A Fast Centroiding Algorithm for Use in Data Analysis of Ultrafast Pulsed Laser Measurements

A THESIS PRESENTED

by

Rachel A. Sampson

to

Stony Brook University's Honors College

in Partial Fulfillment of the Requirements

for the Degree of

Bachelors of Science

in

Physics

Presented December 16, 2015

ABSTRACT

A fast, accurate, and simple centroiding algorithm was developed for data analysis of charged particles detected in a velocity map imaging apparatus and was implemented in both LabVIEW and Matlab. The new algorithm was on average an order of magnitude faster than the previous algorithm used by the group, allowing the average analysis time per frame to be reduced to approximately 1.3 ms in Matlab and 0.5 ms in LabVIEW.

This increase in speed allows for real-time data analysis in LabVIEW and quasi-real-time analysis in Matlab. The fidelity of the new algorithm was found to be comparable to that of the old algorithm. An exploration of the effect of a number of factors on the fidelity and speed of the algorithm was conducted.

The algorithm was tested on data sets taken with two different cameras. One of the cameras gave intensity information for each pixel, while the other camera gave timing information for each pixel. The pixel timing capability is a new and exciting development in the field of ultrafast physics because it allows for unambiguous identification of the fragment source of each hit in the data. This allows for simultaneous imaging of both electrons and ions with a single camera.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Thomas Weinacht. Tom gave me the freedom to explore my interests, but the direction to make my musings productive. His enthusiasm, encouragement, and support never failed to fuel my passion for physics and research and I hope to find an advisor like him in graduate school.

I would also like to express my gratitude to Harold Metcalf and Marty Cohen for supporting me as I started my career in physics research, and for continuing to guide and mentor me throughout my undergraduate career. Their kindness and devotion leave me in awe and I hope to emulate them as I forge my path in the world.

I have all the members of the Weinacht group (Péter Sándor, Arthur Zhao, Spencer Horton, Vincent Tagliamonti, Yusong Liu, and Gayle Geschwind) and the other AMO groups to thank for the good company and good memories. I have really enjoyed my time at Stony Brook and am grateful to have known so many amazing physicists, and people.

Lastly, this thesis wouldn't have been possible without the help of Péter Sándor and Arthur Zhao. I am very thankful for their assistance with coding, never-ending patience, and willingness to sit down with me and explore problems. The TimePix camera used to collect some of the data analyzed in this thesis was provided by Andrei Nomerotski and Merlin Fisher-Levine, who also offered useful discussion and suggestions on the algorithm.

TABLE OF CONTENTS

Page

1 Introduction 1

2 Data Acquisition 5

2.1 Experimental Set-Up 5

2.2 Cameras 7

2.2.1 Basler 7

2.2.2 TimePix 8

3 Centroiding Algorithm 9

3.1 Threshold Image 10

3.2 Prefilter Image 11

3.3 Identifying Hits 13

3.4 Locate Centroid 14

3.5 Timing Information 16

4 Results 17

4.1 Speed 17

4.2 Fidelity 21

5 Conclusion 22

Appendices 25

A Matlab Basler Code 26

B Matlab TimePix Code 30

C LabVIEW Basler Code 36

LIST OF FIGURES

Figure Page

1.1 Timescales at which dynamics appear. 2

1.2 Raw data from ultrafast measurement. 3

2.1 Experimental set-up. [9] 6

3.1 Pseudocode of algorithm. 9

3.2 Graphic of process of thresholding. 10

3.3 Effect of thresholding. 10

3.4 A 4-connected and 8-connected neighborhood. 11

3.5 Illustration of prefiltering process. 12

3.6 Data before and after prefiltering. 13

3.7 Flood-fill algorithm. 14

3.8 Results of object detection. 15

3.9 Raw data and results of centroiding. 16

4.1 Analysis time versus number of hits. 19

4.2 Histogram of analysis times. 20

LIST OF TABLES

Table | Page

4.1 Average image analysis time. 18

LIST OF APPENDIX FIGURES

Figure Page

C.1 Front panel. 37

C.2 Block diagram. 37

Chapter 1: Introduction

Ultrafast physics is the science of measuring processes with picosecond to attosecond resolution. Pico- is 10^{-12}, while femto- is 10^{-15}, and atto- is 10^{-18}. To get a sense of this scale, if you take a second to be the distance light travels in that time, a second would correlate to approximately three-quarters of the distance from the Earth to the Moon [1]. On the same scale, a picosecond (ps) would only be the thickness of a business card [1]. A 10 femtosecond (fs) pulse would be smaller yet, at slightly less than the diameter of a hair and the shortest pulse ever created (67 attoseconds (as)) would be between the size of a virus and DNA [1, 2, 3].

Measuring events at these timescales is very useful because in order to resolve motion, or dynamics, you need a measurement tool with time resolution shorter than the timescale of the motion and many atomic and molecular dynamics occur in the ultrafast regime. For example, molecular rotations occur within picosecond timescales; molecular vibrations happen in the femtosecond regime; and at the attosecond regime, electron wavepacket dynamics can be explored [Fig. 1.1].

While the concept that an event must be measured by something smaller than itself to be accurate might seem abstract. It is easy to understand if you imagine trying to measure the size of a penny with a measuring tape that only has marks every foot. It'd be pretty difficult to get an accurate measurement for the penny. While the example given deals with measuring length, the same is true when

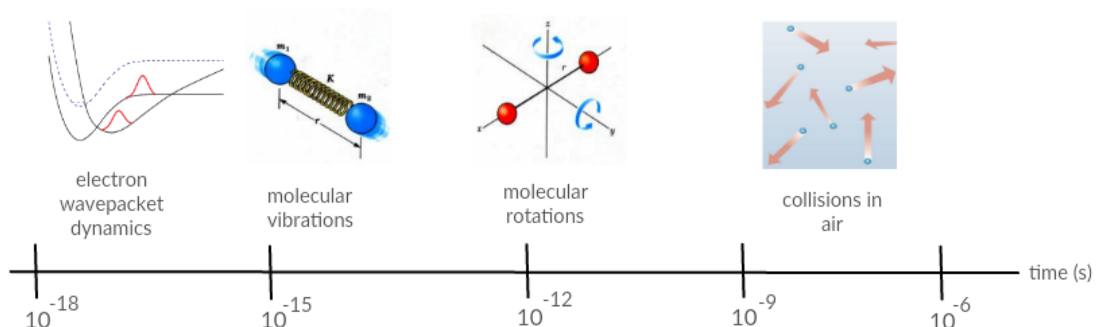

Figure 1.1: Timescales at which dynamics appear.

measuring events in time. This phenomenon can be observed when taking a picture of a squirming child or someone running. The picture comes out blurred because the event being photographed is faster than the camera's shutter, so the image's resolution is poor.

