A Flexible Software Environment for Computer Vision

LEELA GAUTHAM
RAJANEESH KUMAR

This thesis is presented as part of Degree of Master of Science in Electrical Engineering with emphasis on Signal Processing

Blekinge Institute of Technology
March 2014
ABSTRACT

The advancement of open source libraries is playing a key role in today's world. These libraries are being used in most of the real time applications. It is not an easy task to develop a library which is portable and having hundreds of algorithms in it and which is available for free. Here we have Opencv which has hundreds of functions and is being downloaded by millions and the other library PCL is also extensively used open source library for 3D point clouds.

The major problem in using these libraries comes when we are trying to integrate these libraries

In this thesis we have developed a class library based on open source libraries (opencv,pcl) which has been designed and implemented in c++ programming. The developed class library gives its user portability, simplicity, collaboratibility and extendibility. The class library uses qt framework provides a lab environment for students and researchers and also gives the freedom to extend classes. Besides its practical value, the developed class library teaches both object oriented programming and computer vision.

The students spend more time in integration of different open source libraries, so in order to save the time spent by the students in integration and also to make the user use the functions in reliable way. This class library helps the students to overcome this problem and understand both the object oriented programming methodology as well as the Image processing and computer vision. This library is mainly built to maintain portability and extendibility which is the main goal of this thesis.
ACKNOWLEDGEMENT

We take this opportunity to express a deep sense of gratitude towards my thesis supervisor Dr. Siamak Khatibi for providing excellent guidance, encouragement and inspiration throughout the thesis. Without his invaluable guidance, this work would never have been a successful one.

I would like to thank my fellow students who have supported us with their valuable suggestions and discussions throughout this thesis. We would like thank each and everyone who has extended their support during this thesis.

Finally, we would like to thank our parents for supporting us.
# Contents

Abstract 2  
Acknowledgement 3  
List of Figures 6  
List of Abbreviations 7  

1. CHAPTER  
1.1 Introduction .................................................................8  
1.2 Motivation ...............................................................8  
1.3 Aims & objects ............................................................9  
1.4 Research Question ....................................................9  
1.5 Thesis outline ............................................................9  

2. CHAPTER  
Background Theory ..........................................................10  
2.1 Object Oriented Programming ........................................10  
2.2 Object Oriented programming paradigm..............................10  
2.3 Basic concepts of oops ...................................................11  
2.3.1 Classes ...............................................................11  
2.3.2 Objects ...............................................................12  
2.3.3 Data abstraction ......................................................13  
2.3.4 Encapsulation ........................................................13  
2.3.5 Inheritance ..........................................................13  
2.3.6 Polymorphism .........................................................14  
2.4 Opencv ..................................................................15  
2.4.1 Structure of Opencv ...............................................15  
2.5 PCL .................................................................16  
2.5.1 Modular structure of pcl ........................................17  
2.6 Qt .................................................................19  

3. CHAPTER  
3.1 Introduction to SPL ....................................................20  
3.2 Architecture of SPL ....................................................20  
3.3 Class hierarch ............................................................20  
3.4 Role of Administrator ..................................................22  

4. CHAPTER  
4.1 Implementation of spl with qt .......................................23  
4.2 Folder structure of spl library ........................................29  
4.3 Educational Experience with class library .........................31  

5. CHAPTER  
Results ........................................................................32
6. CHAPTER
   Conclusion & future work.................................................................35

7. CHAPTER
   Bibliography....................................................................................36
List of Figures

2.1 OOP paradigm....................................................................................................10
2.2 Example of class box........................................................................................11
2.3 Example of object............................................................................................12
2.4 Example of both class and object.................................................................13
2.5 Derived class....................................................................................................14
2.6 Simple function showing polymorphism......................................................14
2.7 Pcl modules.......................................................................................................17
3.1 Simple diagram of spl library architecture..................................................20
4.1 Interface of qt creator IDE................................................................................23
4.2 Step by step to start a qt gui project...............................................................24
4.3 Testspl project.pro..........................................................................................26
4.4 File testspl.h ....................................................................................................26
4.5 File testspl.cpp................................................................................................27
4.6 File main.cpp ..................................................................................................27
4.7 Edit mode..........................................................................................................28
4.8 Design mode.....................................................................................................28
4.9 SPL structure....................................................................................................29
5.0 Code structure of spl.....................................................................................30
5.1 Simple GUI showing gray scale image.........................................................33
5.2 GUI showing canny image..............................................................................33
List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opencv</td>
<td>Open source computer vision</td>
</tr>
<tr>
<td>PCL</td>
<td>Point cloud library</td>
</tr>
<tr>
<td>IPP</td>
<td>Intel performance primitive</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface</td>
</tr>
<tr>
<td>SDK</td>
<td>Software development kit</td>
</tr>
<tr>
<td>OOP</td>
<td>Object oriented programming</td>
</tr>
<tr>
<td>SPL</td>
<td>Student perception lab</td>
</tr>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>OpenGL</td>
<td>Open graphics library</td>
</tr>
<tr>
<td>OSG</td>
<td>Open scene graph</td>
</tr>
<tr>
<td>LGPL</td>
<td>Lesser general public license</td>
</tr>
</tbody>
</table>
1 CHAPTER

