**B-TECH PROJECT REPORT**

**Development of a low-cost real-time Particle Image Velocimetry system**

Submitted by: -

**RAMAVATH JAGADEESH (200103090)**

**NAVEEN PAREEK (200103082)**

Under the supervision of

**Dr. ATUL KUMAR SOTI**



**DEPARTMENT OF MECHANICAL ENGINEERING**

**INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI**

**CERTIFICATE**

It is certified that the work contained in the thesis entitled “**Development of a low-**

**cost real-time Particle Image Velocimetry**” by “**Mr. Ramavath Jagadeesh**” and

“**Mr. Naveen Pareek,**” has been carried out under my supervision and this work

has not been submitted elsewhere for a degree.

Dr. Atul Kumar Soti

May 2023 Department of Mechanical Engineering

Indian Institute of Technology Guwahati

**DECLARATION**

I declare that this written submission represents my ideas in my own words and where others’ ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will cause disciplinary action by the Institute and can also invoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature with date)

Ramavath Jagadeesh (200103090)

Naveen Pareek (200103082)

Department of Mechanical Engineering,

I.I.T. Guwahati, Assam, India

**ABSTRACT**

This report is the record of our learning about PIV (Particle Image Velocimetry). In this report we recorded our learning of C++ code for analyzing images, processing cross-correlation in C++ and how we can use this code in PIV.

In this project, we first generated code for analyzing velocity fields then tried to implement it on a moving disc. We also recorded our result in this report.

Ramavath Jagadeesh (200103090)

Naveen Pareek (200103082)

Department of Mechanical Engineering,

I.I.T. Guwahati, Assam, India

**ACKNOWLEDGMENT**

We would like to express our gratitude and indebtedness towards Dr. Atul Kumar Soti, Assistant Professor, Department of Mechanical Engineering, IIT Guwahati for giving us the opportunity to take up this project and delve deeper into the field of Fluid Mechanics and Particle Image Velocimetry. We also want to thank him for his constant guidance, patience, empathy, kindness, and friendly behavior while supervising us in the project.

We would like to thank our seniors who helped us whenever we sought their expertise.

At last, we are grateful to our parents and God for everything.

Ramavath Jagadeesh (200103090)

Naveen Pareek (200103082)

Department of Mechanical Engineering,

I.I.T. Guwahati, Assam, India

**INDEX**

Report Title …..............................................................................................03

Certificate.......................................................................................................05

Declaration......................................................................................................07

Abstract...........................................................................................................09

Acknowledgement..........................................................................................11

Index................................................................................................................13

List of Figures.................................................................................................15

Nomenclature....................................................................................17

1: - Introduction................................................................................................22

2: -Literature Review........................................................................................23

3: - Work done in Phase I.................................................................................24

4: - Code generation in C++….........................................................................25

5: - Conclusion and Future Scope ....................................................................39

**LIST OF FIGURES**

Fig. 3.1: - Experimental arrangement for planar 2C-2D PIV in a wind tunnel............23

Fig.4.1: - byte storing in various locations of image’............................................25

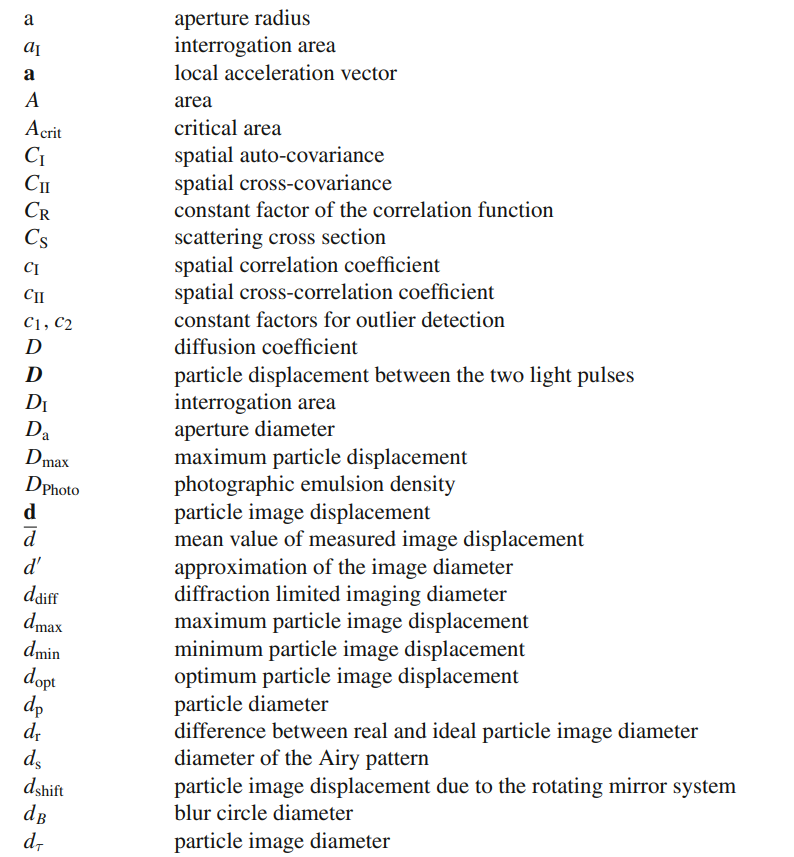
Fig 4.2: - grading of colored images.............................................................................26

Fig.4.3: - As we can see by increasing levels of grey colors, we can make our image more clarified................................................................................................................27