In our lab, we focus on studying molecular dynamics, such as ionization, and the objects we are imaging are charged particles. A detailed description of the experimental set-up can be found in Chapter 2. The basic premise however is that fragments produced by the ionization of a molecule hit a screen causing the screen to fluoresce, or light up. We then take images of this screen using a camera. An example data image is shown in Fig. 1.2.

In Fig. 1.2, the bright spots in the image represent where a charged particle hit the phosphor screen. In order to analyze the molecular dynamics, a large number of hits must be analyzed. This translates into a single data set containing information from thousands of images.

To explore the dynamics, we must identify where hits fell. The naïve way

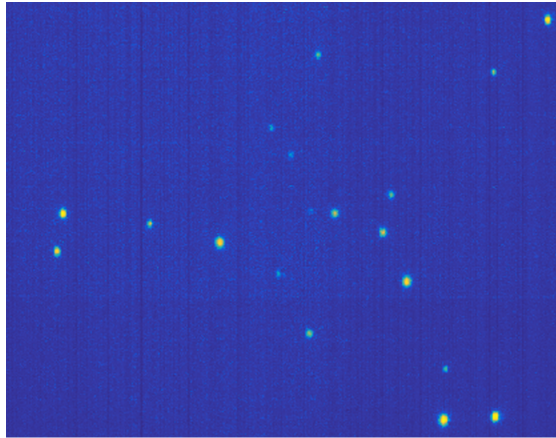

Figure 1.2: Raw data from ultrafast measurement.

of accomplishing this would be to add all the images together and analyze the composite image. Doing so though, does not allow for coincidence measurements and also has problems associated with stray light and camera dark noise. To eliminate the effect of noise and detector efficiency on the data, the centroid of each hit was identified and an image of all the data was compiled by placing a Gaussian at the centroid locations.

Finding the centroid of an object in an image is a fairly common problem in science and there is an abundance of centroiding codes and information on the centroiding problem available [4, 5, 6]. Many of these codes though either run very slowly or are very complicated in nature and rely on large function library databases. In this thesis, we discuss the development of an algorithm which works fast enough for our purposes, while still providing us the precision required.

Chapter 2 contains a brief discussion of the experimental set-up and the two cameras used to collect data. One camera measures light intensity, while the second

gives timing information for every pixel. In chapter 3, a detailed explanation of the algorithm is given. A quantitative analysis of the speed and fidelity of the algorithm is conducted in chapter 4. Lastly, in chapter 5, the results and the significance of the results are discussed.

Chapter 2: Data Acquisition

2.1 Experimental Set-Up

Our experimental set-up consisted of two main components: the laser system and the vacuum chamber.

The laser system used was an amplified Ti:Sapphire system. The system produces 780 nm laser pulses with 1 mJ pulse energy and a minimum duration of 30 fs at a 1kHz repetition rate [10]. Using filament based spectral broadening, pulse duration can be reduced to sub-10 fs with our current minimum pulse duration being 9 fs.

The Vacuum chamber portion of our set-up consists of a traditional time-of-flight (TOF) tube and a velocity map imaging (VMI) apparatus. A molecular beam is sent into the vacuum chamber where the laser beam will pass through the molecular beam, ionizing the molecules. The charged particles created in this process will then be accelerated towards a stack of microchannel plates by exposing them to an electric potential. The time-of-flight tube acts to separate out the times at which the molecules arrive at the MCPs, while the velocity map imaging apparatus acts to make the cause particles of different kinetic energies to arrive at different radii on the MCPs.

The MCPs are a dual-stack chevron MCPs. When the accelerated charged

particles hit the MCP, they produce a shower of electrons. As the electrons travel through the MCP, they will hit the walls many times creating many more showers. By the end of the MCPs, a shower of $\sim 10^6$ electrons has been created. The electrons are then incident on a phosphor screen which fluoresces and an image of the screen is taken using a camera. A discussion of the two cameras used to collect the data is contained in the next section. A more detailed description of the experimental set-up is given in Ref. [7] and Ref. [8].

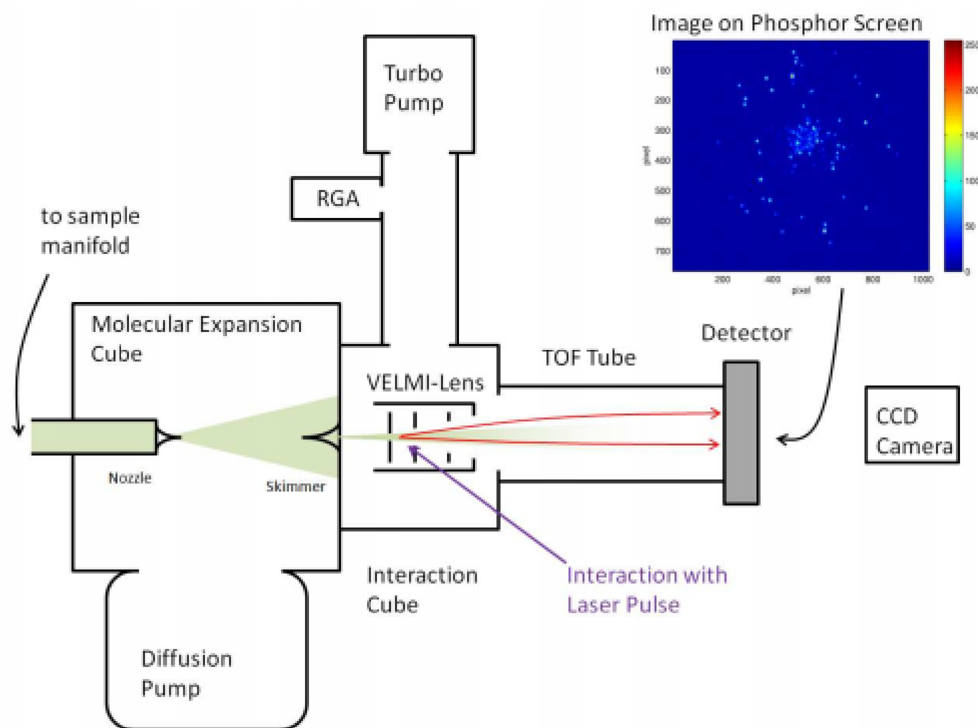

Figure 2.1: Experimental set-up. [9]

2.2 Cameras

The algorithm was tested on data taken from a Basler acA2000-340km and a TimePix camera. The specifications of and differences between the two are discussed below.

2.2.1 Basler

The Basler camera measures light intensity and outputs monochromatic images that are 2048 x 1088 pixels in size [11]. For the purposes of our data acquisition, the camera was used in a mode where only a 360 x 360 pixel portion of the frame was analyzed. This mode was used because it allowed for a 1kHz transfer rate from the camera to the computer. This allows us to transfer the data in real-time.