1.1 Introduction

Object oriented programming plays a major role in the development of new library. Generally the library developed through this concept is more reliable and flexible to use so that the overall time spent is reduced using this object oriented programming. Coming to the computer vision field students are spending most of the time for combining different libraries and their related applications in the lab. There are many libraries in c++ which can be used like Opencv and Point cloud library consisting of different modules related to various fields like Image processing, camera calibration & 3D reconstruction. The point cloud library is also an Image processing library which mainly focuses on 3D point cloud data.

There are some problems in using these libraries by the students in the image processing lab because it is time consuming in combining the libraries and also difficult to predict the functions by the user. In this master thesis we tried to solve these problems faced by most of the students and researchers. So we developed a class library named SPL short for student perception lab, which is mainly dedicated to image processing, camera calibrations, visualization, image analysis. This library is developed based on the Opencv, Pcl library as these libraries covers all the modules that are related to Image processing and consists of various functionalities and can be used easily by the students in the lab. This library is mainly divided into classes and each class consists of its own functionalities.

So this proposed class library is designed in such a way that it is portable and extendable with better GUI creates a lab environment to the students thus saving a lot of time spent and also makes the user predict the functions to obtain the required output.

1.2 Motivation

In today's world the open source software's are being used effectively in library environment. The main advantage of this open source software is that it is free to use by anyone. If we observe in any educational institution or a researcher in the field of computer vision & image processing they are using third party libraries, such as opencv, pcl, Intel IPP, opengl and openscene graph. These Open source libraries are mainly related to computer vision algorithm implementation. But using these libraries and integrating with each other libraries by the user is not an easy task as the student has to spend a fairly large amount of time on integration and predicting the functions.

The motivation of this thesis is to implement a new class library named SPL which is completely dedicated to Image processing and some computer vision related functionalities using Opencv, Pcl and Qt for better GUI.

In this master thesis our supervisor (Dr. Siamak Khatibi) served as a starting point for the work on the overall structure of the library.
1.3 Aims & objectives

The main aim of this thesis is to implement image processing algorithms using open source computer vision library like opencv, Point cloud Library based on qt framework for a better gui. Here we will study different functionalities of opencv, pcl related to image processing and computer vision.

The objectives are as follows:

- To understand how to create a new class library using computer vision functionalities based on oops in c++.
- To make a class library extendable, flexible.
- To make an optimized class hierarchy.
- To implement class library on qt framework.

1.4 Research Question

- How to create a class library for students and researchers?
- How this library can be flexible, portable, simple, powerful Collabratible and extendable?

1.5 Thesis Outline

The thesis report is structured into seven chapters where each chapter covers the various aspects of the report.
Chapter 1 deals with the introduction followed by the aims & objectives, motivation, research question and thesis outline
Chapter 2 deals with the background theory which includes the knowledge regarding various open source libraries.
Chapter 3 provides the brief knowledge about the design of the spl library, class hierarchy.
Chapter 4 deals with introduction of spl implementation of spl with qt.
Chapter 5 deals with results.
Chapter 6 deals with conclusion and further work.
Chapter 7 Bibliography
2 CHAPTER

BACKGROUND THEORY

2.1 Object Oriented Programming

Object oriented programming is a way of approaching the programming. As programs continued to grow, high-level languages were introduced which allow the programmer to handle more complexity. Object-oriented programming promotes the use of software engineering methodology and facilitates software reuse. The main advantage with this OOP is that a problem can be divided and provides a software extension, modification and maintenance. Object-oriented programming with C++ can improve productivity and software quality and offers features such as classes, objects, data hiding, encapsulation, inheritance, polymorphism, and templates [1].

Object oriented programming is a programming paradigm based on the use of objects and their interactions.

2.2 OOP Paradigm

The main motivating factor in the invention of object-oriented approach is to reduce some of the flaws that occurred in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system [2][5]. It holds the data more closely to the functions that operate on it and protects it from modification from outside functions. OOP allows us to decompose a problem into a number of entities called objects and then builds data and functions around these entities. The organization of data and functions in object-oriented programs is shown in Fig 2.1.
The data of an object can access the functions of other objects. However, functions of one object can access the functions of other objects. Some of the striking features of object-oriented programming are:

- Emphasis is on data rather than procedure
- Programs are divided into objects.
- Class structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are held together in the data structure.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.