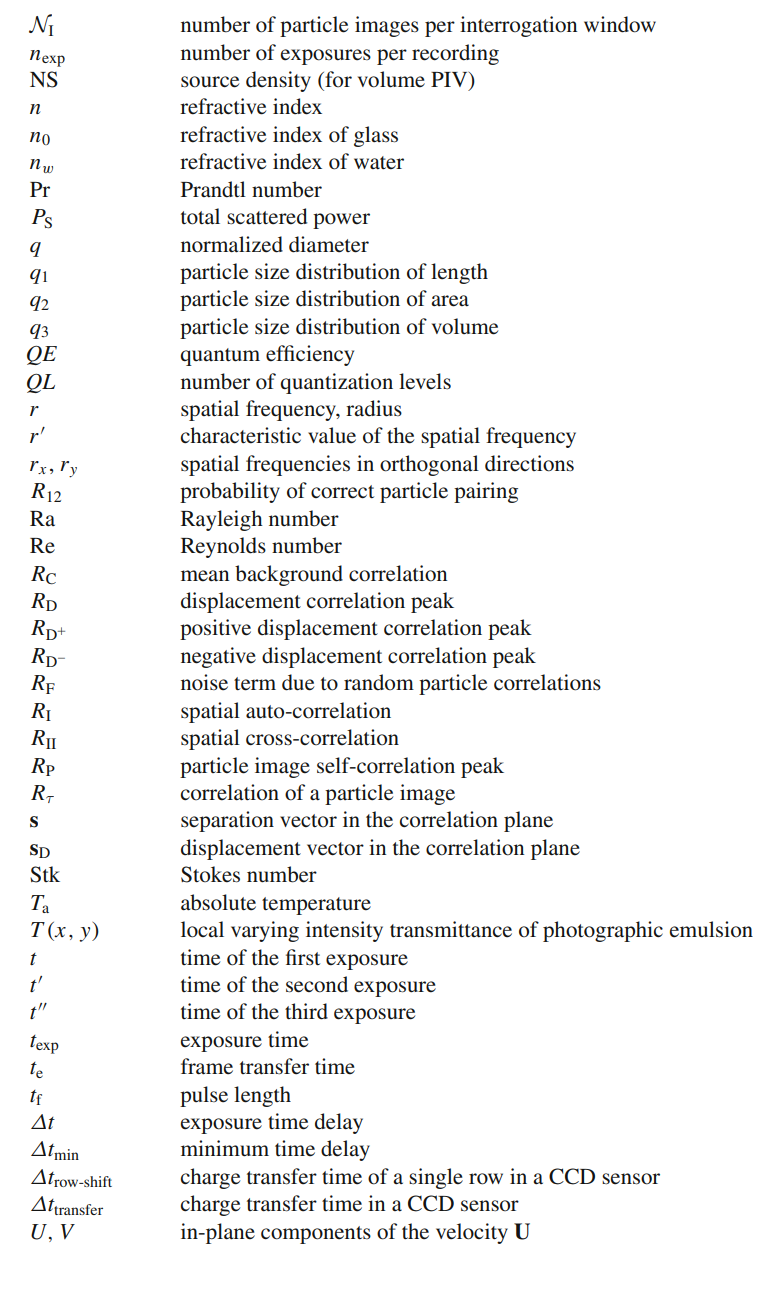
Fig.4.4: - moving a filter mask often referred to as kernel..........................................30

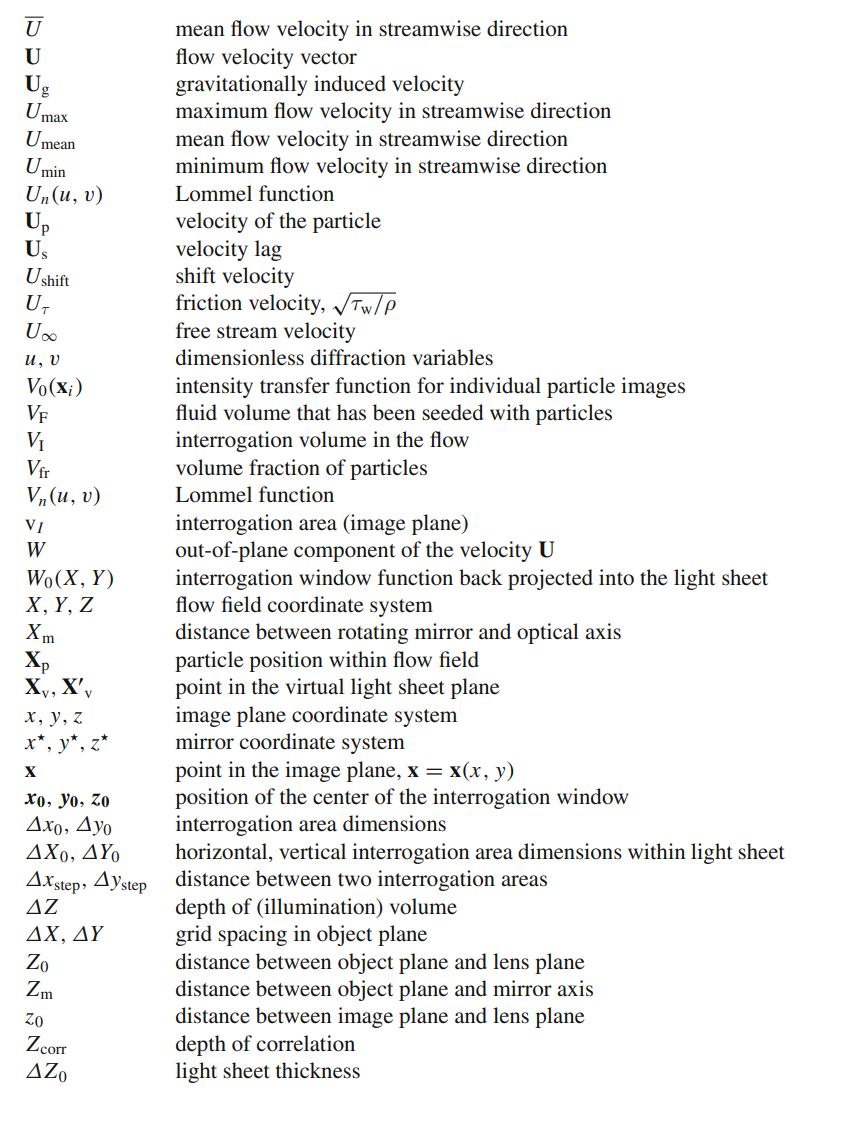
Fig.4.5: - Cross Correlation operation on a matrix.…..................................................31