Using the Basler camera, only ions or electrons can be imaged at one time, not both. This is because the camera only provides intensity information making it impossible to distinguish electrons from ions. In order to image both electrons and ions for coincidence measurements, a fast high voltage switch is used. Initially, a positive electric potential is applied to the ionization fragments causing the produced electrons to accelerate towards the detector. Once the electrons have been detected, a negative potential is applied causing the produced ion to accelerate towards the detector, where its fragment type is identified by its time of flight.

2.2.2 TimePix

The TimePix camera measures timing information for each pixel with 10 ns resolution [12]. The camera output data in the form of a text file, which contained three columns with information on every above-threshold value pixel. The first column contained the x-position of the pixel, the second, the y-position, and the third, the timing information.

In this data set, the intensity information is replaced by the timing information. The timing information is given in reverse number form meaning the largest number corresponds to the light arriving at the camera the fastest and the smallest number corresponds to light arriving at the camera at the latest time. The time stamp can be converted to seconds using a conversion factor.

The usage of the TimePix camera is significant in the field of ultrafast physics as it represents one of the first times a camera with nanosecond timing capabilities is being used in ultrafast. This capability allows us to unambiguously identify hits with the type of fragment they came from. This is useful for coincidence measurements.

Chapter 3: Centroiding Algorithm

In this chapter, a basic outline of the centroiding algorithm is discussed. The full Matlab and LabVIEW code can be found in Appendix A and B respectively.

The input of this code is a grayscale image for the Basler data and a list of x- and y-coordinates plus a time stamp for each pixel light was incident on for the TimePix data. The output of the Basler data analysis is a list of x- and y-coordinates for the centroid of each hit and for the TimePix camera, it is a list of centroid coordinates and maximum and average timing for each hit.

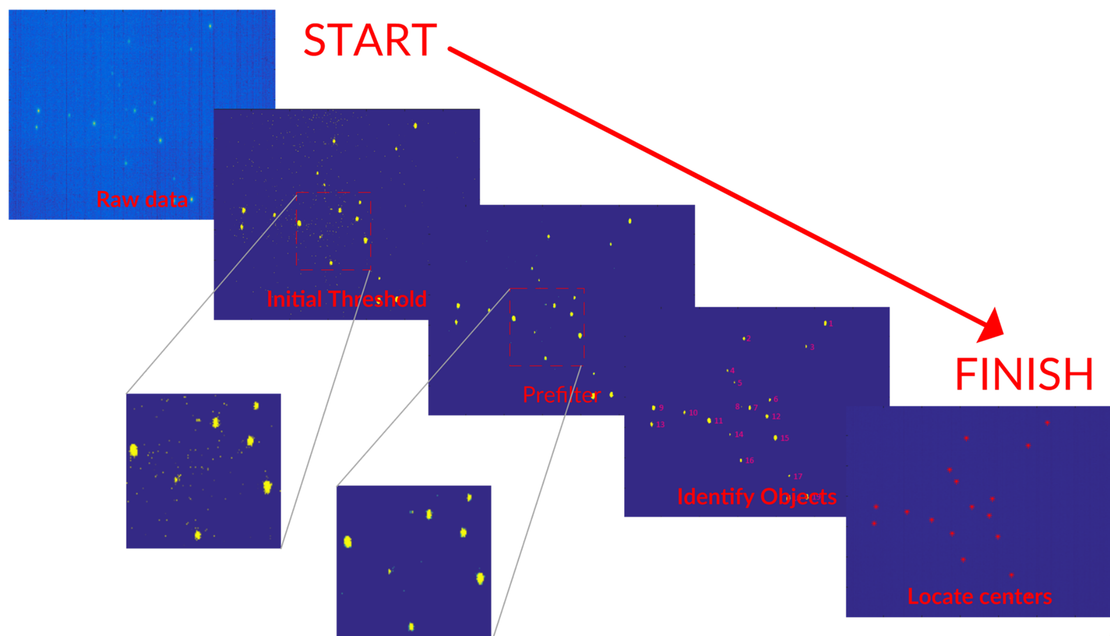

Figure 3.1: Pseudocode of algorithm.

3.1 Threshold Image

The first step of the algorithm, after reading in the raw data, for the Basler data is to threshold the image. This step is skipped for the TimePix data. Thresholding acts to remove low-intensity noise. To threshold an image means to set all values above a certain number, or threshold, equal to one and to set all values below the number equal to zero as shown in Fig. 3.2.

Since a majority of the background noise is low intensity relative to the signal, thresholding the image removes a lot of the background noise from an image. This effect is visible in Fig. 3.3.

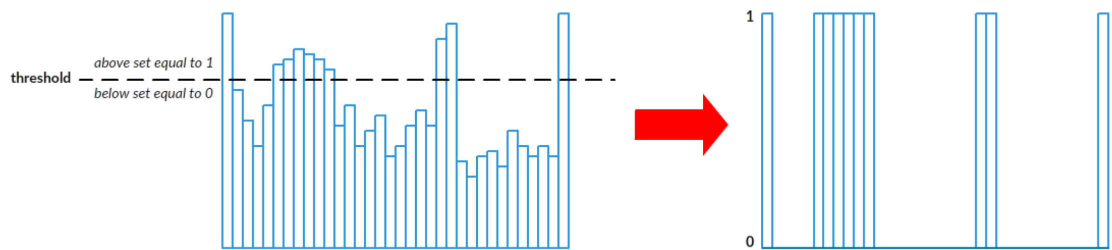

Figure 3.2: Graphic of process of thresholding.

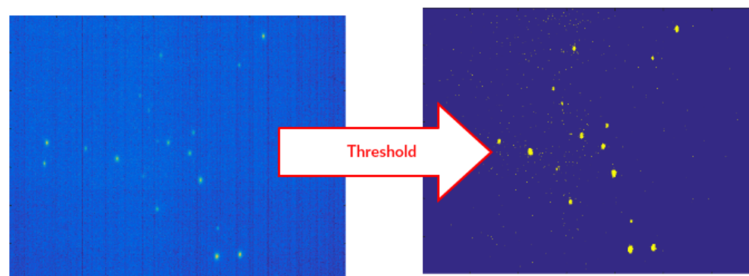

Figure 3.3: Effect of thresholding.

Looking at the raw data, we see that the background of the image is very noisy and the illumination of the background seems to be fairly uniform. After the thresholding though, the background illumination has been eliminated and there are only real hits and very small patches of background noise, all of which are of higher intensity.

3.2 Prefilter Image

After thresholding, there still exists high-intensity noise. The high-intensity noise tends to be small in size, generally only occupying a single pixel. To remove this high-intensity single pixel noise a prefilter is used. The prefilter acts to remove pixels which have no neighbors that are also greater than the threshold. This step removes single pixel high intensity noise.

Neighbors can either be defined as a 4- or 8-connected neighborhood as shown in Fig.3.4. In our data analysis, the 4-connected neighborhood was used.