OOP offers several benefits to the program designer and the user.

### 2.3 Basic Concepts of OOP

It is therefore necessary to understand some of the concepts used extensively in object-oriented programming. We shall discuss in this section the following concepts [3].

1. Classes
2. Objects
3. Data Abstraction
4. Data Encapsulation
5. Inheritance
6. Polymorphism

#### 2.3.1 Classes

A class is used to specify the form of an object and it is used to define the data representation and methods for manipulating data into a package. The data and functions defined within a class are called members of the class. We can create any number of objects belonging to that class. Each object is associated with the data of type class with which they created. When we define a class it does define what the class name means and what kind of object it is and the operations performed by it. For example, mango, apple and orange are members of the class fruit. Classes are user-defined data types and behave like the built-in types of a programming language. For example we define box data type using the keyword class as followed in Fig 2.2.

```cpp
class Box
{
    public:
        double length;  // Length of a box
        double breadth;  // Breadth of a box
        double height;  // Height of a box
};
```

Fig 2.2: example of class box
The keyword `public` determines the access attributes of the members of the class that follow it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as `private` or `protected`.

### 2.3.2 Objects

Objects are the basic run-time entities in an object-oriented system. They may represent a person, i.e. a bank account, a table of data or any item that the program must handle. Program objects should be chosen such that they match closely with the real-world objects. A class provides the blueprints for objects, so basically an object is created from a class [4]. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types.

When a program is executed, the objects interact by sending messages to one another. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other’s data or code. It is sufficient to know the type of message accepted and the type of response returned by the objects shown in Fig 2.3.

```cpp
#include <iostream>

using namespace std;

class Box {
public:
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box

};

int main() {
    Box Box1;       // Declare Box1 of type Box
    Box Box2;       // Declare Box2 of type Box
    double volume = 0.0;    // Store the volume of a box here

    // box 1 specification
    Box1.length = 5.0;
    Box1.breadth = 3.0;
    Box1.height = 7.0;

    // box 2 specification
    Box2.length = 10.0;
    Box2.breadth = 13.0;
    Box2.height = 15.0;

    // volume of box 1
    volume = Box1.length * Box1.breadth * Box1.height;
    cout << "Volume of Box1 : " << volume << endl;

    // volume of box 2
    volume = Box2.length * Box2.breadth * Box2.height;
    cout << "Volume of Box2 : " << volume << endl;
    return 0;
}
```

Fig 2.3: example of object

Both of the objects Box1 and Box2 will have their own copy of data members
The Fig 2.4 shows the public data members of objects of a class can be accessed using the direct member access operator (.)

2.3.3 Data Abstraction

The wrapping up of data and functions into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the object’s data and the program. This insulation of the data from direct access by the program is called ‘data hiding’.

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost, and functions to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. Since the classes use the concept of data abstraction, they are known as Abstract Data Types (ADT).

2.3.4 Encapsulation

Encapsulation is the mechanism that binds together code and data and that protects from outside interference or misuse. It also allows the creation of an object. More simply an object in a logical entity that encapsulate both data and the code that manipulates that data within an object, some of the code and/or data may be private to the object and inaccessible to anything outside the object. In this way an object provides a significant level of protection against some other unrelated part of the program accidentally modifying or incorrectly using the private parts of the object.

2.3.5 Inheritance

Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of hierarchical classification. For example, the bird robin is a part the class flying bird which is again a part of the class bird. As illustrated in Fig 2.5 the principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived.
In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to use the class in such a way that it does not introduce any undesirable effects into the rest of the classes.

### 2.3.6 Polymorphism

Polymorphism is another important concept in OOP. Polymorphism means having several different forms and is mainly related to object methods. For example, an operation may exhibit different behavior in different instances but the behavior depends upon the types of data, used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation.

The Fig 2.6 below illustrates that a single function name can be used to handle different number and different types of arguments.

![Diagram of Polymorphism](image)
Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. Polymorphism refers to programming language ability to process objects differently depending on their data type or class. Polymorphism is considered as a requirement of any true object oriented programming language.

2.4 OpenCV

Coming to the introduction of open source libraries let's get to know about the Opencv library which is used in our thesis. Opencv is nothing but open source computer vision library including hundreds of computer vision algorithms and was introduced originally by the Intel [10]. The library has more than 2500 optimized algorithms and it is extensively downloaded with 2.5M downloads worldwide. This library is written in object oriented languages like C and C++ and can be deployed on Windows, Linux and Mac Osx platforms.

Opencv library mainly focuses on efficiency and optimization and the content of this library is interesting for students and researchers in image processing and computer vision area. The opencv library is divided into various modules and each module will deal with different functions.