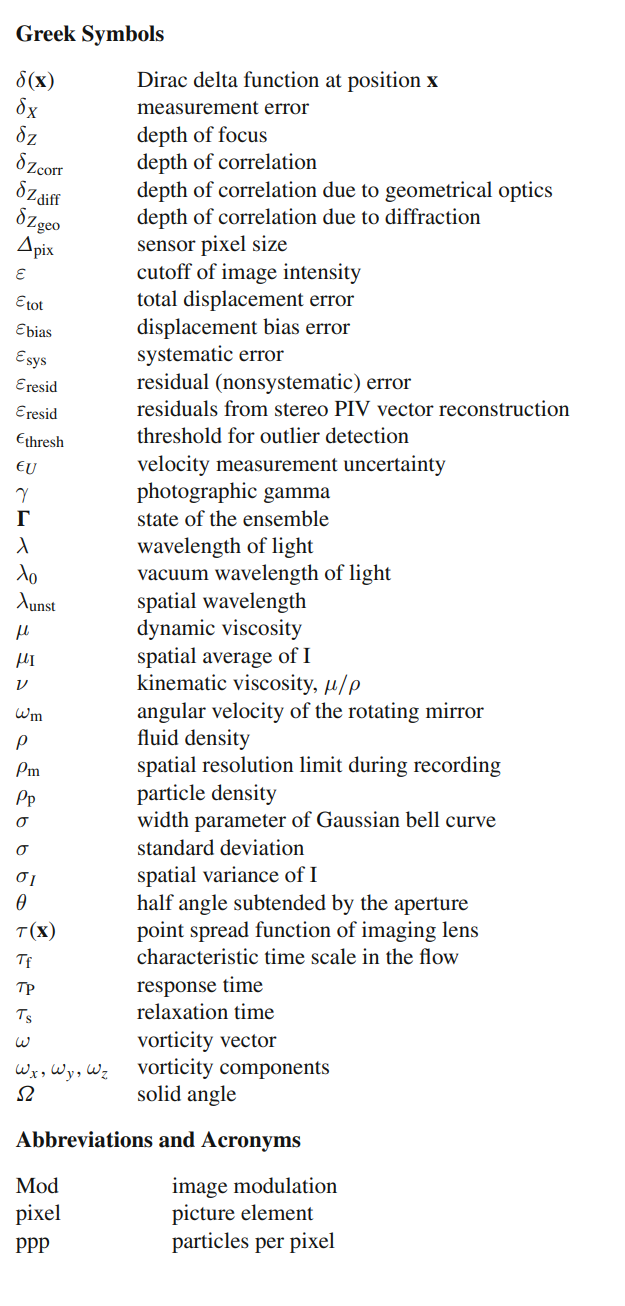
**NOMENCLATURE**



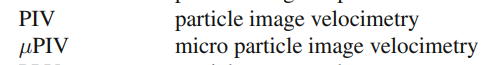
|  |  |  |
| --- | --- | --- |
|  |  | |
|  |  |











**Chapter 1**

**INTRODUCTION**

**1.1 WHAT IS PARTICLE IMAGE VELOCIMETRY?**

Particle Image Velocimetry (PIV) is an experimental technique for measuring the velocity of fluid particles. In this method, we measure the displacement of fluid particles over a brief period and divide it by the time to get the velocity of fluid particles.

For measurement of displacement, neutrally buoyant particles, such as 20 micro size hollow glass spheres for water, are used. Now, we use a planer light sheet to illuminate these particles and their images are recorded by a high-speed camera. Now we calculate the displacement of particles by using these photos captured by the camera by using a technical approach. Velocity can be derived by dividing displacement by time.

**1.2 THE PROBLEM: -**

Usually, in PIV, high-power continuous or pulsed lasers are used for creating light sheets. These lasers are very costly as well as dangerous to work with. So, our main objective is to generate a code for analyzing our images, which we got from the high-speed camera and applying cross correlation on it to get the final velocity and displacement field.

Also, to make the process of image processing faster and to make it suitable for real-time image processing applications, we need to develop code in such a language that reduces the time for processing images, and which directly gives us displacement as well as velocity field in real-time.

**1.3 SCOPE OF THE WORK: -**

Through this project, we learned about Particle Image Velocimetry and learned how to process image and cross correlation in C++ language. In this report, we are trying to record the learning and understanding which we got from this project. We generated a code in C++ language through which we can calculate the displacement field of particles. Later, which is further used to calculate the velocity field of the flow.

**Chapter 2**

**LITERATURE REVIEW**

* We get concepts knoledge of PIV through “**Particle Image Velocimetry a Practical Guide, Markus Raffel • Christian E. Willert Fulvio Scarano • Christian J. Kähler Steven T. Wereley • Jürgen Kompenhans”.** We studied concepts and procedures of measuring velocity flow using PIV, and methods to perform it.
* Then we learned image processing basics from some research paper named **LOSSLESS DIGITAL IMAGE COMPRESSION METHOD FOR BITMAP IMAGES Dr T. Meyyappan1, SM. Thamarai2 and N.M.Jeya Nachiaban3, Image Processing in C Second Edition by Dwayne Phillips.** We got knowledge of bitmap image format from a research paper named **Analyzing the Bitmap Image File and Extracting Different Bit Patterns for Compressing the Image File using Distinct Colour Codes.**

# We learned correlation from many resources like YouTube, google and went through some research works like **Understanding Correlation, January 04, 2017, by Sneha H.L.,** **Cross Correlation, AutoCorrelation -- 2D Pattern Identification written by Paul Bourke**

* **The International Journal of Multimedia & Its Applications (IJMA) Vol.3, No.4, November 2011**

**Chapter 3 WORK DONE IN PHASE I**

We make following conclusions in Phase I: -

* PIV is a method for calculating fluid particles' velocity and has various applications. In this method we mix tracing particles in fluid and click the images of flow by illuminating these tracing particles with the help of lasers, and then we pass these images from our generated code and get the displacement and velocity flow field.
* Here, in our project we generated code on Matlab, later to increase speed and decrease run time to make PIV more suitable for real time operations we can convert this code to C or C++ code.
* After writing code, we verified our code with pre-defined data and two small experiments. further we can do more such experiments to verify our code. Firstly, we tried our experiment with Simple Pendulum, to see its displacement Vector Field, and later we analyze the fan rotation

Following are some topics, which we learned in Phase I: -

**3.1 WHAT IS PIV?**

PIV stands for Particle Image Velocimetry. As the name suggests, it is a method to calculate the velocity of particles with the help of images. It is one of the most preferred method to calculate the velocity of particles of fluid or fluid flow.