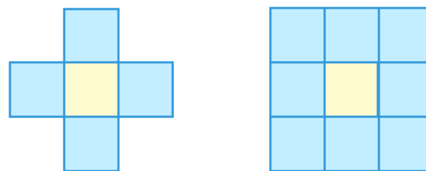

Figure 3.4: A 4-connected and 8-connected neighborhood.

Prefiltering was accomplished by multiplying the thresholded image by shifted copies of itself. A copy of the image in which every pixel was shifted to the left by a single pixel and a copy of the image where every pixel was shifted up by one pixel

was created. These shifted images were then added to the original image to form a composite image. For a pixel to have a value greater than one in the composite image means that one of its neighbors must also be above one. The composite image was then multiplied by the original image.

In Fig. 3.5, the three images on the left represent the original, shifted upwards, and shifted left image. The image on the right represents the composite image, formed by adding the three images on the left and then multiplying by the original image. The white pixels represent pixels that would have a value of one and the yellow pixels denote pixels that would have a value greater than one. The composite image was then thresholded, so only values above one were retained. This step removed single-pixel high-intensity noise by removing pixels that did not have above-threshold neighbors either upwards or to the left.

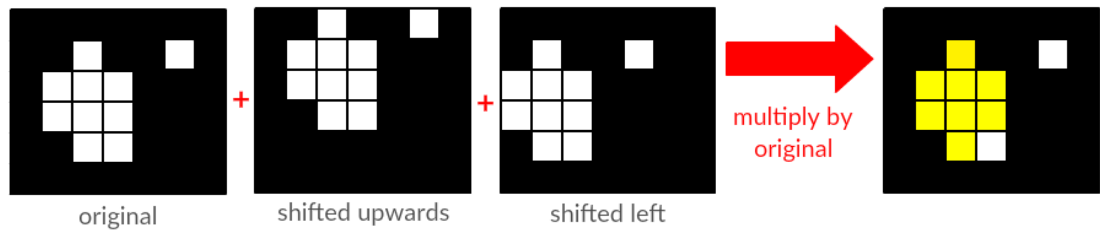

Figure 3.5: Illustration of prefiltering process.

Fig. 3.6 illustrates what a data set would look like before and after prefiltering. It's clear that after prefiltering the image, single pixel noise has been reduced, while retaining the hits that we would like to analyze.

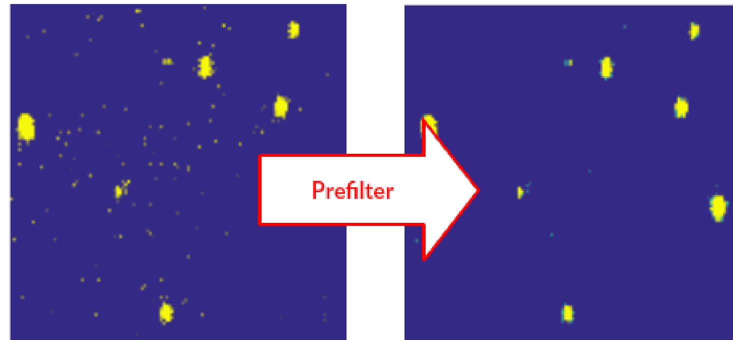

Figure 3.6: Data before and after prefiltering.

3.3 Identifying Hits

At this point, most of the noise has been removed. However, no analysis has been done involving hits in the image. In order to identify the centroid of the hits, the objects in the image must first be identified. Both the Matlab and LabVIEW code make use of built in functions for this step. In Matlab, the function bwconncomp is used; in LabVIEW, the function Count Objects 2 is used.

The Matlab function makes use of a flood-fill algorithm. Flood-fill algorithms are used in Minesweeper and for the bucket tool in many Paint programs [13].

In a flood-fill algorithm, the input image has two colors, a target color and another color. In our case, the target 'color' will be one, while the other color will be zero. The algorithm starts at an initial pixel with the target color. The code then replaces the target color with a new color, say the value 2. Once this is accomplished, the pixel's neighbors will be checked until another target color pixel is found. If a target color pixel is found, the pixel color will be replaced with the

new color. The pixel will then be taken as the new starting point and the process will continue from there. If no target color pixel is found the the object will be considered complete. The rest of the image will then be scanned until another target color pixel is found.

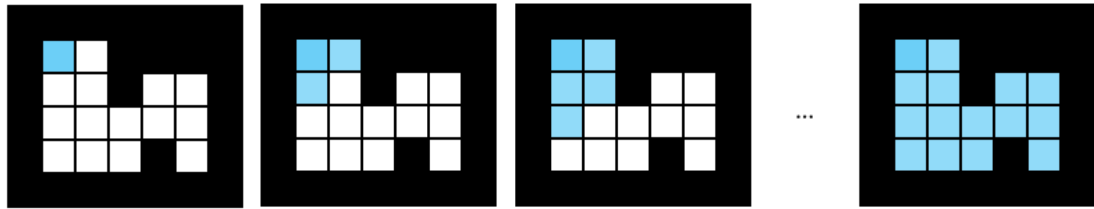

Figure 3.7: Flood-fill algorithm.

When another target color pixel is found that is not the neighbor of one of the pixels in the earlier object, this pixel will be considered a new object and the replacement value assigned to this pixel will be different than the value assigned to the first object. For example, since the first object was assigned 2 as the replacement value, the second object will have a replacement value of 3.

At this step in the process, any objects that are below a specified size are also 'thrown away'. These small objects generally represent noise, rather than a hit. The results of this step are shown in Fig. 3.8.

3.4 Locate Centroid

Next, the algorithm finds the centroid of the object. In order to find the centroid of the object, a center of mass calculation is used. In LabVIEW, the built-in function

Figure 3.8: Results of object detection.

Count Objects 2 is used.

To find the center of mass, a weighted average is taken of each of the pixels within the object. The weighted average is found by multiplying the location of the pixel by its intensity and then dividing by the number of pixels in that dimension for the object. The general formula for the weighted average is given in Eq. 3.1, where x is the position of the pixel and w is the pixel value.

$$\bar{x} = \frac{\sum_{n=1}^{i} x_i w_i}{\sum_{n=1}^{i} w_i} \tag{3.1}$$

The results of this step, a list of x- and y-central coordinates for each hit, will be output by the code. Fig. 3.9 demonstrates the product of this step. All further data analysis is done using these coordinates, a noise free and memory-compact representation of the data.