Opencv has the machine learning library which is very important for the opencv library and now coming to the core part of the Opencv library it mainly contains MLL [11]. Most programmers and students come to know about the role computer vision plays. Now this computer vision is widely being used for processing images and videos on the web. First it was very difficult to have an organized and portable library for various applications and later this was developed by a large number of programmers. Now with the increase of multicores processors and development of computer vision applications there is a large demand for Opencv in several institutions and many are trying to use this library for filtering the image, camera calibration, Motion analysis and Object Tracking.

Now opencv is being downloaded by more than a million users according to statistics. It is even increasing day by day with an average of 30k downloads per month. Generally opencv is written in either C and C++ code in an optimized way and is independent of the Intel performance primitive (IPP), but it will automatically load dynamic link libraries if they are present.

2.4.1 Structure of Opencv Library:

The opencv library is mainly structured in an efficient and optimized way so that the user can use this library easily [5]. Opencv library is divided into various modules and each module is dedicated to a group of computer vision problems. The opencv library is divided into the following modules

- core
- imgproc
- video
- calib3d
- features2d
The above mentioned modules play a key role in the OpenCV library structure and let us see what each module defines and the key features of the module. The Core module comprises of all the basic data structures and the entire machine learning functions that are being used by other modules.

The image processing module is most important for the students as the functions provided in it are mainly related to filtering, Geometric image transformations, Histograms, motion analysis and feature detection. This module covers all the functions related to images under different sub categories.

The next one is video analysis module and this one deals with the subcategories like motion analysis, background subtraction and object tracking algorithm.

The module calib3d mainly deals with 3d data like camera calibration of stereo cameras and Object pose estimation and also deals with algorithms like 3d reconstruction.

The module feature 2d library deals with the salient feature detectors, descriptors and descriptor matchers.

The opencv module Object detect mainly deals with the detection of objects which can be anything like car, face, eyes, hands and so on.

The final module is high gui module and this module mainly deals with the functions that are most commonly used in opencv like for reading and writing images or video to a file. It also provides the UI capabilities that allow us to add track bars and resizing the windows and many other things can be done for the graphical user interface.

We can have a look at some of the opencv concepts which we use in the open source library. In order to use the classes and functions defined in the library can be accessed using cv or "using namespace cv". This is mostly used by the programmers to access all the functions in opencv.

2.5 Point Cloud Library (PCL)

PCL is often referred to as the point cloud library is the new technology which allows 3D perception of data. The pcl library is an advanced library that focuses on 3D data [9]. Now a days with the advancement of technology for acquiring 3D data which is widely used in robotics and also in major fields of science. The PCL is a famous open source which collected hundreds of codes and algorithms to assist developers to handle point cloud data sets for satisfying the needs from industry. By PCL, it can cut down the developing time and cost. Now a day's robots are everywhere and to see the world they need to perceive which can be provided by the point cloud library in the 3D format. The point cloud library consists
of various algorithms for different applications and these modules consist of the functions and classes that are used by millions of users depending on the application.

Not only in Robotics but also in various applications to handle efficiently the point clouds we should have a mechanism that could handle this point cloud and that is nothing but point cloud library. This PCL was developed mainly by the willow garage and later it was released under the BSD license and it is free for commercial and research uses. The PCL is written mainly in C++ language and it is structured in a way that it can be used for various applications in N-D point cloud and geometric applications.

The pcl library consists of numerous states of art algorithms each of which is used for different applications. It consists of the following algorithms that are mainly used to operate on 3D data and they include: Filtering, feature estimation, surface reconstruction, registration, model fitting. Each algorithm is defined by a base class that includes the functionality that can be used for various applications thus by keeping the implementation of algorithm compact as shown in Fig 2.7.

![Fig 2.7: PCL modules.](image)

### 2.5.1 Modular Structure of PCL

The pcl library is a collection of various modules each of which is dedicated to some application [7]. First we have the PCL filters library which mainly deals with the noise removal mechanism for 3D data filtering applications and outlier removal projections.
Pcl_features library mainly contains the mechanism that is used to estimate the 3D features from the point cloud data. 3D features are nothing but the representation at certain 3D point in space, which are nothing but the geometrical pattern in space. This library mainly helps us to find the surface normal's and curvatures in space and to find this, we have a set of points in space and we need to find the closest point in space and depending on the application we opt to use the closest point in the vicinity of p or all the points. The easiest method that we follow to determine the surface normal changes is that Eigen decomposition method.

The pcl_registration library is mainly deals with the process of registration. Here a group of datasets are combined and they are modeled using a process called registration. The main thing here is to find points in these datasets and try to minimize the error between these datasets and if the error falls below some threshold value then the process of registration is said to be complete.