**3.2 EXPERIMENTAL SETUP FOR PIV: -**

In this method, small tracing particles are added to the fluid which is later illuminated in the plane of flow with the help of a laser. Images are captured in a very small-time interval, which is later used to determine the flow velocity with the help of technical methods. The below figure (fig. 3.3) shows the setup of PIV.

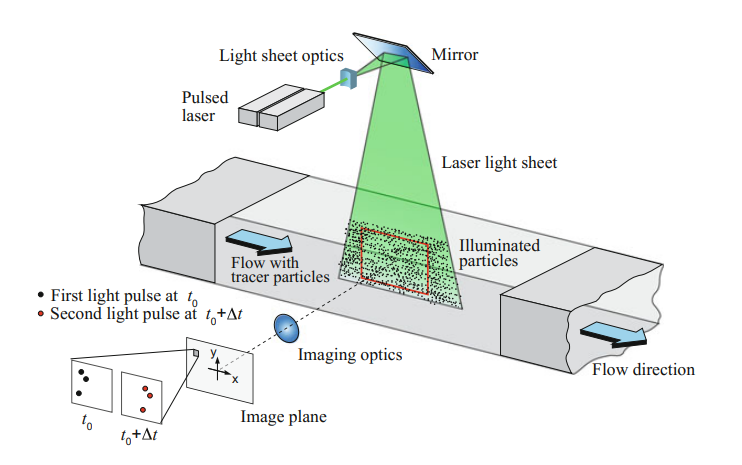


Fig 3.1: - Experimental arrangement for planar 2C-2D PIV in a wind tunnel

Source: - Particle Image Velocimetry a Practical Guide, Markus Raffel • Christian E. Willert Fulvio Scarano • Christian J. Kähler Steven T. Wereley • Jürgen Kompenhans

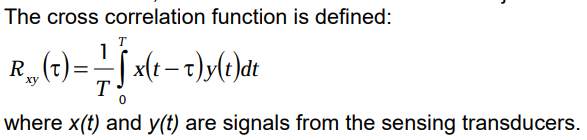
**3.4 APPARATUS OF PIV SETUP: -**

The basic apparatus of PIV contains elements as follows: -

* Tracing Particles: - to track fluid movement.
* Illumination System: - to illuminate the tracing particles.
* Synchronizer: - to synchronize the camera and laser.
* Host Computer: - to store the particle image and conduct image processing.

**3.5 IMAGE EVALUATION METHOD: -**

what is cross-correlation: -



In this phase we will extend our study to get deep knowledge of Cross Correlation and generate a code for image processing, we get from our high-speed cameras.

**Chapter 4**

**GENERATING CODE IN CPP**

After getting images of fluid flow, our main objective is to determine the displacement vector. To fulfill this requirement, In Phase I, we generated a Matlab code which after taking the input of our images showed us displacement vector as output. From this, we got velocity by dividing it by time (multiplying it with the frequency of the camera).

But in this phase our main objective is to develop that code into Cpp or C language to get results faster so that we can use our code for real time applications. For this we will first generate image processing code then cross correlation and then merge them to get final analysis code to work for our application.

**4.1 IMAGE PROCESSING IN C**

There are many image formats such as JPEG, PNG, TIFF, BMP, etc. All files have their pros and cons. For example, PEGs compress automatically — but the image data that you lose when this happens is barely perceptible to the eye. The JPEG file analyses image information and identifies erasable and compressible details based on the image type. This process can even reduce JPEG files to 5-10% of their original size. On the other hand, BMP files are larger than JPEGs. The intention behind BMPs is to hold as much information as possible. So, while JPEGs can get quite large, BMPs are often bigger.

But, for image processing in code we are using a BMP format because it has some advantages over JPEGs. For example, BMP files are of a higher quality than JPEGs.In a BMP image, each pixel has its own specific color. The file may also contain information like color, depth, color profiles, alpha channels and more. This gives BMP images a higher resolution than JPEG files. Their quality and resolution mean that it is easier to edit BMP files because there are more pixels you can manipulate.

**4.1.1 WHAT IS BMP IMAGE?**

BMP stands for bitmap, a raster-based file type designed in the early days of computer graphics to display images independently from devices. Because BMP files are information-rich, they tend to have large file sizes. The Bitmap File Format (BMP) is less complex to understand and decode.

Any image would be structured in the following way:

* **image header –> color table (if any) –> image data**

The BMP image is structured in the same way. It has: **a 54-byte image, 1024-byte color table if present, and the rest is image data.**

Each bitmap file contains a bitmap-file header, a bitmap-information header, a color table, and an array of bytes that define the bitmap bits. The file has the following form:

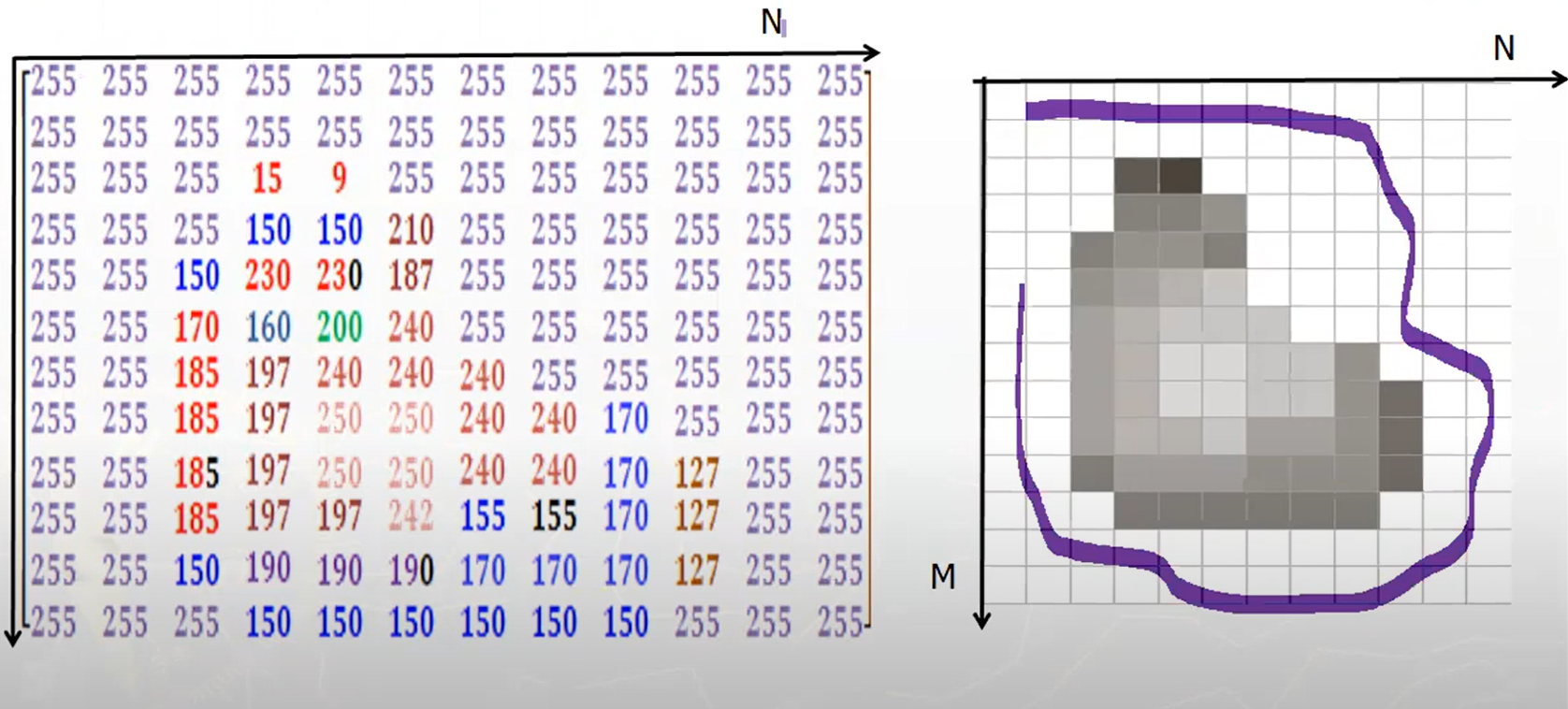
BITMAPFILEHEADER bmfh;

BITMAPINFOHEADER bmih;

RGBQUAD aColors[];

BYTE aBitmapBits[];

These headers, info header and color table etc. Contains information of image in form of bytes. These bytes can be read, written and changed to make any changes in image.

Fig.4.1 byte storing in various locations of image source: - https://youtu.be/qrLxC\_ZhXy4

For example, in the above image, we can see how an image is represented in the form of a matrix by the given max intensity as 255 data and other pure black as 0 and then relative data is stored in each row and column.

Similarly, we can also work on colored images by giving colored matrices value in the shades of primary colors.

Let us take an example, the below image fig. 4.2 a and b we can that how grading is done in three spaces of image by taking white means presence of 93% of all three primary colors and black means absence of all three colors.