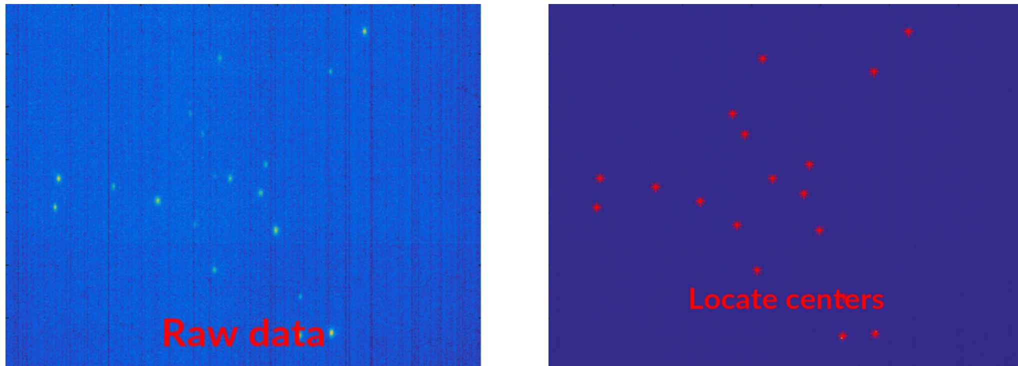

Figure 3.9: Raw data and results of centroiding.

3.5 Timing Information

With the TimePix camera, each pixel measures time, rather than intensity. Therefore the analysis process for the TimePix camera is slightly different. The initial thresholding for the TimePix camera is done by the camera itself and is not included in the data analysis. The camera will only register a hit if it is above a certain threshold, or brightness. If this threshold is not surpassed, the camera will not record the hit.

For each image, the average and maximum time value for each hit is also calculated. This data can be used to identify which fragment the hit came from and characterize the time resolution of the camera.

Chapter 4: Results

The goal of this project was to create a fast and accurate centroiding algorithm for real-time data analysis. The laser used had a repetition rate of 1kHz repetition rate, meaning 1,000 frames of new data were created every second. In order to analyze this data in real-time, or on the fly, the analysis code must also run at a 1kHz repetition rate, or analyze a single data frame in 1 ms.

This analysis should be as accurate as possible at identifying hits and their centroids, so that the data is not skewed during the analysis. A study of the speed and fidelity, accuracy, of the code were conducted and are discussed in the following sections.

4.1 Speed

The goal of this project was to analyze the images at the same speed at which they were collected, so the data could be analyzed in real-time. This goal was achieved with the LabVIEW code, came close to being achieved with the Matlab code which analyzes Basler data, and is still a factor of three off for the Matlab TmePix data analysis. A table of the speeds of each of the algorithms is shown in Table 4.1.

The timing for the algorithms in Matlab was found using the built-in tic and

	Old (ms)	New (ms)	Additional Analysis (ms)
Basler Matlab	5.8	1.3	———
Basler LabVIEW	11.2	0.5	———
TimePix Matlab	———	3.0	0.3 ms

Table 4.1: Average image analysis time.

toc functions, while in LabVIEW, the image analysis timing was found using the VI Profiler. The VI Profiler was used in LabVIEW, rather than the more common Tick Count, because it has μs resolution, whereas Tick Count only has ms time resolution. The times reported included only the time it took to analyze the image, go through all the steps described in Chapter 3, and did not include the time it took to read in the image or write out the list of centroids. For the TimePix data, the time it took to find the maximum time value for each hit was recorded as additional analysis time. The Basler images were 360 x 360 pixels in size, while the TimePix data corresponded to images 256 x 256 pixels in size.

It can be seen from Table 4.1, the new Matlab Basler algorithm is roughly five times faster than the old algorithm and the new LabVIEW Basler algorithm is approximately 20 times faster. This is a sizable improvement.

The Basler Matlab code can analyze approximately 770 frames per second. This is less than the 1000 frames produced per second, but is fast enough for certain applications where images can be presorted prior to analysis based on complementary information allowing us to only analyze a fraction of the 1000 frames per second.

The TimePix analysis takes about a factor of three longer than the 1 ms we

were aiming for. Most of the additional analysis time was added during the object identification portion of the code. The timing for this section could potentially be further reduced by reducing the range of time values which are analyzed. This would remove a good number of noise pixels.

The effect of the number of hits on image analysis time was explored quantitatively using 2040 x 1088 images taken with the Basler camera and analyzed using Matlab. The results are shown in Fig. 4.1. Looking at Fig. 4.1, it seems that there may be a correlation between number of hits and analysis time, but it is not very strong. This suggests analysis time has a stronger dependence on the conditions under which the image was taken than the number of hits. Characterizing analysis time per image as a function of number of hits is important because it informs our choice of molecular density within the molecular beam.

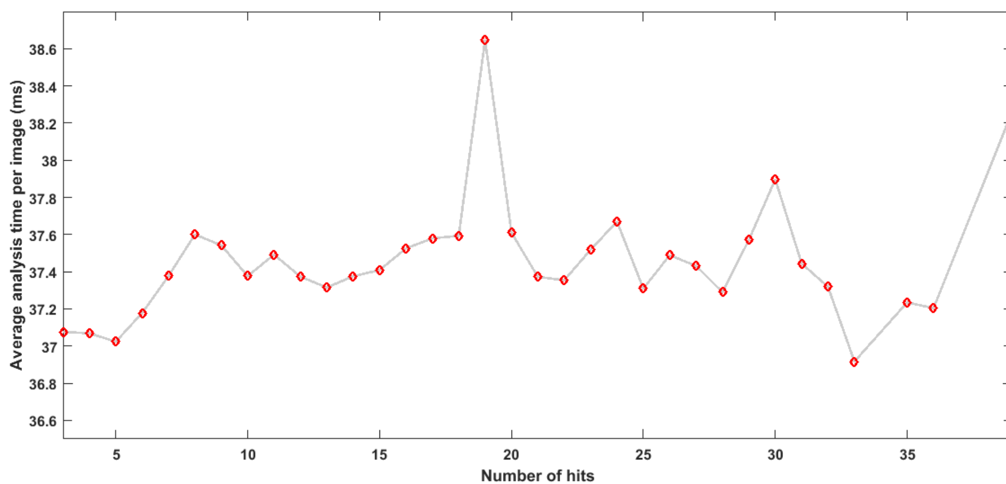

Figure 4.1: Analysis time versus number of hits.

A histogram of the analysis times is shown in Fig. 4.2. The analysis times are generally clustered around some average. It is interesting to note though that the two outliers correspond to the analysis time for the first and second image. On average, the analysis time for the first image was found to be about an order of magnitude larger. This may be due to the fact that Matlab compiles after the program is run. This effect was not observed in LabVIEW, a pre-compiled language, which further supports this hypothesis. This result suggests that it may be wise to feed the Matlab algorithm two noise-free dummy images before analyzing the real data to reduce image analysis times.