The surface library is another one which mainly deals with the surface reconstruction from the 3D scans. The surface reconstruction techniques are generally convex hull, mesh representation or a smoothed resample surface with normal's. The smoothing and resampling method is used when there is a lot of noise in the cloud. Meshing is used when we want to create a surface out of the points and the other method that is convex hull which is used when only the boundaries are need to be extracted.

The Io library is similar to that of the library present in Opencv and this library mainly consists of classes and functions that are used for reading and writing data files and also capturing the point clouds from the devices.

The common library contains the common data structures and methods used by the most of PCL libraries. The core data structures include the Point Cloud class and a multitude of point types that are used to represent points, surface normals, RGB color values, feature descriptors, etc.

The visualization library was built to allow rapid prototyping and visualization of algorithms operating on 3D point cloud data. Similar to opencv’s highgui routines for displaying 2D images and for drawing basic 2D shapes on screen, the library offers: a) methods for rendering and setting visual properties (colors, point sizes, opacity, etc) for any n-D point cloud datasets b) methods for drawing basic 3D shapes on screen (e.g., cylinders, spheres, lines, polygons, etc) either from sets of points or from parametric equations) a histogram visualization module for 2D plots; d) a multitude of Geometry and Color handlers.

The Pcl kdtree library provides the kd-tree data-structure, using FLANN that allows for fast nearest neighbor searches. A kd-tree is a space-partitioning data structure that stores a set of k-dimensional points in a tree structure that enables efficient range searches.

The octree library provides efficient methods for creating a hierarchical tree data structure from point cloud data. This enables spatial partitioning, down sampling and search operations on the point data set. Each node has either eight children or no children. The root node describes a cubic bounding box which encapsulates all points. At every tree level, this space becomes subdivided by a factor.
of 2 which results in an increased voxel resolution. The octree implementation provides efficient nearest neighbor search routines, such as “Neighbors within Voxel Search”, “K Nearest Neighbor Search” and “Neighbors within Radius Search”. It automatically adjusts its dimension to the point data set. Furthermore, a memory pool implementation reduces expensive memory allocation and de-allocation operations in scenarios where octrees need to be created.

2.6 QT

Generally QT is a cross platform application and UI development framework. The name qt came from the letter ‘Q’ was chosen as it looked beautiful in the haavards Emancs font. The letter’t’ stands for toolkit and this was taken from the ‘X’ Toolkit. The company was incorporated originally as the quasar technologies and now it is Trolltech [8].

Qt is a framework written in C++ for developing GUI applications and it can be written only once and can be deployed anywhere since it is cross platform application. QT framework allows professional cross platform graphics application using c++. The Qt allows the programmers to run applications on windows Linux and Mac Os since it is a crossplatform application. This qt sdk is very portable. Qt is available under different licenses including GNU, LGPL 2.1, GNU GPL 3.0 and commercial developer license. The QT has been employed by large organizations in the world for their projects e.g. Adobe, Skype, Wolfram Google, Siemens & Volvo.

In addition to the wealthy Qt class library the developers are provided with Qt designer cross platform (IDE) Qt creator. The Qt software development kit mainly comprises of the Qt creator, Qmake and visual studio add in etc. The user chooses these tools depending on their preferences.

Qt 4 was released and it contains several libraries and the advantage with it is the developers can link only the part they needed in order to use. It contains more than 500 classes and almost 9000 functions. This version of Qt is completely different and redefined with easy to use templates and a flexible 2d painting framework, with powerful text viewing and editing classes and Qt 4 is available both in commercial and free ware for the users. Qt is being used by almost everyone in some way or the other by the developers around the world.
3 Chapter

3.1 Introduction to SPL

Student perception lab (SPL) is collection of many open source libraries integrated together. The aim of Student perception lab is to integrate open source libraries that have been written for various purposes. For example, Opencv is a library for computer vision algorithms, Point cloud library is an open scale project for 2D/3D image and point cloud processing, many of the SPL functions and classes access these libraries' functionality and wrap the functions from the library so that they can be used easily and work in a similar way.

3.2 Architecture of SPL

Spl library is developed on oops concept, as we know that oops is a programming based on the use of object and their interactions. A class is a blueprint or template or set of instructions to build a specific type of object. Every object is built from a class.

3.3 Class Hierarchy

![Fig 3.1: A simple diagram of spl library architecture](image)
In our SPL library, image processing, camera calibration, image or video analysis, visualization is considered [13]. Based on these divisions, we have spl operation, spl calibration, spl processing, spl analysis, spl visual, all these classes are derived from a base library Fig.3.1 shows the suggested library architecture by means of diagram of classes.

