symmetric;

[Y Hpq;Hpq' Z]==semidefinite(size(Hpq,1)+size(Hpq,2));
M=[Y0 zeros(size(Hpq,1),size(Hpq,2)); zeros(size(Hpq,1),size(Hpq,2))']
Z0=regul*eye(size(Hpq,1)+size(Hpq,2));
M=full(inv(M));
F=[Y zeros(size(Hpq,1),size(Hpq,2)); zeros(size(Hpq,1),size(Hpq,2))' Z];
minimize((trace(M*F')));
cvx_precision low;
cvx_quiet(true);
cvx_end;

Y0=full(Y);
Z0=full(Z);
svd(full(double(Hpq))/sum(svd(full(double(Hpq)))))
zit_last=zit;
end

current_rank=sum((svd(Hpq)/sum(svd(Hpq))*100)>pct);
tol=1;
if abs(rank_s-current_rank)<=tol
R(point)=class;
classified_points=[classified_points, point];
end
end
end
r_set=setdiff(r_set,classified_points);
end %big while

A.0.2 Block Hankel Create
function Hankel_Data_Matrix = block_hankel_create(Y);
%create block hankel
m = round(length(Y)/2);
%block hankel matrix
first_element = 1;
last_element = m; % just to make the block hankel matrix squarish
while last_element <= length(Y)
    Hankel_Data(first_element).row = Y(:, [first_element:last_element]);
    first_element = first_element + 1;
    last_element = last_element + 1;
end
Hankel_Data_Matrix = [];
for i = 1:length(Hankel_Data)
    Hankel_Data_Matrix = [Hankel_Data_Matrix; Hankel_Data(i).row];
end

A.0.3 Velocity

function dP = velocity(P, n);
dP = P(n+1:end,:) - P(1:end-n,:);

A.0.4 Rep2

function z2 = rep2(z);
Nf = size(z, 1);
z2(1:2:2*Nf,:) = z;
z2(2:2:2*Nf,:) = z;
Bibliography