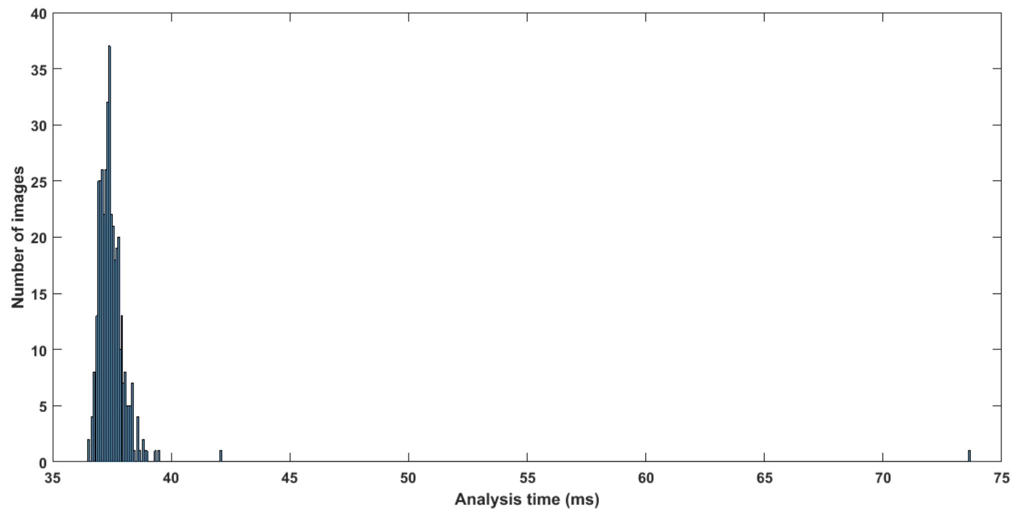

Figure 4.2: Histogram of analysis times.

4.2 Fidelity

Developing an algorithm with high fidelity was essential. It doesn't do much good to write a fast centroiding algorithm, if it is incapable of accurately identifying the centroids of objects in the image. In this section, the fidelity of the algorithm is discussed and characterized.

In order to characterize the fidelity of the algorithm, the number of hits found in an image with the algorithm was compared to the number found manually. Out of 20 Basler images, the Matlab and LabVIEW code both correctly identified the number of hits in 19 of the images. The image with the misidentified hits was different for the two codes suggesting the algorithm used by the built-in object identification functions is different in the two languages.

It was also found that the fidelity of the code was dependent on both the initial threshold value and minimum object size chosen. For both, the fidelity of the code dropped more dramatically if s value below the maximum fidelity value was used than if a value above the maximum fidelity value was chosen. Changing the threshold by as little as 2 from the maximum fidelity threshold was found to change the number of hits identified by 5% if the value was 2 above and 24% if the value was 2 below. The number of hits identified was found to change by 7% if the minimum object size was increased by 1 and 30% if it was lowered by one. This result underscores the importance of choosing an appropriate threshold and minimum object size.

Chapter 5: Conclusion

In this thesis, a fast, accurate, and simple centroiding code for use in data analysis of ultrafast pulsed laser measurements is described. The algorithm was implemented in both Matlab and LabVIEW and was found to be about an order of magnitude faster than the previously used algorithm allowing for real-time data analysis in LabVIEW and quasi-real time data analysis in Matlab.

Overall, the project has been a successful in creating a fast and simple centroiding algorithm. The 1 kHZ analysis rate was reached with the LabVIEW code and is not far off for the Basler Matlab code. Plans are also being made to implement the code in the lab and integrate the algorithm with our data acquisition code.

Future work will focus on further improving the speed of the Matlab Basler and TimePix code. Efforts are currently being directed towards writing a code, which can make use of parallel computing, or in simple terms can analyze more than one image at a time.

Bibliography

[1] A. W. Weiner, *Ultrafast Optics* (John Wiley & Sons, Inc., Hoboken, NJ, 2009).

[2] UW MRSEC Education Group, *What is the Nanoscale?* Website. (2015).

[3] K. Zhao *et al.*, Opt. Lett **37**, 18 (2012).

[4] J. G. Allen, R. Y. D. Xu, and J. S. Jin, *Pan-Sydney Area Workshop on Visual Information Processing* VIP2003[C], (2004).

[5] A. Opelt, A. Pinz, and A. Zisserman. *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference* **1**, pp.3-10, (2006).

[6] L. A. Alexandre, *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems*, (2006).

[7] C. A. Trallero, Ph.D. thesis, Stony Brook University, 2007.

[8] P. Nürnberger. Masters thesis, Stony Brook University, 2003.

[9] D. Geißler, Ph.D. thesis, Stony Brook University, 2013.

[10] A. Zhao, P. Sándor, T. Rozgonyi, and T. C. Weinacht, Phys. Rev. B **47**, 20 (2014).

[11] *Basler acA2000-340km*. BD000581. Rev 6. Basler AG (2015).

[12] C.S. Slater *et al.*, Phys. Rev. A **89**, 011401(R) (2014).

[13] S. Vaingast, *Beginning Python Visualization: Crafting Visual Transformation Scripts* (Apress, New York, 2009).