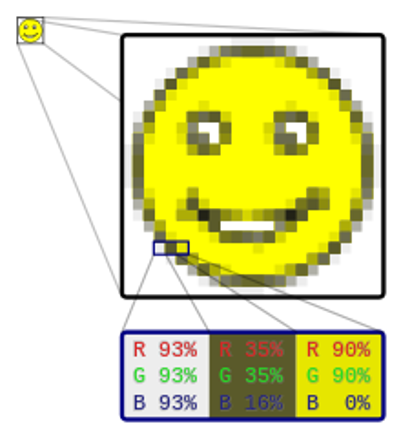
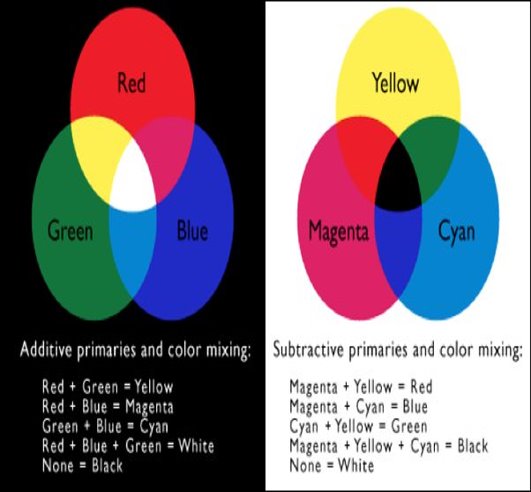
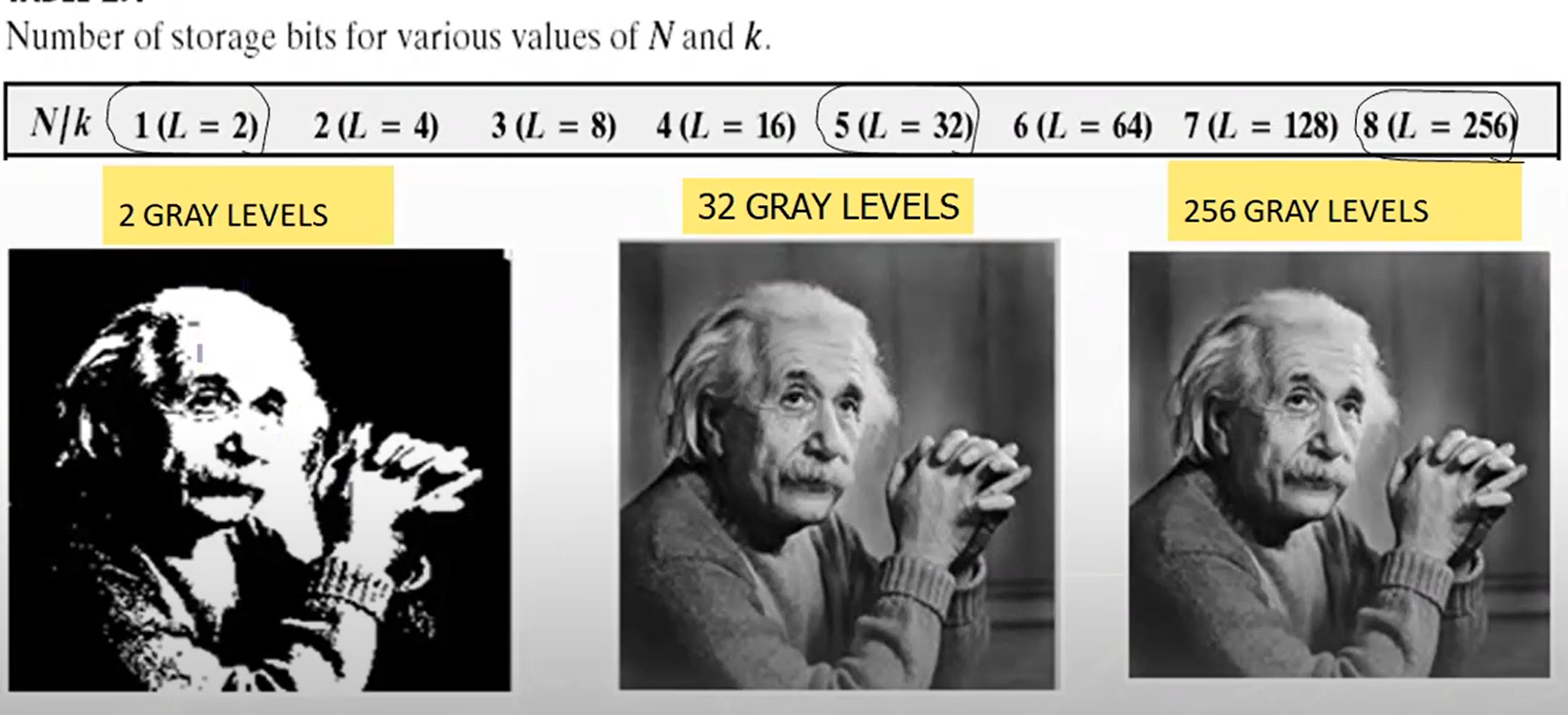


fig.4.2 grading of colored images source: - Google- Wikipedia.

Similarly, we can write and read image files of BMP format in a matrix form. It makes image analysis easy and possible for coding purposes.

Here, it is always dependent on us about how many grades we take between the highest and lowest intensity of presence of color, i.e. White and black, respectively. The greater the numbers of grades, the image will have more details and clarity.

fig.4.3 As we can see by increasing levels of grey colors, we can make our image more clarified. source: - https://youtu.be/qrLxC\_ZhXy4

**4.1.2 GENERATED CODE FOR IMAGE PROCESSING IN C:**

#include <string.h>

#pragma pack(2)

typedef struct {

char type[2];

unsigned int size;

unsigned short reserved1;

unsigned short reserved2;

unsigned int offset;

} BMPHeader;

#pragma pack(2)

typedef struct {

unsigned int size;

int width;

int height;

unsigned short planes;

unsigned short bitsPerPixel;

unsigned int compression;

unsigned int imageSize;

int xResolution;

int yResolution;

unsigned int colorsUsed;

unsigned int colorsImportant;

} BMPInfoHeader;

int main(int argc, char \*argv[]) {

// if (argc < 3) {

// printf("Usage: %s input.bmp output.bmp\n", argv[0]);

// return 1;

// }

// Open input BMP file

FILE \*inputFile = fopen("sample\_1280×853.bmp", "rb");

if (inputFile == NULL) {

printf("Error: could not open input file %s\n", argv[1]);

return 1;

}

// Read BMP header

BMPHeader header;

fread(&header, sizeof(BMPHeader), 1, inputFile);

// Verify BMP file type

if (header.type[0] != 'B' || header.type[1] != 'M') {

printf("Error: invalid BMP file type\n");

fclose(inputFile);

return 1;

}

// Read BMP info header

BMPInfoHeader infoHeader;

fread(&infoHeader, sizeof(BMPInfoHeader), 1, inputFile);

// Print image width and height

printf("Image width: %d\n", infoHeader.width);

printf("Image height: %d\n", infoHeader.height);

// Open output BMP file

FILE \*outputFile = fopen("output1.bmp", "wb");

if (outputFile == NULL) {

printf("Error: could not create output file %s\n", argv[2]);

fclose(inputFile);

return 1;

}

// Write BMP header and info header to output file

fwrite(&header, sizeof(BMPHeader), 1, outputFile);

fwrite(&infoHeader, sizeof(BMPInfoHeader), 1, outputFile);

// Allocate memory for image data

unsigned char \*imageData = malloc(infoHeader.imageSize);

if (imageData == NULL) {

printf("Error: could not allocate memory for image data\n");

fclose(inputFile);

fclose(outputFile);

return 1;

}

// Read image data from input file

fread(imageData, infoHeader.imageSize, 1, inputFile);

// Write image data to output file

fwrite(imageData, infoHeader.imageSize, 1, outputFile);

// Clean up

free(imageData);

fclose(inputFile);

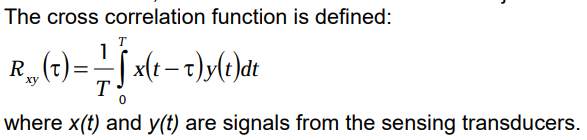
fclose(outputFile);

return 0;

}

**4.2 WHAT IS CROSS CORRELATION?**

Cross-correlation is **a measurement that tracks the movements of two or more sets of time series data relative to one another**. It is used to compare multiple time series and objectively determine how well they match up with each other and, in particular, at what point the best match occurs.