**Base lib:**
The core part of our spl library is the baselib; contains opencv, pcl. The core add-on for the spl library is generally wrapping of libraries rather than solving problems in a novel way. In future if we find something is missing then we can add other libraries like opengl, Glut, glew for graphics for audio we can add rtAudio, portAudio.

All the classes in the intermediate layer are grouped naturally

- SplOperation
- SplCalibration
- SplProcessing
- SplAnalysis
- SplVisualization

**SplOperation**
This class mainly deals with the mathematical operations which are related to matrices, eclipse.of both opencv and pcl libraries.

**SplCalibration**
This proposed class mainly includes the functions that are related to so-called camera calibration and are also related to 3d reconstruction.

**SplProcessing**
This class includes all the functions related to an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, and generic table-based remapping), color space conversion, histograms, and so on. This class is mostly used by the students and researchers in the lab.

**SplAnalysis**
This class includes all the functionalities related to a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.

**SplVisualization**
This class includes all the functionalities related to visualization toolkit and is very much essential class hierarchy. The visualization library was built for the purpose of being able to quickly prototype and visualizes the results of algorithms operating on 3D point cloud data. Similar to OpenCV’s highgui routines for displaying 2D images and for drawing basic 2D shapes on screen, the library offers methods for drawing basic 3D shapes. A histogram visualization module for 2D plots
SplExtendable

If we add any other base lib in the future then the base library may contain different classes and functions. So here we have the provision to add classes depending on the base lib added. For example if we add the opengl in the future then the class can be wrapped further.

SPL

This is the proposed library that we tried to implement. This library consists of splmain.h header file and using this file we can access all the functionalities.

3.4. Role of Administrator

The spl library is designed in such a way that it is flexible and also extendible. If there are any changes in the versions of opencv, pcl and qt then the main task of the administrator is that he has to update the version of the libraries released by replacing the lib files and include files in the baselib folder. After this the administrator adds all the include paths and the libs in the pro file of qt. *Note: The administrator should check the system path*. If the user wants to use some other open source library in the future then it is the administrator job to wrap all the new libraries to the base library and make the new classes available. The section below shows how to start up with qt.
4. Chapter

4.1 Implementation of spl with Qt

The Qt libraries and tools are enclosed in the Qt SDK, available in the Qt website for Windows, Linux and MacOS X. In this work we develop our application under QtFramework version 5.1.1 using the Qt Creator IDE version 2.8.1 for windows. In the following we will detail this process. First we will create a Qt project to our application which is the primary thing of the administrator as shown in Fig 4.1.

![Fig 4.1: Interface of Qt creator IDE.](image)

Let us start by creating a Qt project. Clicking at File→NewFile or Project on the Qt Creator menu bar, we see that Qt offers some project templates. In our case, we choose Qt GuiApplication Fig4.2(a) which creates a minimal application for desktop with a main window, a menu bar, a tool bar, a status bar and a central widget. Following steps depicted in, first we select the QtGui Application option, second we define the project name (b), as well as its path, third we select Desktop as the build target(c) and use Shadow Building to create a separate directory for storing the compiled sources, objects and executable files. Finally, we specify details of the main window interface class as shown in Fig4.2 (d). Qt automatically generated four files:
Fig 4.2(a)

(b)
Fig 4.2: Step by step to start up a Qt GUI project
**testsplproject.pro**: This file, depicted, contains the configurations of the Qt project. It lists the source files of the project, as well as the modules used. Initially, only two Qt modules are added: core and gui. In order to enable Qtspl Module, we add spl file testsplproject.pro, as depicted in Fig 4.3.

```cpp
QT += core gui
greaterThan(QT_MAJOR_VERSION, 4):
QT += widgets
TARGET = Testspl
TEMPLATE = app

SOURCES += main.cpp\
testspl.cpp
HEADERS += testspl.h
FORMS += testspl.ui
```

Fig 4.3: file testsplproject.pro

**testspl. {h,cpp}**: These are the files for the Testspl class in Fig 4.4 and Fig 4.5. In the declaration of this class, we notice the Qt macro Q_OBJECT, which is automatically declared under the private access modifier.

```cpp
#ifndef TESTSPL_H
#define TESTSPL_H

#include <QMainWindow>

namespace Ui {
    class Testspl;
}

class Testspl : public QMainWindow {
    Q_OBJECT

public:
    explicit Testspl(QWidget *parent = 0);
    ~Testspl();

private:
    Ui::Testspl *ui;
};
#endif // TESTSPL_H
```

Fig 4.4: testspl.h generated by Qt creator
Fig 4.5: testspl.cpp

**main.cpp:** This file contains the C++ main function. We will keep main.cpp as simple as we can as shown in Fig 4.6. We will only insert further code on it to format details of spl.

```cpp
#include "testspl.h"
#include "ui_testspl.h"

Testspl::Testspl(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Testspl)
{
    ui->setupUi(this);
}

Testspl::~Testspl()
{
    delete ui;
}
```