APPENDICES

Appendix A: Matlab Basler Code

```
% This program identifies the objects in an image and then finds the
% centroid of each object. The output of this program is a list of object
% centroids.
%
% Adjustable parameters: thrs - value at which initial image is
%                        thresholded
%                        min_size - minimum number of pixels in an object
%                        x_dim - x dimension of image
%                        y_dim - y dimension of image
%
% Written 11/20/2015 by Rachel Sampson


clc;%Clear command window
clear all;%Clear workspace
close all;%Close figures

%==================Create list of files to be analyzed=====================

fileList=dir('M:\2015_02_05_R\test\pic*');%Creates list of all files in the directory
numfiles=numel(fileList);%Number of files in the directory

%========================Adjustable parameters==========================

thrs=37;%Initial threshold
min_size=3;%Minimum number of pixels in an object
x_dim=1088;%X dimension of image
y_dim=2040;%Y dimension of image

%=============================Preallocate variables====================
info=[];%List of centroids
num_hits=zeros(numfiles,1);%Number of objects indentified
timing=zeros(numfiles,1);%Analysis time for image


%Preallocate variables for prefilter
add_vert=zeros(1,y_dim);
add_horz=zeros(x_dim,1);

for r_index=1:numfiles;

    tic;%Start timing

    %==============Read in image=================
    pic = rjpg8c_h(fileList(r_index).name); %Reads in jpg image file
    pic = double(pic(:,:,1));%Creates monochromatic image

    %===========Threshold=====================
    %Removes low-intensity noise

    hit_matrix = gt(pic,thrs);%Threshold image

    %============Prefilter=====================
    %Removes high-intensity single pixel noise
```

```
    up = [hit_matrix(2:end,:); add_vert];%Shift image upwards by one pixel
    left = [hit_matrix(:,2:end), add_horz];%Shift image left by one pixel

    %Adds shifted images and mutiplies by original image
    real_hits = (up+left+hit_matrix).*hit_matrix;

    %Removes pixels that did not have an above-threshold neighbor to the
    %left or up
    BW = gt(real_hits,0.9);%Threshold image

    %==========Finds hits=======================

    CC = bwconncomp(BW);%Finds objects using flood-fill algorithm

    %================Find centroid===============

    %Initialize variables
    row=cell(1,CC.NumObjects);%Pixel x-position
    column=cell(1,CC.NumObjects);%Pixel y-position
    centroid=cell(1,CC.NumObjects);%Centroid of object

    for h_index = 1:CC.NumObjects;

        %Removes hits below the minimum object size
        if size(CC.PixelIdxList{1,h_index},1) < min_size;
            CC.PixelIdxList{1,h_index} = [];
        end;

        %Finds centroid of each object
        if isempty(CC.PixelIdxList{1,h_index}) ~= 1;
            %Converts linear indices to subscripts
            [row, column]=ind2sub([x_dim,y_dim],CC.PixelIdxList{1,h_index});
            %Finds x-position of centroid
            centroid_x=sum(row.*pic(CC.PixelIdxList{1,h_index})./sum(pic(CC.PixelIdxList{1,h_
index})));
            %Finds y-position of centroid
            centroid_y=sum(column.*pic(CC.PixelIdxList{1,h_index})./sum(pic(CC.PixelIdxList{1
,h_index})));
            %Saves centroid of object
            centroid{1,h_index} = [centroid_x centroid_y];
        end;
    end;

    %===================Makes list of centroids==================

    %Preallocate variables
    rows = size(centroid,1);%Number of objects in image
    cols = 2;%Number of values in centroid
    centroids = cell(rows,1);%Cell matrix of centroid positions

    %Concatenate first dimension
    for n=1:rows
        centroids{n} = cat(2,centroid{n,:});
    end
    %Concatenate single column of cells into a matrix
    centroids = cat(1,centroids{:});
```

```
    %Reshapes matrix
    value=size(centroids,2)/2;
    centroids=reshape(centroids,2,value);

    %Calculates number of hits in image
    num_hits(r_index)= size(centroids,2);

    info = [info; centroids'];%Centroid location for each object

    timing(r_index)=toc;%End timing

end;

%Save list of centroids as text file
save('2_5_test_analyzed.txt','num_hits','-ascii')
```

Published with MATLAB® R2015b

Appendix B: Matlab TimePix Code

```
% This program reads in all of the files from a directory and creates a
% false image from the data read in.  The program then identifies all the
% objects in the image larger than a certain area and finds the object's
% centroid and the maximum time value.
%
% Adjustable parameters: min_size - minimum number of pixels in an object
%                        x_dim - x dimension of image
%                        y_dim - y dimension of image
%
% Written 11/29/2015 by Rachel Sampson


clc;%Clear command window
close all;%Close figures

%===================Create list of files to be analyzed=====================

fileList=dir('M:\2015_04_24_R\Run23');%Creates list of all files in the directory
fileList=fileList(~[fileList.isdir]);%Removes directories
numfiles=numel(fileList);%Number of files in the directory

%=====================Load dead pixel mask==================================

load('dead_pixel_mask.mat')%Load dead pixel mask

%=========================Adjustable parameters==========================

min_size=3;%Minimum number of pixels in an object
x_dim=256;%X dimension of image
y_dim=x_dim;%Y dimension of image

%=============================Preallocate variables====================

pic = zeros(256);%Creates false image

info=[];%List of centroids and max time
num_hits=zeros(numfiles,1);%Number of objects indentified
timing_read=zeros(numfiles,1);%Time to read in the txt file and create false
                              %image for each file
timing=zeros(numfiles,1);%Analysis time for image


%Preallocate variables for prefilter
add_vert=zeros(1,y_dim);
add_horz=zeros(x_dim,1);

for r_index=1:numfiles

                                                                     ; %Read in image

    tic;%Starts timing

    %===============Reads in txt file=======================

    [x, y, val] = textread(fileList(r_index).name); %Read in txt file
```

```
%==============Create false image=====================

for i_index=1:numel(x);
    %Removes stripey hits; Removes noise near edges of image
    if val(i_index) > 1 && val(i_index) < 9900;
        %Creates false image
        pic(x(i_index)+1,y(i_index)+1)=val(i_index);
    end;
end;

timing_read(r_index)=toc;%End timing

tic;%Start timing

%=================Remove dead pixels=============

hit_matrix=pic.*mask;%Removes dead pixels

%============Prefilter======================
%Removes high-intensity single pixel noise

hit_matrix=gt(hit_matrix,1);%Thresholds image

up = [hit_matrix(2:end,:); add_vert];%Shift image upwards by one pixel
left = [hit_matrix(:,2:end), add_horz];%Shift image left by one pixel

%Adds shifted images and mutiplies by original image
real_hits = (up+left+hit_matrix).*hit_matrix;

%Removes pixels that did not have an above-threshold neighbor to the
%left or up
BW = gt(real_hits,2);%Threshold image

%==========Finds hits=======================

CC = bwconncomp(BW,4);%Finds objects using flood-fill algorithm

%================Find centroid and max object time===============

%Initialize variables
row=cell(1,CC.NumObjects);%Pixel x-position
column=cell(1,CC.NumObjects);%Pixel y-position
centroid=cell(1,CC.NumObjects);%Centroid of object
max_time = ones(CC.NumObjects,1);%Maximum timestamp in object

for h_index = 1:CC.NumObjects;

    %Removes hits below the minimum object size
    if size(CC.PixelIdxList{1,h_index},1) < min_size;
        CC.PixelIdxList{1,h_index} = [];
    end;

    %Finds centroid of each object
    if isempty(CC.PixelIdxList{1,h_index}) ~= 1;
        %Converts linear indices to subscripts
        [row, column]=ind2sub([x_dim,y_dim],CC.PixelIdxList{1,h_index});
```

```
            %Finds x-position of centroid
            centroid_x=sum(row.*pic(CC.PixelIdxList{1,h_index})./sum(pic(CC.PixelIdxList{1,h_
index})));
            %Finds y-position of centroid
            centroid_y=sum(column.*pic(CC.PixelIdxList{1,h_index})./sum(pic(CC.PixelIdxList{1
,h_index})));
            %Saves centroid of object
            centroid{1,h_index} = [centroid_x centroid_y];
            %Finds maximum timestamp in an object
            max_time(h_index) = max(pic(CC.PixelIdxList{1,h_index}));
        end;
    end;

    %Creates double matrix of centroid locations (cell--> matrix)

    tic;
    rows = size(centroid,1);
    cols = size(centroid,2);
    if (rows < cols)
    centroids = cell(rows,1);
    % Concatenate one dim first
    for n=1:rows
        centroids{n} = cat(2,centroid{n,:});
        q=1;
    end
    % Now concatenate the single column of cells into a matrix
    centroids = cat(1,centroids{:});
    end;

    value=size(centroids,2)/2;
    centroids=reshape(centroids,2,value);%Reshapes matrix

    max_time=max_time(max_time>1);%Removes zeros

    info = [info; centroids' max_time];%Centroid location and average time for each object
    num_hits(r_index)=size(centroids,2);%Number of hits in frame

    timing(r_index)=toc;%End timing
end;

[ion1,ion2,ion3,electrons]=sorthits(info);
visualize_hits(ion1,x_dim,y_dim);
visualize_hits(ion2,x_dim,y_dim);
visualize_hits(ion3,x_dim,y_dim);
visualize_hits(electrons,x_dim,y_dim);

max_time_2=info(:,3);
figure;
h=histogram(max_time_2,3000);%Creates histogram of average times
```

```
function [ion1,ion2,ion3,electrons] = sorthits(info)
% This function sorts the hits by their max time value and outputs
% four lists which include the centroid location for hits for one fragment
%
% Input variables: info - list of centroids
%
% Output variables: ion1 - list of centroids for first ion
%                   ion2 - list of centroids for second ion
%                   ion3 - list of centroids for third ion
%                   electrons - list of centroids for electron
%
% Written 11/29/2015 by Rachel Sampson

[row1 ~]=find(9430<info(:,3) & info(:,3)<=9500);
ion1=info(row1,:);
[row2 ~]=find(9500<info(:,3) & info(:,3)<=9640);
ion2=info(row2,:);
[row3 ~]=find(9720<info(:,3) & info(:,3)<=9772);
ion3=info(row3,:);
[row4 ~]=find(9772<info(:,3) & info(:,3)<9865);
electrons=info(row4,:);
```