Correlation is **a mathematical technique to see how close two things are related**. In image processing terms, it is used to compute the response of a mask on an image. A mask is applied on a matrix from left to right. Mask slides over the matrix from left to right by one unit every time

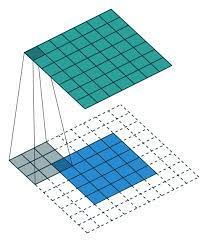


Fig.- 4.4 moving a filter mask often referred to as kernel source: - google

Correlation is the process of moving a filter mask often referred to as kernel over the image and computing the sum of products at each location. Correlation is the function of displacement of the filter.

**4.3 Cross Correlation in C/CPP?**

It was an in-built function in Matlab where we could correlate images and find the displacement of images, but in C/C++ we do not have such type of inbuilt library, so we need to generate a Cross Correlation Function in C/C++ using basic C++ Algorithms and Data Structures.

**4.3.1 WORKING OF CROSS CORRELATION: -**

The cross-correlation of the discrete time signals *x* [*n*] and *y* [*n*] is expressed as

Rxy[m]=∞∑n=−∞x[n]y⋆[n−m]

Cross-correlation of any two given signals can be found via graphical techniques. Here, one signal is slid upon the other while computing the samples at every interval. That is, in the case of digital signals, one signal is shifted by one sample to the right each time, at which point the sum of the product of the overlapping samples is computed.

# We can understand this with an example taken from **Understanding Correlation, by Sneha H.L.** in this example, cross-correlation of the digital signals x [n] = {-3, 2, -1, 1} and y [n] = {-1, 0, -3, 2} can be computed as shown.

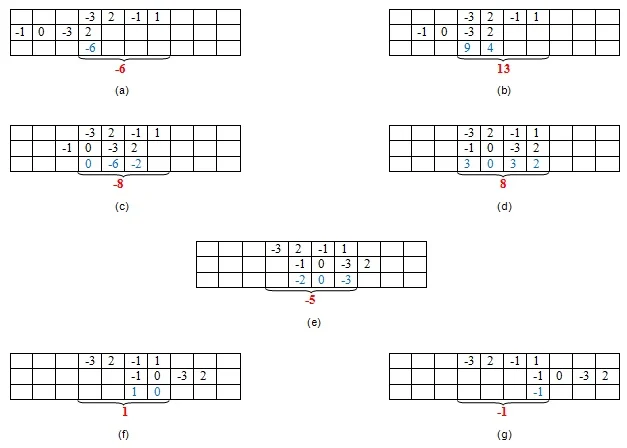


Fig. 4.5: - Cross Correlation operation on a matrix source: -google

Here, the first set of samples (in the first row of every table) refers to the signal x [n] and the second set refers to the samples (in the second row of every table) of the signal y [n].

Next, the samples shown in blue color—those in the third row—are obtained by multiplying the corresponding samples of the first two rows. Finally, we add the samples in the last row (contained in the curly brackets) to obtain the samples of the cross-correlated signal.

Thus, here we see that the samples of the cross-correlated signal Rxy are obtained as {-6, 13, -8, 8, -5, 1, -1}, where 8 is the zeroth sample.

Further, the example presented shows that the sample of the cross-correlated signal is at its highest peak, with value 13, when the last two samples of *y* [*n*] overlap with the first two samples of *x* [*n*]. This is because, in this case, the second signal overlaps with the first at its best, as the two samples in each of the signals are identical.

Hence, it can be concluded that the cross-correlation reaches its maximum when the two signals considered become most similar.

**4.3.2 CROSS CORRELATION CODE: -**

// catch

#include <catch2/catch\_test\_macros.hpp>

// std

#include <algorithm>

#include <cmath>

// to be tested

#include "core/grid.h"

using namespace openpiv::core;

namespace std {

template <typename T>

std::ostream& operator<<(std::ostream& os, const std::vector<T>& ts)

{ char separator[] = ", ";

const char\* ps = nullptr;

for ( const auto& t : ts )

{

os << ( ps ? ps : "") << t;

ps = separator;

}

return os;

}

}

TEST\_CASE("grid\_test - cartesian grid")

{ // utils

using point\_t = rect::point\_t;

auto sort\_grid = []( auto& grid )

{// sort by length of vector and by angle if necessary

std::sort( std::begin(grid), std::end(grid),

[](const rect& lhs, const rect& rhs)

{

const auto& lhs\_tl = lhs.topLeft();

const auto& rhs\_tl = rhs.topLeft();

auto p = [](const auto& \_p){ return \_p[0]\_p[0] + \_p[1]\_p[1]; };

auto a = [](const auto& \_p){ return atan2(\_p[1], \_p[0]); };

if ( p(lhs\_tl) == p(rhs\_tl) )

return a(lhs\_tl) < a(rhs\_tl);

return p(lhs\_tl) < p(rhs\_tl);

} );

return grid;

};

size interrogation{ 32, 32 };

std::vector<rect> expected;

SECTION("margin")

{

auto generated = generate\_cartesian\_grid( {100, 50}, interrogation, 0.5 );

for ( const auto& tl :

{ point\_t(2, 1), point\_t(18, 1), point\_t(34, 1), point\_t(50, 1), point\_t(66, 1),

point\_t(2, 17), point\_t(18, 17), point\_t(34, 17), point\_t(50, 17), point\_t(66, 17) } )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

}

SECTION("exact 1x1")

{

auto generated = generate\_cartesian\_grid( {32, 32}, interrogation, 0.5 );

expected.emplace\_back( rect({0, 0}, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