Fig 4.6: main.cpp

**testspl.ui:** This file contains the XML description of the user interface. This XML file is translated by the Qt User Interface Compiler (ui) to a C++ header file that contains declarations of object widgets used by the application. Fortunately, we do not need to write directly the XML file. Instead, we can edit the user interface using the Qt WYSIWYG editor: the Design mode. When double-clicking on the mainwindow.ui file listed in the Projects Pane, Qt Creator switches to the Design mode. The interface of Design mode contains: (A) components to design the UI; (B) UIWYSIWYG editor; (C) hierarchy of currently created UI components; (D) properties of the UI components. To come back to the text editor.
Fig 4.7: Edit mode

Fig 4.8: Design mode
4.2 Folder structure of spl library

SPL is an open source library that is designed mainly to provide a simple framework for the user. Here the user can access this SPL library easily as it covers all the functions related to various fields.

This library designed here wraps several commonly used libraries inside it like OpenGL for graphics [12], Opencv for computer vision and pcl for point cloud data. This library is mainly designed to be cross compatible. Now we have designed this library to work on windows platform but later this can be made to work on other platforms also. This SPL library is mainly aimed to create a lab environment for the students and researchers.

The SPL library is structured into various folders where each folder has an importance. No matter where we put the library but when we open the folder student perception lab we find a set of folders like apps, baselib, examples, html, spl. These folders comprises of the main structure of the SPL library.

baselib is the core of the spl library because this folder contains all the base libraries that have been used in the development of the spl library. This folder consists of mainly the Opencv library, pcl and Opengl libraries.

The folder examples contains example applications that illustrate the various things that spl library can be used to do. This folder is very much needed for the better understanding as the user has the flexibility to use different examples and understand them so that he can use this when the user tries to create an application.

The html folder mainly contains all the required documentation about the functions of the spl library their use and the modules that are used in this library and everything that the user needs to know about the library.

As we have discussed above in the Fig 4.9 about the folder example is provided with an empty project where the user has the scope to use this empty project with all the includes in the profile by following the procedure given below.
The Fig 5.0 above represents the code structure of our designed class library. Here it is just like a flow chart representing various things that the user can follow. First the user is provided with an empty project at the beginning which is provided with the Qt pro file and it contains the includes and everything so the user does not change this file. After this we have the main.cpp which is the source file where the code starts from here.

After this main.cpp we have the testspl.h which is the header file of our spl project and all the includes are provided in this file. The testspl.cpp is the source file which allows the user to write the code and changes can be made in this testspl.cpp. There is one ui file called testspl.ui which helps the user to create some user interface like depending on the requirement of the user he can create frames, labels, push buttons and many other things.

The next part is the spl library which contains all the classes related to various fields like the image processing, camera calibration, 3D reconstruction, input output operations and various other things that are useful for the students. The user
can change the code using any of the classes depending on his requirement criteria. Here we have used many open source libraries like the opencv ,pcl, Openscene graph etc and the extendable section shown in the flow chart above is left because in the future we can add various libraries depending on the user criteria.

4.3 Educational Experience with Class Library

The class hierarchy presented here was built primarily to provide a large number of primitives for various fields like image processing, visualization, camera calibration. The class hierarchy is also used to teach both object-oriented programming along with computer vision algorithms. Since it contains all the necessary aspects that need to be illustrated. In addition, the library is used to provide laboratory support for an image processing course [15]. Although various other commercially available open source libraries provide similar functionality, they do not offer the same code transparency as this library we developed.

The class library developed here provides good understanding of the object-oriented programming and image processing applications to electronic engineering students at undergraduate and postgraduate levels. In addition, the image processing environment provides an exciting medium for teaching and demonstrating various image processing modules like camera calibration, feature 2d, high gui etc. It is also used to support laboratory experiments.

In summary, the class library has proved an invaluable resource in providing a student centered learning experience.
5. Chapter

Results

In this section the results are presented. The simulation work was done using Opencv 2.4.6, pcl 1.6.0, visual studio c++ 2010 and qt version 5.1.1.

An image, with each pixel value has single sample then it is known as gray scale digital image. Also it conveys only intensity information. This gray scale images are also called as black -and- white, representing the non existence of a chromatic discrepancy. The class part in our algorithm is shown below which opens a color image and coverts it into gray scale.