Published with MATLAB® R2015b

```
function [image]=visualize_hits(fragment,x_dim,y_dim)
%Creates an image of the hits from each fragment by placing a Gaussian at the
%centroid of each detected object
%
% Input variables: fragment - list of centroid locations for a single
%                              fragment type
%                  x_dim - x dimension of image
%                  y_dim - y dimension of image
%
% Output variables: image - image with gaussians placed at centroid
%                            locations
%
% Written 11/29/2015 by Rachel Sampson

sigma=3;%Waist of gaussian
sizeg=[x_dim,y_dim];%Size of image

center = [ceil(fragment(:,1)),ceil(fragment(:,2))];%Center of hits

image = zeros(x_dim,y_dim);%Creates false image

for h_index=1:size(center,1)
    val = gaussC(sizeg, sigma, center(h_index,:));%Places gaussian at centroid
    image = image+val;%Creates composite image
end;

 figure;
 imagesc(image);

function val = gaussC(sizeg, sigma, center);%Generates gaussian mask
[x,y] = ndgrid(1:sizeg(1), 1:sizeg(2));
xc=center(:,1);
yc=center(:,2);
exponent=((x-xc).^2/(2*sigma^2)+(y-yc).^2/(2*sigma^2));
amplitude = 1/(2*sqrt(2*pi));
val=amplitude*exp(-exponent);
end

end
```

Appendix C: LabVIEW Basler Code

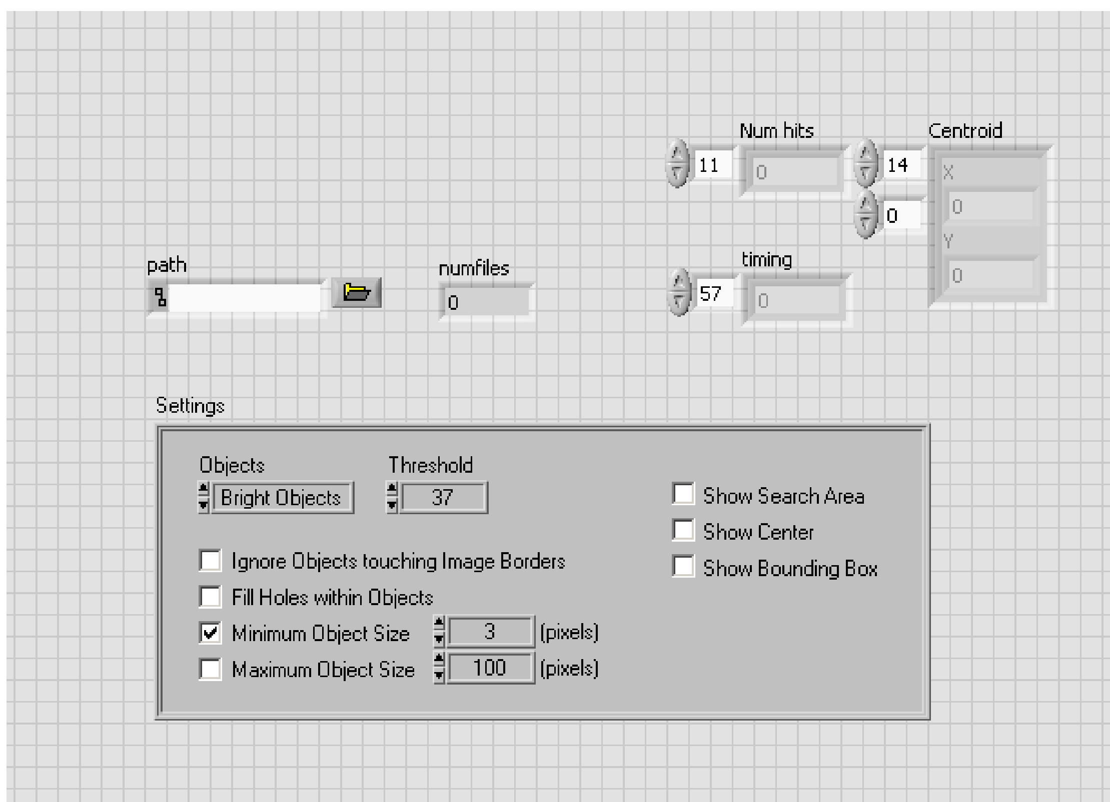

Figure C.1: Front panel.

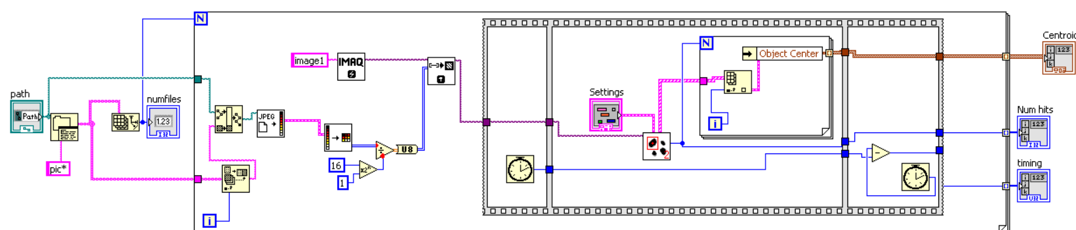

Figure C.2: Block diagram.