}

SECTION("exact 3x3")

{

auto generated = generate\_cartesian\_grid( {64, 64}, interrogation, 0.5 );

for ( const auto& tl :

{ point\_t(0, 0), point\_t(16, 0), point\_t(32, 0),

point\_t(0, 16), point\_t(16, 16), point\_t(32, 16),

point\_t(0, 32), point\_t(16, 32), point\_t(32, 32) } )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

}

SECTION("exact 5x5")

{ auto generated = generate\_cartesian\_grid( {64, 64}, interrogation, 0.25 );

for ( const auto& tl :

{ point\_t(0, 0), point\_t(8, 0), point\_t(16, 0), point\_t(24, 0), point\_t(32, 0),

point\_t(0, 8), point\_t(8, 8), point\_t(16, 8), point\_t(24, 8), point\_t(32, 8),

point\_t(0, 16), point\_t(8, 16), point\_t(16, 16), point\_t(24, 16), point\_t(32, 16),

point\_t(0, 24), point\_t(8, 24), point\_t(16, 24), point\_t(24, 24), point\_t(32, 24),

point\_t(0, 32), point\_t(8, 32), point\_t(16, 32), point\_t(24, 32), point\_t(32, 32) } )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

}

SECTION("3x3, 1 margin")

{ auto generated = generate\_cartesian\_grid( {65, 65}, interrogation, 0.5 );

for ( const auto& tl :

{ point\_t(0, 0), point\_t(16, 0), point\_t(32, 0),

point\_t(0, 16), point\_t(16, 16), point\_t(32, 16),

point\_t(0, 32), point\_t(16, 32), point\_t(32, 32) } )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

}

SECTION("3x3, 2 margin")

{ auto generated = generate\_cartesian\_grid( {66, 66}, interrogation, 0.5 );

for ( const auto& tl :

{ point\_t(1, 1), point\_t(17, 1), point\_t(33, 1),

point\_t(1, 17), point\_t(17, 17), point\_t(33, 17),

point\_t(1, 33), point\_t(17, 33), point\_t(33, 33) } )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) ); }

SECTION("exact 3x3, 50% pixel offset")

{ auto generated = generate\_cartesian\_grid( {64, 64}, interrogation, {16, 16} )

for ( const auto& tl :

{ point\_t(0, 0), point\_t(16, 0), point\_t(32, 0),

point\_t(0, 16), point\_t(16, 16), point\_t(32, 16),

point\_t(0, 32), point\_t(16, 32), point\_t(32, 32) } )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

}

SECTION("exact 3x5, 50/25% pixel offset")

{ auto generated = generate\_cartesian\_grid( {64, 64}, interrogation, {16, 8} );

for ( const auto& tl :

{ point\_t(0, 0), point\_t(16, 0), point\_t(32, 0),

point\_t(0, 8), point\_t(16, 8), point\_t(32, 8),

point\_t(0, 16), point\_t(16, 16), point\_t(32, 16),

point\_t(0, 24), point\_t(16, 24), point\_t(32, 24),

point\_t(0, 32), point\_t(16, 32), point\_t(32, 32) } )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

}

SECTION("exact 3x7, 50/50% pixel offset, non-square interrogation")

{ auto interrogation = size(32, 16);

auto generated = generate\_cartesian\_grid(

{64, 64}, // image size

interrogation, // interrogation size

{16, 8} // offset

);

for ( const auto& tl :

{ point\_t(0, 0), point\_t(16, 0), point\_t(32, 0),

point\_t(0, 8), point\_t(16, 8), point\_t(32, 8),

point\_t(0, 16), point\_t(16, 16), point\_t(32, 16),

point\_t(0, 24), point\_t(16, 24), point\_t(32, 24),

point\_t(0, 32), point\_t(16, 32), point\_t(32, 32),

point\_t(0, 40), point\_t(16, 40), point\_t(32, 40),

point\_t(0, 48), point\_t(16, 48), point\_t(32, 48)

} )

expected.emplace\_back( rect(tl, interrogation) );

INFO( "generated: " << sort\_grid(generated) );

INFO( "expected: " << sort\_grid(expected) );

CHECK( generated.size() == expected.size() );

CHECK( (sort\_grid( generated ) == sort\_grid( expected )) );

**CHAPTER 6**

**CONCLUSION AND FUTURE SCOPE**

In this phase, we went through literature by different authors by various means and tried to gain as much knowledge as possible. Also, we learn about image processing in C, implementation of Cross Correlation in CPP and use our learning to generate a code to study our PIV setup.

**6.1 CONCLUSION: -**

We make following conclusions after going through resources: -

* We can use an image as a matrix and can store data in it.
* We can use this method for applying several types of operations like correlation, Auto Correlation and Cross Correlation.
* Cross Correlation is a method to compare two signals, here images.
* We can compare and can find the position of maximum macthing of two images with the help Cross Correlation function.

**6.2 FUTURE SCOPE: -**

* In this phase, we developed code in C++ language to reduce the runtime and make our code more suitable for real-time applications.
* In the next phase, we will further develop our code and will do as many practicals as possible and will implement GPU to make our code faster and will work on practical PIV setup.

**REFERENCES: -**

* Particle Image Velocimetry a Practical Guide, Markus Raffel • Christian E. Willert Fulvio Scarano • Christian J. Kähler Steven T. Wereley • Jürgen Kompenhans
* https://en.wikipedia.org/wiki/Particle\_image\_velocimetry
* Velocity Field Measurement Using Particle Image Velocimetry (PIV)
* M.D. Atkins, in Application of Thermo-Fluidic Measurement Techniques, 2016
* https://www.adobe.com
* https://abhijitnathwani.github.io/blog/2017/12/20/First-C-Program-for-Image-Processing
* The International Journal of Multimedia & Its Applications (IJMA) Vol.3, No.4, November 2011
* Understanding Correlation by Sneha H.L.
* https://youtu.be/2OW2aLFaV8A
* https://youtu.be/qrLxC\_ZhXy4