Code:

```cpp
void improcessing::openImage()
{
    FileName = QFileDialog::getOpenFileName(this, tr("Open File"),
                QDir::currentPath(), tr("Image Files (*.png *.jpg *.jpeg*.bmp)");
    charFileName = FileName.toLocal8Bit().data();
    image = newimage.splLoadImage(charFileName,CV_LOAD_IMAGE_COLOR);//SPL FUNCTION

    OriginalImage= QImage((const unsigned char*)(image->imageData),image->width,image->height,QImage::Format_RGB888).rgbSwapped();
    ui->label->setPixmap(QPixmap::fromImage(OriginalImage).scaled(ui->label->size()));
}

void improcessing::grayImage()
{
    imgGray=newimage.splLoadImage(charFileName,CV_LOAD_IMAGE_GRAYSCALE);//SPL FUNCTION
    gray = QImage((const unsigned char*)imgGray->imageData,imgGray->width,imgGray->height,QImage::Format_Indexed8);
    gray.setPixel(0,0,qRgb(0,0,0));
    ui->label_2->setPixmap(QPixmap::fromImage(gray).scaled(ui->label->size()));
}
```
Fig 5.1: Simple gui showing the grey scale image.

The Fig 5.1 above is obtained by using our spl library based on qt. It is a simple gui. If we observe in the figure consists of two frames each enclosing a figure. One frame is for input image and the other one is for displaying the output image. We have used our spl function named `splLoadImage()` to convert colour image to gray scale.

Fig 5.2: GUI showing the canny image.
Code:

```cpp
void improcessing::openImage()
{
    FileName = QFileDialog::getOpenFileName(this, tr("Open File"),
        QDir::currentPath(), tr("Image Files (*.png *.jpg *.jpeg*.bmp)"));
    charFileName = FileName.toLocal8Bit().data();
    image = newimage.splLoadImage(charFileName, CV_LOAD_IMAGE_COLOR);//SPL FUNCTION
    OriginalImage = QImage((const unsigned char*)image->imageData, image->width, image->height, QImage::Format_RGB888).rgbSwapped();
    ui->label->setPixmap(QPixmap::fromImage(OriginalImage).scaled(ui->label->size()));
}

void improcessing::grayImage()
{
    imgGray = newimage.splLoadImage(charFileName, CV_LOAD_IMAGE_GRAYSCALE);//SPL FUNCTION
    gray = QImage((const unsigned char*)imgGray->imageData, imgGray->width, imgGray->height, QImage::Format_Indexed8);
    gray.setPixel(0,0,qRgb(0,0,0));
    ui->label_2->setPixmap(QPixmap::fromImage(gray).scaled(ui->label->size()));
}

void improcessing::cannyImage()
{
    imgcanny = imgGray;
    newimage.splCanny(imgGray, imgGray, 50.0, 200.0, 3);//SPL FUNCTION
    canny = QImage((const unsigned char*)imgcanny->imageData, imgcanny->width, imgcanny->height, QImage::Format_Indexed8);
    canny.setPixel(0,0,qRgb(0,0,0));
    ui->label_2->setPixmap(QPixmap::fromImage(canny).scaled(ui->label->size()));
}

void improcessing::sobelImage()
{
    imgsobel = imgGray;
    newimage.splSobel(imgGray, imgsobel, 1, 1, 3);//SPL FUNCTION
    sobel = QImage((const unsigned char*)imgsobel->imageData, imgsobel->width, imgsobel->height, QImage::Format_Indexed8);
    sobel.setPixel(0,0,qRgb(0,0,0));
    ui->label_2->setPixmap(QPixmap::fromImage(canny).scaled(ui->label->size()));
}
```

34
6. Chapter

Conclusion & Future work

In this section we have a class library which has been developed to support the teaching of object-oriented programming and computer vision, along with image processing. This class library is developed using the software visual studio 2010 under windows platform and can be extended to work on other platforms. The class library is developed to provide the students a QT based GUI. All platforms dependent aspects are not included in the library and so this class library can be easily exportable. The most important aspect of this library is that other libraries can be further added easily which makes it flexible.

From the performed class library a new line of work has been created with grow and future improving possibilities. The result of this thesis is a robust, simple and fast which could be implemented in any platform.

It is worth to mention that this thesis has supposed a challenge for us, because we had never worked with open source libraries and c++ programming.

The use of c++ has supposed a great idea in order to make this class library faster so the spent time setting up all the open source libraries has been very useful and it has produced good benefits.

If this project has been interesting, and want to be improved or commercialized in the following section all of the features that can be improved are described.

In the future work we can expand the base library by adding some of libraries which are capable of audio and libraries depending on the student requirement.
We can change or extend the class library structure according to student or researcher criteria.
7. Chapter

Bibliography

[1] B Stroustrup, the c++ Programming Language AT&T Laboratories
1991 .pp 361-474


(last visited 11/7/2013)


http://cmake.org/cmake/help/documentation.html
